University of Oslo
Department of Informatics

Empirical Realization Ranking

Erik Velldal

A thesis submitted for
the degree of philosophiae
doctor (Ph.D.)

December, 2007

Abstract

This thesis develops a new approach to the problem of indeterminacy in grammar-based natural language generation (NLG). The problem of indeterminacy concerns the fact that, for a given input semantic representation, the grammar might allow for several (i.e. thousands) alternative surface realizations. While the traditional approach to dealing with this problem is to rank the generated strings using a surface-oriented n-gram language model (LM), this thesis develops a linguistically informed approach based on features that are keyed to the internal structure of the realizations. The approach extends on the methodology previously used for statistical parsing and statistical unification-based grammars, and adapts it to the context of generation. This allows us to train treebank-based discriminative realization rankers based on modeling frameworks such as Maximum Entropy (MaxEnt) and Support Vector Machines (SVMs). The training data is based on the novel notion of a generation treebank, which we show how to automatically create on the basis of an existing parse-oriented treebank.

For reference, we also develop an n-gram-based LM trained on a large corpus of raw text. Our experimental results show that the use of a discriminative model trained on just a few thousand items in a generation treebank, gives significantly better ranking performance than the use of a traditional surface-oriented LM. Moreover, we show that even better results can be obtained by combining the two modeling approaches. This is done by including the LM as an additional feature in the discriminative model. Evaluation scores are reported for several data sets and using a range of different automated metrics. We also include results for a manual evaluation carried out by a panel of external anonymous judges.

The hybrid system for surface realization described in this thesis is currently integrated for target language generation in the Norwegian–English machine translation (MT) system LOGON. We also show how the realization ranker is used together with a global end-to-end reranking model for selecting the final output of the MT system.

Acknowledgments

There are many people that I am indebted to for their support or contribution to the work in this thesis. Above all I want to thank my adviser, Stephan Oepen. I cannot overstate his relentless support and commitment, coupled with his knowledge about virtually every aspect of the field. In addition to the many hours spent discussing both theoretical and technical issues, we have spent much time peer-programming many of the extensions to the [incr tsdb()] system that deal with the statistical modeling.

Oepen has also played an important role by virtue of being the technical manager for the LOGON project which the work in this thesis forms part of. Funded by the Norwegian Research Council's program for language technology (KUNSTI; *KunnskapsUtvikling for Norsk SpråkTeknologI*), the LOGON project is a collaborative effort with participants from all of the three largest Norwegian universities (Oslo, Bergen and NTNU in Trondheim). I would like to thank all the members of LOGON for their invaluable feedback on preliminary versions of the approach developed in this thesis, and for making it very enjoyable to be a part of the team. In particular I would like to Dan Flickinger for help on the ERG grammar and help on the treebanking procedures. The work in this thesis could not have been carried out without the parse treebanks provided by Flickinger. I am also grateful to Jan-Tore Lønning for his persistent support on many different matters of the thesis process, both practical and theoretical, and especially in helping out with some of the more mathematical aspects. In addition to being the project leader for LOGON, Lønning is the head of the Logic and Natural Languages (LNS) research group at the Department of Informatics, University of Oslo (UiO) where I have been working for these last few years. Thanks are due also to the rest of the members the LNS group for their support throughout the period of my thesis writing. There are several other people at UiO who have provided much useful advice and encouragement, and in particular I want to thank Liv Ellingsen, Anders Nøklestad, and Lars Nygaard.

During the work on this thesis I spent a semester at The Department of Cognitive and Linguistic Sciences at Brown University, and I am indebted to Mark Johnson for hosting my stay there. I would also like to thank the other members

of the BLLIP (Brown Laboratory for Linguistic Information Processing) group at Brown, and Eugene Charniak and Sharon Goldwater in particular.

Special thanks are due to Emily M. Bender, director of the Professional Master's in Computational Linguistics Program at the University of Washington. With the help of Bender, several MA students in the program were recruited as anonymous judges for a manual evaluation of some of the models developed in this thesis. I am also very grateful to all of the anonymous judges who took the time to participate in the evaluation.

There are many people who I would like to thank for their comments on earlier versions of the work in this thesis. First I would like to thank the other participants of the 2005 NLG course at NGSLT (The Nordic Graduate School of Language Technology), and in particular Eva Forsbom for useful discussions related to BLEU and other similarity metrics, and for generously handing over her code for computing the NEVA score. I would furthermore like to thank the members of the DELPH-IN network, and John A. Carroll and Francis Bond in particular, for many helpful discussions and for providing feedback on presentations of preliminary results. I also want to thank to the members of the Machine Learning reading group at UiO, as well as the anonymous reviewers of various conference papers. Last but not least, I am very grateful to Rob Malouf for help on the TADM package and for technical advice in relation to the discriminative models, both MaxEnt and the SVMs.

The work described in this thesis depends heavily on free software, both in the *libre* and *gratis* sense. A collective "thank you" is therefore also due to the community of open-source developers who choose to make their code freely available, free of both proprietary restrictions and cost. This pertains not only to developers of software, but more generally to developers of computational resources in a broad sense, such as corpora, annotations etc.

Finally, thanks to Tuva for providing the best reason to finally bring closure to this project—ahoy me hearty!

Technical This thesis was typeset in LaTeX 2_ε, using BibTeX with the *apacite* package for generating citations and the bibliography. All graphs and histograms have been produced using *gnuplot*, while the tree structures are based on the LaTeX packages *rtrees* and *qtree*. All other figures have been drawn using *xfig* and *dia*.

Erik Velldal, Oslo, December 2007

Contents

Chapter 1

Introduction

This thesis develops a novel approach to the problem of *indeterminacy* in grammar-based *natural language generation* (NLG). More specifically, we are working with the task known as *surface realization*, which deals with mapping an abstract semantic representation into a surface string in a natural language such as English. The problem of indeterminacy concerns the fact that there will typically be several such alternative surface realizations corresponding to a given semantic representation. All of these alternative realizations will be what we regard as *paraphrases*, meaning that they are all semantically equivalent, at least in a strict truth-conditional sense. Although all of these competing paraphrases will also be well-formed according to the underlying grammar, some of them will usually sound much more natural and fluent than others. For the symbolic generation system considered in this thesis, we often see several hundreds and thousands of different surface strings generated for a token input semantics. In order for the system to be practically useful, then, we need a principled way of ordering the alternative hypotheses, and selecting the final output string to be presented to the user. This is the task we refer to as *realization ranking*.

The traditional approach to this problem is to score and rank the surface strings using a generative n-gram-based language model (LM). A standard n-gram model is purely sequential and surface-oriented, and conditions the probability of each word in a sentence only on the $n-1$ preceding word forms in the sequence. We here present an alternative and linguistically informed approach based on discriminative models trained[1] on treebank data. By developing the notion of a *generation*

[1] A brief note on terminology: The use of terms such as *learning* and *training* might seem strange to readers unfamiliar with the lingo of empirically oriented NLP. Many of the methods used in this thesis are related to the field of *machine learning* (ML). This is a field that is concerned with data-driven models that can, in a loose sense, be said to *learn* from experience. By this we mean that they are designed to pick up on statistical (i.e. distributional) regularities in the provided data. Based on these statistical estimates, the model or *learner* should be able to make generalizations

treebank, we are able to adapt and extend on the methodology of state-of-the-art statistical parsing and statistical unification-based grammars, thus making it applicable to the context of NLG. This allows us to train discriminative models for realization ranking in a similar manner as when training models for statistical parse disambiguation. Instead of just considering the linear sequence of surface forms as in traditional LMs, our treebank models use features that are keyed to the internal structure of the realizations.

The specific surface realization module we are working with is currently used for target generation within the Norwegian–English machine translation system LOGON (Oepen et al., 2004; Lønning et al., 2004). The LOGON project has developed a working prototype system for high-quality automatic machine translation (MT) of text within the domain of Tourism, and the general aim is for high precision rather than for very broad coverage and robustness. Moreover, the core of the LOGON system is founded on symbolic and rule-based methods, implementing an approach based on deep linguistic analysis. However, as the problems of indeterminacy and ambiguity arise at several stages throughout the translation pipeline, these rule-based methods are furthermore complemented by data-driven methods for statistical ranking and choice. The main focus of this thesis, however, is restricted to the problem of ranking generator outputs. Although the particular generator we use is embedded in an MT system, our ranking task is restricted to the context of generation. This is an important distinction. It means that we are not concerned with ranking sentences as translations of a foreign source sentence, but rather as realizations of a semantic representation. This restriction in scope also makes our goal more general: We are working with the problem of indeterminacy in natural language generation in a broad sense, not necessarily confined to the context of generation for MT. An exception to this restriction is when we in Chapter 9 briefly discuss a model for end-to-end reranking for MT. This discussion also complements the core theme of the thesis by showing briefly how the hybrid generator we develop is integrated into the overall MT pipeline, and how realization ranking interacts with other system components and the reranking. The presentation of the MT reranker also demonstrates additional applications of the modeling techniques used throughout the thesis.

The remainder of this introductory chapter is organized as follows. In the next section we first give a high-level introduction to the general task of natural language generation, including a presentation of the specific generator used within the LOGON MT system. Section 1.2 provides a discussion of the problems of indeterminacy in relation to wide-coverage grammar-based generation, which is the

and predictions when faced with new and unseen data. While the terms *training* and *learning* are routinely used for referring to this estimation process, *training data* refers to the data that forms the empirical basis for the estimation.

general problem we attempt to tackle in this thesis. The basic foundations for our approach are described in Section 1.2.1, which also gives a general discussion about the issue of statistically guided choice in parsing and generation. In Section 1.3 we present an outline of the thesis itself, briefly summarizing the various chapters. Finally, in Section 1.4, we summarize some of the most important contributions and achievements, and also look at some examples of work by other colleagues in the field that represent continuations of the ideas developed in this thesis.

1.1 Natural Language Generation

As a sub-field of *natural language processing* (NLP), *natural language generation* (NLG) is concerned with making computers produce utterances in a natural language, such as English or Norwegian, on the basis of some computer-internal representation. When trying to define the basic task of NLG, Reiter and Dale (2000) start off by contrasting it with *natural language understanding* (NLU). Intuitively, the two processes can be understood as inverses of each other. While NLU concerns a mapping from natural language to a computer-internal representation, NLG concerns a mapping from such a representation into natural language (Reiter & Dale, 2000, sec. 1.1.1). In NLU we are given an utterance, and the task is to analyze its syntactic structure and interpret the semantic content of that utterance. In NLG on the other hand, we move in the other direction. The goal is to produce a natural language utterance that best expresses the meaning that we want to convey.

NLG systems come in many different shapes and sizes, from the very simple to the very complex and sophisticated. The simplest systems rely on so-called *canned text*, where generation is largely a matter of retrieving predefined stored text. *Template-based systems* are a notch more sophisticated. While still making heavy use of stored text, they may also include simple transformations and filling-in of "blank" (i.e. variable) fields or placeholders. The more advanced systems are so-called *feature-based* systems, which provide a more flexible approach (Hovy, 1997). Distinctions within the language (such as tense, utterance type, etc.) are encoded as features or attributes. This class of NLG approaches includes the *grammar-based* systems, where linguistic constraints and the relations between meaning and form are encoded in an externally specified grammar. This type of generation is sometimes also referred to as *syntactic generation* (van Noord & Neumann, 1997), and the input is typically specified as an abstract logical-form semantic representation. As we shall see, the particular NLG system underlying the work described in this thesis is an instance of this type. Note that, there is also a difference between systems as to whether they perform *multi-sentence* or

single-sentence generation. For the former type, the aim is to generate coherent discourse or running text, while the latter limits its attention to individual utterances, irrespective of the larger discourse context. It is this latter type that we will be focusing on in this thesis.

In the setting of natural language generation it is common to distinguish between two main stages in the process pipeline: *content planning* and *surface realization* (Busemann, 1995). The first stage is sometimes also called *deep generation*, and is concerned with determining the specific content to be communicated, and structuring the relevant information into an abstract representation. Within the planning phase, some researchers also distinguish between *document planning* or *text planning* on the one hand, and *sentence planning* or so-called *microplanning* on the other (Reiter & Dale, 2000, ch. 5). The first stage would then take care of planning the actual content, perhaps consulting a knowledge base, splitting the text into sections and paragraphs, ensuring text coherence, and so forth. The second phase would take care of tempus, voicing, agreement, aggregation[2], and so forth.

Now, given the abstract representation produced by the planning phase, the realization stage then maps this into a linguistic structure and a surface text. In other words, surface realization takes us from an abstract text specification to an actual sequence of words in a natural language.

The first stage of generation is typically described as being concerned with *what to say*, while the second is concerned with *how to say it*. This broad division of labor is sometimes also referred to as *strategical* versus *tactical* generation. Note that the separation of the two subtasks is by no means sharp and clear-cut, and systems will differ with respect to how they organize the flow of events: Some might decide on syntactic structure or lexical choice in the planning phase, while others include this as part of the realization phase. It will also of course be possible to further divide the process into smaller chunks of more specialized subtasks. Some systems might even have a speech synthesizer at the end of the pipeline. Finally, generators also differ with respect to the level of integration between the different modules and some might also include various feedback mechanisms, making the flow of control more complex than the simple pipeline architecture suggested above.

1.1.1 Generation in LOGON

The embedding context of the generation task we are working with in this thesis is that of the LOGON MT system for Norwegian-to-English translation. The over-

[2] Aggregation refers to the process of removing redundancy in the text, so as to make it shorter, more readable, and less repetitious (Dalianis, 1996).

all architecture of the system builds on a *semantic transfer* approach which can be broken down into three main components: (i) First, a syntactic and semantic analysis of a Norwegian source sentence yields language-specific logical-form semantic representations. (ii) A transfer step then maps these representations into translationally equivalent language-specific English representations. (iii) Finally, the transferred semantic representations are passed to the generator to produce English sentences. Moreover, the semantic representations themselves are based on the framework of *Minimal Recursion Semantics* (MRS; Copestake, Flickinger, Malouf, Riehemann, & Sag, 1995; Copestake, Flickinger, Pollard, & Sag, 2006). As described in Section 4.2, an MRS representation provides a relatively "flat" logical-form semantic structure, and an important property of MRS is its support for *underspecification*, especially with respect to scope relations. The actual generation itself is carried out by the generator component of the open-source grammar engineering system *Linguistic Knowledge Builder* (LKB; Copestake, 2002). Developed by Carroll, Copestake, Flickinger, and Poznanski (1999) and Carroll and Oepen (2005), this is a chart-based, lexically-driven surface realizer that can generate sentences from (possibly underspecified) logical-form semantics such as MRS. As the final component in the LOGON MT pipeline, the generator produces English target strings on the basis of MRS representations as transferred from the source analysis. The English grammar that governs this generation is the *LinGO English Resource Grammar* (ERG; Flickinger, 2002). This is a general-purpose and wide-coverage lexicalist grammar specified using the unification-based framework of *Head-Driven Phrase Structure Grammar* (HPSG; Pollard & Sag, 1994), and includes MRS for capturing semantics. Having been developed in a completely declarative way using a typed feature-structure logic, the grammar furthermore has the property of being bidirectional and can be used for both parsing and generation.

In terms of our general overview of different types of generation systems above, we see that the specific generator we assume here deals with so-called tactical generation or surface realization, as most of the decisions corresponding to the planning phase will be given by the analysis of the source text. In other words, the decisions concerning *what* we want to say are largely specified by the source language utterance and the subsequent transfer of semantic representations. The task of specifying *how* we want to say it then proceeds by realizing the semantic MRS representation as a well-formed English surface form according to the ERG grammar. In other words, our particular generation task can be further categorized as grammar-based generation from logical forms. Now, there is a general problem that faces natural language generators of this type. This is the problem of *indeterminacy*. As the grammar will typically license a set of candidate grammatical strings for each semantic input, the generation process itself is non-deterministic. This is in contrast to many of the smaller and simpler

types of NLG systems, where the constraints in the symbolic or rule-based system fully specify the output, meaning that the system only constructs a single candidate sentence for a given input. However, this deterministic mode of generation is not flexible enough to be suited for large-scale generation with a wider coverage of linguistic phenomena. In the next section we shall see several examples of how wide-coverage grammar-based NLG systems need to deal with the problem of *choice* before they can finally present the user with a single output sentence.

1.2 Indeterminacy

As is commonly accepted, there are usually many ways to express a given meaning in a natural language, some more effective or natural-sounding than others. Although this is most certainly true for us as human language users, it is also true for grammar-based NLG systems such as the LKB generator. For a given input MRS, the generator will usually come up with a choice of several possible paraphrases. Figure 1.2 shows some examples of alternative outputs when generating from a single (underspecified) MRS using the LinGO ERG: While the grammaticality of all the realizations is guaranteed with respect to the underlying grammar, clearly some outputs are far more fluent than others. In the data sets used in the current study, the input semantics can optionally be underspecified with respect to information structure, which means that all grammatically allowed passive and topicalized constructions will be included in the set of possible paraphrases. Other issues that give rise to indeterminacy in generation are, among others, optional complementizers and relative pronouns, ordering of intersective modifiers, and lexical and orthographic variations.

The information that a grammar provides us, is exactly that which is suggested by its name; grammaticality. However, there will usually be many different grammatical sentences that are semantically equivalent (in truth conditional terms). Grammaticality alone then, is not enough to determine what humans judge "natural", "fluent", or "right." As pointed out by, among others, Abney (1996), whereas *grammaticality* (at least in computational terms) is an absolute or two-valued property (a given sentence either is or is not grammatical according to a given grammar), *naturalness* is a matter of degree. Moreover, as the coverage and scope of the underlying generation grammar increases, so does typically also the number of realizations that it can produce for a given meaning. In the introduction to an overview chapter on NLG, and under the heading *Significant Gaps and Limitations*, Hovy (1997, sec. 4.1.2) makes the following claim in relation to what he calls *generation choice criteria:*

> *Probably the problem least addressed in generator systems today is the one that will take the longest to solve. This is the problem of guid-*

The panorama gains grandeur as you approach the summit.
The panorama gains grandeur as the summit, you approach.
The panorama, as you approach the summit, grandeur is gained by.
The panorama gains grandeur as by you, the summit is approached.
The panorama, as by you, the summit is approached, grandeur is gained by.
The panorama, as the summit is approached by you grandeur is gained by.
Grandeur, as you approach the summit, the panorama gains.
Grandeur, the panorama gains as you approach the summit.
Grandeur, the panorama gains as the summit, you approach.
Grandeur is gained by the panorama as you approach the summit.
Grandeur, the panorama gains as the summit is approached by you.
Grandeur, as by you, the summit is approached, the panorama gains.
Grandeur, as the summit is approached by you, the panorama gains.
Grandeur, the panorama gains as you, the summit is approached by.
Grandeur is gained by the panorama as the summit is approached by you.
Grandeur is gained by the panorama as by you, the summit is approached.
As you approach the summit, grandeur, the panorama gains.
As you approach the summit, the panorama gains grandeur.
As you approach the summit the panorama gains grandeur.
As you approach the summit, grandeur is gained by the panorama.
As you approach the summit, the panorama, grandeur is gained by.
As you approach the summit, by the panorama, grandeur is gained.
As you, the summit is approached by, by the panorama, grandeur is gained.
As you, the summit is approached by, the panorama, grandeur is gained by.
As the summit, you approach, grandeur, the panorama gains.
As the summit is approached by you, grandeur, the panorama gains.
As the summit is approached by you, the panorama gains grandeur.
As the summit is approached by you, grandeur is gained by the panorama.
As the summit is approached by you, by the panorama, grandeur is gained.
As the summit is approached by you grandeur is gained by the panorama.
By the panorama, as you approach the summit, grandeur is gained.
By the panorama, grandeur is gained as you approach the summit.
By the panorama, as the summit is approached by you, grandeur is gained.
By the panorama, grandeur is gained as the summit is approached by you.
By the panorama, as is the summit approached by you, grandeur is gained.
By the panorama, as the summit is approached by you grandeur is gained.

Figure 1.1: Example of alternative surface realizations when generating from an MRS using the LinGO ERG. Unless the input semantics is specified for aspects of information structure (e.g. requesting foregrounding of a specific entity), paraphrases include all grammatically legitimate topicalizations and passivizations. Other sources of generator indeterminacy include, for example, the optionality of complementizers and relative pronouns, permutation of (intersective) modifiers, and lexical and orthographic alternations. (Note that this example only shows a small subset of the sentences actually generated for this item.)

> *ing the generation process through its choices when multiple options exist to handle any given input.* [...] *As long as generators remain fairly small in their expressive potential then this problem does not arise. However, when generators start having the power of saying the same thing in many ways, additional control must be exercised in order to ensure that appropriate text is produced.*

For the data sets used in this thesis, the generator will often come up with hundreds—sometimes even thousands—of different candidates for a given semantic input. Moreover, the number of alternative realizations is expected to further increase as the coverage of the system is broadened and as the system is extended to generate from packed, ambiguous transfer outputs. It is therefore essential to have a principled and scalable method for selecting the final target realizations. As we discuss in the next section, one of the central aims of this thesis is to build specialized statistical models that can provide a handle on this problem.

1.2.1 Statistical Choice

Over the last twenty or so years, the use of data-driven or empirical methods has surged within the field of natural language processing (NLP). One of the reasons for this trend that is particularly relevant for our setting, is that there are many language problems for which it is very difficult, or impossible even in principle, to manually define a solution in terms of explicit rules or hard constraints. There are many phenomena in natural language that simply cannot be described in terms of absolute dichotomies of right or wrong, and where it is more fruitful to think in terms of soft constraints and a graded continuum of appropriateness. Trying to differentiate the alternative paraphrases hypothesized by the generator provides us with exactly such a case. When faced with alternative orderings among attributive adjectives, for example, or a particular lexical choice, it is often impossible to state precisely why one variant sounds better than the other. Empirical methods can provide us with a principled way of modeling this type of uncertainty or gradedness.

Compared to other fields within NLP, there is not a very long tradition for empirical methods in NLG. The work that is widely regarded as pioneering the use of more empirically oriented approaches in NLG is the development of the hybrid Nitrogen system (Knight & Hatzivassiloglou, 1995; Langkilde & Knight, 1998a) in the late 1990s. Nitrogen implements a so-called *generate-and-select* set-up. First the generator constructs a set of alternative hypotheses, and then a statistical model is applied for scoring the corresponding surface strings. In the approach of Langkilde and Knight (1998a), the statistical model is an n-gram language model (LM), and the corresponding string probabilities are taken to indicate "flu-

ency". As described in Section 2.1, an n-gram model factorizes the probability of a sentence into the product of the individual word probabilities, and each word probability is only conditioned on the $n-1$ words preceding it in the sequence. Note that, in the generate-and-select approach of Nitrogen, the strings are scored according to a *bigram* model (i.e. $n = 2$).

In addition to its successor HALogen (Langkilde, 2002), the Nitrogen approach of using n-gram statistics has later been used in many other NLG systems, such as those described by Bangalore and Rambow (2000), Ratnaparkhi (2000), White (2004), Habash (2004), and others. These, and many other systems, are reviewed in Chapter 3 of this thesis. One advantage of using n-gram LMs is that they are relatively easy to estimate, and they can be trained on "raw" unannotated text. However, there are also many limitations inherent to the n-gram approach. The most obvious such limitation, as already pointed out by Langkilde and Knight (1998b), is that an ordinary n-gram language model cannot capture long-range dependencies and dependencies between non-contiguous words. An important part of this problem is, of course, the fact that a purely surface oriented n-gram model will fail to capture dependencies that show a structural rather than sequential regularity. The deeper structures of the strings are ignored entirely. Neither can the model capture dependencies that hold between more than n words. These are some of the reasons why it seems reasonable to assume that the quality of the generator rankings can be improved if we aim to go beyond the abilities of the standard n-gram models, and try to incorporate more information about the linguistic structure of the realizations.

Compared to NLG, models for statistical selection have received a lot more attention within the area of NLU or *parsing*. In many ways, the field of statistical parsing is much more mature than the field of statistical generation, and statistical parse selection models have proved especially well-suited for capturing soft constraint that are difficult to encode directly in the grammar or to define in terms of explicit rules. In our case, working with grammar-based generation using a linguistically fine-grained and wide-coverage HPSG grammar such as the ERG, there are certain areas within statistical NLU that immediately stand out as particularly interesting. The work on learning *stochastic unification based grammars* (SUBGs), as pioneered by Abney (1997) and Johnson, Geman, Canon, Chi, and Riezler (1999), are among these. As any large-scale wide-coverage grammar of a natural language is destined to be massively ambiguous, there is an immediate need to be able to efficiently order the various hypotheses in a systematic way. Johnson et al. (1999) show how *conditional log-linear models* can be used for efficiently estimating statistical parse disambiguation models for large-scale unification-based grammars. As further described in Section 2.3, log-linear models are defined in terms of *feature functions* that can be designed to record arbitrary properties of the structures that we are interested in modeling. Moreover, in relation to parse

selection, the models are furthermore estimated on the basis of *treebanks*. Generally speaking, a parse treebank is a corpus where strings have been annotated with grammatical structure. In the case of treebanks based on unification grammars, the sets of available parses licensed by the grammar for each string have typically been manually disambiguated in order to indicate which is considered to be preferred or optimal. In the work by Johnson et al. (1999), a discriminative or conditional model is then estimated to maximize the probability of the preferred parses relative to all the other non-preferred parses, while the features are defined over the grammatical productions in the trees. As noted by Johnson et al. (1999), there is typically an infinite number of possible SUBGs corresponding to a given unification-based grammar. Depending on the particular feature functions that we use and the corpus used for training, different models will result.

Note that, while the particular SUBG estimated by Johnson et al. (1999) is couched in the framework of *Lexical Functional Grammar* (LFG; Kaplan & Bresnan, 1982), the overall approach generalizes also to other unification-based formalisms. Toutanova, Manning, Flickinger, and Oepen (2005) follow a similar approach when training discriminative log-linear models for parse selection on the HPSG-based *Redwoods treebank* (Oepen et al., 2002). This treebank is annotated in accordance with the ERG, i.e. the same grammar that we use for generation within LOGON, including semantic analysis in the form of MRS.

Now, there are several similarities between the tasks of ranking parses and ranking realizations. The former task is the problem of selecting a preferred string given an input analysis, while the latter is the problem of selecting a preferred analysis given an input string. We see that there is a relation of *inverse similarity* between these two problems. While parsing attempts to recover the underlying meaning and structure of a given surface utterance, generation attempts to express a given meaning as a surface utterance. In both directions of processing, however, the underlying grammar will usually license many possible hypotheses, and correspondingly there is a need for ordering these hypotheses in a principled and systematic manner. We see that the two ranking tasks parallel each other closely, and in both cases the goal is to find the optimal output structure under some set of constraints. Recognition of this similarity between parse ranking and realization ranking is at the heart of the approach we develop in this thesis. As an alternative to the linear n-gram models that have traditionally been used for statistical generation, we will here instead draw inspiration from the existing and well-established methodology of statistical parsing and try to adapt this for the task of statistical generation.

Given the good results achieved by Toutanova et al. (2005) for parse selection on the Redwoods treebank, based on the same underlying grammar as used for target generation in LOGON, this work provides us with a natural starting point. Moreover, the treebanks developed within the LOGON project also instantiates

the characteristic Redwoods-approach to treebanking, as further described in Section 5.2. Note, however, that there are some essential changes that must be made with respect to the structure and information in the treebanks before we can use them for training discriminative models for realization ranking.

As noted above, the discriminative parse selection models are trained by maximizing the probability of all the preferred analyses relative to all the alternative and non-preferred analyses. This gives us a statistical model for the distribution of parses conditioned on a given input string. For the purpose of realization ranking, however, we are interested in modeling a somewhat different distribution, viz. the distribution of strings given the semantics. In analogy to the approach described above, estimating such a model would mean maximizing the probability of the preferred realizations relative to all the alternative and non-preferred realizations. However, as there is an implicit directionality inherent to the annotations of traditional parse-oriented treebanks, they do not immediately offer the kind of training data we require. The optimality relations encoded in these treebanks are conceived as mappings *from* strings *to* analyses. As we argue in Chapter 5, however, for the purposes of training a statistical "generation grammar", it seems reasonable to make the assumption that the treebanked strings can also be treated as optimal realizations of the treebanked semantics. In other words, the suggestion is to view the optimality relations in the treebank as *bidirectional* or *symmetric*. What this effectively means in practise, is that we take the original sentences in the corpus to define the reference realizations for the corresponding treebanked semantics. Now, recall that the MRS component of the HPSG annotations in the Redwoods-style treebanks can also be used as input to the LKB generator. As a next step then, we can take the semantics of the originally treebanked analysis and exhaustively generate all the possible paraphrases that express this meaning, as licensed by the underlying grammar. The paraphrases matching the original string in the underlying corpus will be labeled as the optimal or preferred candidate. In sum this gives us the training data necessary for estimating a discriminative realization ranker. As described in more detail in Chapter 5, we refer to this type of extended treebank resource as a *symmetric treebank* (Velldal, Oepen, & Flickinger, 2004). When we are only interested in the set of relations relevant to the problem of realization ranking, we sometimes also use the term *generation treebank* (Velldal et al., 2004).

Similarly to the parse selection models developed by Toutanova et al. (2005), the treebank models for realization ranking we develop here will use structural features defined over the grammatical derivation of the realizations. The main type of models we will develop is the class of conditional log-linear models known as *maximum entropy* (MaxEnt) models. However, we will also be experimenting with another type of feature-based discriminative learning framework, namely *support vector machines* (SVMs). We here follow the approach described

by Joachims (2002) for learning *ordinal ranking functions*, based on a generalization of maximum-margin SVM classifiers.

Of course, we would also like to assess the performance of the linguistically informed treebank model relative to a traditional surface-oriented n-gram LM. We therefore train and test a series of different LMs, leading up to a 4-gram model trained on the a plain (unannotated) text version of the 100-million-word *British National Corpus* (BNC). The results presented in Chapters 7 and 8 show that a MaxEnt model trained on an in-domain generation treebank with a few thousand items gives substantially better ranking performance than the 4-gram BNC LM. However, in order to take advantage of the different strengths of the respective modeling frameworks, we also train a version of the MaxEnt model that includes the scores of the language model as a separate feature. This combined model obtains significantly better results than any of its individual component models alone. Some of the main results for these experiments are summarized in Section 1.4.

1.3 Outline of the Text

The purpose of this section is just to give an overview of how the thesis itself is structured, highlighting the main points of the different chapters and sections. In the section that immediately follows, we close off this introduction by summarizing the main contributions of the thesis. In addition to giving a concise presentation of the most important developments and results, we also look at examples of work by other researchers where the methodology presented here is applied to other languages and other grammatical frameworks. In Chapter 2 we start off with a brief introduction to the mathematical foundations of the various statistical modeling frameworks that we use, starting with the approach of *n-gram language models* (LMs) in Section 2.1. After presenting the basic properties of these models, including issues such as *data sparseness* and *smoothing*, we also discuss some of the problems or limitations that are typically associated with LMs. Section 2.2 then gives a brief introduction to some basic concepts from *information theory*, leading over to the framework of log-linear models or *maximum entropy models* (MaxEnt) in Section 2.3. We here discuss issues such as *conditional log-likelihood estimation* and *regularization*. In Section 2.4 we give a brief introduction to the framework of *support vector machines* (SVMs). Similarly to MaxEnt, this is another type of discriminative models that has quickly gained prominence in the NLP community in the recent years. We start by looking at the usual notion of SVMs as *maximum-margin binary classifiers*, and then go on to see how they can be generalized for learning preference relations from *ordinal ranks*. Note that Chapter 2 is not meant to give a complete introduction to any of these modeling frameworks,

but rather to provide some minimal background for an uninitiated reader.

In Chapter 3 we review some of the previous work within the field that is relevant to our approach to realization ranking. Naturally, the bulk of this chapter concerns previous research on *statistical generation*, which we present in Section 3.2. However, we also look at research from other sub-fields such as *statistical parsing* and *statistical machine translation* (SMT). As we shall see, there are many similarities between the ranking tasks that arise in relation to parsing, SMT, and NLG, and an important motivation for the approach developed in this thesis is to take advantage of some of these similarities. Most relevant in relation to parsing, as we also touched upon above, is the work done on *stochastic unification based grammars* (SUBGs), and in particular the work by Toutanova et al. (2005) on training discriminative log-linear models on the *Redwoods treebanks*, which we describe in Section 3.3.2.

Chapter 4 gives a more detailed presentation of the architecture of the larger LOGON MT system. In addition to taking a closer look at the LKB generator, we also look at the other system components devoted to analysis and transfer. We provide some more background on the semantic representation language of MRS, the ERG grammar, as well as the [incr tsdb()] system (Oepen, Netter, & Klein, 1997; Oepen & Flickinger, 1998). As we shall see, the latter system plays an important part in the organization of both the system development and the annotated resources within LOGON.

As mentioned above, one of the starting points of the approach developed in this thesis is the previous work on statistical parse selection on the Redwoods treebanks (Toutanova et al., 2005). We also mentioned that the development of the in-domain treebanks within the LOGON project also instantiates the Redwoods-approach to treebanking. In Chapter 5 we describe some characteristics of the Redwoods treebanks in more detail. This chapter also explains the process of how we can use a parse-oriented treebank to bootstrap the automatic creation of a *generation treebank*.

In developing the various rankers, a fair amount of software design and implementation had to be carried out, in order to add support for large-scale statistical modeling to the open-source [incr tsdb()] system. In Chapter 6 we describe our so-called *experimentation environment*, which broadly refers to the collection of utilities for training and testing the various statistical models investigated in this thesis, all implemented as extensions to the underlying [incr tsdb()] system. The first part of the chapter mainly concerns the issue of making large-scale model tuning computationally tractable, particularly in the context of using large feature sets (on the order of several hundred thousands, and even millions, of distinct features) extracted from treebanks. The latter half concerns various aspects of *evaluation*. In this relation, Section 6.3 describes the selection of *scoring metrics* that we use when evaluating ranker performance, such as exact match accuracy

and the string-similarity metrics word-accuracy (WA) and NEVA (Forsbom, 2003). Section 6.4 furthermore describes the various hypothesis tests that we use for testing the statistical significance of any observed differences in these scores.

Chapter 7 is somewhat more hands-on in spirit, as it describes the actual development of the various realization ranking models. We start off in Section 7.1 with the experiments with generative n-gram language models trained on the BNC. In Section 7.2 we then go on to present the discriminative maximum entropy models trained on the features defined over our generation treebanks. Evaluation results for all models are presented as we go along, testing on the development data. Building on a manual error analysis in Section 7.2.5, contrasting the LM and MaxEnt rankers, Section 7.2.6 presents a *combined* model where the LM scores are added as a separate feature in the MaxEnt model. In Section 7.3, we finally describe the development of SVM rankers, using the same feature set as the combined MaxEnt model.

Given the best performing rankers from the respective modeling frameworks that we experiment with in Chapter 7, the final evaluation of their performance is presented for a set of held-out test data in Chapter 8. Complementing the usual automatic evaluation, Section 8.1 also presents results of a manual, human evaluation effort, carried out by a panel of external and anonymous judges (the "questionnaire" or evaluation form that was used for this task is included in Appendix A).

As said, the focus of this thesis is on extending a symbolic generator with a statistical layer for dealing with the problem of indeterminacy. Although the generation system is also embedded in a larger Norwegian-to-English MT system, we have emphasized that the models we develop are designed for ranking realizations, not translations. However, as we shall see in Chapter 9, the realization ranking also plays an important part in the subsequent ranking of target translations, which we refer to as *end-to-end reranking*. As described in Chapter 9, and also Section 4.4, target generation is not the only component in the MT pipeline that is prone to non-determinism. Both source language analysis and semantic transfer in LOGON may output multiple hypotheses, and both components are therefore equipped with their own statistical layers for ranking and selection. For each component, the statistical ranking is used for extracting n*-best lists* of the top-ranked candidates, which are then passed on to the next component downstream in the pipeline. This cascading n-best mode of operation greatly reduces the number of end-to-end hypotheses considered by the system. At the end of the process chain, after generating n-best lists of target realizations, we apply a discriminative reranker in order to choose a final translation. In addition to using the scores of the per-component rankers, the reranking model also includes several global features of the source and target sentence pairs, as described in Section 9.4.1. Chapter 9 also gives a brief presentation of the MRS ranking module of the transfer compo-

nent. The development of the parse ranking machinery will be presented in less detail, however, as this did not form part of the work carried out in the scope of this thesis. We finally conclude the thesis with a summary of the main results in Section 10.

1.4 Main Results and Contributions

This thesis introduces a new approach for developing statistical models for ranking the surface realizations produced by a grammar-based natural language generator. The approach extends on the methodology previously applied for statistical parse selection and adapts it to the problem of selecting generator outputs. In doing so, we introduce the notion of a *generation treebank*. These are treebanks where semantic representations are paired with their corresponding sets of paraphrases as licensed by an underlying grammar, with labels indicating which paraphrases are considered optimal for each semantics. We also show how a standard parse-oriented treebank, annotated with a declarative and constraint-based grammar, can be extended to a generation treebank in a fully automatic way, proposing to refer to the totality of such an extended treebank resource as a *symmetric treebank*.

On the basis of an in-domain generation treebank of roughly four thousand items, we successfully train discriminative realization rankers using the frameworks of both maximum entropy modeling (MaxEnt) and support vector machines (SVMs). Our rankers are trained using structural features defined over grammatical derivation trees. In order to compare the performance with previous approaches to statistical generation, we also train a 4-gram language model (LM), using an unannotated version of the 100 million word general-domain BNC. Evaluation results for several data sets show that the discriminative treebank models consistently outperform the traditional generative n-gram models. When testing the models on held-out data we see that the LM obtains an exact match accuracy of 52.60%, contrasted with 71.31% for the MaxEnt ranker. The MaxEnt model also outperforms the LM according to string-similarity metrics such as word accuracy and NEVA (a modified version of BLEU; Papineni, Roukos, Ward, & Zhu, 2002). Note that the average string length in this data is roughly 17.5 tokens, while the average number of realizations per item is roughly 52.8. However, we also present parallel results for another version of the same data set, where the attributes related to information structure (IS) are underspecified. This leads to an even greater degree of generator indeterminacy, yielding an average of 116 realizations per input semantics (with a maximum of 3360 realizations generated for a single item). On the IS-underspecified version of the held-out data the difference in ranking performance is even more pronounced: The LM obtains 48.71% accuracy, contrasted

with 71.20% for the treebank model. All the differences in evaluation scores are found to be statistically significant at the level of $\alpha = 0.05$ using a two-tailed and paired application of the non-parametric Wilcoxon singed-rank test and sign-test.

The relative differences in ranking performance for the sequential language model and the discriminative treebank model are also confirmed through a human evaluation effort carried out by a panel of anonymous judges, all of which are native speakers of English. With the kind assistance of Prof. Emily M. Bender, a group of seven MA students within the *the Professional Master's in Computational Linguistics Program* at the University of Washington, were recruited to judge the relative quality of alternative generator outputs as selected by the respective models. The results of this evaluation clearly indicate that the judges found the sentences selected by the MaxEnt model to be of better quality than those selected by the LM. Various tests also show that the level of inter-judge agreement is high. On the system-level, considering the total average rank values assigned by the judges, we get a Spearman rank correlation coefficient of $\rho = 1$, meaning that the judges unanimously agreed on the relative ranks of the models.

A manual error analysis contrasting the LM ranker and the MaxEnt ranker shows that the models commit many non-overlapping errors of different types. In order to take advantage of the different strengths of the respective models we therefore also train a version of the MaxEnt model that includes the scores of the LM as a separate feature. This combined model performs better than any of the two individual models alone, achieving an exact match accuracy of 73.98% on the held-out data (72.21% for the IS-underspecified version). When tested using ten-fold cross-validation on the development data, the combined log-linear model obtains 74.25% accuracy (on the standard version), compared to 53.75% for the LM.

Using the same feature configuration as in the combined MaxEnt model, we also experiment with various types of SVMs for learning ordinal ranking functions, following the approach of Joachims (2002). However, the best performer—a linear SVM based on binary ranks only—is not able to outperform the best performing MaxEnt model, although the differences are not detected as statistically significant. In sum we see that, for the task of ranking generator output, discriminative models estimated on generation treebanks significantly improve on the performance of n-gram language models.

Much of the work carried out for the completion of this thesis has been related to extending the [incr tsdb()] system, improving its support for large-scale experimentation with statistical learners. The discriminative models developed in this thesis have been trained on data sets comprising several hundred thousands, and even millions, of distinct features extracted from treebanks. One of the perhaps most important additions to the experimentation environment has therefore been the implementation of various *caching mechanisms* for efficient handling of rich

feature sets. This makes it computationally tractable to run batch experiments (e.g. "grid" searches for parameter tuning) on comparatively large data sets. In addition to the various levels of caching, we have also implemented support for various scoring metrics, significance tests, various modes of n-fold cross-validation, controlled batch experimentation, as well as integration with different external packages for model estimation such as the CMU SLM toolkit (Rosenfeld, 1995), TADM (Malouf, 2002), and SVM^{light} (Joachims, 1999).

The work of this thesis forms part of the LOGON project for high-precision Norwegian–English machine translation (MT). The project is funded by the Norwegian Research Council's program for language technology (KUNSTI; *KunnskapsUtvikling for Norsk SpråkTeknologI*), and represents a collaborative effort with participants from the three largest Norwegian universities (Oslo, Bergen, and Trondheim). The generation system that we describe here is currently in active use for target language generation in the MT prototype system developed within LOGON. In this context the generator operates in a mode where it produces n-best lists which are then further reranked using a global discriminative log-linear model (Oepen et al., 2007). Note that the n-best lists of realizations are efficiently extracted using a strategy for *selective unpacking* from a packed forest representation, as introduced by (Carroll & Oepen, 2005). The unpacking algorithm is based on a guided search using discriminative treebank models as developed in this thesis.

The LOGON project has many connections to the international NLP community, especially with respect to the DELPH-IN network (Deep Linguistic Processing With HPSG), a loose organization of researchers from various sites around the world collaborating on issues related to deep linguistic processing and ways to combine symbolic and statistical methods. Moreover, the code developed for experimentation in this thesis forms part of the open-source repository distributed by the DELPH-IN network. The experimentation environment is in active use within the DELPH-IN community, and realization rankers similar to those we here develop for the English ERG grammar have already been developed for other languages and other grammars. For example, Dr. Berthold Crysmann at the Language Technology Lab of DFKI (*Deutsches Forschungszentrum für Künstliche Intelligenz*) in Saarbrücken, as part of the DELPH-IN consortium, has trained realization rankers for the HPSG-based *German Grammar* (GG; Müller & Kasper, 2000; Crysmann, 2005). It should also be emphasized that our experimentation environment for statistical modeling is not restricted to the "direction" of realization ranking. For example, again within the DELPH-IN community, large-scale parse selection models have successfully been trained on the Japanese Hinoki treebank (Bond et al., 2004). These experiments also confirm the *scalability* of our experimentation environment, as Hinoki contains more than 65,000 sentences annotated with the JACY grammar (Siegel & Bender, 2002), comprising a total of more than

5,250,000 candidate parses.

During the work on this thesis, preliminary results and developments have been published and presented at several international conferences and workshops, including the *3rd Workshop on Treebanks and Linguistic Theories* (TLT 2004), *the 10th Machine Translation Summit* (MT-Summit X 2005), *the 2006 Conference on Empirical Methods in Natural Language Processing* (EMNLP 2006), the *10th International Conference on Theoretical and Methodological Issues in Machine Translation* (TMI 2007), and others. In addition to this, developments have also been presented at more informal venues, such as the annual summits of the DELPH-IN network and the LOGON project meetings. In sum this means that the ongoing development of this thesis has continuously been carried out in close contact with the larger NLP community.

Moreover, many of the ideas developed in this thesis are already being pursued by other researchers within the field. For example, in the work by Nakanishi, Miyao, and Tsujii (2005) at the Tsujii Laboratory at The University of Tokyo, our empirical realization ranking approach have been reimplemented for another chart-based generator based on the Enju grammar, an English HPSG grammar extracted from the Penn Treebank (Miyao, Ninomiya, & Tsujii, 2004). Closely paralleling the experiments of Velldal and Oepen (2005), Nakanishi et al. (2005) train and test several different realization rankers; a bigram model trained on the BNC, a log-linear model based on syntactic features of a generation treebank, as well as a combination of the two. In order to train and test the log-linear model, Nakanishi et al. (2005) created a generation treebank for the Wall Street Journal (WSJ) portion of the Penn Treebank (Marcus, Santorini, & Marcinkiewicz, 1993). By converting the Penn Treebank annotations to HPSG representations, including semantic relations in the form of predicate argument structures, all possible paraphrases can be generated for each analysis while labeling the original treebanked strings as the references—thus creating a symmetric treebank. In the results of Nakanishi et al. (2005), the log-linear treebank model is consistently shown to perform better than the n-gram-based LM, not even being outperformed by the combined model. The latter result is probably partly due to the fact that the LM model was only trained on a sub-set of the BNC, as well as using a lower order of $n = 2$ (compared to $n = 4$ as used here).

As said, our realization rankers are trained on symmetric treebanks annotated with the ERG HPSG grammar for English, including semantic representations in the form of MRSs. However, the overall approach is not specific to the particular language, grammar, or formal frameworks used here. As noted above, Nakanishi et al. (2005) replicate many of the results using another HPSG grammar that instead uses predicate argument structures for "meaning" representation. Moreover, at the Institute for Natural Language Processing (IMS at the University of Stuttgart, there is an ongoing project that works on developing symmetric treebanks using a

large-scale LFG grammar for German. Lead by Prof. Christian Rohrer, the project entitled *Combining contextual information sources for disambiguation in parsing and choice in generation*,[3] has adapted several of the models developed in this thesis for the generator module integrated in the Xerox Linguistic Environment (XLE; Maxwell & Kaplan, 1993). For the purpose of *parse selection*, this LFG grammar engineering platform implements the approach presented by Riezler et al. (2002), as already adapted for German. Much like in our own setup, given the reversibility of the underlying LFG grammar, the XLE system can also generate surface realizations on the basis of input f-structures. Following the procedure outlined in Velldal et al. (2004) and Velldal and Oepen (2005), Cahill, Forst, and Rohrer (2007) construct a symmetric treebank of roughly 8600 items (all associated with more than one available realization) on the basis of the TIGER Treebank (Brants, Dipper, Hansen, Lezius, & Smith, 2002). Using the symmetric treebank, Cahill et al. (2007) are able to adapt the XLE parse selection functionality for the task of realization ranking. Paralleling the models of Velldal and Oepen (2005), Cahill et al. (2007) report results for realization rankers based on both a surface-oriented LM baseline system as well as a log-linear model trained on the symmetric treebank (also including the LM as a feature). Note that the LM is trained on the 200-million-word Huge German Corpus (a collection of text from German newspapers and magazines).

Using a range of different evaluation measures such as BLEU, exact match accuracy, and a ranking score (based on the rank value of the reference string), Cahill et al. (2007) report a substantial improvement when using the combined log-linear model trained on treebank data compared to using the baseline LM alone. Although there is only a slight increase in BLEU score (0.065), the relative error reduction is 29% in terms of exact match and 49% for the ranking score.

In conclusion, this dissertation contributes to the field of hybrid, grammar-based language generation in multiple ways, both methodologically and technologically. By successfully adapting well-established modeling approaches from the field of statistical parsing, transferring them to the field of generation, we are able to train treebank-based discriminative models that greatly improve on the traditional generative n-gram-based approaches. Our approach is also general enough to be applicable to other grammar-based generation systems and other grammatical formalisms than what we use for the particular experiments in this thesis.

[3] The project, internally dubbed *D2*, forms part of a larger project called *Disambiguation in Context* (*Sonderforschungsbereich 732*) at the University of Stuttgart, funded by the German Research Foundation. For more details about the D2 sub-project, see the original project proposal: http://www.uni-stuttgart.de/linguistik/sfb732/data/projects/d2.pdf.

Chapter 2

Mathematical Preliminaries

> *But it must be recognized that the notion "probability of a sentence" is an entirely useless one, under any known interpretation of this term* (Chomsky, 1969).

In this chapter we will review the mathematical foundations of the various statistical models that are applied to the task of realization ranking in this thesis. There are three main frameworks that we will be working with; n-gram language models (LMs), maximum entropy (MaxEnt) models and support vector machines (SVMs). It is not the purpose of this chapter to give a complete introduction to any of these framework, but rather to provide some background for the uninitiated reader. Moreover, while this chapter takes a quite general view on the models and only describes their mathematical foundations, details of the actual implementation, training, tuning and testing of these models are the topics of Chapters 6, 7, and 8. We review the different frameworks in turn, starting with *n-gram language models* in Section 2.1. This is followed by a brief introduction to some basic concepts from *information theory* in Section 2.2, most importantly the notion of *entropy*. This concept has traditionally played an important role in relation to the evaluation of language models, and also provides a natural transition to the framework of *maximum entropy models*, which is the topic of Section 2.3. Finally, in Section 2.4, we look into the theory of *support vector machines* and its application to the task of learning ranking functions.

Before we get started we include a brief comment on notation. Let X be a discrete random variable over an event space $\mathcal{X}$. We will use $p(x)$ to denote the probability of a particular instantiation of X according to the probability mass function p, i.e. $p(x) = Pr(X = x), x \in \mathcal{X}$. However, we will sometimes also use $p(x)$ to denote the entire distribution and let x act as an abstract placeholder rather than an actual instantiation. Moreover, we will sometimes let X represent both the random variable and the event space $\mathcal{X}$, but the meaning will always be

clear from the context.

2.1 N-gram Language Models

The goal of statistical language modeling is to estimate, as accurately as possible, the distribution of strings in natural language. Language is thus viewed as a stochastic process generating sequences of words. In the following we will often use w_1^k as a compact notation for a string or sequence of words $w_1, \ldots, w_k$. A statistical language model (SLM, or just LM for short) is simply a probability distribution $p(w_1^k)$ over strings W that is meant to reflect how likely a string w_1^k is to occur as a sentence.

There are several frameworks commonly used for statistical language modeling, but by far the most widely used is the framework of n-gram-based language modeling. In fact n-gram modeling is perhaps the single most widely used method within all of statistical NLP. While originally applied to the task of speech recognition, these types of models have been successfully applied to a wide range of different tasks such as machine translation, part-of-speech tagging, parsing, and information retrieval. Moreover, an n-gram model does not necessarily need to be defined on the word level. For instance, phoneme level n-grams could be useful for acoustic modeling, while a model on the character level could be useful for applications such as spell-checking or auto-completion. Finally, note that there are also many application areas outside of NLP where sequential modeling plays an important role, such as in bio-informatics in relation to analyzing gene sequences. The rest of this section gives a brief description of the basic building blocks of n-gram models.

The problem of language modeling is often formulated as that of predicting the next word in a sequence, given the other words that precede it. By the chain rule of joint probabilities, the probability of a sequence of words can then be factorized as

$$p(w_1, \ldots, w_k) = p(w_1)p(w_2|w_1)p(w_3|w_1, w_2) \ldots p(w_k|w_1, \ldots, w_{k-1}) \quad (2.1)$$

However, the number of parameters that would be involved in modeling this full distribution would be prohibitively large. In an n-gram model the estimation problem is made more manageable by relying on a so-called *Markov assumption*. This means that the probability of a given word is taken to only depend on the $n-1$ words preceding it. Just to be clear about terminology, an n-gram in itself is just a sub-sequence of n elements from a sequence $w_1, \ldots, w_k$, and in an n-gram model the probability of the ith word in a given sequence is conditioned on the last $n-1$ words in its history. An n-gram model is sometimes also referred to as an $(n-1)$th order Markov model (Manning & Schütze, 1999, sec. 6.1.2).

Given that the probabilities of the individual words are taken to only depend on the $n-1$ previous words, the probability of a string $w_1, \ldots, w_k$ is just the product of its individual word probabilities, computed as

$$p_n(w_1^k) = \prod_{i=1}^{k} p(w_i | w_{i-n+1}^{i-1}) \tag{2.2}$$

In order to make $p(w_i|w_{i-n+1}^{i-1})$ well-defined also for the $n-1$ first words of a sequence, it is common to assume that the strings are padded on the left[1] with a special token such as `<s>`, to indicate the beginning of the string. Another common assumption, which is what we will use for the LM experiments in this thesis, is that we only include a single such sentence start marker `<s>`, and that the model instead *backs off* of to compute the probability of the first $n-1$ words w_i based on a history of the $i-0$ preceding words. For example, for a trigram model (i.e. $n=3$) we would compute the string probability as

$$p_n(w_1^k) = p(w_1)\, p(w_2|w_1)\, p(w_3|w_1, w_2) \prod_{i=4}^{k} p(w_i | w_{i-n+1}^{i-1}) \tag{2.3}$$

However, assuming we always pad all strings with the start symbol, we can ignore the initial unigram probability $p_n(\texttt{<s>})$. It is also common to make the simplifying assumption that no important dependencies hold across sentence boundaries, and that the training corpus is assumed to be segmented into sentences (with `<s>` marking the boundaries). The probability for a collection of sentences T can then be computed as a product of the individual sentence probabilities, as in

$$p_n(T) = \prod_{w_1^k \in T} p_n(w_1^k) \tag{2.4}$$

Furthermore, in order to define a proper probability distribution over all strings independent of their length, the strings are usually padded on the right with a designated symbol such as `</s>` which is included in the product in Equation (2.2) (Chen & Goodman, 1998). Without such an end marker, the model as it stands above would only define a proper distribution (i.e. the probabilities sum to unity) over strings of the same length. Accordingly, the sum of the probabilities of all strings would be infinite.

Now, given the machinery we have introduced so far, Equation (2.5) shows an example of how the probability of a string such as *have a nice day* is computed

[1] We here assume left-to-right directionality.

assuming a bigram model (i.e. $n = 2$):

$$\begin{aligned} p_n(\textit{have}, \textit{a}, \textit{nice}, \textit{day}) = & \, p_n(\textit{have} \mid \text{<s>}) \times \\ & \, p_n(\textit{a} \mid \textit{have}) \times \\ & \, p_n(\textit{nice} \mid \textit{a}) \times \\ & \, p_n(\textit{day} \mid \textit{nice}) \times \\ & \, p_n(\text{</s>} \mid \textit{day}) \end{aligned} \tag{2.5}$$

So far we have mostly talked about the form of the model. We now turn to focus more on the problem of estimating the actual probabilities. It is easy to compute the *maximum likelihood* (ML) estimate for an n-gram model, i.e. the probability estimate that maximizes the probability of the training data. In order to compute the ML estimate of $p_n(w_i|w_{i-n+1}^{i-1})$ we can simply use the *relative frequencies*, as in

$$p(w_i|w_{i-n+1}, \ldots, w_{i-1}) = \frac{c(w_{i-n+1}, \ldots, w_i)}{c(w_{i-n+1}, \ldots, w_{i-1})} \tag{2.6}$$

where $c(w_i^k)$ denotes the frequency of occurrence of the sequence w_i^k in the training corpus. However, as we shall see in the next section, the simple ML estimate may not give us a model that generalizes well when presented with new data.

2.1.1 Data Sparseness and Smoothing

In most cases, n-gram models for NLP purposes are trained for $n = 2$ or $n = 3$, (called *bigram* models or *trigram* models respectively), and very rarely for $n \geq 5$. The value of n is usually relatively low due to both the number of parameters involved in such models and also the general problem of data sparseness. Even with low values of n and large amounts of training data with millions of words, there will still be lots of perfectly reasonable n-grams that are left unobserved. And if even a single component n-gram in a sequence is unobserved, the ML probability of the entire string will be zero, as zeroes will propagate through the multiplication of word probabilities. *Zipf's law*[2] is often found to fit well with empirical counts gathered from corpus data. This empirical law describes the general tendency of there being a small number of events that occur with high frequency, while there is a large number of events that occur with a low frequency. The long tail of rare events means that, no matter how large the training corpus,

[2] Zipf's law, formulated by the Harvard linguist George Kingsley Zipf, states that for many frequency distributions, the relationship between the frequency of an event f and its rank r (according to frequency) obeys $f \propto \frac{1}{r}$ (Zipf, 1935). Since the distribution of words is observed to obey Zipf's law it is sometimes said to have a Zipfian distribution. This essentially means that a language generally has a small number of very frequent words, an intermediate number of medium frequent words, and a large number of infrequent words.

there will always be many n-grams that remain unobserved or have a frequency too low to make any reliable estimates. Another important factor contributing to this situation is of course implied by Chomsky's well-known dictum that language use is a creative process (Chomsky, 1965). Natural language continuously sees the addition of new words and new combinations of words.

All in all, we see that the simple ML estimate will underestimate the probability of many strings. In machine learning terminology, the problems described above can be seen as an instance of the problem of overfitting. Although the ML distribution gives us the model that best fits the training data, it does not necessarily generalize well when applied to new data. To alleviate this problem and obtain more accurate models, it is important to apply some kind of *smoothing method*. The term reflects the fact that the purpose of these methods is to flatten the otherwise spiky maximum likelihood distribution and make it more uniform. The process is also known as *discounting* since the methods typically work by discounting the probabilities of non-zero n-grams and then redistributing this among unobserved or low-frequency events.

Although there exist a range of different smoothing methods that are commonly used when estimating n-gram models, they typically fall within one out of two main categories; *interpolated* models and *back-off* models (Chen & Goodman, 1998). In the former type of methods, the probability of a word is computed as a linear interpolation of higher-order and lower-order models. In the latter type of methods, the probability estimate for a given n-gram falls back on a lower order model if its observed frequency in the training data is below some given threshold. The main difference between the two types of approaches is that, while a back-off model only uses a lower-order distribution for determining the probability of unobserved or low frequency n-grams, an interpolated model always uses a combination of distributions of different orders for all n-grams. These general strategies can be implemented with a range of different discounting methods for modifying the counts that form the basis of the probabilities. However, it is not within the scope of this chapter to try to cover the issue of smoothing in its entirety. For readers looking for a more complete introduction, Chen and Goodman (1998) provide an overview and comparative evaluation of many of the most commonly used discounting methods, such as Good-Turing estimation, Additive Smoothing, Witten-Bell, Jelinek-Mercer, Kneser-Ney, Katz, Absolute Discounting, and others. Here we will briefly describe only one such method, so-called *Witten-Bell discounting* (corresponding to discounting type C in Witten & Bell, 1991), which we will be using as part of a back-off method as described above. The particular Witten-Bell back-off strategy that our models use correspond to the implementation provided in the Carnegie Mellon Statistical Language Modeling

Toolkit[3] (CMU SLM toolkit; Clarkson & Rosenfeld, 1997).

Under the Witten-Bell approach (Witten & Bell, 1991), the discounting is performed on the basis of how many distinct word types that are observed to follow a particular context. For example, consider an event w_{i-n+1}^{i}, where we want to predict the probability of the word w_i based on the context or history w_{i-n+1}^{i-1}. To make the exposition less cumbersome, we here simply refer to the history as h and the word we want to predict as w. Furthermore, let $c(h, w)$ refer to the frequency count of the entire event, while $c(h, *)$ is the total frequency of the preceding history. Now, let $t(h)$ be the total number of distinct words following h in the training data. The discounted Witten-Bell probability can then be computed as

$$p_{\text{WB}}(w|h) = \begin{cases} \frac{c(h,w)}{c(h,*)+t(h)} & \text{if } c(h, w) \geq 1 \\ \frac{t(h)}{c(h,*)+t(h)} & \text{if } c(h, w) = 0 \end{cases} \tag{2.7}$$

For an unseen event (h, w), the probability will be higher when the type count $t(h)$ is higher (relative to the overall frequency of h). In other words, the probability reserved for an unobserved event will be higher if the history tends to occur with many different words. A context that is followed by fewer types will have a smaller amount of its probability mass discounted and redistributed for unseen events. Of course, one problem with the formulation as it stands in Equation (2.7), is that the probability is still undefined (or zero) in cases where also the history h is unseen. As mentioned above, however, the language models implemented in the CMU SLM toolkit follow a back-up approach, where such cases are handled by falling back on a model that uses a lower value of n, i.e. conditions the probability on a shorter history. Of course, the interpolation of the different models is weighted in a way that ensures that the overall model defines a proper probability distribution.

2.1.2 Vocabulary

A special case of the problem with zero-probability n-grams arises in relation to specifying the vocabulary of the LM. In some restricted application settings, where all the words occurring in the test data are known to have occurred in the training data, one might be able to use a model with a closed vocabulary. But in the more typical setting, one needs to deal with the case of unobserved unigrams, i.e. words in the test data that did not occur in the training data. Given the Zipfian distribution of word frequencies mentioned earlier, many of the words that actually do occur will typically only be seen once or a few times, making it hard to form any

[3] For more information, see http://www.speech.cs.cmu.edu/SLM/toolkit.html.

reliable predictions. The common solution is to explicitly enumerate the vocabulary (e.g. according to some frequency cut-off in the training data) and then build an open-vocabulary model. Under both training and testing, all out-of-vocabulary (OOV) tokens are then mapped to a special symbol designating an unknown word, such as <unk>. When estimating the model, the counts for this token will then conflate the counts for all OOVs in the training data. At the time of application, all unobserved words in the test data will be mapped to this token.

There are also many other important choices involved in specifying the vocabulary of a model. Many of these choices are related to normalization techniques, which can potentially help remedy the data-sparseness problem by reducing the number of unique tokens in the data. Relevant examples of such normalization steps include stemming or lemmatization, conflating spelling variants, normalizing numbers, date and times, normalizing case, removing punctuation and formatting, and many other measures. In fact, a significant part of the effort that goes into crafting an accurate n-gram model is typically spent on issues related to pre-processing of the data.

Before we round off this section it is important to point out that there exist many extensions to standard n-gram models as we have described here. For example, the technique of using a special token <unk> for OOVs as described above can be seen as an instance of the more general technique of using word classes. In fully class-based models the estimation is preceded by a clustering step, so that probabilities can be computed over sequences of classes instead of words. Class-based models can also be combined with standard word-based models, performing class-expansion only for certain words such as proper names, time-expressions etc. By making abstractions over individual words, class-based models can help alleviate the data sparseness problem, similarly to smoothing and normalization. Other variants of n-gram models include cache models, topic-based models, and skipping LMs, to name a few (Stolcke, 2002).

2.1.3 Problems Related to N-gram Models

A common criticism of n-gram-based LMs concerns their inherent limitation of not being able to capture dependencies that go beyond the $n-1$ span of the history. In fact, a simple n-gram model will even fail to capture all the potential dependencies within this span. For example, consider a back-off trigram model presented with the string *red fruity wine*. If this particular n-gram does not occur in the training data, the model will back-off to the bigram *fruity wine* when predicting the probability of *wine*, regardless of how many times the bigram *red wine* has been observed. As we see, long-range dependencies and dependencies between non-contiguous words reveal inherent short-comings of simple n-gram models. An important part of this problem is, of course, the fact that simple n-

gram models are purely surface-oriented and thereby fail to capture dependencies that show a structural rather than sequential regularity. The deeper structures of the strings are ignored entirely. Generally, the explicit linguistic knowledge that goes into a standard n-gram model is usually modest or nil, and the main criticism of these models is usually related to their lack of theoretical purity in some form. Nonetheless, in terms of empirical results, n-gram models have proved hard to outperform on many tasks.

2.1.4 Evaluation

A central task in statistical modeling is evaluation. As mentioned above, n-gram models have been applied to a wide range of different problems within NLP. In general, the most useful evaluation will necessarily have to be task-based in order to be directly relevant for the particular problem at hand. However, depending on the application setting, task-based evaluation may be computationally expensive and time-consuming, and it is practical to have a quick way of evaluating a model during development. Language models are typically evaluated using measures that in some way reflect the likelihood of some test corpus according to the estimated distribution. The two most commonly used such measures are called *cross-entropy* and *perplexity*. Perplexity is simply the inverse of the geometric average probability assigned to each word in the test set by the model. Cross-entropy can be derived by taking the logarithm of the perplexity. The cross-entropy of a model with respect to a test sample expresses the number of *bits* needed to encode the sample according to the model distribution, and is thereby a direct measure of performance for the particular task of text compression (Chen & Goodman, 1998). However, it is often assumed that lower entropy rates also correlate well with the performance level on many other tasks. In order to explain how these evaluation measures are motivated and defined, we take the opportunity to include a brief detour via *information theory*, before we turn to the framework of conditional maximum entropy models.

2.2 Basic Concepts from Information Theory

In this section we present a few basic information-theoretic concepts which play an important role in relation to evaluating language models and also in relation to the principles underlying maximum entropy models, which we describe in Section 2.3. The field of information theory arose from studies on *data compression* and *transmission*, as pioneered by Claude Shannon in the late 1940s. One of the key quantities in information theory is *entropy* (Shannon, 1948), which is a measure of the average *uncertainty* in a random variable. For a random variable X,

distributed according to a probability mass function $p(x)$, its *entropy* is defined as

$$H(X) = -\sum_{x \in X} p(x) \log p(x) \tag{2.8}$$

We here follow the convention of defining $0 \log 0 = 0$. Note also that, we will sometimes write $H(p)$ instead of $H(X)$ to denote the entropy of the random variable X that is distributed according to p. When using logarithms to base 2, the entropy is said to be measured in *bits*.[4] This means that we can think of each outcome as being represented by some binary number. Entropy is then an expression of how many bits on average are required to describe the random variable assuming an optimal encoding scheme (Cover & Thomas, 1991). We see that the entropy is highest under the *uniform* distribution, i.e. when all outcomes are equally probable. In this case the entropy is simply $\log |X|$, where we take $|X|$ to denote the number of possible events. When all events are equally probable, the entropy increases monotonically with the number of possible events. As said, one of the main concerns of information theory is how to best encode information, and the entropy of a random variable provides a lower bound on the average number of bits needed to represent that variable (Cover & Thomas, 1991). With a *non-uniform* distribution, the coding scheme can take advantage of the fact that some events are more likely than others, and assign labels that require a smaller number of bits to encode these events, thereby reducing the average code length.

The lowest possible entropy (0) is achieved if one of the values has a probability of 1. This intuitively makes sense, as there is then no randomness involved, the outcome is constant and fixed, and there is no uncertainty associated with the variable. Entropy can also be expressed as the *expected value* of the random variable representing the function $\log \frac{1}{p(X)}$, as in

$$H(X) = \sum_{x \in X} p(x) \log \frac{1}{p(x)} = E_p \left(\log \frac{1}{p(X)} \right) = -E_p \left(\log p(X) \right) \tag{2.9}$$

This definition makes it clearer that entropy represents a weighted average, and the quantity $-\log p(x)$ can intuitively be interpreted as the amount of uncertainty associated with the particular event x.

The entropy measure is foundational to many other important measures within information theory. One such measure that is commonly used to model the "distance" between a "true" probability distribution p and some other distribution q, is

[4] The base used when computing the entropy reflects the number of symbols in the alphabet we use for coding the random variable. Since bits are based on the alphabet $\{0, 1\}$ with only two symbols, we use logs to base 2. Note that the number of symbols in the assumed alphabet does not even necessarily need to be an integer. For instance, when using natural logarithms with the base e the entropy is said to be measured in *nats*.

the *relative entropy*. The measure is also known as Kullback-Leibler divergence, and it is defined as

$$\begin{aligned} D(p\|q) &= \sum_x p(x) \log \frac{p(x)}{q(x)} \\ &= E_p \log \frac{p(X)}{q(X)} \end{aligned} \tag{2.10}$$

As in Cover and Thomas (1991, ch. 2), the definition above assumes the convention that $0 \log \frac{0}{q} = 0$ and $p \log \frac{p}{0} = \infty$. In the typical LM setting, p represents the empirical distribution of the data while q is some estimated model that we want to test. As can be seen from Equation (2.10), relative entropy can be expressed as the expected logarithm of the likelihood ratio, and can be understood as a measure of the inefficiency (in terms of wasted bits) that results from encoding X according to the approximation q instead of p. It is always non-negative, and zero only in the case of identity ($p = q$). If we knew the true distribution p we could devise a code with expected description length $H(p)$, but if we instead use a code devised for another distribution q, we would need $H(p) + D(p\|q)$ bits on average to encode the random variable (Cover & Thomas, 1991, ch. 2). This latter quantity is sometimes labeled *cross-entropy*, or $H(p, q)$. In other words, cross-entropy is a measure of the average number of bits needed to identify an event from a set of possibilities, if a coding scheme is based on a given probability distribution q, rather than the "true" distribution p.

$$\begin{aligned} H(p, q) &= H(p) + D(p\|q) \\ &= -\sum_x p(x) \log p(x) + \sum_x p(x) \log p(x) - \sum_x p(x) \log q(x) \\ &= -\sum_x p(x) \log q(x) \end{aligned} \tag{2.11}$$

Since $D(p\|q) \geq 0$, it follows that $H(p, q) \geq H(p)$. Note also that, just as in the case of entropy, cross-entropy can also be expressed as an expectation:

$$H(p, q) = -\sum_x p(x) \log q(x) = -E_p \left(\log q(x)\right) \tag{2.12}$$

In the introduction to the section on language modeling we mentioned how language can be viewed as a stochastic process, and Section 2.1.4 noted how we could evaluate a statistical model on the basis of cross-entropy. Ideally we would like to measure the cross-entropy between our model q and the true distribution of language p. If we make the simplifying assumption that language defines a so-called stationary ergodic processes (in essence this describes a random process

that will not change its statistical properties with time), we can apply an important result known as the *Shannon-McMillan-Breiman theorem* that says that the cross-entropy between a true distribution p and a model q can be approximated with respect to a large sample sequence $x_1, \ldots, x_N$ by computing

$$H(p, q) \approx H(x_1^N, q) = -\frac{1}{N} \log q(x_1^N) \tag{2.13}$$

In the context of language modeling, the token count N is usually taken to include the end-of-sentence marker `</s>`, but not the start marker `<s>`. Note also that, because $H(p)$ is fixed, but unknown, minimizing $H(x_1^N, q)$ as in Equation (2.13) is equivalent to minimizing relative entropy of q to p.

As mentioned in Section 2.1.4, another commonly used measure for evaluating LMs, especially within the field of speech recognition, is perplexity. Given the definition of cross-entropy given above, perplexity is defined as

$$P(x_1^N, q) = 2^{H(x_1^N, q)} \tag{2.14}$$

Just as with cross-entropy, the perplexity is inversely related to the average probability assigned to the words in the test sample, which means that we want to minimize the perplexity of our model.

In the next section we turn to have a look at the framework of maximum entropy modeling.

2.3 Maximum Entropy Models

The family of maximum entropy (MaxEnt) models provides a very flexible modeling framework that allows one to combine disparate and overlapping sources of information in a single model without making unwarranted independence assumptions. MaxEnt models have been widely used for a range of tasks in NLP, including parse selection (see e.g. Johnson et al., 1999; Miyao & Tsujii, 2002; Malouf & van Noord, 2004) and reranking for machine translation (see e.g. Och et al., 2004). These models sometimes also go under other guises such as log-linear models, exponential models, Random Fields and Gibbs distributions. In many cases the differences between these models only pertain to the theoretical motivation rather than their practical implementation. In this section, however, we focus on conditional maximum entropy models. To make the exposition more concrete, we try to use terminology that makes it directly relevant for the particular problem that we will be dealing with in the chapters to follow; training a model on vectors of structural features extracted from treebank data, with the purpose of ranking surface realizations that are generated for given meaning representations.

A MaxEnt model is given by a set of *feature functions* and corresponding *weights*. The specified feature functions describe properties of the data points, and the associated set of learned weights determines the contribution or importance of each feature. The real-valued features can describe arbitrary properties of the data points. In our case the relevant event space consists of pairs of semantic representation and realizations. Before we go on, let us first introduce some new notation. We will take s to denote a semantic representation, and r to denote a realization generated for some s. Note that, while we have used w_1^k to denote a surface string in the preceding discussion, we here use r to denote a more general notion of a realization which may also include a structural representation. Now, the generator function $\mathcal{Y}(s_i) = \{r_1, \ldots, r_m\}$ gives the set of possible realizations licensed by the grammar for a semantic representation s_i. We will sometimes just write $\mathcal{Y}_i$ for short. Let our training data be given as $X = \{x_1, \ldots, x_N\}$ where each x_i is a pair (s_j, r_k) for which $r_k \in \mathcal{Y}(s_j)$ and r_k is annotated in the treebank as being a correct realization of s_j. Note that we might have several different members of $\mathcal{Y}(s_j)$ that pair up with s_j in X. Given a set of d features, each pair of semantic input s and hypothesized realization r is mapped to a feature vector $f(s, r) \in \Re^d$ where each dimension records the value of one of the d specified features. The goal is then to estimate a vector of weights $\lambda \in \Re^d$ on the basis of these feature vectors, which will determine our model q_λ. Note that this chapter only gives an abstract presentation of the MaxEnt framework. Details of the actual use and implementation of this framework, such as the definition of the actual features we use, are deferred to later chapters.

The estimation of the weights is governed by a set of *constraints*. The constraints require that the expected values of the features with respect to our model q_λ should equal their expected values with respect to a target distribution $\tilde{p}$. We will let $\tilde{p}(s, r)$ denote the empirical distribution as given by the relative frequencies of the training data, i.e.

$$\tilde{p}(s, r) = \frac{c(s, r)}{N} \tag{2.15}$$

Just as in the preceding sections, $c(s, r)$ denotes the counts in the training corpus and $N = \sum_{s,r} c(s, r)$. A realization labeled as preferred in training data can be thought of as having occurred once, while all the realizations labeled as dispreferred have a count of zero. The expected value of a given feature f_i with respect to the empirical distribution $\tilde{p}(s, r)$ can then be given as

$$E_{\tilde{p}}(f_i) = \sum_{r,s} \tilde{p}(s, r) f_i(s, r) \tag{2.16}$$

On the other hand, the expected value of feature f_i with respect to the model

distribution $q_\lambda(s, r)$ is given as

$$E_{q_\lambda}(f_i) = \sum_{r,s} \tilde{p}(s) q_\lambda(r|s) f_i(s, r) \tag{2.17}$$

where $\tilde{p}(s)$ is the marginal empirical distribution of s in the training data. Given these definitions of expected feature values under the relevant distributions, the constraints imposed by the maximum entropy principle can be stated as

$$E_{\tilde{p}}(f_i) = E_{q_\lambda}(f_i) \tag{2.18}$$

We see how the empirical expectations of the feature functions represent prior knowledge which is included in the model in the form of constraints on possible models. For a set of d feature functions, d corresponding constraints are imposed on the model. However, the property of satisfying the constraints on feature expectations does not uniquely determine a model. There will typically be a large or even infinite number of possible models $\mathcal{Q}$ that represent solutions to this set of constraints. This is where the principle of maximum entropy comes into play. This principle states that, of all the models $q_\lambda \in \mathcal{Q}$, we should pick the one which is most uniform. Recall from Section 2.2 that the most uniform distribution for a given event space is the one with the greatest entropy, as defined in Equation (2.8). In other words, we want to find the following model:

$$q_\lambda^* = \arg\max_{q_\lambda \in \mathcal{Q}} H(q_\lambda) \tag{2.19}$$

It can be shown that there is guaranteed to be a unique model $q_\lambda^* \in Q$ which has the highest entropy and satisfies all the constraints.

The MaxEnt principle can be interpreted as choosing the model with the highest degree of uncertainty, subject to the given constraints. This might at first seem like a misguided strategy, given the discussion of various entropy-based measures for model evaluation in Section 2.2, where we emphasized that it was desirable to minimize the entropy of the models. However, entropy can also be seen as a measure of the simplicity of a model. When presented with a choice between alternatives, with no knowledge to guide us, any assumptions other than all of them being equally likely would seem unwarranted. The MaxEnt principle is typically explained as an *Occam's Razor* argument for model selection, in the sense that we should never choose a model that is more complicated than necessary for explaining the empirical data. Choosing the model with the highest entropy means choosing the model that makes the fewest additional assumptions about the data beyond what is encoded by the constraints on feature expectations. Or, as one of the pioneers of maximum entropy modeling, Jaynes (1957), succinctly states it:

> *Information theory provides a constructive criterion for setting up probability distributions on the basis of partial knowledge, and leads to a type of statistical inference which is called the maximum-entropy estimate. It is the least biased estimate possible on the given information; i.e., it is maximally noncommittal with regard to missing information. (Jaynes, 1957)*

The goal of finding the model in $\mathcal{Q}$ that maximizes the entropy can equivalently be formulated in terms of minimizing the relative entropy, as defined in Equation (2.10) above. We want to minimize the divergence between the empirical target distribution $\tilde{p}$ and the model distribution q_λ on the one hand, and between the model and the uniform distribution q_0 on the other hand:

$$q_\lambda^* = \arg\min_{q_\lambda \in \mathcal{Q}} D(\tilde{p} \| q_\lambda) \tag{2.20}$$

and

$$q_\lambda^* = \arg\min_{q_\lambda \in \mathcal{Q}} D(q_\lambda \| q_0) \tag{2.21}$$

A conditional MaxEnt model of the probability of a realization r given the semantics s, has the following parametric form (Berger, Della Pietra, & Della Pietra, 1996):

$$q_\lambda(r|s) = \frac{1}{Z_\lambda(s)} \exp\left(\sum_{i=1}^{d} \lambda_i f_i(s,r)\right) \tag{2.22}$$

The so-called partition function Z_λ is defined as

$$Z_\lambda(s) = \sum_{r' \in \mathcal{Y}(s)} \exp\left(\sum_{i=1}^{d} \lambda_i f_i(s,r')\right) \tag{2.23}$$

and acts as a normalizing constant for each s, ensuring a proper probability distribution.

As defined in Equation (2.22), MaxEnt models belong to the *exponential family* of models, and are often also known as *log-linear* models. The name reflects the fact that the logarithm of the probability corresponds to a linear combination of the feature values, as in

$$\ln q_\lambda(s|r) = (-\ln Z_\lambda) + \sum_{i=1}^{d} f_i(s,r)\lambda_i \tag{2.24}$$

Note that while the terms *log-linear* or *exponential* models refer to the parametric form of the model, *maximum entropy* modeling describes a specific strategy or guiding principle for how to estimate the parameters of these models.

It is also worth noting that when scoring the candidate realizations that are generated for a given semantics, we will only be interested in their relative rank order, and not the associated scores in themselves. Since $Z_{\lambda(s)}$ will be constant (for a given s), it therefore suffices to compute their non-normalized scores. In order to find the most probable realization $\hat{r} \in \mathcal{Y}(s)$ for a given semantics s, we can use the following decision rule;

$$\hat{r} = \arg\max_{r \in \mathcal{Y}(s)} q_\lambda(r|s) \tag{2.25}$$

$$= \arg\max_{r \in \mathcal{Y}(s)} \sum_{i=1}^{d} \lambda_i f_i(s, r) \tag{2.26}$$

2.3.1 Minimum Divergence Models

It is also possible to give a more general formulation of maximum entropy models if we explicitly include the so-called *reference distribution*[5] q_0, as in the following definition:

$$q_\lambda(r|s) = \frac{1}{Z_\lambda(s)} \, q_0(r|s) \, \exp\left(\sum_{i=1}^{d} \lambda_i f_i(s, r)\right) \tag{2.27}$$

Z_λ is correspondingly defined as

$$Z_\lambda(s) = \sum_{r' \in \mathcal{Y}(s)} q_0(r'|s) \, \exp\left(\sum_{i=1}^{d} \lambda_i f_i(s, r')\right) \tag{2.28}$$

The reference distribution offers a way for prior knowledge to be incorporated directly into the model by imposing a prior distribution over the data. In the case of maximum entropy modeling, the guiding principle is exactly to not make any assumptions about the data except for the feature value constraints derived from our training data, and so q_0 is the uniform distribution. Since this will be a constant given the semantics we are conditioning on, $q_0(r|s) = 1/|\mathcal{Y}(s)|$, this can be left out in the case of maximum entropy models. One can, however, also replace this uniform distribution by some other reference distribution q_0 to incorporate prior knowledge in the model. According to Equation (2.21) above, we will in this case instead be minimizing the divergence between the model and this reference distribution. This yields what is known as a *minimum divergence* model[6] (Berger

[5] Note that there are several terms in use for what we here call a reference distribution. Other terms commonly seen in the literature include *base distribution* and *default distribution*. Some authors also reserve the term reference distribution to the empirical distribution $\tilde{p}$.

[6] Note that the entropy of the resulting model q_λ will necessarily be lower in the case of minimum divergence models, compared to when q_0 is taken to be the constant uniform distribution.

& Printz, 1998). The more general term denoting these types of models is simply *maximum entropy / minimal divergence* models (MEMD).

2.3.2 Maximum Likelihood Estimation

As defined in Equation (2.19) above, learning a MaxEnt model amounts to finding the values for the parameter vector λ that satisfy the constraints on expected feature values, and also uniquely determine the model with the highest entropy. It turns out that solving the constrained optimization problem of finding the model $q_\lambda \in \mathcal{Q}$ with the greatest entropy, is equivalent to solving the unconstrained optimization problem of finding the weight vector λ that maximizes the log-likelihood $L\lambda$ of the training data X:

$$\hat{\lambda} = \arg\max_{\lambda \in \Re^d} L_\lambda(X) \tag{2.29}$$

Thus, two different approaches —maximum likelihood and maximum entropy— lead to the same solution. The maximum entropy model satisfying the constraints in Equation (2.18) with $\tilde{p}$ being the empirical distribution derived from the training set, is the same as the maximum likelihood solution for the family of log-linear models defined in Equation (2.22). Note furthermore that, since we here will try to estimate a conditional model directly —an issue we return to below— the weights are optimized according to the *conditional* log-likelihood function (Johnson et al., 1999), which is defined as

$$L_c\lambda(X) = \sum_{(s,r)\in X} \tilde{p}(s,r) \log q_\lambda(r|s) \tag{2.30}$$

The log-likelihood function is concave in the model parameters and reaches a global maximum where its gradient is zero (Malouf, 2002). The gradient of the log-likelihood function is given as the vector of its partial derivatives with respect to its parameters λ:

$$G(\lambda) = E_{\tilde{p}}(f) - E_{q_\lambda}(f) \tag{2.31}$$

Generally there is no analytical solution to the problem of finding λ^*, and estimation is carried out using some iterative optimization algorithm. Based on the divergence between the estimated distribution $q_\lambda^{(k)}$ and the empirical distribution $\tilde{p}$, the model parameters are updated to some new estimate $\lambda^{(k+1)}$. The parameters are iteratively refined in this way until the decrease in divergence between iterations falls below some threshold. Although most of the relevant optimization methods are similar from this very high-level view, they can differ greatly with respect to how the actual updates are computed. For a more in-depth presentation of some

of the available algorithms, the reader is referred to Malouf (2002), who compares the performance of several optimization methods for estimating conditional MaxEnt models. Traditionally, the most commonly used such methods for estimating maximum entropy models are *Generalized Iterative Scaling* (GIS; Darroch & Ratcliff, 1972) and *Improved Iterative Scaling* (IIS; Della Pietra, Della Pietra, & Lafferty, 1997). More recently, however, the trend seems to be toward using more general function optimization methods such as *conjugate gradient* and *limited-memory variable metric* methods. In the experiments reported by Malouf (2002), the limited-memory variable metric algorithm of Benson and Moré (2001) substantially outperforms the other methods with respect to computational efficiency (measured in terms of number of iterations, number of log-likelihood or gradient evaluations, and total running time).

Note that the subscript c that adorns the log-likelihood function given in Equation (2.30) is there to highlight the fact that we are using the *conditional* log-likelihood, sometimes known as the *pseudo-likelihood* (Johnson et al., 1999). This reflects the fact that we are trying to directly model a conditional distribution, instead of a full joint distribution. In the machine learning literature this distinction between estimating joint and conditional models is sometimes known as generative and discriminative modeling, respectively.

In many settings within NLP, trying to maximize the joint likelihood of the training data would mean that the partition function must sum over an unmanageable large space of events (e.g. all possible sentences). By restricting ourselves to computing a conditional distribution we can greatly reduce the computational cost by only considering the subspace of events that are licensed by the input event that we are conditioning on. Moreover, as noted by Johnson et al. (1999), for certain problems in computational linguistics we are actually just interested in the conditional distributions. In our case we are interested in the conditional distribution of realizations given the input semantics, while we are not really concerned with the marginal distribution of the semantic representations. If we estimated a full joint-distribution $q(s, r)$, the conditional distribution could be computed using Bayes' rule as

$$q(r|s) = \frac{q(s|r)q(r)}{q(s)} = \frac{q(s,r)}{\sum_{r'} q(s,r')} \tag{2.32}$$

However, when using a discriminative approach we instead proceed to estimate a conditional model $q(r|s)$ directly. Given a semantics s, the model then aims to maximize the probability of a preferred realization r relative to the other realizations $r' \in \mathcal{Y}(s)$. Intuitively, this is all we are really interested in for the ranking problem. That is, for the task of picking the best realization, we can essentially ignore the marginal distribution of semantic representations $q(s)$. In discriminative modeling the focus is not so much on modeling the underlying distributions as it is on optimizing the mapping from inputs to desired outputs.

2.3.3 Regularization

As described above, MaxEnt models can perhaps be thought of as inherently including a mechanism for smoothing, in the sense that the maximum entropy principle ensures that we select the model that is maximally uniform while obeying the empirical constraints. However, we have also seen that the training procedure is an instance of maximum likelihood estimation, with all the associated problems we described in relation to data sparseness and language modeling in Section 2.1.1. As is typically the case when relying on ML estimation (or conditional ML as described above), MaxEnt models are prone to overfitting. This tendency is especially pronounced when training on sparse data using many features. The ML approach to computing λ amounts to assuming a uniform prior on the parameter values (i.e. for each parameter we assume that all possible values are equally likely). Another approach to the estimation problem is to include a prior probability of the model, independent of how well the model fits the data. A widely used approach is to replace the likelihood function of Equation (2.30) with a penalized version that imposes a Gaussian prior over each of the model parameters (Chen & Rosenfeld, 1999; Johnson et al., 1999; Malouf & van Noord, 2004).

$$\hat{\lambda} = \arg\max_{\lambda} L_c(\lambda) + \log \prod_{i=1}^{d} \frac{1}{\sqrt{2\pi\sigma_i^2}} \exp -\frac{(\lambda_i - \mu)^2}{2\sigma_i^2} \tag{2.33}$$

$$= \arg\max_{\lambda} L_c(\lambda) - \sum_{i=1}^{d} \frac{\lambda_i^2}{2\sigma_i^2} \tag{2.34}$$

The second term of the objective function in Equation (2.33) is the regularization term, defining a zero mean normal distribution over the parameter values. By promoting less extreme parameter values, this regularization term can reduce the tendency of log-linear models to over-fit the training data. By drawing the parameter values towards zero, the resulting model will be more uniform, similarly to the smoothing techniques discussed in relation to n-gram models in Section 2.1.1 above. In addition to improving accuracy, the use of a regularized likelihood function tends to also reduce the number of iterations needed for convergence during estimation (Malouf & van Noord, 2004).

In the rewriting steps of Equation (2.33) we drop the term $\frac{1}{\sqrt{2\pi\sigma_i^2}}$ as it will be a constant for each i and so does not have any effect on our maximization, and the omission of μ simply reflects the fact that we use a distribution with mean zero. The specified value of the variance parameter σ^2 determines the relative contribution of the prior and the modified penalized likelihood function, and thereby the degree of smoothing. If we look at the penalty term in Equation (2.33) we see that (i) having parameters with absolute values that are closer to zero leads to a smaller penalty, and (ii) using a smaller value for the variance leads to larger penalties in

overall (i.e. increasing the degree of smoothing). Specifying lower values for σ^2 gives a more peaked prior distribution. This will require more data to force the parameter values aways from zero. An optimal value of the variance parameter σ^2 is typically determined empirically by testing on held-out data. Note that the prior distribution on the parameters also could be defined in many other ways. Another commonly used prior is the exponential distribution, which can be used to more aggressively force some of the parameter values all the way down to zero, thereby effectively combining feature selection and regularization in the same step Riezler and Vasserman (2004). In the literature one often finds that Gaussian and exponential priors are referred to as L2 and L1 regularization, respectively.

In maximum likelihood estimation, the parameters are viewed as having a fixed but unknown value which should be estimated to maximize the likelihood of the data. Each hypothesis is taken to be equally likely. However, when estimating λ according to Equation (2.33) we seek to maximize the product of the conditional probability of the data and the prior probability of the parameters. This is sometimes interpreted from the perspective of *Bayesian learning*, as a way of choosing the *maximum a posteriori hypothesis* (MAP). In the Bayesian approach, the parameters are viewed as random variables for which we have some a priori knowledge. This *a priori* distribution is then revised in the light of observations to give the *a posteriori* distribution of the true parameter values. That is, for a data set D we seek the hypothesis λ, given as

$$\hat{\lambda} = \arg\max_{\lambda} p(\lambda|D) \tag{2.35}$$

$$= \arg\max_{\lambda} \frac{p(D|\lambda)p(\lambda)}{p(D)} \tag{2.36}$$

$$= \arg\max_{\lambda} p(D|\lambda)p(\lambda) \tag{2.37}$$

In the context of our discussion, the terms $p(D|\lambda)$ and $p(\lambda)$ would correspond to the likelihood function and the Gaussian prior respectively.

Before we leave the topic of smoothing in relation to MaxEnt models, it is also worth noting that another smoothing technique could be to include some strategy for feature selection. This effectively means discarding many of the constraints in the model, and a model with fewer constraints will generally also be more uniform (Chen & Rosenfeld, 2000). The simplest such strategy is simply to impose some frequency cutoff on the features. Leaving out features for which we have the fewest observations also means leaving out those features for which we have the most unreliable estimates.

2.4 Support Vector Machines

In this section we will review another family of discriminative machine learning methods that has rapidly achieved a widespread use and popularity within the statistical NLP community in recent years; Support Vector Machines (SVMs; Boser, Guyon, & Vapnik, 1992; Vapnik, 1998). Our main interest here is the SVM approach developed by Joachims (2002) for learning *ordinal ranking functions*. However, a more common use of SVMs is for learning binary *classifiers*. One advantage of this simpler classification task, from a presentational point of view, is that it makes it easier to describe some of the basic motivations and theoretical concepts underlying the SVM paradigm. Furthermore, as we shall see later, the task of learning ranking functions can be recast in terms of learning a classification function. We will therefore start this section with a brief presentation of SVMs as applied to classification, and then later see how this extends to the case of learning ranking functions. Note however, that an in-depth presentation of all the technical details that underlie the SVM approach would be out of scope for our current purposes. Instead we aim to give a concise and high-level description, providing an intuitive understanding of the underlying concepts, while leaving out some of the more mathematically involved results of statistical learning theory which SVMs build on.

2.4.1 SVM Classification

The learning task for SVMs is typically formulated in terms of training a binary classifier. Let us assume that our training data for this learning task is given by a set of pairs $T = \{(x_1, y_1), \ldots, (x_n, y_n)\}$, consisting of n data points $x_i \in \Re^d$ and associated class labels $y_i \in \{-1, +1\}$. The dimensions of the input space $\Re^d$ correspond to feature functions that encode properties of the data items, just as for the feature vectors $f(s, r)$ in the MaxEnt set-up described in Section 2.3 above. However, in contrast to the MaxEnt approach, the SVM approach has a geometric rather than a probabilistic view on the problem. If the data points we want to classify are viewed as points in a geometric space, the aim is to find the best separating hyperplane in the space. A hyperplane is represented by a normal vector w and an offset b. The corresponding linear decision function learned by the classifier is given as

$$F_w(x) = w \cdot x + b \tag{2.38}$$

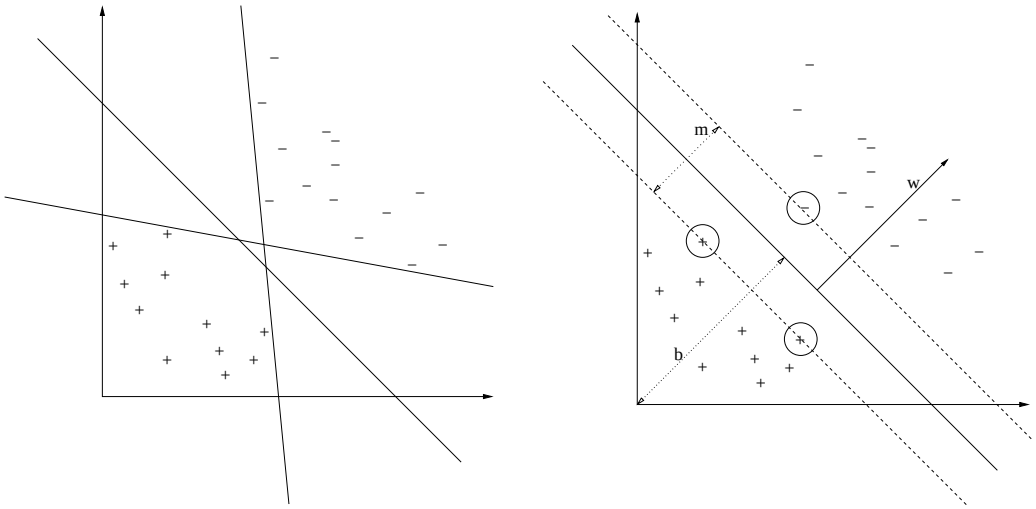
In order to assign a class label to some data point x, we can simply take the sign of $F_w(x)$. The vector w is orthogonal to the separating hyperplane and determines its orientation. The scalar b, know as the *bias*, determines the offset of the hyperplane from the origin. If b was not present or set to zero, the effect would simply be that

w always passed through the origin.

Let us for now assume that the two classes are linearly separable. The hyperplane defined by a normal vector w and bias b correctly separates the data if

$$y_i(w \cdot x_i + b) \geq 1 \text{ for all } 1 \leq i \leq n \tag{2.39}$$

However, there will then generally be infinitely many possible planes that correctly classify the training data. Figure 2.1(a) below shows an example of several possible planes correctly classifying a set of linearly separable data points in two dimensions.



(a) Possible separating planes (b) Max margin plane

Figure 2.1: The figure to the left shows examples of some possible separating planes in a two-dimensional feature-space that correctly classifies the linearly separable data of two classes labeled $+$ and $-$. In the figure to the right, we see an example of a maximum margin classifier with a plane that correctly classifies the linearly separable data from two classes, labeled $+$ and $-$. The margin is denoted by m, while w is the parameter vector and b is the bias. The circled points lying on the boundaries of the margin (i.e. the dashed lines) are the support vectors.

Now, out of all the possible hyperplanes that correctly separate the data, which should we choose as our classifier? The SVM answer to this question is to choose the hyperplane that gives the maximum margin separation between the classes (Cristianini & Shawe-Taylor, 2000). In other words, the SVM learner will pick the hyperplane that is furthest away from any of the data points in both classes. SVMs are therefore an example of so-called *maximum margin* classifiers. This approach to selecting a separating hyperplane seems to accord well with our intuitions. Quite intuitively we would expect that, by making sure the margin space

between the decision boundary and the nearest points in each class is as large as possible, the classifier will be maximally robust with respect to changes in the distribution of the data and have the best generalization capabilities when applied to new examples. Formally, the justification for max-margin learning is embedded in the principle of *structural risk minimization* from statistical learning theory (Vapnik, 1998).

An example of a maximum margin plane that correctly classifies a data set in two dimensions is given in Figure 2.1(b). The circled data points lying on the parallel planes that define the margin are the *support vectors* (SVs). As can be seen from the figure, the decision boundary is entirely defined in terms of the SVs, which are the points lying closest to the hyperplane. Geometrically the size of the margin, the distance between the dashed lines in Figure 2.1(b), is given as $2/\|w\|$. This means that, in order to maximize the size of margin m, we can try to minimize the norm of the parameter vector

$$\|w\| = \sqrt{\sum_i w_i^2} \tag{2.40}$$

while at the same time satisfying the constraints of Equation (2.39).

The problem of finding a separating hyperplane as described above is known as *hard margin* optimization. However, in cases where our training data contains noise or outliers, such a separating hyperplane might not exist. Cortes and Vapnik (1995) therefore extended the ideas above to also handle such *non-separable* data. When a separating hyperplane cannot be found, a solution can still be approximated by allowing for some misclassified examples. The training error is measured by so-called *slack variables* ξ_i, which are then used for penalizing the objective function for misclassifications. The corresponding learning problem is known as *soft margin* optimization. The parameters of the corresponding maximum-margin hyperplane are derived by solving the following optimization problem:

Minimize: (2.41)

$$V(w, b, \xi) = \frac{1}{2} w \cdot w + C \sum_{i=1}^{n} \xi_i$$

subject to the constraints (2.42)

$$\forall 1 \leq i \leq n : \; y_i(w \cdot x_i + b) \geq 1 - \xi_i$$

$$\forall 1 \leq i \leq n : \; \xi_i \geq 0$$

The penalty factor $C > 0$ governs the trade-off between the training error and margin size. This constant must be specified by the user, and by increasing the value of C the margin is made "harder".

The minimization problem stated above is an instance of a convex quadratic programming (QP) optimization problem, and there exist several algorithms for solving such problems. In this thesis we will be using the freely available SVM^{light} toolkit[7] which is an implementation of the optimization techniques described by Joachims (1999). However, most optimization methods do not attempt to directly solve the primal problem defined in Equations (2.41) and (2.42) above. Instead one tries to solve the *dual formulation* of the learning problem, which is defined as follows (Cristianini & Shawe-Taylor, 2000):

Maximize: (2.43)

$$W(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j (x_i \cdot x_j)$$

subject to the following constraints: (2.44)

$$\forall i : \; C \geq \alpha_i \geq 0$$

$$\sum_{i=1}^{n} \alpha_i y_i = 0$$

The goal is then to find a vector of parameters α_i representing the relative contribution of each example x_i in the solution. We see that the slack variables of the linear penalty function in Equation (2.41) are not present in the dual formulation of the problem. Instead, the constant C appears only as an additional constraint on the Lagrange multipliers in Equation (2.44).

The actual support vectors correspond to the examples x_i for which $\alpha_i > 0$. These are the examples that lie closest to the hyperplane. When it comes to applying the classifier to a new instance x, the dual formulation of the decision function in (2.38) can be stated as

$$F_w(x) = \sum_{i=1}^{n} \alpha_i y_i x_i \cdot x \; + b \tag{2.45}$$

Note that the bias b is specified so that $y_i F_w(x_i) = 1$ for all SVs (i.e. for each x_i with $\alpha_i > 0$).

When reading the dual formulation of the optimization problem in Equation (2.43) and the decision function in Equation (2.45), it should be recognized that the weight vector w that represented the solution to the primal problem defined in Equations (2.41) and (2.42), can here be seen expressed as a linear combination of the points in the training data:

$$w = \sum_{i=1}^{n} \alpha_i y_i x_i \tag{2.46}$$

[7] See `http://svmlight.joachims.org/` for more information on the SVM^{light} package.

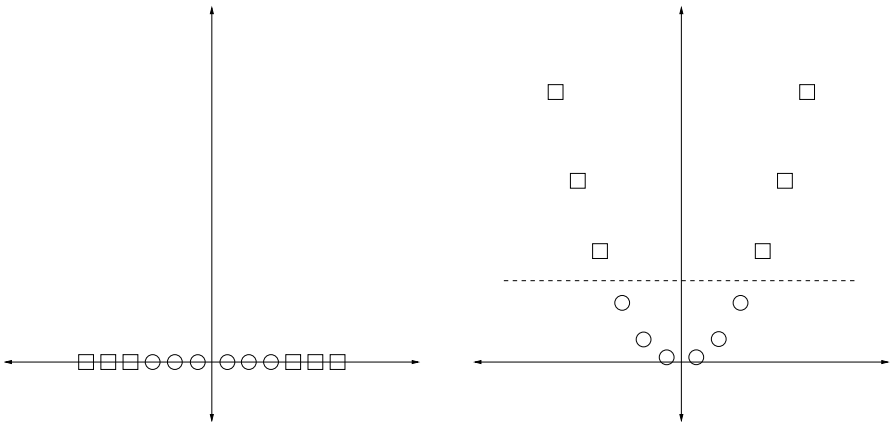
When looking at the definition of the scoring function in Equation (2.45) above, we see that it is only the SVs that end up having any influence. This furthermore means that the solution corresponding to the weight vector in Equation (2.46) typically will be quite sparse, since only features of the support vectors get a non-zero weight.

So far the learner we have described can only acquire linear decision boundaries. As said, when the data is not linearly separable in the given input space the use of slack variables means we can still find a solution by allowing some errors. However, another possibility for actually overcoming this limitation on linearity, is to map the data into a higher-dimensional (possibly infinite) feature space and try to find a solution there instead. A linear solution might exist in this high-dimensional feature space which is highly non-linear with respect to the original input space. As we shall see, by taking advantage of some properties of the dual representation just described, SVMs can accomplish this in an effective way by the use of so-called *kernel functions*.

Up until now, we have dealt directly with the data points x_i, where each element x_{ij} is taken to encode the value of some specified feature function. However, a key property of so-called kernel methods like SVMs is to make use of an additional mapping $\Phi(x_i)$, which might include non-linear transformations and combinations of the original features. As an example of how a simple mapping can make a difference with respect to separability, consider the binary classification problem in Figure 2.2 below. In the one-dimensional representation in Figure 2.2(a), only the x-axis is specified, and the data points are not linearly separable. However, in the two-dimensional representation in Figure 2.2(b), we include the additional mapping $y = x^2$, rendering the data linearly separable.

An important property of the dual representation in Equation (2.43) is that the data is only ever used in the form of dot products. Linear learning algorithms such as SVMs, that can be cast in terms of dot products, are known as *kernel methods*, and the so-called *kernel trick* allows us to easily formulate non-linear variants of such methods in terms of the Φ projections. A kernel function is simply a function that returns the dot product of the mapped representations of the two arguments x_i and x_j in the feature space defined by Φ, i.e. $K(x_i, x_j) = \Phi(x_i) \cdot \Phi(x_j)$. One benefit of using kernels is that $K(x_i, x_j)$ can be specified directly, so that it is not necessary to explicitly carry out the mapping from the original input space to the higher-dimensional feature space. This means that the computational cost associated with the learning problem is not necessarily proportional to the dimensionality of the feature space defined by Φ.

There are a few properties that a kernel function must possess to ensure that it actually corresponds to some feature space. In addition to being symmetric, they are typically also required to satisfy Mercer's theorem, which states that the matrix $K = (K(x_i, x_j))_{i,j=1}^n$ must be positive semi-definite, i.e. it has no non-negative



(a) One dimension (x-axis only) (b) Two dimensions ($y = x^2$)

Figure 2.2: An example of a two-class data set (class membership corresponding to circles and boxes) represented in two different feature spaces; one in which only values for the x-axis are specified, and one with the additional mapping of $y = x^2$. In the one-dimensional input space the data points are *not* linearly separable, in its two-dimensional image, however, they are.

eigenvalues (Cristianini & Shawe-Taylor, 2000).

There are many ways that a kernel function can be defined and tailored to the specific problem at hand. However, some of the more standardly used kernels (in addition to the linear kernel) include *polynomial* kernels and *radial basis functions* (RBF). Given two input vectors x and y, these kernels are defined (with various kernel parameters) as follows, each corresponding to the dot product in some feature space:

$$\text{Linear:} \qquad K(x, y) = x \cdot y \tag{2.47}$$

$$\text{Polynomial:} \qquad K(x, y) = (x \cdot y + c)^d \tag{2.48}$$

$$\text{RBF:} \qquad K(x, y) = \exp\left(-\gamma \|x - y\|^2\right) \quad \text{for } \gamma > 0 \tag{2.49}$$

RBFs where the kernel parameter is of the form $\gamma = \frac{1}{2\sigma^2}$ are also known as Gaussian kernels.

This concludes our short review of SVM classifiers. In the next section we turn to the issue of how SVMs can be adapted for the task of learning ranking functions.

2.4.2 SVM Ranking

For the purpose of ranking documents in information retrieval, Joachims (2002) develops an SVM approach for learning ranking functions. The approach has similarities to ordinal regression, but the problem is formulated in terms of structural risk minimization, yielding a large margin approach similar to the SVM classifiers described above. The method is similar to the SVM approach to ordinal regression developed by Herbrich, Graepel, and Obermayer (2000). Based on relevance judgments from user data, Joachims (2002) uses an SVM for learning preference relations among retrieved documents relative to an input user query. In the following, however, we describe the method of Joachims (2002) in terms the problem explored in this thesis; learning a ranking function from a set of preference relations on sentences, relative to a given input semantics. The goal is to learn a ranking function that reflects that, for a given semantics s, some realization r_i is ranked higher than r_j. We will write $r_i \prec_s r_j$ to denote such a binary preference relation. Furthermore, we take $\prec$ to define a strict partial order on the set of possible realizations (i.e. it is irreflexive, asymmetric, and transitive).

Let the training data be given by a set of triplets $X = (s_i, r_j, y_{ij})$, where s_i is a semantic representation, $r_j \in \mathcal{Y}(s_i)$ is a realization, and $y_{ij} \in \Re$ is a label. Typically the range of y_{ij} will be restricted to $[0, 1]$, or even $\{0, 1\}$, but it can be defined to take on any real value. y_{ij} can be manually specified or be automatically defined by, for example, some string-similarity metric measuring the similarity between a reference sentence and the candidate realizations. In any case, regardless of the kind of scoring scheme we use, we would not be interested in the actual scores themselves, but only the relative ranks that the scores impose on the examples. On the basis of the label values y, each semantics s in our data is associated with an ordering relation $\prec^*_{s_i}$ over pairs of realizations in $\mathcal{Y}(s_i)$. We will sometimes just write $\prec^*_i$ for short. $\prec^*_i$ is defined by the relative ordering imposed by the values of y that are assigned to the realizations in the training data. To be precise, given two triplets $(s_i, r_j, y_{ij}) \in X$ and $(s_i, r_k, y_{ik}) \in X$, where $j \neq k$, we say that $r_j \prec^*_{s_i} r_k$ iff $y_{ij} > y_{ik}$. In the following, to make the exposition less cumbersome, we usually just refer to the training data by way of this encoded preference relation $\prec^*$, instead of by explicitly referring to the triplets in X. If we have n distinct semantic representations s_i in our training data, we get n ordering relations $\prec^*_i$, each defined on the corresponding space $\mathcal{Y}_i \times \mathcal{Y}_i$.

Now, the goal is to learn an ordering relation $\prec$ which as closely as possible reflects this original ranking $\prec^*$. This match can be measured by the number of *inverted pairs* between the two orderings. An inversion occurs if the two ranking functions disagree on the ordering of a given pair, that is if $r_k \prec_i r_j$ while $r_j \prec^*_i r_k$. Let $Q^{\prec}_i$ correspond to the number of inverted pairs incurred by f for a

particular training item s_i.

$$Q_i^{\prec} = \sum_{jk} \begin{cases} 1 & \text{if } r_j \prec_i r_k \ \wedge \ r_k \prec_i^* r_j \\ 1 & \text{if } r_k \prec_i r_j \ \wedge \ r_j \prec_i^* r_k \\ 0 & \text{otherwise} \end{cases} \tag{2.50}$$

The objective is thus to learn a ranking function from a family of functions f that minimizes the number of such inverted pairs with respect to the training data, $\frac{1}{n}\sum_{i=1}^{n} Q_i^{\prec}$. Joachims (2002) points out that this can be seen as analogous to the objective of minimizing the number of training errors in relation classification as described above, while the target here is not a class label but a binary ordering relation.

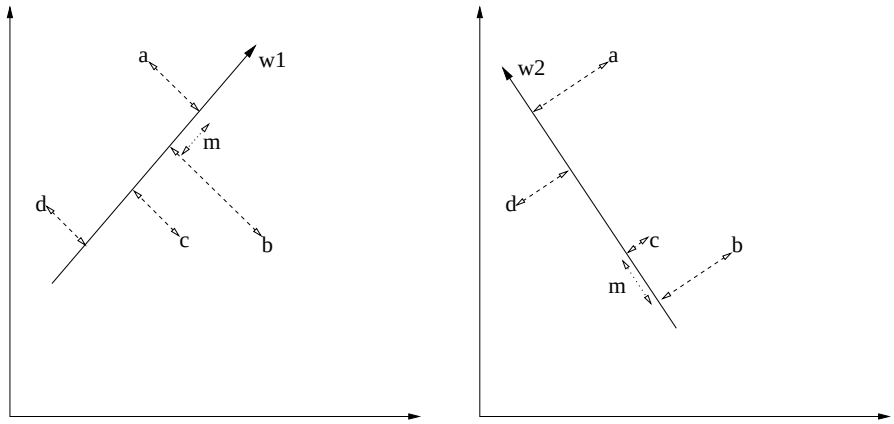
In order to make the exposition clearer, we will here write $\Phi(s, r)$ directly when representing the feature vectors of the training examples, instead of going by way of a feature vector $x = f(s, r)$ and then defining the additional mapping into a feature space $\Phi(x)$. The type of functions that the SVM ranker learns are linear ranking functions defined as

$$r_j \prec_i^w r_k \iff w \cdot \Phi(s_i, r_j) > w \cdot \Phi(s_i, r_k) \tag{2.51}$$

As described by Joachims (2002), the data points are ordered by their signed distance to a hyperplane with normal vector w or, equivalently, by their projections onto w. An example of this is shown in Figure 2.3, where two different rankings are imposed on four data points in two dimensions (where a given point corresponds to a mapping $\Phi(s_i, r_j)$). Note that, the bias b is actually assumed to be 0 for the linear ranking functions considered here and is therefore also left out in Equation (2.51). For the class of linear ranking functions in Equation (2.51), the problem of minimizing the number of inverted pairs is equivalent to finding the w which fulfills the maximum number of the following constraints:

$$\begin{aligned} &\forall (j,k) \text{ s.t. } r_j \prec_1^* r_k : \ w \cdot \Phi(s_1, r_j) \ > \ w \cdot \Phi(s_1, r_k) \\ &\ldots \\ &\forall (j,k) \text{ s.t. } r_j \prec_n^* r_k : \ w \cdot \Phi(s_n, r_j) \ > \ w \cdot \Phi(s_n, r_k) \end{aligned} \tag{2.52}$$

Now, just as for the SVM classifiers described above, the maximum margin solution can be approximated by the use of slack variables ξ. As shown in Figure 2.3, the margin m here corresponds to the distance between the two projections that are closest among all the points which we are trying to order (Joachims, 2002). The formulation above leads to the following optimization problem (Joachims,



(a) SVM ranker $w1$ (b) SVM ranker $w2$

Figure 2.3: Examples of the rank order imposed on four data points by the weight vector learned by two different SVM rankers. In the first example (left) we get $a \prec b \prec c \prec d$, with a as the top ranked candidate. In the second example we get the ordering $a \prec d \prec c \prec b$.

2002, Sec.4.2):

Minimize: (2.53)

$$V(w, \xi) = \frac{1}{2} w \cdot w + C \sum \xi_{i,j,k}$$

subject to the following constraints: (2.54)

$$\forall (j,k) \text{ s.t. } r_j \prec_1^* r_k : \; w \cdot \Phi(s_1, r_j) \; > \; w \cdot \Phi(s_1, r_k) + 1 - \xi_{1,j,k}$$

$$\ldots$$

$$\forall (j,k) \text{ s.t. } r_j \prec_n^* r_k : \; w \cdot \Phi(s_n, r_j) \; > \; w \cdot \Phi(s_n, r_k) + 1 - \xi_{n,j,k}$$

$$\forall (i,j,k) : \; \xi_{i,j,k} \geq 0$$

As before, C governs the trade-off between maximizing the margin size and minimizing the training error.

The connection between the classification task described in the preceding section and the ranking task we describe here is revealed when rewriting the constraints in the optimization problem above. The ranking constraints listed in Equation (2.54) can equivalently be expressed as follows:

$$\forall (j,k) \text{ s.t. } r_j \prec_i^* r_k : \; w \left(\Phi(s_i, r_j) - \Phi(s_i, r_k) \right) > 1 - \xi_{i,j,k} \tag{2.55}$$

Writing the constraints in this form makes it clear that the ranking task formulated by Joachims (2002) is actually equivalent to that of a binary *classification*

task defined for the pairwise *difference vectors* corresponding to the ordered data points. This means that the SVM ranker can be trained using the same set-up and similar methods as described for the SVM classifiers in Section 2.4.1. A new set of training data T corresponding to the classification task can be derived from X as follows. For each pair of realizations r_j and r_k in X that are associated with a preference relation $r_j \prec_i^* r_k$ (i.e. $y_{ij} > y_{ik}$), we compute their difference vector $x_l = \Phi(s_i, r_j) - \Phi(s_i, r_k)$. Depending on whether a given r_j is preferred to a given r_k or the other way around, the newly constructed training example x_l is labeled $y_l = +1$ or $y_l = -1$ respectively. The pair (x_l, y_l) is then added to T. Note that, we only need to add one training example for each pair r_j and r_k. For instance, if a positive example has already been added to T for $\Phi(s_i, r_j) - \Phi(s_i, r_k)$, then we do not need to add an additional negative example for $\Phi(s_i, r_k) - \Phi(s_i, r_j)$, since these would correspond to equivalent constraints on the model.

Equation (2.51) shows how an SVM ranker orders a set of possible realizations relative to a given semantic input s. Just as when applying the MaxEnt models described in Section 2.3, each candidate is scored by taking the dot product of its feature vector and the learned weight vector w. The candidates can then be sorted accordingly. However, just as for the scoring function for the SVM classifiers in Equation (2.45), the weight vector and the ranking function can be expressed as a linear combination of the support vectors in the training data. This opens up the possibility of using kernels and learning non-linear ranking functions.

This concludes our foundational review of the various machine learning methods that we will later put to use for the problem of realization ranking. In the next chapter we move on to get familiar with some of the previous work done within the field in relation to this and related tasks.

Chapter 3

Previous and Related Work

3.1 Overview

As noted in Section 1.1, most traditional natural language generation (NLG) systems are rule-based and strictly symbolic in nature. These earlier systems are "deterministic" in the sense that they only construct a single candidate string for a given input. Looking at more recent NLG research, however, there is a growing tendency to allow for some degree of indeterminacy in the generation process and let the system pursue several possible choices simultaneously. In cases where multiple choices are available, data-driven methods can be used for discriminating between the possible realizations. In this section we will take a look at some of these more recent NLG systems that incorporate a statistical component for making decisions when multiple generation choices are possible, thereby bearing similarities to the approach we develop in this thesis. Although the aspect we want to focus on here is the use of empirical methods for ranking and selecting realizations, we sometimes also need to delve into the details of particular generation algorithms as well. This is because the two processes are often tightly intertwined, or, in cases where the empirical methods work independently of the generation algorithm proper, the decisions made by one component can have consequences for the type of decisions left to be made by the other.

It is a somewhat curious fact that, when compared to other areas of NLP, the use of statistical methods was adopted rather late within the sub-field of NLG. For example, within the field of parsing or natural language understanding (NLU), which in many ways is the natural counter part of NLG, there is a much longer tradition of using data-driven methods. Just as there might be several possible strings corresponding to a given semantics in generation, there might be several possible analyses corresponding to a given string in parsing. However, while the number of works on realization ranking is relatively modest, numerous studies exist on the

task of parse disambiguation using corpus-based statistics. Moreover, there are are many obvious similarities between the ranking tasks within grammar-based generation and parsing. By tapping into some of the state-of-the-art methods for parse disambiguation one might hope to transfer some of these results and insights over to the problem of realization ranking. In Section 3.3 we review some of the previous work on parse selection that is most relevant to our own experiments on realization ranking. Finally, in Section 3.4, we also present a few selected examples on work on discriminative reranking for *statistical machine translation* (SMT). This is yet another task which in many ways neighbors on the problem of realization ranking. In one way or another, any SMT system needs to include something that resembles a generation step in order to produce the target sentence. As there should be no need to point out, given that our own generation task is actually embedded in an MT system, the need to efficiently handle the problem of indeterminacy is very much present in any large-scale MT system.

3.2 Statistical Generation

Within the field of NLG, the development of the hybrid Nitrogen system (Knight & Hatzivassiloglou, 1995; Langkilde & Knight, 1998a) at ISI in the late 1990s is widely regarded as the work that pioneered the use of more empirically oriented approaches. Because the Nitrogen system, and its successor HALogen (Langkilde, 2002), provided much of the ground-work that other systems have later extended on, we will devote a little bit more space here on this particular project than on some of the other data-driven systems that have since followed.

After starting off by looking into the n-gram-based method used in Nitrogen and HALogen, we will look at some other projects that follow a similar approach, such as (Bangalore & Rambow, 2000; White, 2004; Habash, 2004; Ratnaparkhi, 2000). We then review the hybrid Amalgam system which is based on decision tree classifiers for guiding the generation process (Gamon, Ringger, & Corston-Oliver, 2002). Finally we will be looking at a few approaches that use models similar to *probabilistic context-free grammars* (PCFG) as are used in parsing (Humphreys, Calcagno, & Weise, 2001; Belz, 2005; Cahill & Genabith, 2006).

3.2.1 Nitrogen and HALogen

The work on the seminal Nitrogen system started in the context of the knowledge-based Japanese-English machine translation system JAPANGLOSS (Knight et al., 1995), where target language generation was initially performed by the Penman generator (The Penman Project, 1988). This is a grammar-based sentence realizer based on the Nigel generation grammar for English (Mann & Matthiessen, 1985),

using the framework of Systemic Functional Grammar (SFG). Knight and Hatzivassiloglou (1995) noted two important sources for disfluencies in the output of their generation component. On the one hand there was missing knowledge on the part of the source language analyzer, such as incomplete or inaccurate representations of number, definiteness or time information which can be difficult to infer from Japanese. Such missing information does not necessarily arise because of deficiencies in the analyzer, but can be due to properties of the source language itself. This is a typical example of why it sometimes makes sense to allow for underspecification in MT; There might be information that needs to be made explicit in the target language, but which simply is not provided in the source language. In such cases it can be a sensible strategy to leave the possibilities open at the point of analysis, and instead leave the decision to the target realizer. On the other hand, Knight and Hatzivassiloglou (1995) also reported problems related to gaps in the knowledge of the generator itself. For instance, the generation lexicon would contain incomplete or inaccurate knowledge of lexico-syntactic constraints, selectional restrictions or collocational information. In order to ensure robustness in cases of missing information, the Penman generator relies on pre-specified defaults (such as giving preference to singular noun phrases, *in* over *on*, *that* over *who*, nominalizations over clauses, active voice, the alphabetically first synonym for open-class words, and so on).

As noted by Knight and Hatzivassiloglou (1995), however, such heuristics often fail to produce fluent and natural-sounding output, and the problems only get worse when trying to scale up the system. Of course, another available strategy is simply to fall back on a random choice among the possible alternatives in such cases, but this would not be a much more principled way of attacking the problem. These issues lead Knight and Hatzivassiloglou (1995) to suggest an alternative approach; modeling fluency as likelihood. This way Knight and Hatzivassiloglou (1995) formulated the realization task as a two-stage process; first generate all the possible sentences for a given semantic input, and then pick the most likely one according to an n-gram-based statistical language model (LM). The likelihood of a candidate realization is simply computed as the joint probability of the words in the string. In an n-gram model this joint probability is further decomposed into the product of the conditional probability of each word given the $n-1$ words preceding it. The framework of n-gram language modeling was already widely used in other areas of NLP such as speech recognition and machine translation, when Knight and Hatzivassiloglou (1995) adopted it for this *generate-and-select* task, and Section 2.1 above provides some more background and mathematical details for this framework.

To create a statistical language model for English, Knight and Hatzivassiloglou (1995) estimated bigram probabilities from an unannotated text collection of 46

million words taken from the WSJ corpus[1]. In HALogen, however, the successor of the Nitrogen system, a larger bigram model was trained on a corpus of 250 million words of WSJ texts (Langkilde, 2002). Given an estimated LM, the various English target sentences that are generated for a given input are then scored and ranked according to their (log-)probabilities.

In most cases, many sub-phrases will be shared across the different possible surface realizations that are generated. This means that a lot of information will be duplicated if one sets out to explicitly list all of them. Also, the scoring function would need to do a lot of duplicated work, evaluating many of the same phrases over and over again. So, in order to have the scoring function be applied more efficiently, Nitrogen first maps the generated sentences into a more compact representation that avoids duplicating all of the parts that are common to many sentences. In the earlier versions of Nitrogen this was done by using a *word lattice* (Knight & Hatzivassiloglou, 1995) —a directed acyclic graph where the different sentences are effectively represented as different transition paths. Given such a lattice representation, the scoring phase can *extract* the best sentence without the need to explicitly enumerate all of the alternatives. The actual sentence extraction is implemented as a beam search that retrieves the N highest scoring paths with respect to the language model (Knight & Hatzivassiloglou, 1995).

Although a lattice structure provides a way of efficiently representing some of the shared structure, some redundancy still remains. This also means that some of the computations involved in scoring also need to be repeated several times, thus making the extraction less efficient. There are also other problems associated with the use of word lattices. Langkilde and Knight (1998a) note that the number of competing realizations produced by their system usually grows exponentially with the length of the phrase, and that the lattice structures typically end up being too large for an exhaustive search to be practically feasible. The approximation offered by the heuristic N-best search that is used instead is not guaranteed to recover the mathematically optimal solution. In later versions of the Nitrogen and HALogen systems, the generated sentences are therefore instead mapped to a more compact *forest* representation (Billot & Lang, 1989; Langkilde, 2000).

A lattice structure can be converted to a forest by assigning a label to each unique arc in the graph, and letting these labels correspond to nodes in an AND/OR tree. All duplicated arcs in the lattice will thereby be referenced by one and the same node label in the forest. Alternatively, the same structure can be represented as a set of non-recursive context-free rewrite rules (Langkilde, 2000).

Given a generation forest, extracting the most likely sentence according to the LM can be carried out efficiently using a bottom-up dynamic programming tech-

[1]The Wall Street Journal, available from the Linguistic Data Consortium, http://www.ldc.upenn.edu/

```
(e1 / eat
    :subject (d1 / dog)
    :object (b1 / bone
                :premod (m1 / meaty))
    :adjunct (t1 / today))


(e2 / eat
    :agent (d2 / dog)
    :patient (b2 / bone
                 :premod (m2 / meaty))
    :temporal-locating (t2 / today))
```

Figure 3.1: Example inputs to HALogen

nique. The method involves breaking down the score of each node in the forest into an *external* score and an *internal* score. While the external score depends on features of the sibling nodes, the internal score is context-independent and is stored within the node once it is computed. Each phrase will have a set of externally relevant features that contribute to the external context-dependent scores of its sibling nodes. In the case of an n-gram model, the externally relevant features are simply the first and the last $n - 1$ words of the phrase. For each node in the forest, one only needs to keep track of the phrase with the best internal score for each unique combination of externally relevant features. This means that an exponential number of phrases can be pruned away while the optimal solution is still guaranteed to be recovered in the end.

The meaning representations used by Nitrogen are encoded in the so-called Abstract Meaning Representation language (AMR), which is based on the SENSUS knowledge base (Knight & Luk, 1994) and WordNet (Miller, Beckwith, Fellbaum, Gross, & Miller, 1990). A feature of the generation grammar is that it allows for a continuum of underspecification in the input AMR structures, and relations or properties can be specified on different levels of abstraction, either syntactically or semantically. Examples of two input structures for HALogen are shown in Figure 3.1 (Langkilde, 2002). The ordering of relations in the input does not affect their ordering in the output, except in cases where the same relation (e.g. modifier and adverbial relations) occurs multiple times at the same level of nesting. Flags can also be set to control the amount of permutations allowed in the output in such cases.

Two levels of rules are invoked during the symbolic processing. Before the input semantic structure is mapped to a word lattice, it first goes through a so-called

recasting phase where the structure is reformulated to a less abstract and more syntactically detailed representation. This rewriting step is a one-to-many mapping, producing structures that lie closer to the final linearization, and the amount of rewriting needed will of course also depend on the specificity or abstraction level of the original input structure. After the recasting of the AMR feature structure, a set of grammar rules is then recursively applied, in a bottom-up compositional fashion, to map it into a packed lattice or forest representation. The number of complete realizations encoded in the packed structures that Nitrogen produces is reported to typically be on the order of several hundred thousands. As further described below, the simplicity of the lexical, morphological and grammatical knowledge in the generator means that the system massively overgenerates by design, leaving the bulk of linguistic decisions to the bigram model in the extraction phase.

Nitrogen's *many-paths, two-level* generation model, as Knight and Hatzivassiloglou (1995) dub it, means that the grammar and the lexicon can be greatly simplified by shifting a lot of the linguistic decision-making to the statistical component. In Nitrogen, the main task of the initial symbolic level is to provide a mapping from semantic to syntactic relationships. Detailed linguistic information is left out and the system trusts the secondary statistical extraction phase to make proper choices for a range of phenomena such as word choice, number, determinateness, subject-verb agreement, verb tense, voicing, verb sub-categorization, and more (Langkilde & Knight, 1998b).

Also, the lexicon is greatly simplified by delegating responsibility to the statistical component. A great deal of lexical knowledge is thereby left out of the underlying knowledge base, such as subcategorization of verbs, gradability of adjectives, and countability of nouns (Langkilde & Knight, 1998a). In addition, aspects of both inflectional and derivational morphology are also left to the extraction module. The rules and tables in the generator knowledge base are greatly simplified, and in some cases this means deliberately overgenerating (e.g. *photo* $\rightarrow$ *photos* / *photoes*, or *potato* $\rightarrow$ *potatos* / *potatoes*), and leaving it to the LM to pick the correct forms (*photos* or *potatoes*). Since the incorrect forms will typically receive a very low likelihood from the corpus estimates, they will be ranked low by the statistical LM and thereby be discarded in the post-generation extraction phase.

Examples like this illustrate the division of labor that exists between the statistical and symbolic components in the hybrid Nitrogen system. Compared to the use of rigid defaults, the use of corpus-based likelihood estimates provides a much more robust and flexible way to handle underspecification or otherwise missing information in the input or the knowledge base. Furthermore, automatically acquired corpus-based statistical knowledge, here in the form of n-gram probabilities, is also used to replace parts of manually crafted resources. Hybrid

systems can thereby be suited to overcome the traditional bottleneck in creating large-scale NLG systems, the process of knowledge acquisition.

An obvious limitation of the extraction component of the Nitrogen system is that it entirely relies on the use of unigram and bigram statistics. As also pointed out by Langkilde and Knight (1998b), an ordinary surface-based n-gram language model cannot model long-range dependencies between non-contiguous words. Neither can it capture dependencies between more than n words. In relation to this, Langkilde and Knight (1998b) discuss the potential benefits of moving to n-grams of a higher order, such as trigrams. The same inherent limitations will still hold of course, but Langkilde and Knight (1998b) furthermore claim that moving to larger n-grams might even lead to worse performance because of problems related to data sparseness. Langkilde and Knight (1998b) argue that one would need to store exponentially more data and apply more extensive smoothing to avoid zero scores. Furthermore, since many more trigrams than bigrams will necessarily have zero counts, Langkilde and Knight (1998b) argue that moving to a trigram model might introduce more mistakes because of the smoothing of the counts relative to unigram counts. However, these arguments do not seem to take into account the use of back-off models, which are now common-place within the field of language modeling. In such a set-up, if the observed count for a given n-gram is below some threshold, one simply backs off to a lower order model, $n - 1$. There are also other variations over this basic idea, which we reviewed in Section 2.1 where we described the technicalities of language modeling in more detail.

3.2.2 Other Systems

Another early hybrid generator is the FERGUS system (Flexible Empiricist/Rationalist Generation Using Syntax) described by Bangalore and Rambow (2000). Similarly to Nitrogen, FERGUS too uses an n-gram language model for extracting the most likely surface realization from a word lattice. However, the generator also includes a tree-based stochastic model, in addition to a wide-coverage grammar for English called XTAG (XTAG Research Group, 2001). This is a hand-crafted, general-purpose and declarative grammar based on the lexicalized Tree Adjoining Grammar (TAG) formalism. There are three main modules involved in the generation process in FERGUS; After a dependency tree is given as input to the system, the stochastic *Tree Chooser* performs syntactic annotation of the nodes to produce a semi-specified derivation tree; the grammar-based *Unraveler* then maps the possible realizations of this structure to a word lattice; and finally the stochastic *Linear Precedence (LP) Chooser* extracts the most likely linearization.

The input to FERGUS takes the form of a dependency tree representing lexical predicate-argument structure. Nodes in the dependency tree are only labeled with

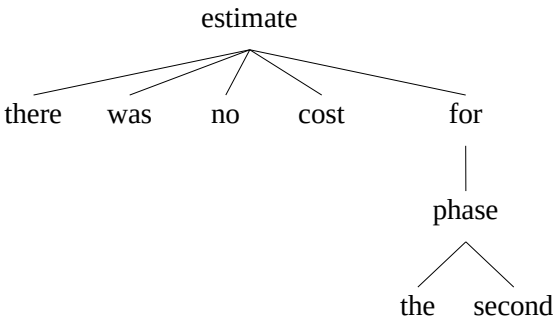


Figure 3.2: Example of input to FERGUS

lexemes, and in the first stage of generation the input is syntactically annotated using a stochastic tree model in a process that is described as being analogous to supertagging (Bangalore & Joshi, 1999), but annotating words in a tree instead of in a linear sequence. The supertags here constitute references to elementary trees from the underlying TAG grammar and encode syntactic properties of a lexical head such as grammatical function, subcategorization frame, realization of arguments, etc. The stochastic tree model is trained on an annotated corpus of XTAG derivations for one million words of the WSJ.

The semi-specified derivation trees produced by the Tree Chooser might still be ambiguous with respect to adjunct and argument positions if their syntactic role is unspecified. In the next stage, the Unraveler maps a supertagged tree to a word lattice that represents all the alternative possible linearizations. The possible orderings of nodes in the tree are determined by the underlying XTAG grammar.

In the final stage of the generation process, the LP Chooser extracts the most likely realization from the lattice with respect to an n-gram language model. The n-gram statistics are collected from unannotated text from the WSJ, similarly to the set-up described for Nitrogen above, although Bangalore and Rambow (2000) train a trigram model instead of a bigram model.

The input dependency structures that FERGUS operates from are fairly specified, and although FERGUS performs some lexical choice and syntactic choice, the bulk of the necessary decisions to be made are related to word order. Figure 3.2 shows an example taken from Bangalore, Chen, and Rambow (2001) that illustrates what the dependency trees given as input to the system look like. In the experiments described by Bangalore et al. (2001) the (unordered) nodes in the input tree must include all words (including function words), in fully inflected form (which the authors admit is unrealistic for actual applications). However, in terms of argument structure and syntactic roles, the input might optionally be par-

tially or completely underspecified. As noted by Bangalore and Rambow (2000), the input structures required by FERGUS must be said to be more syntactic than semantic in nature.

A system that more closely resembles our own LKB generator, is the realization module described by White (2004), in that it performs generation from logical form input. Furthermore, the grammar-based generation algorithm itself is chart-based,[2] similarly to the approach taken in the LKB generator (Kay, 1996; Carroll et al., 1999; Carroll & Oepen, 2005). The realizer is part of the open source OpenCCG package, based on the framework of *combinatory categorial grammar* (CCG). White (2004) proposes a number of techniques for making chart realization more efficient, including the use of n-gram models for search and pruning, which are the methods most relevant for our purposes.

Just as the Nitrogen and FERGUS systems, the OpenCCG realizer relies on n-gram statistics for selecting the final string to be output by the system. However, the approaches described so far all follow a generate-and-select set-up. In these two-level approaches, the statistical models are applied in a separate post-processing step. The OpenCCG realizer implements two different modes of realization. One mode runs this traditional two-stage process with (symbolic) packing followed by (statistical) unpacking (White, 2005). This mode produces a forest structure followed by a bottom-up extraction stage, similarly to the approach described by Langkilde (2000). In the other mode, however, OpenCCG implements a proposal by Varges and Mellish (2001) where the statistical ranking is integrated in the chart algorithm itself in a so-called *anytime search*. This means that a best-first search is performed with a specified time limit for when the system should return the best available realization discovered so far (as soon as at least one complete realization has been constructed). The implementation relies on treating the agenda as a priority queue sorted by n-gram scores (White, 2004). During the bottom-up generation process, edges are scored as soon as they are built, and ordered on the agenda accordingly, before being successively moved to the chart and combined. The search is over when the agenda is empty and no more combinations can be made, or when the time limit for the anytime search has expired.

For the LM used for scoring the edges, White (2004) uses a 5-gram back-off model. The order of the model drastically extends on those used in other systems such as Nitrogen and FERGUS, but this is possible because the parameter estimation is done over much smaller data sets, using 25-fold cross-validation. The model also includes expansion into semantic classes for certain named entities.

In the OpenCCG realizer, the n-gram scores are also used for N-best pruning of edges. By specifying a limit on the maximum number of edges in the chart that can have different strings but equivalent categories and cover the same parts

[2] More details on the notion of chart-based generation are provided in Section 4.3.

of the input semantics, the system prunes away the edges whose strings have the lowest n-gram score (White, 2004). Although edge pruning does not affect the possibility of finding a complete realization, it can prevent the system from ever finding the optimal candidate with the best LM score.

As the focus in the OpenCCG project is on precision and high-quality output rather than wide coverage, the grammars used by White (2004) overgenerate only mildly, in contrast to the massive overgeneration seen with the generation grammars used in Nitrogen and HALogen. The main indeterminacy here is related to ordering of adjectival and adverbial modifiers, and most of the decisions pertaining to lexical choice are already specified in the input.

In any case, the main motivation for integrating anytime search in the chart realization is to increase the efficiency and running times, which are often crucial in "online" applications of NLG such as dialog-systems or machine translation. Depending on how one chooses to configure the thresholds governing the time cut-offs in the anytime search, one can control the trade-off between processing time and accuracy. White (2005) notes that for single-best output, the anytime mode can provide significant time savings by cutting off the search early. However, for producing an N-best list or even performing a complete search, the two-stage packing/unpacking mode can be more efficient.

We have now seen several instances of hybrid NLG systems that use surface-oriented n-gram models to assist in selecting the most fluent realization. The work reported by Habash (2004) extends on this approach by also including n-gram counts of words in *dependency relations*. Habash (2004) records so-called *structural n-grams* that model parent-child dependency relations in order to perform lexical choice as well as structural choice for NPs. Structural n-grams can capture longer range dependencies between words and the relations are also more general by being based on uninflected lexemes instead of surface forms as used in standard n-gram models. Based on pairs of parent-child lexemes collected from 127,000 parsed sentences from the English UN corpus, Habash (2004) trains a model totaling more than 500,000 structural bigrams for 40,000 lexemes.

The particular system reported on by Habash (2004) is called EXERGE (Expansive Rich Generation for English) and is used for target language generation in the Spanish–English MT project Matador (Habash, 2003). Habash (2004) describes the EXERGE module as lying somewhere in between HALogen and FERGUS in terms of both input complexity and the balance of statistical vs. symbolic components.

In the first phase of the generation process, after a dependency structure is given as input to the system, a rule-based component performs structural expansion and syntactic assignment to produce a forest of syntactic dependencies. The structural n-gram model is also used for expanding the syntactic structure of noun phrases in this forest. In the next step, a bottom-up dynamic programming al-

gorithm then applies the structural n-gram model for pruning ambiguous nodes in the forest, effectively performing lexical choice. This pruning is done in order to reduce the size of the packed word forest that is created in the following stage of the realization process, using a rule-based generation grammar. Finally the ranking module of HALogen (Langkilde, 2000) is applied for extracting the best string using a standard linear bigram model. To sum up, Habash (2004) uses two levels of n-gram models to help determine the final surface string produced by the generator: In addition to looking at the conditional probabilities of words given the preceding word in a linear sequence, the system also looks at the joint probabilities of parent and child lexemes in a dependency structure.

The use of dependency-based n-gram counts is also found in the machine learning approach to template-based realization proposed by Ratnaparkhi (2000). Instead of just recording parent-child relations as in Habash (2004), Ratnaparkhi (2000) includes *siblings* and *grandparents* in the model. As mentioned in Section 1.1, the use of templates represents a simpler approach to the task of surface realization, where generation amounts to filling in the open slots of otherwise more or less fixed phrases. This simple approach can work well for limited domains with little expected variation in input and output. Even within limited domains however, the need to manually maintain and write the templates may represent a bottleneck to scaling up the system. Ratnaparkhi (2000) therefore proposes a fully data-driven method for generating and instantiating templates. The semantic representations used in the system are simple attribute-value pairs, representing simple propositions in the domain of air travel such as the arrival and departure of flights (e.g. {DESTINATION=Seattle, DEP-DAY=Friday}). Instead of relying on a manually created knowledge-base or grammar, the system learns from a training corpus of 6000 generation templates, i.e. phrases where the values of relevant attributes have been replaced by the attribute names themselves. The goal is then to automatically learn the optimal attribute ordering and lexical choice for a given set of attribute-value pairs (e.g. “A flight to Seattle leaves on Friday”).

In order to learn the most probable sequence of words for a given attribute-value set, Ratnaparkhi (2000) uses a maximum entropy model. The overall framework of maximum entropy modeling was described in Section 2.3, and it here suffices to recall that the model encodes relevant information about the instances in the training data by a set of specified feature functions. Ratnaparkhi (2000) compares two implementations of his approach, using two different sets of feature functions. For each word in the training data, the first version of the system records local information in the form of bigrams and trigrams in the history of the phrase. The features also record information about the attributes remaining to be realized in the given phrase, although the model ignores the actual values of attributes and these are simply filled in once a template has been generated. Given an estimated model and an input set of attribute-value pairs, a heuristically

constrained, breadth-first search tries to construct N complete realizations that express all the attributes once and only once, and then picks the most probable candidate.

In a second implementation of the system, Ratnaparkhi (2000) achieved better accuracy by defining additional features to also include *syntactically related* words in the history. By first having the templates in the training corpus annotated with syntactic tree structures, the features of the second model also record siblings, parent and grandparent relations for each word, resulting in a more accurate model.

Unlike the other NLG modules we have reviewed in this section, the template-based approach described by Ratnaparkhi (2000) is not a hybrid generator, combining statistical and symbolic approaches. Instead it directly learns a mapping from attribute-value pairs to phrases. However, this template-based approach is quite limited in scope, and the overall approach is difficult to scale to more complex domains. Furthermore, although a fully trainable approach eliminates the need for creating hand-coded grammars, lexicons, ontologies and other labor-intensive resources, it does so at the expense of accuracy.

We have so far seen several examples of n-gram-based approaches to the task of statistically guided generation, including approaches that incorporate additional information about syntactic dependencies in their models. A quite different approach is taken in the Amalgam project of Gamon et al. (2002), which uses several specialized classifiers to help determine linguistic choices in the generation process. This surface realizer has been implemented for French, English, and German for different technical domains (e.g. software manuals).

The input to Amalgam is a logical form dependency graph, with fixed lexical choices for content words. This graph is then mapped into a syntax tree through a series of linguistic operations, both manually engineered and machine learned, finally yielding a surface string to be read off the leaf nodes of the tree. The various operations and transformations that are applied to produce this final syntax tree have been formulated as *classification tasks*, with corresponding dedicated *decision tree* classifiers. The classifiers are trained to make choices regarding syntactic labeling, ordering, auxiliaries, negation, morphological case, extraposition of relative clauses, aggregation, punctuation, and more (Corston-Oliver, Gamon, Ringger, & Moore, 2002). Through the successive application of its multiple special-purpose decision tree classifiers, Amalgam performs a greedy search towards the final hypothesized string.

The various classifiers have been trained on automatically parsed data, consisting of 100,000 sentences from technical manuals. To produce the appropriate training data, the syntactic analyses resulting from the parsing have been further processed to produce logical forms. The features used in classifier training are then defined over LF nodes and their corresponding syntactic nodes.

In the German instantiation of Amalgam, the realizer hosts a total of twenty-one decision trees, of varying complexity. Not all of the operations are implemented through the use of decision trees, however. For some types of contexts for which there is not sufficient training data to reliably learn the conditions of how and when to apply them automatically, manually defined operations are used (e.g. certain global movement operations such as raising, Wh-movement, or movement of relative pronouns). Many of the simpler morphological operations are also manually specified. Furthermore, for determining the optimal ordering of modifiers within a constituent, Corston-Oliver et al. (2002) apply a generative language model defined using bigram features over syntactic trees.

What we see in Amalgam is a kind of modularization, where different subtasks in the generation process are handled by different sets of task-specific rules and learners. Other examples of this strategy are found in the work of Shaw and Hatzivassiloglou (1999) and Malouf (2000), who implement several different machine learning strategies for the isolated task of ordering prenominal modifiers. Malouf (2000) tests a selection of methods for learning the order of prenominal adjectives in English, including memory-based learning (MBL) classifiers, bigram models, and a positional probability model.

The remaining examples of hybrid realizers that we will be looking at all bear similarities to PCFGs, as used in parsing. Recall that, a PCFG associates probabilities to the rules X $\rightarrow$ α_i of a context-free grammar (where X is a non-terminal and α is a sequence of terminals and non-terminals), so that $\forall_i \sum_j P(X_i \rightarrow \alpha | X_i) = 1$. The probability of a particular tree is then given by the product of the probabilities of all the rules that have been used for constructing it. The first PCFG-based realizer we turn to is that of Humphreys et al. (2001), which, similarly to Amalgam, was developed at Microsoft Research. One goal of their approach is to directly reuse resources initially developed for doing analysis.

The starting point of Humphreys et al. (2001) is a broad-coverage grammar that they describe as having a context-free backbone extended with an extensive set of fine-grained and possibly context-sensitive conditions on each rule. On the basis of this grammar they train a generative lexicalized PCFG, described as being similar to that of (Collins, 1997). The model is built by parsing a corpus of 25,000 sentences using the (non-probabilistic) grammar, and then simply recording the frequencies of the rule applications without any intervening manual annotation. A simplified generation grammar is then derived from the original analysis grammar, basically extracting its context-free backbone. Since the detailed rule conditions originally associated with the rules of the analysis grammar have been discarded, the generation grammar massively overgenerates. Then, the PCFG that was initially trained for the analysis grammar, is applied *as is* in order to constrain generation.

The scope of the system presented by Humphreys et al. (2001) is quite sim-

ple and is used for restating natural language data-base queries, for the purpose of confirmation and disambiguation. Analysis of the natural language queries provides a representation of argument structure which is used as input to the generator. All lexical choices are specified in the input, including prepositions and determiners. The first stage of generation performs a one-to-one-mapping of the relations in the input representation to syntactic terms. The second stage then assigns a linear ordering to the syntactic nodes using the simplified generation grammar, and the single final path is determined by the scores assigned by the same PCFG as used in analysis. Finally, a third and rule-based stage of the system generates the inflected forms for the leaf nodes of the tree, producing the final string.

Another research effort that bears some resemblance to the PCFG approach of Humphreys et al. (2001), is the weather forecast system described by Belz (2005), a study which forms part of the larger research project COGENT.[3] Using a generate-and-select approach, the efficiency and accuracy of three different statistical models are compared. One of the models uses the same set-up as in (Langkilde, 2002), extracting realizations from a packed forest representation using a bigram model. The other two approaches tested by Belz (2005) are based on a single model trained on treebank data, and then applied with two different search procedures, Viterbi or a greedy search.

As a practical NLG module, the weather forecast system is quite restricted. The vocabulary only covers some 90 words, but this also means that even small amounts of training data (close to 23,000 words are used for training the various models reported on by Belz (2005)) can easily give very good coverage. The generation task itself consists of mapping numerical vectors of wind data, encoding speed, direction and time period, into natural language statements about wind characteristics (Belz, 2005). The underlying generation grammar is given as a set of context-free rules with atomic arguments.

The treebank-based model itself is defined as follows. Each string in the training corpus is annotated with its corresponding set of possible derivations from a context-free grammar. No form of disambiguation is used in this process, counting all possible derivations for a given string as being equally good.[4] The frequency counts are then obtained for the individual CFG rule applications, adding $1/n$ to the count of each occurrence, where n is the number of alternative derivations for a sentence. After smoothing the counts using *add-one* smoothing, a standard

[3] COntrolled GENeration of Text. A research project focusing on wide-coverage generation. COGENT is a joint effort between the University of Sussex and the University of Brighton, but also involves informal collaboration with the LOGON project. See http://www.itri.brighton.ac.uk/projects/cogent for more background.

[4] The training data can thus perhaps not be said to have been *treebanked* in the traditional sense of the word.

maximum likelihood estimate computes probability distributions over alternative decisions.

When the resulting model is applied by doing a greedy search through a generation forest for a given input, the single most likely decision at each point during generation is chosen, resulting in very efficient extraction but without any guaranty of finding the globally most likely realization. When instead applied using the Viterbi algorithm, maximizing the joint probability of all decisions, the most likely generation process is guaranteed to be found, but at considerably higher computational cost. The cost of extracting the most likely realization with respect to the bigram model, however, is a lot more expensive still. Belz (2005) attributes this to the overhead of loading the bigram model to score the packed representations. Moreover, for the weather forecast task described in Belz (2005), the bigram approach turns out to give significantly better results than using the treebank-based model. The treebank model is in many ways similar to a standard PCFG, although trained on non-disambiguated data. Of course, an obvious question is whether better results can be obtained for the treebank model if the derivations in the training data were first disambiguated and if other smoothing methods were applied before computing the MLE. It might also be possible to obtain better results using a *lexicalized* model, especially since the vocabulary is so small. The vocabulary size, taken together with the fact that the data sets exhibit only a small number of different syntactic structures, probably also explains the good accuracy of the bigram model.

The final hybrid NLG system we will mention in this section is the recent PCFG-based generator of Cahill and Genabith (2006). This surface realizer uses a probabilistic grammar that is trained on a treebank automatically adapted to approximate a unification-based generation grammar. More specifically, the annotations added to the treebank data are based on the framework of Lexical Functional Grammar (LFG). As shown by Abney (1997), however, a PCFG-based model can only provide an approximation to constraint-based formalisms such as LFG and HPSG.

The unification-based LFG formalism uses two levels of representation for describing the linguistic information in a sentence: *c-structure* (or constituent structure) represents the external structure of a sentence in the form of a phrase structure tree. On the other hand, *f-structure* (or functional structure) represents the internal structure, describing abstract syntactic functions in the form of an attribute-value matrix. Nodes in the c-structure are linked to features in the f-structure by a mapping function, ϕ.

The realizer presented by Cahill and Genabith (2006) tries to find the most likely tree, given an input f-structure. The training data for the underlying probability model is based on the methods described in Cahill, Burke, O'Donovan, Genabith, and Way (2004) for automatically annotating an existing treebank with

f-structure information. With the purpose of automatically acquiring stochastic LFG approximations, nodes of the trees in the Penn-II treebank are annotated with f-structure equations which are then passed to a constraint solver to produce a full f-structure for the tree. On the basis of this treebank, Cahill et al. (2004) train a PCFG where the rules are annotated with structural f-structure correspondences, such as

$$\text{NP}(\uparrow \text{OBJ} = \downarrow) \rightarrow \text{DT}(\uparrow = \downarrow)\ \text{NN}(\uparrow = \downarrow) \tag{3.1}$$

The probability estimates are based on a simple MLE for the production frequencies in the (f-structure annotated) treebank. As with a standard PCFG, the probability of a tree is decomposed as the product of all the probabilities of the productions that occur in the tree, but in this case the productions are annotated with f-structure information. Using the resulting model together with the constraint solver, unseen text can be parsed into trees with associated f-structures.

The more interesting part for our purposes, is that Cahill and Genabith (2006) train a similar model for the purpose of implementing statistically guided LFG-based generation. However, to make the model more geared towards generation, the probability of a rule is conditioned not only on its LHS as described above, but also on the f-structure features that are ϕ-linked to the LHS of the rule, such as PRED, SUBJ, COMP, TENSE and so on. This model is then applied using a chart-based generation algorithm with Viterbi pruning. Punctuation, finally, is handled in a separate post-processing step.

There is one important issue that has been more or less absent in this section's discussion of different realizers; evaluation. Generally, the increasing focus on more rigid evaluation methodologies within NLP as a whole is often attributed to the increasing use of empirical methods. Given the late adoption of empirical methods within NLG, it is perhaps not so surprising that this sub-field has also been late at picking up on methods for quantitative and automatic evaluation. Furthermore, there are also many factors that contribute to complicating the evaluation task in NLG. For one thing, acceptability and grammaticality are hard to quantify, and there are often several different realizations that are acceptable as output. Ratnaparkhi (2000), for example, refrains from including any automatic measures on these grounds. Humphreys et al. (2001) too only report manual and informal evaluation of their results. With the advent of more empirical methods within NLG, however, many researchers have now adopted the use of string-based metrics as used in MT and speech recognition. The most common metrics are BLEU (Papineni et al., 2002) as used by Langkilde (2002), White (2004), Habash (2004), Belz (2005), Cahill and Genabith (2006), and variants of edit distance as used by Bangalore and Rambow (2000), Corston-Oliver et al. (2002), Langkilde (2002), Belz (2005), Cahill and Genabith (2006). We will return to these and other evaluation measures in Section 6.3 where we give more details. Such

metrics are valuable for regression testing and comparing different versions of a particular system, but when the scores for different systems are computed for different test data they do not provide a basis for directly comparing different systems in a meaningful way. For many other sub-fields within NLP, the community has long since settled on common data sets and evaluation tasks that facilitate the comparative evaluation of different approaches. Such resources are still largely unavailable for NLG. Moreover, since many realizers are tightly tuned to a particular domain, it is difficult to define shared tasks that would make it possible to draw direct comparisons across different systems. Nonetheless, some of the projects we have looked at above have taken some first steps in this direction, by using test data traditionally used when reporting results on statistical parsing. Bangalore and Rambow (2000), Langkilde (2002), Cahill and Genabith (2006) evaluate their systems on sentences taken from section 23 of the WSJ treebank. This requires that the bracketed treebank data first be preprocessed to produce appropriate input structures. Various realizers also often differ with respect to how much information is already specified in the input, and this is of course an issue that further complicates the interpretation of evaluation measures. We will give a more formal treatment of the different evaluation measures in Section 6.3.

Finally, there are some works on statistically guided generation that, more than any of the other systems we have reviewed in this section, are closely related to the approach developed in this thesis, viz. the work of Nakanishi et al. (2005) and Cahill et al. (2007). The main reason why we have not here included any details for these studies, is that they closely parallel much of the what we will outline in the following chapters. Similarly to our approach, Nakanishi et al. (2005) and Cahill et al. (2007) train treebank-based discriminative models for realization ranking, using generators built on constraint-based grammars. While Nakanishi et al. (2005) work with an HPSG-based grammar for English, Cahill et al. (2007) work with an LFG-based grammar for German. Some more details on the experiments and results of these studies were provided in Section 1.4 in the introduction. Now, the approach of both Nakanishi et al. (2005) and Cahill et al. (2007) are based on ideas originally proposed by Velldal et al. (2004), Velldal and Oepen (2005), and Velldal and Oepen (2006). As we shall see, this approach to realization ranking is motivated by recognizing the similarity toward the task of *parse selection*. Simplifying slightly, the task of ranking analyses in parsing can be seen as the *inverse* of the task of ranking realizations in generation. In both parsing and generation, we wish to determine some most likely structure given the constraints of the grammar, the difference being in the direction of processing: In the former case we start from a surface string, while in the latter we start from a semantic representation. Taking this similarity as our point of departure, we next turn to review some of the previous work on the problem of parse selection.

3.3 Statistical Parse Selection

Reiter and Dale (2000) note that one possible conceptualization of the generation task, is to view it as a series of choices. This conceptualization seems fitting for many of the approaches we reviewed in the preceding section. Reiter and Dale (2000) further contrast this with parsing, which they suggest might best be understood as a task of managing competing hypothesis. However, there are also many obvious similarities between the tasks of parsing and generation, and both conceptualizations seem to be able to accommodate both tasks, depending on perspective. The last few systems we described in Section 3.2 provide interesting parallels to the approach that is developed in this thesis, in the sense that they rely on techniques previously applied for the purpose of analysis. For instance, in the project described by Humphreys et al. (2001), there is an explicit emphasis on directly reusing resources developed for parsing. The starting point of this thesis shares a similar emphasis, although at a more abstract level. Instead of reusing the actual resources themselves, we seek to reuse an overall methodology and set of techniques, and then adapt the resources to the specific setting of generation. Our perspective on the task of realization ranking is given by recognizing its similarity to the task of *parse selection*. The former task is the problem of selecting a preferred string given an input analysis, while the latter is the problem of selecting a preferred analysis given an input string. This relation of "inverse similarity" between the two tasks forms the starting point for our approach, and this section is therefore devoted to discussion of some previous work on statistical disambiguation in parsing. This will thereby also provide some more background for the methods that we will develop in later chapters.

There are two main motivations for introducing a stochastic element in parsing: *robustness* and dealing with *ambiguity*. In many real-world applications, the input to a natural language analyzer can be expected to be noisy and contain errors. Moreover, even for a broad-coverage grammar, there will always be cases that the grammar fails to cover or that are simply deemed out-of-scope. A strict grammaticality requirement will leave the parser brittle in the face of such input. Furthermore, even ungrammatical strings are often comprehensible, and certain statistical parsers can enable us to assign an interpretation even to ungrammatical input. In generation, however, the issue of robustness looks a bit different. For one thing, we do not want the system to produce ungrammatical output. Furthermore, although we want the realizer to be able to cope with a wide range of semantic inputs, it seems reasonable to expect that these input structures are well-formed. A stronger parallelism exists with respect to the second main motivation for statistical grammars in parsing, that of constraining ambiguity. This is especially evident in cases where the same reversible grammar is used for both parsing and generation. The flip-side of grammatical *coverage* is *ambiguity*, and a statistically

guided parser enables us to *disambiguate* by ranking the competing structures and selecting a single preferred analysis. Although we want to allow for a wide range of grammatical phenomena to obtain good coverage when parsing, we want to restrict ourselves as to not produce unnatural-sounding output when generating. Although the underlying grammar ensures the semantic *fidelity* of a generated realization with respect to the input, it does not automatically ensure its *fluency*, and there will typically exist a multitude of different ways in which the grammar can express the same meaning. Just as on the parsing side, a statistically guided generator is needed so that we can rank the competing structures and select a single preferred output.

3.3.1 Stochastic Unification Based Grammars

Research on statistical grammars and parse selection has a long history and has produced a tremendous body of work. The literature on this topic is far too comprehensive to be covered in this section, and we will here only be touching on some of the more recent efforts that are more directly linked to the approach we will be pursuing for realization ranking. The most important line of work in this respect is that related to training stochastic versions of constraint-based grammars such as LFG and HPSG, or what Johnson et al. (1999) call *stochastic unification-based grammars* (SUBGs).

Historically, most of the work on statistical parsing has focused on probabilistic context-free grammars (PCFGs), as we also briefly touched upon in relation to realization ranking above. PCFGs are easily estimated from a bracketed corpus. The corpus can be annotated according to an existing grammar, or the annotations can themselves be taken to implicitly define a grammar. The maximum likelihood estimates for the productions can be computed as simple relative frequencies of their occurrences in the corpus. However, the strong independence assumptions of these models are known to render them linguistically inadequate for many purposes. Recall that the probability of a subtree in a PCFG does not depend on words not dominated by the subtree or on nodes in the derivation outside the subtree. Although the nodes of a tree can be enriched to include information about lexical heads and ancestor relations, this quickly inflates the parameter space and increases the problems related to data sparseness.

The more powerful family of unification-based grammars (UBGs) such as HPSG and LFG, can encode much richer syntactic and semantic constraints. However, as shown by Abney (1997), the method used for defining weights for a context-free grammar does not transfer to the case of unification-based grammars. The non-local dependencies created by the unification constraints, break the simple tree structure otherwise induced by the productions of a PCFG (Johnson et al., 1999). This furthermore has the consequence that simply computing relative frequencies

does not generally result in a maximum likelihood estimate in the case of SUBGs. In order to account for the context-sensitive dependencies of unification-based grammars, Abney (1997) shows how a maximum likelihood distribution can be computed using *log-linear models* (as described in Section 2.3), with parameters estimated using a Monte Carlo-based gradient ascent procedure. The estimation procedure finds the optimal parameters λ that maximize the log-likelihood of the training corpus according to the model.

The overall framework introduced by Abney (1997) was further refined by Johnson et al. (1999) to make parameter estimation efficient enough to be practically feasible even for grammars of more realistic size. As further described in Section 2.3, a log-linear model represents a given object as a set of feature functions f with corresponding model weights λ. The crux of finding the maximum likelihood solution for a log-linear model is computing the expected values of f under the distribution specified by λ. In the initial model proposed by Abney (1997), finding the MLE for a SUBG would involve summing over the set of all possible grammatical or well-formed analyses. Abney (1997) tried to solve this by using a Monte-Carlo method where the expected values are computed from generated random samples. As an alternative, Johnson et al. (1999) suggested using a so-called *pseudo-likelihood* estimator that finds parameters that maximize the *conditional* probabilities of the annotated parses given the strings in the training data. The parametric form of a conditional log-linear model can be seen in Equations (2.22) and (2.23) of Section 2.3, and the function for computing the conditional or pseudo-likelihood of the training data is given in Equation (2.30). This estimation method only requires summing over the possible syntactic analyses of the strings in the training data. While originally proposed by Besag (1975), this form of estimation procedure has also been used by Berger et al. (1996) in the context of maximum entropy models for machine translation, and by Jelinek (1998) in the contexts of speech recognition. More details on the estimation procedures and likelihood function can be found in Section 2.3.

As in the case of PCFGs, the training data for a SUBG consists of a set of strings paired with their preferred parses. However, the discriminative training of conditional log-linear SUBGs as described in Johnson et al. (1999) also takes advantage of *negative examples* in the sense that the weights are chosen to maximize the probability of the preferred parses relative to the non-preferred ones for each string in the training corpus.

It might be worth noting at this point that a PCFG can itself be represented as a log-linear model. If we let the features record the frequencies of the productions used in a given tree, and let the corresponding weights be the log-probabilities of these productions, we get a log-linear model where the normalization term sums to 1. The maximum likelihood estimate for the parameters of this model is obtained by computing the production probabilities as relative frequencies from the

training corpus. In the case of SUBGs, however, even if we define feature functions to be indexed on productions as in a PCFG, the relative frequency-approach will generally not yield the maximum likelihood estimates we want due to the non-local dependencies of the productions. With the estimation procedures used by Abney (1997) and Johnson et al. (1999), the features need not be restricted to productions, but can instead encode arbitrary and non-local properties of the linguistic structure in question. As commented by Johnson et al. (1999), *specifying the features of a SUBG is as much an empirical matter as specifying the grammar itself*. We look at more examples of feature functions below.

The maximum likelihood estimate for a PCFG gives us a *generative model* that predicts a full joint distribution of strings and trees, $p(\text{string}, \text{tree})$. In other words, it learns both the distribution of strings $p(\text{string})$ as well as the distribution of trees given strings, $p(\text{tree}|\text{string})$. Johnson et al. (1999) note that the conditional distribution intuitively seems important for the disambiguation task while the marginal distribution does not. Better results might therefore be achieved by directly optimizing the conditional likelihood, as described above. This means learning a conditional distribution $p(\text{tree}|\text{string})$ but not a full joint distribution. As we shall see below, the experimental results of Toutanova et al. (2005) on the Redwoods treebank show that discriminative models indeed outperform generative models for the parse selection task.

3.3.2 Parse Disambiguation on the Redwoods Treebanks

Several variants of the approach developed by Abney (1997) and Johnson et al. (1999) of using log-linear models for SUBGs have later been described by other authors working on parse disambiguation (Osborne, 2000; Toutanova, Manning, Shieber, Flickinger, & Oepen, 2002; Toutanova et al., 2005; Malouf, 2002; Riezler et al., 2002; Miyao & Tsujii, 2002; Malouf & van Noord, 2004). Particularly relevant for the experiments of this thesis, however, is the work of Toutanova et al. (2005) on training conditional log-linear models for parse selection on the LinGO Redwoods HPSG treebank (Oepen et al., 2002). The treebanks used for developing and testing the LKB generator in the LOGON project also follow the Redwoods approach to treebanking, and the details of these resources and methods are further described in Section 5.2. Here it suffices to note that each string in the treebank is annotated with an HPSG analysis licensed by the LinGO *English Resource Grammar* (ERG; Flickinger, 2002), as further described in Section 5.2. The HPSG signs are typed feature structures that encode fine-grained syntactic information, and also include a logical-form meaning representation based on *Minimal Recursion Semantics* (MRS; Copestake et al., 1995, 2006) which we present in more detail in Section 4.2.

As further described in Section 5.2, the fundamental data structure of the Red-

woods resources is the *derivation tree*. The internal nodes of a derivation tree correspond to rule schemas of the underlying grammar, such as the head-complement or head-adjunct schemas, while the preterminals of the trees are fine-grained lexical identifiers (which are usually mapped to the more abstract notion of lexical types when defining the features of the models). The derivation trees recorded in Redwoods are significantly different from the phrase structure trees found in many traditional resources such as the Penn treebank, and encode much more fine-grained information. Note also that the full HPSG-sign can be reconstructed from the derivation tree by reference to the underlying grammar. Toutanova et al. (2005) train and test several disambiguation models with a range of features defined over derivation trees, including both generative and discriminative models. We will briefly describe the different models and features below.

The first generative model is a PCFG with production features defined for the rule schemas in the derivation trees of the HPSG analyses in the treebank. The model parameters are specified to maximize the likelihood of the set of preferred derivation trees of sentences in the training treebank. In a second model, the set of basic production features is also extended to include ancestor information for up to a maximum of four dominating nodes. Capturing more context in this way can be especially helpful for a grammar such as the ERG which produces trees that are fully binarized.

For comparison, Toutanova et al. (2005) also train similar PCFG models using more conventional phrase structure trees derived from the derivation trees. However, the disambiguation accuracy is higher for the models trained on Redwoods' native derivation tree representations.

As mentioned before, the HPSG-signs recorded in Redwoods also include a deep semantic representation based on the MRS formalism. Another PCFG-style model is trained on semantic dependency trees that are extracted from these MRS representations. Toutanova et al. (2005) also implement a conventional trigram HMM tagger. This defines a joint probability distribution over preterminal tag sequences and yields of the derivation trees, smoothed using linear interpolation of lower order models.

Finally, several of the generative models are combined in a single model by linear interpolation. The component models include: The PCFG using ancestor information trained on derivation trees; the PCFG trained on semantic dependency trees; and finally a simplified version of the HMM tagger that only uses the trigram tag probabilities (i.e. the scores can be seen to correspond to the transition probabilities of the original tagger). In the resulting combined model, the score for a given parse is then computed as a linear interpolation of the log-probabilities of the individual models.

Now, for all the generative models listed above, Toutanova et al. (2005) go on to train corresponding discriminative log-linear models. The individual models

are defined using exactly the same feature sets, but are trained to maximize the conditional log-likelihood of the treebank data. Toutanova et al. (2005) find that, for all the different model configurations, the discriminative variants substantially outperform their generative counterparts, resulting in relative reductions in error rate of up to 28%. For the combined model, the discriminative variant (combining all features in a single model) resulted in a 13% error reduction.

There are several important points resulting from the study of Toutanova et al. (2005). When comparing the different PCFG-style models, using production features defined over derivation trees gave better results than with standard phrase structure trees. Extending the feature set to include ancestor information also gave a boost in accuracy. On the other hand, Toutanova et al. (2005) found that the information in the semantic dependency trees did not seem to contribute significantly to disambiguation. Finally, for all the different model configurations, the generative formulations proved to be outperformed by the discriminative variants. For the individual component models as well as for the final combined models, the accuracies of the discriminative models were higher.

In this thesis we will develop discriminative log-linear models for realization ranking that in many ways parallel the discriminative models for parse disambiguation described by Toutanova et al. (2002) and Toutanova et al. (2005). The features recording local sub-trees in the derivation trees, as well as the ancestor information and the tag trigrams, form the starting point of the feature set we use for training the discriminative realization rankers. These features, as well as some further extensions of them, are defined in more detail in Section 5.5. As we shall see, however, in order to make treebanks useful for training the kind of model we need, some adaptions of the existing resources are necessary. The discriminative model of Toutanova et al. (2005) gives the conditional probability of an analysis given a string. The required training data for such a model consists of all possible analyses for a set of strings, as well as an indication of what parses are considered the correct or preferred ones. In our case we want to model the distribution that goes in the opposite direction. What we are interested in is a model of the conditional probability of a string given a semantic analysis. In order to train such a model we need a treebank resource that encodes preference relations of strings relative to the semantics, and not the other way around. This is the kind of resource we refer to as a *generation treebank* (Velldal et al., 2004). In Chapter 5 we describe how a generation treebank can be automatically constructed on the basis of an existing Redwoods (parse-oriented) treebank.

3.4 Reranking for Statistical Machine Translation

Before we close off this chapter on previous and related work, a few comments should be offered on the work on *statistical machine translation* (SMT). Although SMT systems often do not include a separate component for generation, the task obviously demands that target language is generated in some sense. These systems will need to face many of the same issues as dedicated NLG systems, such as ensuring the fluency of the target string. The translations are usually produced by direct mappings from the source to the target, without going by way of any intermediate semantic representations.

One line of research within SMT which has many parallels to the task that we are trying to pursue here, is the work on *discriminative reranking*. In a reranking set-up, some underlying SMT system first delivers an n-best list of candidates, and then an additional discriminative model is applied to rank this list before the top-ranked translation is finally selected for output. This is also a common set-up in parse selection, in which case a discriminative model is estimated for n-best lists produced by a baseline statistical parser.

Let f be a source sentence in a foreign language and e be a translation in the target language. In the traditional noisy-channel perspective on MT, as used in the seminal IBM models (Brown et al., 1990), the problem of finding the candidate that maximizes the probability $p(e|f)$ is decomposed into two generative models; a conditional translation model $p(f|e)$ and a target language model $p(e)$.

$$\hat{e} = \arg\max_{e} p(e|f) = \arg\max_{e} p(f|e)p(e) \tag{3.2}$$

As an alternative to the noisy-channel approach, Och and Ney (2002) show how discriminative maximum entropy models can be used for defining translation models where the posterior probability $p_\lambda(e|f)$ is modeled directly. Various such log-linear models are used for extending a baseline SMT system by adding new feature functions. One advantage of this arrangement is that the reranking model can take advantage of global features that might be unavailable to the baseline system. Another advantage is that one can define features that directly targets specific weak spots of the baseline system.

Och and Ney (2002) and Och et al. (2004) give many creative examples of feature functions, including complex features such as various alignment models, in addition to the baseline translation model itself. By letting the corresponding λ's act as model scaling factors, such a log-linear translation model is even shown to subsume the traditional noisy channel model as a special case. Och et al. (2004) also report on a range of experiments employing annotated training data such as treebanks and using features based on shallow and deep syntactic parsers. Unfortunately, the outcome of these experiments which try to incorporate more

linguistic information in the models are not particularly positive (although this is explained as possibly being due to small data sets, interactions with other components such as statistical parsers, etc.).

Although Och and Ney (2002) work in the setting of reranking n-best lists of a baseline statistical MT-system, many of the same issues carry over to our setting of ranking the output of a symbolic NLG system. However, in our own experiments, as we shall see in Chapters 7 and 8, training on treebanks and using syntactic features seem to benefit the ranking performance to a much larger degree. Moreover, it is worth noting that the approach of Och and Ney (2002) will be more directly relevant when we in Chapter 9 turn to describe our preliminary experiments with training a discriminative model for end-to-end reranking within LOGON. At that point we take a brief departure from the isolated sub-task of ranking generator output, and instead focus on the task of reranking the target translations of the overall hybrid MT system, based on the presentation of Oepen et al. (2007).

This chapter has presented many examples of previous work that bear similarities to the task of statistical realization ranking which is the focus of this thesis. We have seen examples of work on statistical generation, parse disambiguation, and discriminative reranking for SMT. Of all of these, the work that is most important for the realization ranking approach that we will be developing over the following chapters, is the work by Toutanova et al. (2005) on training discriminative log-linear models for parse selection on the Redwoods treebank. We described this approach in Section 3.3.2 above, and we will also return to it in later chapters. In the next chapter, however, we turn to present the Norwegian–English MT system that our hybrid generator system forms part of; LOGON.

Chapter 4

The LOGON System

The work on realization ranking presented in this thesis is set within the larger context of the Norwegian–English machine translation project LOGON[1] (Lønning et al., 2004; Oepen et al., 2004). The goal of the LOGON project is to develop a prototype system that performs high-quality automatic machine translation for text in the domain of tourism, and back-country hiking in particular. The system architecture is centered around semantic transfer and implements a "deep", knowledge-rich approach that focuses on linguistic precision. The main pillars of the system are founded on *rule-based* or *symbolic* approaches. This point is something which sets the LOGON project apart from much of the other ongoing research efforts within MT today, which generally seem to have a bias towards statistical and empirical methods. In contrast, the overall approach in LOGON is founded on linguistically rich methods and resources. The general aim is for high precision, rather than for very broad coverage and robustness. Although the core of the LOGON system consists of symbolic or rule-based methods, these are complemented by stochastic methods for the task of managing ambiguity[2] and indeterminacy, as described in Section 4.4 below.

The LOGON project is a collaborative effort with participants from all of the three largest Norwegian universities (Oslo, Bergen, and NTNU in Trondheim), and with funding from the Norwegian Research Council's program for language technology (KUNSTI[3]). The project is working closely with researchers from other

[1] The LOGON project web-pages can be found at http://www.emmtee.net/.

[2] The term *ambiguity* is most commonly used in relation to *sense distinctions*, for example when referring to the existence of different interpretations of a word or different structural resolutions. In this text we will sometimes use the term ambiguity in a somewhat broader sense, referring to the availability of alternative branches in an non-deterministic process in general. Where and when we use the term in this broader sense should always be clear from the context.

[3] *KunnskapsUtvikling for Norsk SpråkTeknologI*, a program set up for the period 2001–2006 with the goal of promoting knowledge development for Norwegian language technology. For more information, see http://program.forskningsradet.no/kunsti.

sites such as the Universities at Stanford, Saarbrücken, Cambridge and Sussex, as well as companies such as NTT[4] and PARC,[5] and has also hosted several international guest researchers. Much of this exchange has taken place by way of the DELPH-IN[6] network, a loose organization of researchers from various sites around the world collaborating on issues related to deep linguistic processing, combining linguistic and statistical methods. Note also that, even though the main language pair of the LOGON prototype is Norwegian and English, rudimentary, bidirectional versions of the system have also been developed for German–English and Japanese–English, in collaboration with other members of the DELPH-IN network.

This chapter gives a brief run-through of all the main components of the LOGON system. We will, however, be putting emphasis on the particular components that are more directly relevant for the purposes of this thesis, which is the generator, the target language grammar, and the semantic formalism.

4.1 System Overview

The LOGON system builds on a relatively conventional semantic transfer architecture, and can be broken down into three main parts: (i) First, a syntactic and semantic analysis of a Norwegian source sentence produces language-specific logical-form semantic representations. (ii) A transfer step then maps these representations into language-specific English representations. (iii) Finally, the transferred semantic representations are passed to the generator to produce English sentences. The logical-form semantic representations are couched in the framework of *Minimal Recursion Semantics* (MRS; Copestake et al., 1995, 2006). An MRS is a flat, event-based representation of semantics, where the meaning of a structure essentially is given as a multi-set (i.e. a bag) of labeled relations, together with an additional set of constraints on scope relations. We provide more details of the MRS representations in Section 4.2 below. A central aim in LOGON has been to treat the semantic transfer representations independently from any particular grammar formalism. This has allowed for a novel aspect of the system, in that analysis and generation are based on different grammatical frameworks. While analysis of the Norwegian source is based on the framework of *Lexical Functional Grammar* (LFG; Kaplan & Bresnan, 1982), generation in the target language is based on *Head-Driven Phrase Structure Grammar* (HPSG; Pollard & Sag, 1994). The two

[4] Nippon Telegraph and Telephone Corporation. For more information on the Natural Language Research Group at NTT, see http://www.kecl.ntt.co.jp/icl/mtg/.

[5] Xerox Palo Alto Research Center. For more information on NLP research at PARC, see http://www.parc.xerox.com/research/projects/natural_language/.

[6] Deep Linguistic Processing With HPSG. See http://www.delph-in.net for more background.

components support a uniform representation on the semantic level, however, as provided by the MRS formalism.

Both the first and last stages in the translation pipeline, i.e. source analysis and target generation, draw heavily on pre-existing tools and resources. On the Norwegian side, the LFG-based NorGram (Dyvik, 1999) grammar is used, and processing is carried out using the PARC Xerox Linguistic Environment (XLE; Maxwell & Kaplan, 1993). To our best knowledge, NorGram is the most comprehensive computational grammar for Norwegian, and currently its lexicon includes more than 80,000 lexemes. For integration into the transfer-based architecture of LOGON, NorGram has been extended with an additional module that derives MRS representations, in addition to the two other levels of representation standardly found within LFG; constituent structure (c-structure) and functional structure (f-structure). While c-structure represents the hierarchical composition of a sentence in the form of a phrase structure tree, f-structure provides a representation of grammatical relations and syntactic properties. In a process known as *co-description*, the different components of the c-structure and f-structure representations in LFG are defined to correspond through a set of linking equations. In NorGram, an MRS representation is projected off the f-structure in a similar fashion.

On the English side, in turn, LOGON relies on the *English Resource Grammar*[7] (ERG; Flickinger, 2002) of the LinGO[8] project. In contrast to NorGram, the ERG is couched in the framework for which the MRS formalism was originally developed; HPSG. The ERG is a general-purpose and wide-coverage lexicalist grammar. After making some additions and adjustments, mostly to accommodate domain-dependent vocabulary, the grammar provides accurate analyses for most of the reference translations in the LOGON development corpora. The grammar is developed and applied using the *Linguistic Knowledge Builder* (LKB; Copestake, 2002), an open-source[9] grammar engineering system. Also part of the LKB system is the tactical generator module that LOGON uses for producing English sentences in accordance with the ERG. This is a chart-based, lexically-driven surface realizer that can generate sentences from underspecified and flat semantics such as MRS. The LKB generator is discussed in more detail in Section 4.3 below, but in the next section we turn to the component that provides the link between source language analysis and target language production; semantic transfer of MRS meaning

[7] An online demo of the ERG is available at http://www.delph-in.net/erg/, in addition to release notes and download instructions.

[8] Linguistic Grammars Online is a research project at the Center for the Study of Language and Information (CSLI), Stanford University, devoted to developing linguistically precise HPSG grammars and tools for grammar-engineering. The LinGO lab is also a founding member of the DELPH-IN initiative. See http://lingo.stanford.edu/ for more information.

[9] Both the LKB and ERG, as well as large parts of the LOGON MT system itself, are part of the open-source DELPH-IN repository; see http://www.delph-in.net/ for background.

representations.

4.1.1 Semantic Transfer

The LinGO ERG has previously been applied also in the Verb*mobil* German–English speech-to-speech MT system (Wahlster, 1997). Similarly to LOGON, the Verb*mobil* project used a semantic transfer approach, while its domain was appointment scheduling and travel planning. Several of the ideas on how to best implement the transfer process have been carried over from the Verb*mobil* project to LOGON. Specifically, transfer is treated as a resource-sensitive rewrite process. Governed by a set of *MRS transfer rules* (MTRs), fragments of an MRS for the source language are successively replaced to produce an MRS for the target language. There has also been research within LOGON on how to further improve the transfer mechanism. Two important outcomes of this are the organization of transfer rules in a hierarchy of types, as well as a chart-based treatment of transfer ambiguity. Moreover, although one of the main intended uses of MRS was exactly as a transfer formalism in machine translation (Copestake et al., 1995), LOGON appears to be the first system to actually implement this. As said, Section 4.2 provides a brief presentation of the MRS formalism itself.

The task of manually defining lexical transfer rules is one of the major bottlenecks in the development of a transfer-based system such as LOGON. To alleviate this labor-intensive process, the LOGON project has also featured research on the automatic acquisition of transfer rules from machine-readable bilingual dictionaries, using hand-crafted transfer rules as templates (Nygaard, Lønning, Nordgård, & Oepen, 2006). As for the current version of the system, the rule base of the transfer module comprises a total of 14,500 MTRs. Roughly 10,000 of these rules have been auto-generated for nouns and adjectives found in Kunnskapsforlaget's Norwegian–English dictionary (Henriksen, Haslerud, & Eek, 2001).

All the pieces of information exchanged between the various components of the LOGON system take the form of MRSs. This aspect of the system can also be seen in the chart diagram illustrating the system architecture in Figure 4.1 (Oepen et al., 2004). The communication between the transfer component and the two grammars (for Norwegian analysis and English generation) is constrained in terms of a *Semantic Interface* specification (SEM-I; Flickinger, Lønning, Dyvik, Oepen, & Bond, 2005). This means that the transfer step can be carried out, by and large, without knowledge of the grammar internals. The SEM-Is for the two grammars also exhaustively enumerate the sets of valid semantic predicates that define the transfer vocabulary. For each predicate in the vocabulary it furthermore lists its set of appropriate roles and corresponding value constraints.

4.1.2 Data-Driven Development and Testing

Although the LOGON system itself is not a data-driven or empiricist system, the development cycles within the project are. The project places a strong emphasis on empirically oriented testing and evaluation of system performance against corpus data. To this end, the project maintains several (freely available) treebanks based on in-domain corpora, constructed following the LinGO Redwoods approach to treebanking described by Oepen et al. (2002). The central organizing facility for evaluation and testing within the project is the open-source [incr tsdb()][10] profiling environment (Oepen et al., 1997; Oepen & Flickinger, 1998). [incr tsdb()] is closely integrated with the LKB system and the XLE, and it provides several specialized tools for regression testing and performance profiling of constraint-based grammars. It also defines a uniform format for test and reference data, as well as performance data itself. A central philosophy embodied in [incr tsdb()] is that the profiling methodology otherwise found in relation to software engineering, also can be beneficial to constraint-based NLP systems (Oepen & Flickinger, 1998; Oepen & Callmeier, 2004).

LOGON does not depend on [incr tsdb()] only for profiling purposes. Another important feature of [incr tsdb()] is its support for using the Parallel Virtual Machine protocol (PVM; Geist et al., 1995). For communication between processes, PVM implements a message-passing model that facilitates parallelized and distributed computing across multiple networked machines. The LOGON MT architecture itself is essentially built as an extension of [incr tsdb()], with a central controller using its PVM-based API for communication between the various components in the system, as seen in Figure 4.1.

4.2 The MRS Formalism

> *Our aim is to develop a minimally structured but descriptively adequate representation, which allows for various types of underspecification and facilitates generation and the specification of semantic transfer equivalences.* Copestake et al. (1995)

Rather than being a semantic theory in itself, Minimal Recursion Semantics, as introduced by Copestake et al. (1995), is a representation language for describing semantic structures. MRS takes a so-called neo-Davidsonian or event-based approach to semantics, with explicit variables for events in a relatively flat representation. The meaning of a given structure is compositionally expressed by a bag

[10] Pronounced *tee ess dee bee plus plus*. The homepage of the system is located at http://www.delph-in.net/itsdb/.

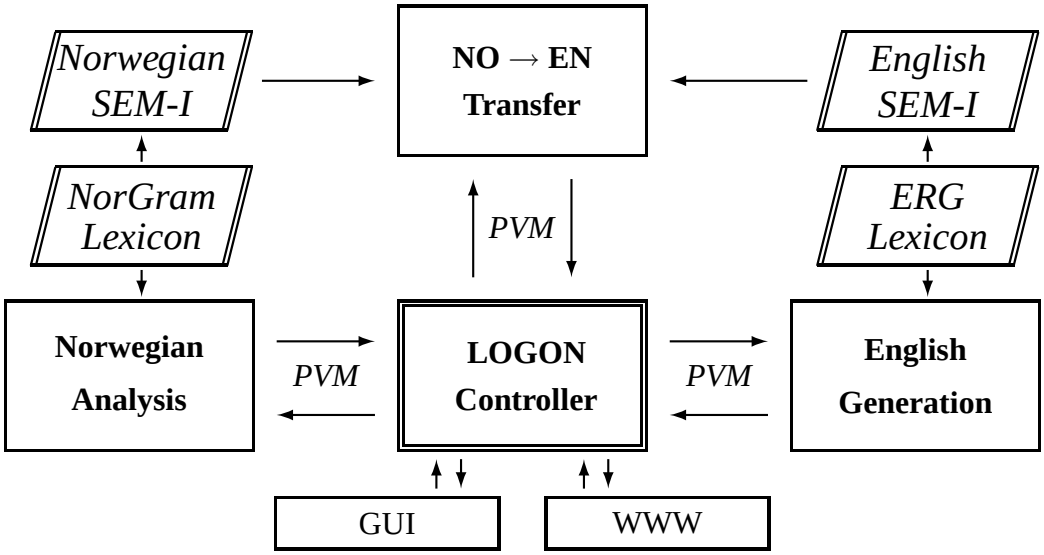


Figure 4.1: Schematic LOGON system architecture: the three core processing components are managed by a central controller that passes intermediate results in the form of MRSs through the translation pipeline. The Parallel Virtual Machine (PVM) layer provides the protocol used for inter-process communication, facilitating distribution, parallelization, failure detection, and roll-over.

of so-called *elementary predications* (EPs). An EP corresponds to a single relation and its associated arguments, and an important feature is that these relations are never embedded within one another, hence recursion in MRS is indeed minimal.

When doing translation, it is often desirable to leave some semantic distinctions and ambiguities underspecified. Ambiguities can be computationally expensive (or even impossible) to resolve, and we do not want to arbitrarily commit to a specific reading. For related language pairs, such as Norwegian and English, it is also often the case that lexical or structural ambiguities parallel each-other in the two languages so that the best solution might be to leave such ambiguities intact in the translation, yielding a notion of ambiguity preserving MT (Copestake et al., 1995).

An important property of MRS is exactly the support for underspecification, and especially with respect to scope relations. The MRS representations contain a special type of meta-variables called *handles* or *labels*, that allow us to refer to particular predications and their relative position in the scope hierarchy. These variables make it possible to maintain a flat representation while still representing scope relations. Restrictions on scope are specified by a set of handle constraints

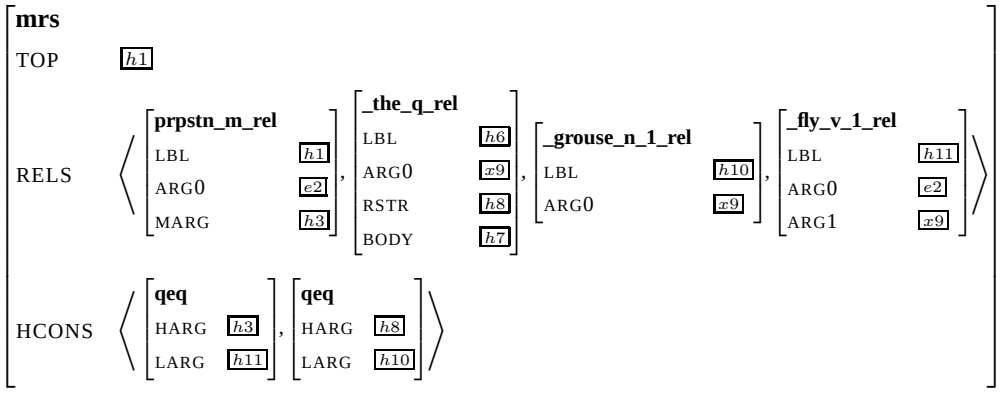


Figure 4.2: MRS feature structure for the sentence "*The grouse flew*."

that define dominance relations over the flat set of labeled relations (i.e. the bag of EPs) in the representation. Several relations can have the same label, which puts them at the same position in the scope hierarchy (and corresponds to logical conjunction). An important point is that the handle variables can be underspecified, which means that multiple scopes can be represented by a single structure. With respect to the support for underspecification, MRS is closely related to a broader family of flat semantics, including other approaches such as Quasi-Logical Form (QLF; Alshawi & Crouch, 1992), Underspecified Discourse Representation Theory (UDRT; Reyle, 1993) and Hole Semantics (Bos, 1995). In MRS, underspecification of scope relations is accomplished using underspecified variables and handle constraints, while underspecification with respect to lexical choice can be done using generalizations over predicate classes.

Similarly to other components of HPSG, MRS can be represented in terms of feature structures. In Figure 4.2 we see an example of an MRS for the sentence "*The grouse flew*", expressed as a feature structure of type **mrs**. MRSs are also often expressed using the somewhat more readable notation seen in Figure 4.3, as a tuple of three elements: a top handle, the list of EPs, and the handle constraints. We will go through the specific elements of either representation in more detail below.

As seen in Figure 4.2, the **mrs** type feature structure contains three attributes: TOP,[11] RELS and HCONS. The RELS attribute holds the bag of EPs, and the value of TOP is the handle of the highest scoping EP. Each EP in turn has a handle and a number of semantic arguments. The HCONS attribute lists the constraints on

[11]The MRS shown in Figure 4.2 uses a slightly simplified inventory of attributes compared to that of Copestake et al. (2006). TOP here corresponds to HOOK|LTOP.

$\langle h_1$,
{h_1:prpstn_m(h_3), h_6:_the_q(x_9,h_8,h_7),
h_{10}:_grouse_n_1(x_9), h_{11}:_fly_v_1(e_2,x_9)},
{$h_3 =_q h_{11}$, $h_8 =_q h_{10}$}$\rangle$

Figure 4.3: MRS tuple for the sentence *The grouse flew*.

possible scopes of the EPs in RELS. These constraints are called *qeq constraints*, sometimes written as $=_q$, which stands for equality modulo quantifiers (Copestake et al., 2006). If a handle argument h and a label l are constrained to be $h =_q l$, then the handle argument must be filled either directly by the given label (i.e. $h = l$) or by the label of some other quantifier which in turn has its body argument filled by l. In this way the $=_q$ relation either encodes equality or a specific relation of outscoping. In the feature structure notation these constraints are represented by the type **qeq**, where the attributes HARG and LARG correspond to the handle (or *hole*) and the label respectively. Note also that we have used prefixes to indicate the typing of the variables in both modes of representation, such as e for event variables, x for instances, and h for handles. MRSs typically also use thematically neutral role labels (ARG$_0, \ldots,$ARG$_n$) and leave the interpretation of the role labels for each predicate to a downstream process. In the triple notation of Figure 4.3, this is reflected in the canonical ordering of the argument positions.

As seen in Figure 4.2, the MRS also includes an addition to the feature structure inventory standardly found in HPSG. The values of the RELS and HCONS features are taken to be *unordered lists* of EPs and handle-constraints respectively. In truth conditional terms, MRS implicitly assumes conjunction between relations that share the same handle variable. In a fully scope-resolved MRS, the handles will be constants, and this expression can be thought of as a syntactic variant of a predicate calculus expression (Copestake et al., 2006). Given the possibility of underspecification, a given MRS can correspond to several such logical forms. In the next section we turn to the issue of generating English strings on the basis of an underspecified MRS logical form.

4.3 The LKB Generator and the ERG

In this section we briefly highlight the main properties of the realizer of the LKB system, as presented by Carroll et al. (1999) and Carroll and Oepen (2005). As said, this is the generator that produces target sentences in the final stage of the LOGON translation pipeline, and it is also the generator that we will be using for

the experiments on realization ranking in this thesis.

The LKB realizer follows a *lexically-driven* approach to generation, which is suitable for lexicalist grammars such as HPSG, where most of the information is encoded directly in lexical entries or lexical rules, as opposed to phrasal grammar rules. The semantic input to generators following this kind of approach is typically given as a bag of lexical predicates with associated variables that capture the semantic relations between the items. For the particular instantiation of the LKB realizer used in the LOGON system, this type of flat semantics is provided in the form of MRS as described in Section 4.2 above. Tactical surface generation from an MRS logical form then proceeds in accordance with the lexicalist ERG grammar. Note that, although the generator mainly follows a lexically-driven approach, grammar rules may also contribute to the semantics.

Generation in LKB is implemented using a bottom-up, chart-based algorithm (Shieber, 1988; Kay, 1996), as familiar from chart-based parsing. In parsing, the edges of the chart cover orthographic units of the given input string. In generation, on the other hand, the edges cover relations of the given semantics, thus giving up the notion of adjacency found in parsing with contiguous constituents only. The entire generation process includes three main steps; (i) lexical look-up and population of the chart, (ii) chart generation using a forest structure for packing local indeterminacies, and (iii) unpacking of the generation forest. We will quickly go through the individual steps one by one.

Prior to generation, all rules and lexical entries are indexed by the relation they contain (as well as any available abstractions over this relation). Then, in the first step of generation, the system retrieves lexical entries that are indexed on the relations of the input semantics. Variables in the lexical entries are instantiated to correspond one-to-one with the variables in the given MRS (through a process dubbed *Skolemization* by Carroll & Oepen, 2005). Next, lexical and morphological rules are applied, possibly instantiating further relations of the input semantics. The empty chart is then populated with edges that correspond to the instantiated entries and rules, each pointing to the semantic relations of the input that they cover.

There is not always a straightforward one-to-one mapping between relations in the input semantics and the instantiated lexical entries or lexical rules in the initial chart. We will not delve into the finer details of this issue, but just briefly mention the four main reasons why this complication might occur. (i) Due to underspecification or ambiguity of predicates in the input, a single relation in the semantics might instantiate several distinct lexical entries (analogous to lexical ambiguity during parsing). (ii) A single lexical entry can potentially correspond to multiple relations (just as a single lexical entry can correspond to several words in parsing). (iii) Although the generator mainly follows a lexicalist approach, constructions may still potentially introduce relations. This is only allowed, however,

if the relevant relations are already present in the input semantics. (iv) The system also allows for lexical entries that do not introduce relations (analogous to empty categories in parsing). However, since such entries may have negative consequences for the overall efficiency, they are only added to the chart if they are licensed by special trigger rules that are defined to match specific properties of the input semantics.

After the initial phase of lexical look-up and instantiating the chart with inactive edges, the second step is the actual chart generation. As said, each edge is associated with a set of relations in the input. Chart generation then proceeds in a bottom-up and head-first fashion, matching inactive edges against existing active edges, or creating an active edge by matching an inactive one against the head daughter of a rule. Before two edges are combined in a construction, the generator first checks to make sure that the relations they cover do not overlap. Generation is complete when all inactive edges covering the entire input MRS have been found.

Similarly to chart parsing, chart generation can be computationally expensive, and the complexity of chart-generation is exponential in the worst-case scenario. If we make the simplifying assumption that each EP corresponds to a single lexical item, then an input MRS containing n EPs can in the worst case yield $O(2^n)$ edges, each covering a different subset of EPs (Carroll & Oepen, 2005). Several techniques for improving processing efficiency are described by Carroll and Oepen (2005). One of the techniques that has been shown to have the greatest impact on efficiency is the use of local ambiguity packing for keeping the number of edges covering a given subset of relations down to a minimum. Similarly to the use of parse forests (Billot & Lang, 1989) in relation to chart-based parsing, edges that cover equivalent parts of the input can also in generation be "packed" into a single representation. Since the generator in our case is working in a unification-based universe, the equivalence test is implemented as a test for subsumption on feature structures (Oepen & Carroll, 2000).

One particularly persistent source for the potential complexity in chart-generation is provided by intersective modification (e.g. as in *high rugged Norwegian mountains*). If the linear ordering of modifiers is unconstrained by the grammar, there will be $n!$ different possible phrases for n modifiers (e.g. as in *rugged high Norwegian mountains*, *Norwegian high rugged mountains*, etc.). The problem is two-fold; one representational and one algorithmic. On the one hand there is the problem of efficiently representing all the permutations in surface order that the modifiers give rise to. This is handled by local ambiguity packing in the generation forest as described above. On the other hand there is the problem of ensuring local completeness during generation. This is tackled by a mechanism for so-called *index accessibility filtering* described by Carroll and Oepen (2005), which makes sure that every relevant unit is included before embedding in a larger structure.

After the chart generation phase is complete, we have a compact representation of all the possible realizations in the form of a forest structure. The final stage of the generation process is unpacking the generation forest. As we have noted several times by now, there might be several possible realizations available for a given semantics, and as we noted in relation to the case of intersective modifiers, the number of edges in the chart (and thereby the number of candidate realizations) can be exponential in the worst-case. Depending on the complexity of the input, it is therefore essential to have a means of ranking the output. Accomplishing this task is, of course, exactly the purpose of the stochastic models we will develop in this thesis. However, this complexity also means that an exhaustive unpacking of all the candidates in the forest structure would imply an exponential explosion in processing time. As a solution to this, Carroll and Oepen (2005) present a dynamic programming algorithm for selective unpacking of the generation forest, avoiding the need to enumerate all the possible candidates. By guiding the search through the forest using exactly the type of maximum entropy models developed in this thesis, the selective unpacking step can extract a list of only the n best candidates (according to a given model), thus saving a considerable amount of computation. The selective unpacking procedure is guaranteed to extract the exact n-best list of optimal candidates according to the maximum entropy model. An important final step before accepting something as a possible output is a post-generation check to make sure the complete edges cover all the EPs in the input and that they are in fact compatible with the semantics.

In the introduction to this chapter we mentioned that the symbolic or rule-based backbone of the LOGON system is augmented with stochastic methods for the purpose of ambiguity management. It is this interplay of different types of methods that we take a closer look at in the next section.

4.4 Branching Ambiguity

The way it is standardly depicted, the translation pipeline consists of three stages, as seen in Figure 4.4: Analysis → Transfer → Generation. As said, the LOGON translation model is centered on semantic transfer of MRSs. Analysis maps a Norwegian source string into an MRS representation; a transfer grammar maps this source MRS into a target MRS; generation, finally, maps the transferred MRS to a target string. However, each of these components are prone to produce ambiguities in one form or another: There might be several admissible parses of the source string; the transfer grammar might produce several target MRSs for a given source MRS; and finally the generator might produce multiple target language realizations of a given MRS. This means that, at each junction point in the pipeline, the process may potentially branch out, giving rise to a downstream propagation of ambigu-

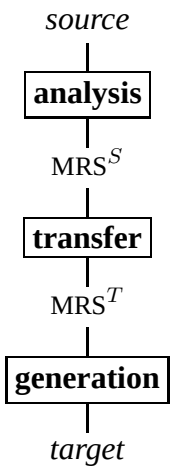


Figure 4.4: Simplified view of the MT pipeline: A source sentence is presented to the system and analyzed. The corresponding source MRS is then passed through transfer to produce a target MRS, which is finally realized as a string in the target language.

ities or indeterminacy. This means that the translation pipeline in reality looks more like the process outlined in Figure 4.5 below. We see that the translation process naturally contains several choice points, and although it might be useful during testing and development to maintain and track the full set of hypotheses at each point through the system, this is of course not a viable option in an applied version of the system. For one thing, running the system in full fan-out mode is of course computationally expensive and time consuming. But more importantly, an exhaustive list of possible outputs will typically not be very useful to the end-user.

At each point in the pipeline there is a need to select one or a few preferred candidate hypotheses that we want to pass on. This is important for reasons of both efficiency and usability. Statistical modeling can here provide a way of managing such ambiguities (in a broad sense), by enabling us to treat the notion of preferences in a principled and systematic way. Building probabilistic models can enable us to rank, and thereby ultimately filter or select, the output of the different stages of the translation process. On the part of analysis, such functionality is already available through the XLE system, which implements statistical parse selection as described by (Riezler et al., 2002). In joint work with the TrePil[12] project (Rosén, Smedt, Dyvik, & Meurer, 2005), the LOGON group in Bergen has

[12]The TrePil project works on developing methods for the semi-automatic construction of a treebank for Norwegian. See http://gandalf.aksis.uib.no/trepil/ for more information.

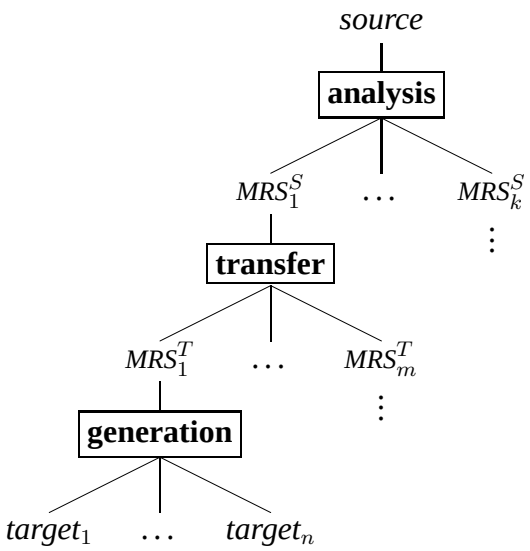


Figure 4.5: A branching MT process where each component produces indeterminacy that propagate onward.

done some initial experiments with training a discriminative log-linear model on a small Norwegian treebank. This is then used for disambiguating and ranking the Norwegian analyses. For the transfer component, the initial attempt at ranking the target MRSs is using a generative language model trained on a corpus of MRS triplets derived from 8000 treebanked sentences in the LOGON corpus (as further described in Section 4.4.1). A "sentence" in the training corpus corresponds to a dependency triplet extracted from a normalized (i.e. variable free) dependency graph representation of a given MRS (Oepen & Lønning, 2006). A trigram back-off model is then trained on the resulting set of triplets, and subsequently used for ranking competing MRSs after transfer.

Of the three intersection points to potentially fork the pipeline as in Figure 4.5, it is the last one that has received most attention in terms of original research within the LOGON project. As should be clearly established by now, this problem of dealing with the indeterminacy created by the generator is indeed the topic of this thesis, and we will describe the application of several generative and discriminative models for the purpose of ranking competing realizations generated for a given input MRS. It is important to note, however, that although the ranking experiments described in this thesis are embedded within the context of an MT system, we will not be primarily concerned with the task of ranking translations as such. The focus here is on the isolated, but still more general, sub-task of ranking

realizations produced by a tactical generator. To be sure, the realization rankers we build in this thesis play an important role in end-to-end ranking of output in LOGON, in combination with the other ranking modules just described for parsing and transfer. However, our immediate focus is not on end-to-end ranking for MT, but rather on ranking the output of a grammar-based generator. In Chapter 9 we return to the issue of end-to-end reranking for the LOGON MT system and see how the realization rankers we develop fit into this larger picture.

4.4.1 Data and Domain

We round off this chapter with a few remarks regarding the domain and the sources of the data used for development and testing in LOGON. We mentioned in Section 4.1.2 that the LOGON project has placed a strong emphasis on corpus-based testing throughout the development cycles. Parts of this treebanked data will also play an important role for the experiments on realization ranking in this thesis.

As said, the domain that was chosen for the LOGON prototype system is tourism. One important practical aspect of working with the tourist domain is that it is relatively easy to get hold of text material where both Norwegian and English versions are available. Even so, high-quality parallel texts are resources that are far from abundant. In practice, the domain has therefore been further delineated by the available text sources, which has narrowed it down to hiking in the Norwegian mountains and backcountry. The most important sources of parallel texts have been various published tourist brochures with information about hiking trips. Most of this material was originally published bilingually, and contains an English translation in addition to the Norwegian original. Besides the translation provided in the published version, two additional sets of English translations have been commissioned by the project. These have been produced by two different professional translators. As for instructions, both translators were asked to provide a rather direct translation into idiomatic American English, in the sense that they should not add any extra information in order to provide background, cultural context, and so on. Furthermore, one of the translators was specifically asked to remain as faithful as possible to the grammatical form and lexical choice of the original source, while still producing idiomatic and fluent English. The initial construction and processing of these corpora —including sentence segmentation and alignment, in addition to tagging and lemmatization by the Oslo-Bergen tagger (Hagen, Johannessen, & Nøklestad, 2000)— have been carried out by the Text Laboratory[13] at the University of Oslo. Parts of the Norwegian version of the texts and the full English version have furthermore been treebanked and manually annotated with the correct parses. The main data set is internally dubbed

[13] See http://www.hf.uio.no/tekstlab/ for more information.

JHPSTG, an acronym reflecting the fact that it is actually built out of three smaller treebanks called *Jotunheimen* (JH), *Prekestolen* (PS), and *Turglede* (TG). Together these treebanks comprise the so-called LOGON parallel tourist corpus. The JH and PS sub-corpora respectively contain approximately 27,000 and 3,700 words in Norwegian, and come with two commissioned translations into English, in addition to the one original English version. The TG sub-corpus contains 13,000 words in the Norwegian version, but only comes with two commissioned English translations (i.e. there was no original English version). In the following, we will refer to this collection of treebanks simply as the *Tourist* data set. As we shall see in Chapter 5, there exists two main parts of this corpus within LOGON. One is the main Tourist corpus which is used as *development data*, and the other is the smaller collection of *held-out data* which is reserved for final testing.

In addition to the texts taken from published booklets, other material has also been gathered from various websites providing information on tourism in Norway. For example, the (non-parallel) *Rondane* treebank, for which we will also see experimental results in later chapters, contains sentences of native-English text collected from on-line tourist guides.

Further details and statistics on the properties of the relevant data sets are provided in the next chapter, where we turn to describe how the English versions of the treebanks are adapted for the purpose of training and testing our statistical realization rankers.

Chapter 5

Symmetric Treebanks

In this chapter we will be taking a detailed look at the *treebanks* that provide the empirical basis for our experiments. Generally speaking, a treebank is a data resource where strings have been annotated with grammatical structure, typically in the form of parse trees. We will be using treebank data when evaluating the different statistical rankers, and also when training the SVMs and MaxEnt models. The training of our language model, on the other hand, is done on raw text, more specifically on an unannotated version of the British National Corpus (BNC[1]). However, we will postpone the presentation of this data set until Section 7.1, and reserve the current chapter to the annotated treebank data. We start out this chapter with some discussion of how treebanks are traditionally used when training discriminative models for *parse selection*. After revisiting the connection between parse ranking and realization ranking, we move on to explain how a treebank can be converted to a form that is suitable for the latter. This draws upon the notion of so-called *symmetric treebanks*, as we explain later. Following this initial high-level discussion of treebanks, Section 5.2 describes the particular *Redwoods* framework for treebanking (Oepen et al., 2002) which our data sets are based on. After presenting the Redwoods approach, as well as the ERG grammar underlying the annotations, Section 5.3 moves on to spell out the actual steps of how we derive a symmetric treebank required for training our discriminative realization rankers. In relation to this, Section 5.4 includes some further discussion of the theoretical implications and underlying assumptions of the treebanking procedure, specifically the notion of *bidirectional optimality*. Finally, Section 5.5 defines the features that are extracted from the treebank data and used when encoding realizations in the MaxEnt and SVM models.

[1]For more information on the BNC, see http://www.natcorp.ox.ac.uk/.

5.1 Treebanks for Parsing and Generation

The approach to realization ranking developed in this thesis starts out by recognizing its similarity to the task of *parse selection*, i.e. choosing among competing analyses of a natural language utterance. As discussed in Section 3.3 already, these problems can intuitively be seen as the inverses of one another. On the one hand, there is the task of selecting among the analyses delivered by a parser for a given string. On the other hand, there is the task selecting among the realizations produced by a generator for a given semantic analysis. The purpose of the current section is to point out some of the differences and similarities with respect to the training data that these two tasks require. We will also propose a novel bootstrapping method for automatically constructing the training data required for a realization ranker on the basis of an existing data set for parse selection.

Let us for a moment stick with the task of parse selection. Let us also assume that we are working with some kind of discriminative learner such as the MaxEnt or SVM models described in Chapter 2. When training a discriminative model for parse selection, the distribution that one is interested in is the conditional probability of an analysis given a string. This distribution is typically estimated from treebank data, consisting of strings that have been *annotated* with some kind of grammatical analyses. These annotations can include various levels of syntactic and semantic information.

The annotation step is typically a rather labor-intensive process, whereby a linguist manually specifies the correct analysis for each string. In grammar-based approaches to treebanking (such as the approach we will be dealing with here), the optimal or correct parse is usually selected by manually disambiguating the analyses delivered by a parser with respect to an underlying computational grammar. In terms of terminology, we will often also refer to the dichotomy of preferred/dispreferred training examples as *optimal/sub-optimal* or *gold/non-gold*. Furthermore, to isolate the task of identifying and marking the gold candidates, we will sometimes refer to this as *labeling*. The labeling of gold examples is otherwise often treated as being subsumed by the process of annotation.

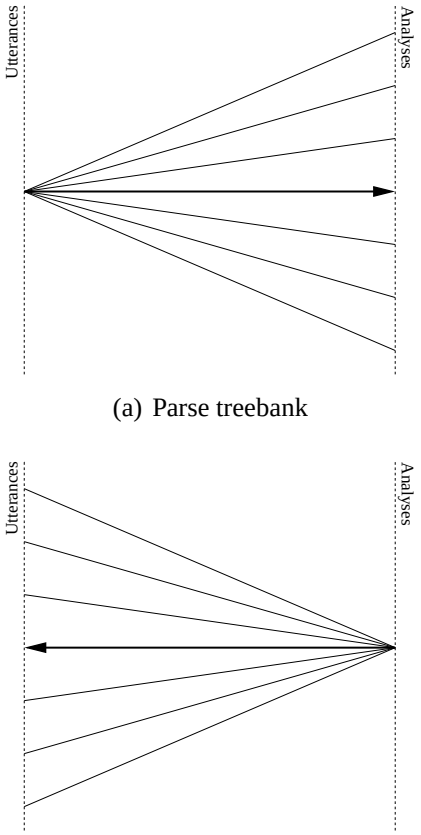
As described in Sections 2.3 and 2.4, the data examples must be represented as feature vectors when presented to the model. Each feature typically describes some syntactic property which is extracted from the treebank data, such as the application of a particular grammatical rule. Now, given such annotated and labeled data, the goal is to estimate a distribution that, for each treebanked string, maximizes the probability of the preferred parse over all the other competing candidates. Note that, although it is usually only the preferred parses that are considered part of the treebank proper, we believe it is useful to view the entire set of available parses (both optimal and sub-optimal) as first-class data objects in the treebank. One reason for this is that the discriminative models can only learn from

ambiguous examples, which means that both optimal and sub-optimal candidates must be included in the training data. Furthermore, instead of just labeling a single hypothesis as the correct one, while treating the competing hypotheses as if they were all equally bad, we sometimes want to work with a more graded notion of right and wrong. For example, all the candidates could be weighted according to some quality measure, perhaps based on their similarity to the reference. We will have more to say on this later in Section 7.2.7sec:svm-ranker.

As described above, a statistical model for parse disambiguation provides us with a distribution of parses conditioned on a given input string. For the purpose of realization ranking, however, our interest lies in a different distribution. What we want to model here is the distribution of strings given the input semantics. Such a model will enable us to discriminate between the multiple paraphrases that can be generated for a given meaning representation. There is, however, some clear sense of symmetry in the two ranking tasks, although the directionality of processing and relevant distributions are, in a sense, reversed. This apparent symmetry suggests that it might be worth trying to train models for realization ranking in a similar fashion as for parse selection. One of the main questions we want to pursue in this thesis therefore, is whether the same type of models and data sources that are currently used in state-of-the-art statistical parsing, can also be adapted for the task of statistical realization ranking. More concretely, we will be concerned with the use of discriminative models that are trained using syntactic features of treebank data in a way that is inspired by that of Toutanova et al. (2005).

So far this section has described how treebanks can be used for training models for parse ranking, and that our goal now is to try to follow a similar approach for the direction of realization ranking. Note however, that there is also some sense of directionality implicit in the structure of a traditional treebank itself. The optimality relations that a treebank encodes are usually conceived as mappings *from* strings *to* analyses. This relation is represented in Figure 5.1(a) below, where the arrow represents the optimality relation and the other arcs correspond to competing (sub-optimal) parses. In other words, the arrow points from the observed string to the disambiguated gold analysis. To best understand this figure it should be read as if we had zoomed in on a single item in the treebank.

The suggestion of Velldal et al. (2004) is to view these optimality relations as *bidirectional* or *symmetric*. This means that the original utterance is also treated as an optimal realization of the corresponding semantics in the treebanked analysis. This gives us a way to obtain a set of semantic inputs paired with their labeled optimal realizations. As a next step, we can take the semantics of the originally treebanked analysis and exhaustively generate all the possible paraphrases that express this meaning, as licensed by the underlying grammar. This results in sets of relations as those illustrated in Figure 5.1(b). This kind of expanded treebank re-



(a) Parse treebank

(b) Symmetric treebank

(c) Generation treebank

Figure 5.2: Single-item view of different types of treebanks. Arrows indicate preference or optimality relations. (a) Parse treebank of utterances paired with possible analyses. (b) Symmetric treebank including all paraphrases of the treebanked semantic analyses, and assuming bidirectional optimality relations. (c) Generation treebank of semantic analysis paired with possible realizations.

source is what Velldal et al. (2004) termed a *symmetric treebank*. A symmetrized treebank consists of

(i) gold-labeled pairs $\langle$*utterance*, *analysis*$\rangle$ that are considered bidirectionally optimal,

(ii) the set of competing analyses for each *utterance*, and

(iii) the set of competing paraphrases for each *analysis*.

Note that the bidirectional optimality pertains to the level of strings and semantics, which are the input to the parser and the generator respectively.

In contexts where we need to be clear about the specific "side" of a symmetric treebank that we are referring to, we will sometimes use the terms *parse treebank* and *generation treebank* (Velldal et al., 2004; Velldal & Oepen, 2005). The

side corresponding to the generation treebank is isolated in Figure 5.1(c). This is exactly the data that we need in order to train a realization ranker in a similar way as we described for discriminative parse selection models.

So far our exposition of treebanks has been rather abstract, but in the following sections we will try to make the ideas sketched above more concrete. Section 5.2 provides a presentation of the particular Redwoods approach to treebanking. Section 5.3 then gives a step-by-step description of how we construct the treebanks we actually use for our realization ranking experiments. This is based on the fully automated procedure presented by Velldal et al. (2004) for producing symmetric treebanks from existing resources.

5.2 Redwoods and the ERG

The Redwoods treebank[2] is a collection of annotated corpora from various domains (e.g. transcribed scheduling dialogs, ecommerce email, and lately tourism text). The corpora are available under an open-source license and currently comprise some 15,000 annotated utterances. Although not all the treebanks we are using in this thesis form part of the official Redwoods release, they have been created using the same methodology and the same underlying grammar. Much of the discussion in this section therefore pertains to Redwoods not as a specific data set, but more generally as a family of treebanks sharing the same basic methodology. Note also that it is not unlikely that at least parts of the LOGON Tourist data will be included in future releases of the official Redwoods treebank.

In the previous section we talked about treebanks rather loosely and without making many qualifications. However, there exists many distinct types of treebanks, differing hugely with respect to the type and granularity of information that is encoded, as well as to how the annotation process itself is carried out. The Redwoods treebanks have been created by what can be called a *grammar-based* approach. By this we mean that the treebanks are annotated in accordance with an existing hand-crafted grammar. More concretely, each string in the Redwoods corpora is annotated with an HPSG analysis assigned by the LinGO *English Resource Grammar* (ERG; Flickinger, 2002). All linguistic information in the annotations is grounded in the external grammar, and this is a key aspect of the Redwoods approach to treebanking (Oepen et al., 2002; Oepen, Flickinger, Toutanova, & Manning, 2004). This can be contrasted with approaches where the treebank annotations are themselves taken to implicitly define a grammar. By instead anchoring the annotations to an external grammar, the internal consistency of the treebank is guaranteed, It also makes the resource more *dynamic*, in that annotations can

[2] See `http://redwoods.stanford.edu/` for further information on the Redwoods initiative and access to the data available to date.

easily be updated to reflect revisions in the grammar as it develops and improves over time. Oepen et al. (2002) describe a method for semi-automatically updating and maintaining treebanks with respect to changes in the grammar, based on the notion of *elementary discriminants* (Carter, 1997). Simply put, these correspond to basic, differentiating properties of local ambiguities in the parse forest. By toggling the activation of these markers the annotator can usually disambiguate a parsed string in very few steps. Another aspect of the dynamic nature of the Redwoods approach, is how the rich HPSG signs are easily mapped to various other external formats.

The implementation of the all the procedures that go into developing treebanks (such as the procedures for synchronizing a treebank with a given grammar version, the generation of paraphrases when symmetrizing a parse treebank, the labeling procedures, and so forth), is based on the tight integration of [incr tsdb()] (see Section 4.1.2) and the LKB system (see Section 4.3). Together these systems provide an extensive software suite for grammar engineering and profiling, in which the facilities for construction and maintenance of treebanks form part of the same set of tools that is used for regression testing of grammars. This tight integration should not be surprising given the grammar-based approach to the construction of the treebanks, where a given treebank is intimately connected to a given version of the grammar. Furthermore, internally in [incr tsdb()], each treebank is stored in the form of a so-called *profile*. In addition to the grammatical annotations themselves, a profile also records many types of performance statistics about the underlying system (e.g. parser or generator) and grammar, all stored in the form of a relational DB.

Let us now turn to the grammar itself, and the annotations that are used in Redwoods. As described in Section 4.1, the ERG is a general-purpose and wide-coverage computational grammar of American English. It is a lexicalist grammar, couched in the framework of *Head-Driven Phrase Structure Grammar* (HPSG; Pollard & Sag, 1987, 1994). The HPSG signs are typed feature structures that encode fine-grained syntactic information. Each sign also include an underspecified MRS representation of the semantics, couched in the framework of Minimal Recursion Semantics (MRS; Copestake et al., 1995, 2006). As described in Section 4.2, an MRS is a flat, event-based representation of semantics, where the meaning of a structure essentially is given as a bag of labeled relations, together with an additional set of constraints on scope relations.

The current version[3] of ERG comprises a total of 27,659 lexical entries, for 19,468 distinct stems. The grammar associates each lexical item with a *lexical type*, organized in a type hierarchy of 856 types. Furthermore, each lexical item is

[3]The version of the ERG that we use in this thesis corresponds to the grammar of the first official public release of the LOGON system, as of January 2007.

also associated with one or more *semantic relations*, which are used when creating the MRS representation for the treebanked strings. The current rule set of the grammar comprises 62 lexical rules and 185 phrase structure rules.

The fundamental data type of the Redwoods treebanks is the *derivation tree*, and this is the type of structure recorded in the [incr tsdb()] profiles. The internal nodes of these trees correspond to identifiers of rules in the underlying grammar, such as the head-complement or head-adjunct schemas, while the preterminal yields correspond to identifiers of the lexical entries. As we shall later see in Section 5.5, the derivation tree representations also form the basis for the feature types that we extract for the discriminative treebank models. However, in that case the lexical identifiers are first mapped to their corresponding lexical entry types in the grammar. A sample derivation tree for the sentence *The dog barks* is shown in Figure 5.2, where the pre-terminal nodes have first been mapped to the such abstract lexical types.

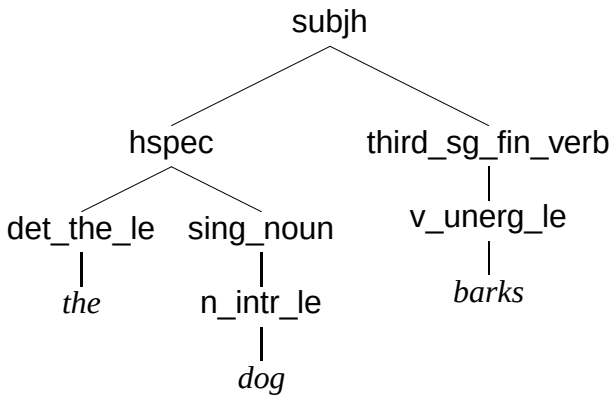


Figure 5.2: Sample HPSG derivation tree for the sentence *the dog barks*. Phrasal nodes are labeled with identifiers of grammar rules, and (pre-terminal) lexical nodes with class names for types of lexical entries. Note that, while the native derivation tree format has pre-terminals corresponding to lexical identifiers, this figure shows a somewhat modified format where these identifiers are mapped to one of the ERG's 856 abstract lexical types. This is the representation that forms the basis of our treebank features, as defined in Section 5.5.

Note that the full HPSG signs can be reconstructed from the stored derivation trees by reference to the grammar, making available the complete syntacto-semantic analysis. For example, this gives us access to the MRS-representation of the semantics, as shown in Figure 5.3 below. As should be clear, the derivation trees recorded in Redwoods are significantly different from the phrase structure

$\langle h_1$,
{h_1:prpstn_m(h_3), h_6:_the_q(x_9,h_8,h_7),
h_{10}:_dog_n_1_rel(x_9), h_{11}:_bark_v_1_rel(e_2,x_9)},
{$h_3 =_q h_{11}$, $h_8 =_q h_{10}$}$\rangle$

Figure 5.3: MRS for the sentence *The dog barks*.

trees found in many traditional resources such as the Penn Treebank,[4] and—by virtue of deriving full HPSG signs—encode much more fine-grained information.

5.3 Creating Symmetric Treebanks

In this section we describe the individual steps of how we "symmetrize" an example of a Redwoods treebank within the [incr tsdb()] environment, using the LKB generator and the ERG grammar. Our starting point is the parse-oriented version of the Tourist treebank, which comprises a total of 8146 treebanked strings. For each string, the existing treebank records the annotated correct analysis as assigned by the ERG, and additionally includes all alternate (competing but dis-preferred) analyses. The task we describe here is essentially to paraphrase the entire set of treebanked strings by generating from the semantics associated with their gold analyses. Generated realizations that match the original string are in turn then labeled as gold. The end result is a *generation treebank* that we can use for training a discriminative realization ranker. The reader is referred to Section 4.4.1 for further details on the domain and text types that this corpus contains.

The actual procedure is straightforward and consists of three main steps:

(i) Paraphrasing, i.e. exhaustively generating all the possible realizations for each gold MRS.

(ii) Labeling, i.e. matching the generator output against the original treebanked strings to identify gold realizations.

(iii) Pruning, i.e. throwing away items that are not relevant for learning.

In the following we go through each of these steps in turn, before finally summarizing some properties of the resulting generation treebank, as well as some other generation treebanks created using the same procedure.

[4] See http://www.cis.upenn.edu/~treebank/ for more information on the Penn Treebank.

5.3.1 Paraphrasing

First, for each analysis that has been manually labeled as the correct reading, we exhaustively generate all possible realizations for its MRS. In other words, for each string (and its hand-annotated intended meaning) in the original treebank, all semantically equivalent paraphrases admitted by the grammar are generated. Note that, the notion of semantic equivalence should here only be understood in *truth conditional* terms, in the sense that the paraphrases share the same MRS (in a normal-form representation). It is also worth pointing out that the grammar we are using for generation is the same grammar that is initially used for analysis. Furthermore, the *grammaticality* of all the competing candidates is guaranteed by the generator with respect to the input grammar.

For the 8146 MRSs in the Tourist data set that we start out with, we are able to successfully generate from 7856 of them. Note that some of the slight "loss" that we experience in generation is actually caused by explicitly specified limits on computational parameters such as the maximum number of edges in the chart and maximum generation time per input MRS (here set to 50,000 edges and 3 minutes, respectively).

5.3.2 Labeling

The next step is to automatically label the preferred realization(s). For each item this requires that we compare the generated realizations to the original input in the parse treebank, and the goal is to identify the ones that match this reference. These realizations will in turn define the *gold standard* or *references* in the generation treebank. Before we can do this, however, we need to define what is to count as a match. This can be done on various levels. For instance, in this thesis we experiment with two different such "alignment" strategies, comparing the realizations at either the level of full *derivations* or only their *yields*.

In the first alignment mode, i.e. comparing the full derivations, the realizations are matched node for node. In this mode, two realizations with the same surface form are not counted as a match unless their derivation histories are also the same. It is possible to allow for some slack, however, by defining *equivalence classes* over labels. This allows us to treat distinct node labels as if they were identical. We can also opt to ignore certain types of labels entirely during the matching.

In the other alignment mode, we compare the original strings in the parse treebank against the yields of the generated trees. To be precise, we do not compare the actual surface strings themselves, but rather the *pre-terminal* yields of the generated derivations. As described in the previous section, the pre-terminal yields correspond to sequences of lexical identifiers. This allows for some more flexibility in the matching process, for instance by defining equivalence classes

over labels just as we did for the derivation matching. In this alignment mode, all realizations that have the same yield will be treated as equivalent. This furthermore means that we often end up labeling *multiple* realizations as gold for a single input MRS. One particular factor that often leads to this situation is punctuation. Although punctuation is actually controlled by the grammar, we effectively ignore punctuation in the matching. As ERG implements punctuation by means of lexical rules, punctuation is not visible at the level of the pre-terminal yields. This means that realizations that differ only with respect to punctuation will be treated as equivalent when identifying the gold realizations.

Out of the 7856 items that we successfully generate for, we are able to successfully label 6422 items (using yield matching). The reason why some items are lost in the labeling step is that we are not always able to re-generate a realization that matches the original input string in the parse treebank. There are several possible sources for such mismatches. However, many of them are not necessarily very interesting from a linguistic point of view and may only concern alternate spellings, abbreviations, etc. For example, while an original string in the parse treebank may have used the shorthand forms *no.* or '-', the strings we generate for the corresponding gold MRS always use the words *number* or *to* respectively. In the labeling step we then get a mismatch for the corresponding lexical labels (i.e. number_abb_n1 vs. number_title, and x_to_y_np_sg_- vs. x_to_y_np_sg_to). Although many of these may seem trivial to accommodate in isolation, it would still be a tedious undertaking to define exception rules for all of these cases. There are also other types of mismatches where the relations involved would be too hard to capture by simple equivalence classes, or ones that are related to treebanking errors. Yet others again are caused by limitations on what is currently represented within the framework of MRS (and its use in the ERG), and also by idiosyncrasies of the generation algorithm itself. We give a few examples of various mismatches below, where the string in (a) corresponds to the original entry in the parse treebank, and (b) is an example of a string that is generated for the same gold semantics. The portions of the strings that are relevant for the discussion is highlighted in boldface. Note that, for readability we here choose to show the actual surface strings, and not the pre-terminal yields which are what we are actually matching.

One example is the case of contracted negations:

(5.1) Tourist item [14382]:

(a) Still, you **mustn't** forget to look up once in a while.

(b) Still, you **must not** forget to look up once in a while.

Such cases are currently not captured as equivalences by our simple matching procedure, as they would require an asymmetry in token positions. Similar problems arise when the original string contains contracted auxiliaries (e.g. *there's*).

Example (5.2) shows a mismatch caused by the fact that locative inversions are suppressed in generation. Furthermore, *perhaps* has for pragmatic reasons been analyzed as semantically vacuous in the parse treebank and will therefore not re-surface in generation.

(5.2) Tourist item [10842]:

(a) **Not quite as high**, but **perhaps** just as impressive, is a trip up Styggehø or Hestlegerhø.

(b) A trip up Styggehø or Hestlegerhø is **not quite as high**, but just as impressive.

Just as with *perhaps*, there are also other words and phrases that are currently not reflected in the semantics and that give rise to similar problems. For instance, this is the case with certain adverbs such as *however* and *for example*. An example of the latter is provided in Example (5.3) below. Since the semantics of this adverb is currently not represented in the treebanked MRS, it can never re-surface in generation, and we subsequently fail to identify a realization that matches the reference.

(5.3) Tourist item [13731]:

(a) We know, **for example**, that some **8000** years ago, the forests reached higher than they do today.

(b) We know that some **DecimalErsatz** years ago the forests reached higher than they do today.

A similar effect is incurred by constructions that involve the use of *both . . . and* or *either . . . or*. In such cases the generator normalizes by using a simple conjunction, leaving out the *either* and *both*. Consequently, we fail to find a match between the original string and the generated realizations, and once again we fail to label any of the realizations as gold.

Note that, Example (5.3) above also reveals something about how numerical entities are handled in the grammar. The actual numerical values are not represented as distinct predicates in the MRS, but their presence is encoded by abstract place-holders. In the generated surface strings they appear as entries such as *DecimalErsatz*, *DateErsatz* and *DecadeErsatz*. However, note that the tokens *8000* and *DecimalErsatz* are not part of the reason for the mismatch between the two (pre-terminal) yields of Example (5.3) above, as their corresponding labels (numval-card4digit and card_gle respectively) are covered by one of the aforementioned equivalence classes.

Section 5.4 below includes some further comments with respect to the implicit assumptions underlying the way we align realizations in this labeling step, and specifically with respect to the notion of bidirectionality.

5.3.3 Pruning

As said, when symmetrizing the Tourist treebank we end up with 6422 items for which we can successfully identify one or more reference realizations. However, not all of these items are useful to us as training data for the discriminative models. The models can only learn from items that display some real indeterminacy. By this we mean that the MRS has more than one generated realization and, furthermore, that not all of the available realizations are marked as gold. If this is not the case then the treebanked MRS will not present us with any real problem of choosing among alternatives. The last step of constructing our training data is therefore to prune away such irrelevant treebank items. After pruning the ones that only have a *single* realization we are down to 5539 items. The items thrown away in this step are typically very short sentences, headings, etc. After further cutting away the items where *all* of the realizations are labeled as *preferred* (e.g. in cases where the competing candidates only differ with respect to punctuation) we are finally left with a net total of 3921.

5.3.4 The Generation Treebanks

To sum up, this section has described the stepwise construction of the Tourist generation treebank that we will later be using for training and development. It includes all possible paraphrases for each gold semantic representation in the initial parse treebank. Realizations are labeled as gold if they match the original string in the treebank corpus. The final data set includes a total of 3921 items (i.e. MRSs with their corresponding set of realizations). These items have all survived a long chain of processing and share the following properties:

(a) The original string has been successfully parsed and disambiguated.

(b) The corresponding gold MRS successfully generated with more than one realization.

(c) One or more (but not all) of the realizations were successfully aligned to the original string and labeled as a gold reference.

Although the treebanking procedure above was described using the Tourist data set as an example, we follow the exact same steps also when producing other generation treebanks for our experiments. Another key data set for the experiments in this thesis is the generation treebank constructed for the LOGON *held-out data*. This data set is based on material that was set aside during the creation of the Tourist corpus and reserved for testing purposes. To make sure that the final system evaluation is as pure and strict as necessary, the held-out data has been sealed off from inspection by any of the LOGON system developers during

the actual development phase. In the same vein, the LOGON Tourist held-out data set can also serve as the basis for a generation treebank used for the final system evaluation of the realization rankers. Note that we will report experimental results also on the development set in the form of cross-validation, computing the average performance across folds. However, the test scores computed for the held-out data will provide a more strict indication of the true ranking performance, as the rankers have not been tuned to this data set and we have not inspected the data during the development and selection of models.

Table 5.1 summarize both of these generation treebanks, breaking down the data along several dimensions. The items are first split into bins according to their number of realizations (i.e. what would correspond to "ambiguity rate" in parsing terms). For each bin the table then shows the corresponding number of items, average sentence length, average number of realizations, and average number of realizations labeled as gold. As can be seen from the tables, the total average

Tourist Development Data

Aggregate	Items	Words	Trees	Gold	Baseline
$100 \leq n$	369	27.5	360.0	8.0	3.33
$50 \leq n < 100$	230	24.7	73.5	4.9	6.64
$10 \leq n < 50$	1144	20.6	22.5	3.3	17.08
$5 \leq n < 10$	868	15.3	6.9	2.2	32.13
$1 < n < 5$	1310	13.9	3.2	1.4	45.64
Total	**3921**	**18.1**	**47.3**	**3.0**	**28.05**

Tourist Held-Out Test Data

Aggregate	Items	Words	Trees	Gold	Baseline
$100 \leq n$	17	26.0	572.5	7.8	2.20
$50 \leq n < 100$	24	24.1	78.1	5.6	7.55
$10 \leq n < 50$	83	20.0	23.4	3.2	15.86
$5 \leq n < 10$	57	14.6	6.7	2.1	32.21
$1 < n < 5$	88	14.0	3.1	1.4	45.08
Total	**269**	**17.6**	**52.8**	**2.9**	**27.28**

Table 5.1: Some core metrics for the generation treebanks we use for training and testing. The data items are aggregated relative to their number of realizations. The columns are, from left to right, the subdivision of the data according to the number of realizations, total number of items, average string length, average number of realizations, average number of references, and finally the baseline for expected accuracy by random choice.

length of the 3921 candidate realizations in the Tourist generation treebank we use for development is 18.1. For the 269 items in held-out data set the average string length is just slightly shorter, at 17.6. Note that these figures refer to the length of *tokenized* strings, which means that for example punctuation is treated as separate units. More details on the tokenization process are provided in Section 7.1.1 (note however that the normalization performed before computing the average string lengths does not include end-of-sentence markers). In the development data, we see that the average number of generated hypotheses per item is 47.3, while the number is 52.8 for the held-out data. Not surprisingly, we find a higher number of candidate realizations for the items which also have a longer average string length.

Although the degree of non-determinism, i.e. the average number of available candidates, is obviously an important factor characterizing the difficulty of the ranking task, another important factor is the number of candidates labeled as gold, as described in Section 5.3.2 above. This figure is around three for both data sets. On the basis of these of these two properties, we can compute a *random choice baseline* to more directly indicate the difficulty of the ranking task. This corresponds to the average exact match accuracy we could expect to obtain if we were to select candidate realizations completely at random. As we see from the final column of Table 5.1, the baseline figure for the full development treebank is 28.05%, while slightly lower for the held-out data, at 27.28%.

Note that, the fact that these Redwoods style treebanks are built on top of a grammar (and not the other way around) means that they can be dynamically updated to reflect revisions of the grammar. As noted in Section 5.2 of this chapter, Oepen et al. (2004) developed a semi-automated procedure to (re-)synchronize a parse treebank with a new version of the grammar. An updated version of a corresponding generation treebank can be produced by repeating the automated procedure we have described in the current section.

There is one final point which has not yet been mentioned in relation to how we choose to produce the generation treebanks, and this regards the degree of specificity in the semantic representations. Recall from the discussion in Section 4.2, that a key aspect of the MRS formalism is that it allows for varying degrees of underspecification in the representations. One important dimension in this respect is the notion of information structure (IS). In the construction of the Tourist data set discussed above, the input MRSs include the attributes PSV and TPC. These attributes encode *topicalization* and *passivization* respectively, and can be used for requesting foregrounding of a specific entity. However, this information may also be underspecified in the MRS or it can optionally be suppressed in generation. In this case the generated realizations will also include all grammatically legitimate topicalized and passivized constructions. Consequently, it would in practise be up to the statistical model we train to decide whether or not to use a passive or active voicing. It can, of course, be debated what is the appropriate division of la-

bor here, i.e. whether such aspects related to information structure and argument structure should be left to the surface realizer (or, more precisely, the statistical ranker), or if it should instead be determined during the phase of strategical generation and deep planning.[5] Given that we are only dealing with single-sentence generation here, in the sense that we are not looking at properties of a wider discourse context, it can easily be argued that making such decisions at this point in the pipeline would be ill-founded, and that they are better left to the planning component. When the hybrid generator we develop is used in the LOGON MT system, this is indeed the mode that is used, in the sense that this information is passed on through transfer from the source analysis. However, in order to also test our hybrid set-up on a slightly harder ranking task with a larger degree of indeterminacy, we additionally include results for data sets that do not include this information. Table 5.2 below summarizes the generation treebanks we construct for the LOGON development data and held-out test data where we disregard the IS markers in the semantics during the paraphrasing phase.

As can be seen from the fourth column (*Trees*) in Table 5.2, the average number of realizations per input MRS is more than doubled compared to the IS-specified data set in Table 5.1. For the development data we generate an average of 115.7 realizations per input MRS, 112.3 for the held-out data. As the degree of non-determinism is much higher, the random choice baseline for exact match accuracy (last column) is correspondingly lower, down to around 16.5% for both data sets. Also note that, while the topicalized or passivized sentences are often expected to be longer due to extra punctuation, bi-phrases, etc., we see that the average string lengths for the items in the IS-underspecified data sets are actually shorter (ticking in at a total of 16.6 and 17.0 for the development and held-out set respectively). One reason for this is that some of the items for which we would expect to see the longest sentences do not successfully generate in the underspecified set-up. The reason for this is simply that we are more likely to hit time-outs, due to the increased non-determinism. Note that the distribution of test items according to average string length is shown in the histogram of Figure 5.4 (the corresponding held-out distribution is in Figure 8.1, in Chapter 8). On the other hand, observe that we also end up with quite a few more items in final data-sets for the non-IS treebanks: We get roughly 800 more items for the development data and 80 for the held-out data. So, although we might loose some items due to time-outs, we have also gained a few. The reason for this is that more items are likely to survive the pruning step as described in Section 5.3.3. In the underspecified mode, more items are likely to display proper non-determinism, i.e. have more than one

[5] See Section 1.1 for a discussion of strategical vs. tactical generation. When embedded in a transfer-based MT system such as LOGON, the strategical phase can be seen to naturally correspond to the phase of source analysis and the subsequent semantic transfer.

realization. This means that more items can be considered relevant for the ranking task and be included in the final data set. Moreover, the increased number of topicalized and passivized constructions also means that there is less chance of finding items where all of the candidates have been labeled as gold, which would again result in pruning.

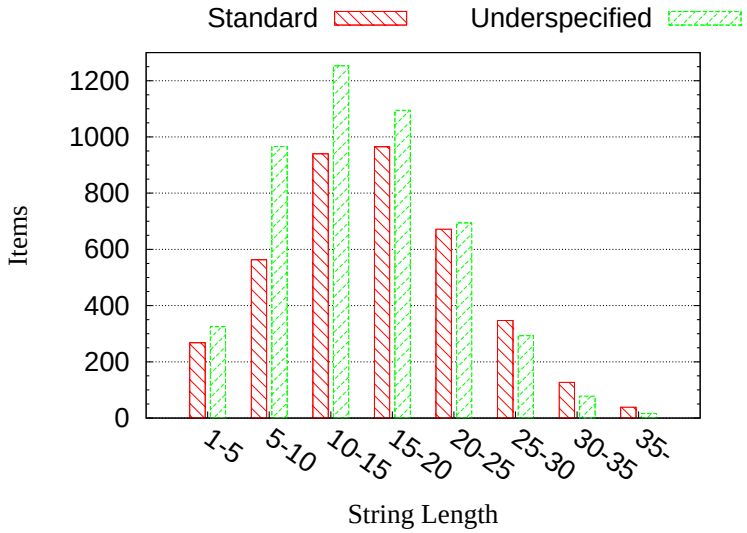


Figure 5.4: **Average string length** of items in the development data. *Primary* shows the distribution of average string length for the 3921 items in the standard test data, while *underspecified* shows the same for the 4720 items in the treebank generated with IS-underspecification. A bin such as '5-10' comprises strings that contain more than or equal to five words, but less than ten.

This section has described the stepwise procedure of how we construct the generation treebanks we will be using for development and held-out testing of the discriminative rankers. In the next section we turn to some more high-level discussion about some of the implicit assumptions underlying the treebanking procedure. In particular we will revisit the notion of bidirectionality, which forms the basis of the way we identify gold realizations in the labeling step of Section 5.3.2.

5.4 Bidirectionality and Superoptimality

In a traditional parse treebank, the pairs of surface strings and analyses encode a preference or optimality relation between strings and their analyses. Each annotated analysis is taken to represent an optimal interpretation of the corresponding observed surface form. As described above, the strategy developed here for utilizing treebanks for generation comes with the additional assumption that, without

Tourist Development Data, underspecified for IS

Aggregate	Items	Words	Trees	Gold	Baseline
$100 \leq n$	966	24.4	468.0	4.4	1.58
$50 \leq n < 100$	433	20.9	71.8	3.2	4.63
$10 \leq n < 50$	1607	17.2	23.8	2.2	10.83
$5 \leq n < 10$	842	12.1	6.9	1.7	25.02
$1 < n < 5$	872	11.3	3.3	1.3	41.82
Total	**4720**	**17.0**	**112.3**	**2.5**	**16.62**

Tourist Held-Out Test Data underspecified for IS

Aggregate	Items	Words	Trees	Gold	Baseline
$100 \leq n$	74	23.0	472.9	4.7	1.33
$50 \leq n < 100$	29	20.7	74.2	2.6	3.42
$10 \leq n < 50$	113	16.8	22.7	2.1	10.53
$5 \leq n < 10$	69	11.7	6.9	1.7	26.07
$1 < n < 5$	64	12.5	3.2	1.2	39.71
Total	**349**	**16.6**	**115.7**	**2.5**	**16.41**

Table 5.2: Some core metrics for the additional versions of the generation treebanks where realizations are generated from MRSs with underspecified information structure markings, giving added indeterminacy with respect to such phenomena as passivization and topicalization. As in Table 5.1, the first column shows how the data items are binned according to their number of realizations. The other columns are; total number of items, average string length, average number of realizations, average number of references, and finally the baseline accuracy corresponding to random choice.

introducing too much distortion, this optimality relation can also be taken to hold in the *reverse* direction. In other words, for the purpose of discriminative training, the observed string is also treated as an optimal way of expressing (the semantics of) the reference analysis. This gives us a means of "bootstrapping" the training data that we need on the basis of an existing parse treebank. As described in Section 5.3, it is this assumption of bidirectionality which guides the labeling of gold realizations after generating the paraphrases for the treebanked strings.

Now, during the work on this thesis, the core ideas in relation to the treebanking have been presented at several international conferences and workshops, as well as in informal discussions with colleagues. When it comes to this notion

of bidirectionality, the general impression from these exchanges is that there is some division between people from certain "camps" in the field. While people with a background in machine learning and empirical NLP are typically prepared to accept the premises of this arrangement without hesitation, researchers with a background from theoretical linguistics are sometimes more reluctant. The purpose of this section is therefore to provide some additional discussion of both why the assumption of bidirectional optimality is so useful, as well as to what degree such an assumption is warranted. The section ends with a short detour by way of the linguistic framework known as *Optimality Theory* (OT), where we look at some of the ideas of bidirectional optimization developed there.

First of all, what do we actually mean by optimal? Admittedly, this has remained a rather vague and fleeting notion in the text so far. In relation to parsing, the notion of an optimal candidate is perhaps more clear-cut. It will often (but far from always) be the case that only a single parse clearly stands out as *the* correct reading. In generation on the other hand, the situation can be a bit more fuzzy. Many of the alternative realizations will often not be radically different from each other, and sometimes it might be difficult to point out a single surface string as the ultimate candidate. However, we need to define some kind of gold standard in order to formulate the problem as a so-called *supervised learning* task. Machine learning separates between unsupervised and supervised learning. An unsupervised learning task is one where we are trying to uncover hidden structure or latent variables that are not directly observable in the training data. This is in contrast to supervised learning, which is what we are dealing with here, where it is necessary to guide the learner by labeling the training examples with whatever information we are trying to learn. For the ranking task this means that we need to pick one or more examples for each item in the training data and tell the learner that these are the candidates we would like it to prefer. Now, one way of doing this would be to manually inspect each and every realization, marking the ones we believe to be the best or optimal. However, this would be a very time-consuming and labor-intensive task. This might perhaps become clearer when considering the fact that, for the 3921 items in the Tourist treebank, there are a total of 195,242 distinct realizations. The whole idea of constructing symmetric treebanks as described in the previous sections is that it allows us to bypass this step. By assuming reciprocity in the preference relation between the strings and the semantics in the original annotations, we can "bootstrap" the training data we need on the basis of an existing parse treebank. Instead of manually choosing the preferred expression of a given semantics ourselves, we trust the expression chosen by the original speaker, as it occurs in the corpus, to be a good candidate in most cases. Intuitively, the choice made by the original human speaker would seem to provide us with a solid starting point for guiding a statistical model for machine generated language.

We can also try to motivate the approach by following a route that is somewhat

less pragmatic, albeit arriving at the same conclusion. Let us first observe a few basic facts surrounding the enterprise of grammatical annotation. Many of these will undoubtedly seem like stating the obvious. Nonetheless, when taken together, these simple observations will hopefully make it clear how and why the assumption of bidirectionality, for modeling purposes, can be motivated and justified.

Let us start by establishing a simple, yet important, property of the corpus data underlying the treebanks. This concerns the fact the corpora we are working with are composed of naturally occurring utterances produced by real language users. What we mean by "naturally" and "real" here is simply that the strings have not been artificially constructed by linguists for experimental or explanatory purposes, but are observations of actual language use. Each string has been produced by an actual language user, and for the means of communicating some message.

Given the corpus of language data, let us now turn to the process of annotating parses when constructing the initial treebank. What does it actually imply to label a given parse s_i as the correct or preferred analysis for a given surface form r_j? First of all, the fact that one is doing annotation in the first place usually means that one already subscribes to some kind of abstract grammatical framework. In our case this abstract framework is HPSG. More specifically, we are working with HPSG as implemented in the ERG. Now, when we label a given parse as being the correct one for a given utterance, we make a claim, at some level or another, that this particular parse represents the intended interpretation of that utterance. Of course, we would never find a computational linguist or grammar writer claiming that these abstract grammatical representations actually are accurate models of how we as human language users mentally represent language. On the other hand, most would probably still object to a claim in the opposite extreme, that their representations are entirely inadequate to this end. The whole enterprise of parsing would seem rather futile if we did not believe that, at some level, our syntacto-semantic annotations are a meaningful approximation to the way language is used and understood by humans.

So, when we are annotating a given string, we are in some way committing to the belief that this particular semantic analysis can be taken to represent, in some approximate way, the actual idea that some real language user wanted to express at the time it was uttered. In other words, when a sentence is included in our semantically annotated treebank, we are in some sense making the assumption that a (presumably rational and competent) language user generated this very sentence as an expression of the treebanked semantics. Some speaker out there found this very sequence of words to be the best way to express the given meaning. Now, from the perspective of NLP as a data-driven and empirically oriented discipline, we would expect that language users are granted some authority on the issue of how to best express a given meaning. We would generally expect language users to be experts when it comes to formulating expressions that convey the meaning

that they want to communicate, and that they do so in an effective and natural-sounding way. For the purpose of training a model for generation ranking, it does not seem overly radical then to let the originally observed expressions be treated as the optimal realizations for the corresponding treebanked semantics. To be sure, language users are not perfect, in the sense that they sometimes express themselves in a manner that is *less* than optimal in terms of conveying the meaning that they want to convey. But that is not the claim we are trying to make here. The point is merely that their efforts probably represent a good enough approximation to the "true" optimal (if anyone believes such a thing to exist), that they can be used for guiding a statistical learner.

To sum up, when treebanking a corpus, we are making the claim that a given parse is the correct analysis for a given string. In a Redwoods treebank, the analysis also contains a semantic component in the form of an MRS. Now, for the purpose of training a discriminative realization ranker, we would like to also be able to treat the observed string as an optimal expression of the semantics. By assuming that (i) the treebanked semantics corresponds fairly well with the intended meaning, and (ii) that language users are fairly good at expressing the meaning they intended, this seems like a reasonable arrangement.

5.4.1 Bidirectionality and Superoptimality in OT

Before we leave this topic, it is worth noting that the issue of bidirectionality has also been discussed by many researches working within the framework of *Optimality Theory* (OT; Prince & Smolensky, 1993). Central to OT is the idea of evaluating the *harmony* between inputs and outputs with respect to a set of *ordered constraints*. The approach has been applied to a wide range of areas within linguistics such as phonology, morphology, syntax, semantics, and pragmatics. However, there exist several alternative conceptions of the main components in OT. One modification concerns the *direction of the optimization*, which in the standard formulation is taken to be unidirectional, from inputs to outputs. For example, Blutner (2000) and Zeevat (2000) both suggest various notions of *bidirectional optimization*. Several different notions of bidirectionality have been suggested by these and other authors in attempts to account for how the processes of linguistic production and interpretation can mutually constrain each other. Simplifying somewhat, the argument for two-way optimization can be condensed in the following two statements. When trying to express a meaning, other possible meanings must be taken into account. When trying to interpret an expression, other possible expressions must also be considered. The idea is that we are simultaneously taking into account the perspective of the listener and the speaker, during both production and interpretation. Of course, this kind of appeal to bidirectionality is closely related to the well-known Gricean conversational maxims

(Grice, 1975). These pragmatic principles assume two basic and conflicting economical interests: speaker's economy and hearer's economy. Given the view that communication through language is a cooperative effort, both sender and recipient are obliged to strive to balance between these opposing principles. For example, the speaker is expected to make the process of interpretation as smooth as possible and should therefore take care to avoid violation of interpretation constraints on behalf of the recipient (Zeevat, 2000).

In the OT literature, form–content pairs in which both elements are optimal with respect to each other (by some definition of optimality), are called *superoptimal*. However, optimality and superoptimality can be defined with various degrees. In the terms used by Blutner (2000), the gold-labeled form and content pairs in our symmetric treebanks would be assumed to be *strongly superoptimal*. This means that the form is taken to be the top ranked candidate given the content, and vice versa. In reality, this notion of superoptimality is probably too strong for many cases. As a simplifying assumption for setting up our supervised training task however, it seems to provide us with a reasonable starting point.

5.5 Feature Templates

The overall theme of the current chapter is treebank data. We have described the construction of the generation treebanks that we use for training the discriminative learners. We have seen how the treebanks relate pairs of semantic specifications s_i and realizations r_j. More specifically, we are dealing with pairs of MRSs and derivation trees, as shown in Figure 5.3 and Figure 5.2 respectively. Recall, however, from the description of SVMs and MaxEnt models in Chapter 2, that the data must be presented to the learner using a *vectorial representation* $f \in \Re^d$. Each vector element f_i encodes the value of a specified feature function which describe some property of a realization. Furthermore, for each feature f_i, the learned model will have a corresponding weight λ_i. When presenting the modeling frameworks in Chapter 2, we gave these features a rather abstract treatment as unspecified functions from the space of form–content pairs to the space of real values. In this section, however, we present the particular treebank features that we use in more detail. To be more precise, what we will actually be presenting are the general feature *templates*. We never need to explicitly specify all the individual features themselves. The actual features are produced automatically in a sense, by matching the templates against the training instances in the treebank. Each feature template will then be instantiated any given number of times, yielding the actual features.

As we go through the various feature types, Table 5.3 will show examples of instantiations of these features for the tree in Figure 5.2.

For the purpose of parse selection, Toutanova et al. (2005) train several discriminative log-linear models on the Redwoods parse treebank, using features defined over HPSG derivation trees. The feature sets used in these models provide the starting point also for our generation experiments. Recall from the previous discussion that the non-terminals of a derivation tree represent the *construction types* and *lexical identifiers* of the underlying HPSG grammar. For the purpose of feature extraction, however, these latter pre-terminal lexical entry identifiers are first mapped to the more abstract *lexical entry types* (LE-types) within the grammar. Now, the basic feature set of our discriminative realization rankers is defined in the same way as for the CPCFG-S model of Toutanova et al. (2005). In this set-up, each feature captures a local sub-tree from the derivation, limited to depth one. In other words, these features record the particular productions observed in a given tree, such as hspec → det_the_le sing_noun. In Table 5.3 these features correspond to feature template # 1. The *value* of a given type # 1 feature corresponds to the number of times a given expansion occurs in the tree.

To reduce the effects of data sparseness, feature type # 2 in Table 5.3 provides a back-off mechanism for these configurational features. While the type # 1 features record the full sequence of daughters in the local sub-trees, the type # 2 features reduce this to only one of the daughters in turn. We sometimes refer to these features as *active edge* features, in analogy to the notion of active edges in chart-based generation/parsing. Note that, since the context that gets recorded is reduced (as compared to the type # 1 features), there will also be significantly fewer unique features instantiating this template, while the total number of occurrences of each of these features will be correspondingly higher.

Conversely, to facilitate sampling of *larger* contexts than just sub-trees of depth one, feature template # 1 also allows for various degrees of *grandparenting*.[6] By specifying an additional parameter to the template, the recorded information can be extended to include various levels of ancestor annotation. This ancestor parameter is available also for the active edge features (type # 2), which means that, for a given node, we extract a non-branching path through the tree that includes a single daughter together with an upward chain of dominating nodes. In Table 5.3, the level of grandparenting is indicated by the first integer in the instantiated type # 1 and # 2 features. For the experiments that we report later, we include a maximum of 4 grandparents.

We have seen several extensions of the basic type # 1 feature above, adding information about ancestor nodes and/or active edges. Another extension of this basic structural feature adds information about balance or skew of constituent weight among sister nodes, trying to capture generalizations about *relative constituent*

[6] By *grandparenting* we refer to all use of ancestor information, regardless of whether we for the current node are including a parent, grandparent, great grandparent, etc.

Type Id	Sample Features
1	⟨0 subjh hspec third_sg_fin_verb⟩
1	⟨1 △ subjh hspec third_sg_fin_verb⟩
1	⟨0 hspec det_the_le sing_noun⟩
1	⟨1 subjh hspec det_the_le sing_noun⟩
1	⟨2 △ subjh hspec det_the_le sing_noun⟩
2	⟨0 subjh third_sg_fin_verb⟩
2	⟨0 subjh hspec⟩
2	⟨1 subjh hspec det_the_le⟩
2	⟨1 subjh hspec sing_noun⟩
3	⟨1 n_intr_le *dog*⟩
3	⟨2 det_the_le n_intr_le *dog*⟩
3	⟨3 ◁ det_the_le n_intr_le *dog*⟩
4	⟨1 n_intr_le⟩
4	⟨2 det_the_le n_intr_le⟩
4	⟨3 ◁ det_the_le n_intr_le⟩

Table 5.3: Examples of structural features extracted from the derivation tree in Figure 5.2. The first column identifies the feature template corresponding to each example; in the examples, the first integer value is a parameter to feature templates, i.e. the depth of grandparenting (types 1 and 2) or n-gram size (types 3 and 4). The special symbols △ and ◁ denote the root of the tree and left periphery of the yield, respectively.

weights. We can derive a basic notion of *skew* (s) simply as the standard deviation of constituent lengths over sister nodes within a local sub-tree. Constituent length is here simply measured by the number of input tokens that it spans. To reduce data sparseness, we normalize the s values into three bins or classes according to the following three conditions; $s = 0$, $0 < s < \sqrt{2}$, or $\sqrt{2} \leq s$. Template type # 1 is then correspondingly extended by adding an extra integer parameter, indicating the class identifier of s on the corresponding sub-tree. Furthermore, to obtain more fine-grained statistics over constituent weight distributions, we also add another variant of this feature, where each daughter label is annotated with the number of words covered by that node. Again the values are binned, this time according the whether we have a single word, up to four words, four to eight, or more than eight words.

In addition to the dominance-oriented features defined above, our models also include features that are more linearly oriented. The features of type # 3 and

4 record n-*grams of lexical types*, extracted from the pre-terminal yields of the derivation trees. In loose analogy to HMM part-of-speech tagging, the n-grams of lexical types capture syntactic category assignments. The difference between templates # 3 and # 4 only regards *lexicalization*, as the former additionally includes the surface token associated with the rightmost element of each n-gram (again, this can be seen as loosely corresponding to the emission probabilities in an HMM tagger). An additional parameter for both of these features is the *size* of the n-grams, and for the experiments here we use a maximum n-gram size of 4. Note, however, that the *optimal* n-gram size is something that must be determined empirically. The same is true for the many other feature parameters, such as the optimal level of grandparenting, various frequency cut-offs, and so forth. Additionally, we would of course also want to asses the effect of including a particular feature type in the first place. In general then, we see that there is a large space of possible feature configurations that need to be experimentally explored. Moreover, the number of actual features that get extracted quickly grows to very large numbers. For example, for the Tourist treebank we end up extracting more than a million distinct features, given the templates defined above. Finally, note that there is typically also a range of learner-specific parameters that need to have their optimal values experimentally specified as well. For any data set of a reasonable size, this type of experimentation quickly becomes computationally demanding. The next chapter goes into some more details on the challenges posed by this kind of search and model tuning, as well as some of the methodological and technological advances to the experimentation environment that we realized for the purpose of this thesis.

This concludes our chapter on the treebank data. We began this chapter by pointing out some of the differences between models geared toward parsing and generation, and then introduced the notions of symmetric treebanks and generation treebanks. This also included some discussion of the notion of bidirectional optimality. After giving an overview of some of the properties particular to Redwoods treebanks and the underlying ERG grammar, we presented the details of the specific treebanks that we use for experimentation in this thesis. Finally, we defined the feature types that we extract from this data. Before we turn to describe the actual ranking experiments in Chapters 7 and 8, the next chapter documents some of the implementational details of the experimentation environment that we use. While the current chapter has focused on the data sets themselves, in terms of both treebanks and the features we extract from them, the next chapter focuses on techniques for managing and processing this data.

Chapter 6

Experimentation Environment

In this chapter we present some of the key details related to the implementation of what we broadly refer to as our *experimentation environment*. By this we mean the collection of facilities for training and testing the various stochastic models investigated in this thesis, implemented as extensions to the [incr tsdb()] system, as further described in Section 4.1.2. Roughly speaking, the first half of the current chapter will be concerned with aspects of (efficient) *data handling*. The latter half will be concerned with aspects of *evaluation*.

The initial discussion in this chapter emphasizes the need for efficient handling of the treebank features, especially during the development and experimentation stages when we typically need to run a large number of repeated experiments. We start out, in Section 6.1, by providing some more background and motivation for this claim, and argue that advanced data management is a prerequisite to making experimental model development computationally tractable for larger data sets. Then, in Section 6.2, we go on to take a brief peek under the hood of our experimentation machinery to reveal some of the specific implementation details that make efficient large-scale experimentation possible. The purpose of this chapter is thus to shed light on some of the more practical aspects of model building and experimentation. These are the kind of issues that are often not discussed in the literature. However, as any NLP practitioner knows all too well, a significant part of the actual workload is typically tied to data processing challenges, such as managing large data sets. The project presented in this thesis is no exception. In Section 6.3, we turn to look at some of the various *scoring metrics* we have incorporated in the system for evaluating the different realization rankers. For any differences in performance revealed by these evaluation measures, we have also integrated a selection of *hypothesis tests* that we use to determine the statistical significance, as described in Section 6.4.

6.1 Parameter Search and Model Tuning

During the development phase of model building, there is typically a large space of parameters that need to be empirically tested. The specific values that give the best model performance are generally not known *a priori*, nor can they generally be analytically determined. This means that we typically need to perform cycles of repeated experiments in order to find the best (or at least a reasonable) value for each of them.

On the one hand, there are parameters related to the specific learner. Note that we are not here talking about the model parameters that we are trying to estimate, such as the λ-vector for a MaxEnt model. What we are talking about here are instead the various parameters governing the estimation process itself. For instance, for the MaxEnt models one can specify such things as the minimum improvement in log-likelihood between iterations before stopping, and the variance of the prior weight distribution. For the SVMs there are numerous parameters related to the specific kernel, as well as shared parameters such as the penalty factor governing the trade-off between the training error and margin size. As we shall see later, some of these estimation parameters can have a rather drastic impact on model performance.

In addition to the learner-specific parameters, there are also the various variables that relate to the feature functions. As described in Section 5.5, these include such things as the level of grandparenting, the span of lexical type n-grams, different strategies for defining frequency cut-offs, and so forth. Additionally, we would of course also like to assess the contribution of the individual feature types themselves. All in all, we see that there is a large pool of parameters that need to be gauged and tuned. On top of this, the various parameters will often interact in subtle ways, so that changing the value of one parameter has consequences for what is the best value for another. For example, what is the best value of the variance parameter of the MaxEnt regularizer can depend on which subset of features we choose to include. When parameters interact in this way it is necessary to do cross-experimentation, exploring the full "grid" of parameter values.

Although it should already be clear at this point why it might be necessary to perform many rounds of experiments, there is yet another factor to complicate things further. As is typically the case in NLP, the data sets available for training and testing are relatively small. Small not in terms of required storage, but in terms of providing reliable statistics for the complex distributions we are trying to model. We therefore want to take care to "squeeze" the maximum out of the little data that we have available. For example, setting aside a portion of the data for held-out testing during development would mean loosing valuable training data. We would also risk ending up with a held-out test set that is too small to really be informative. One common method to alleviate this problem is the use of

n-fold cross-validation. To give an example, consider the case of *ten-fold* cross-validation, which is what we actually often use during the development stage. This means that the training data set is first split up into ten equally sized and non-overlapping parts, or *folds*. We then repeatedly train a model on nine of the folds, and use the one remaining fold for testing. After ten full rounds we will have tested on the entire data set, and we can compute the average performance over the ten folds. The reason we are including this technique in the present discussion is, of course, that it contributes to further increase the already high number of training cycles that need to be run. First, we have the relatively large space of interacting parameters that need to be experimentally tested, and, secondly, for each of these experiments we need to train and test ten different models (or whatever n number of folds we choose to use).

Now, the first step in training an SVM or a MaxEnt model, is to extract a set of features. Depending, in part, on the definition of the features, this extraction phase can be computationally expensive. Recall that, for the 3921 different items in the generation treebank that comprise the primary Tourist development data, we have a total of $195,242$ distinct realizations. For the IS-underspecified version, we have 4720 data items with a total of $530,082$ distinct realizations. It should come as no surprise then that traversing all the derivation trees in the treebank, and extracting features as defined in Section 5.5, is a task that is computationally very expensive. In fact, the feature templates described in Section 5.5 yield a total of $1,182,587$ instantiated feature types for the primary development data. For the IS-underspecified version, the number of distinct feature types is $1,921,636$. Now, as will be clear when we later discuss various experimental configuration options such as frequency cutoffs, etc. (see e.g. Sections 7.2.1 and 7.2.3), we will typically only use a sub-set of this feature pool in a given experiment. Still, the number of features activated for a given experiment will typically be on the order of several hundred thousands, at the minimum.

Recall from Section 5.2 that the [incr tsdb()] profiles record realizations in the form of derivation trees, stored in textual form in a data base. For feature extraction it is necessary to reconstruct the full HPSG sign from these derivation trees by reference to the underlying grammar, again adding to the computational cost. Note that some additional feature types even involve applying *auxiliary models*. For example, we will later be adding a feature that encodes the likelihood of the surface yields as computed by a separate n-gram language model. In sum, it should be clear that feature extraction can easily become an expensive process. The computational overhead associated with extraction, combined with the need to perform repeated cycles of training as outlined above, means that it is absolutely paramount that features are handled in an efficient manner.

At this point it is perhaps worth spending a few remarks on why we are not more concerned with the other aspects involved in experimentation. One round

of experimentation can generally be broken down into two main stages; training and testing. The former corresponds to estimating a model from the training data, while the latter corresponds to evaluating it on test data. However, a prerequisite for both of these steps is that the data is converted to the appropriate feature vector representation. Moreover, in our case, both the estimation and application of the models is handled by the use of external and specialized toolkits, interfaced to [incr tsdb()]. As further described in Chapter 7, we use SVM^{light} (Joachims, 1999) for the SVM modeling and TADM (Malouf, 2002) for MaxEnt modeling. Section 6.2.3 also includes some more details on TADM, which is the toolkit that we have used most extensively for our experiments with discriminative models, and for which the integration into [incr tsdb()] is most mature. Nonetheless, both of these toolkits implement highly optimized routines for the specific numerical optimization tasks. This means that the factors that we can try to further optimize ourselves are related to intermediate data management and feature extraction. In many cases, this is also the actual bottleneck in the experimentation process. To be sure, at least when working with SVMs, also the training process itself can be notoriously slow. But in the case of MaxEnt, estimation is usually quite fast. In both cases the actual application of the final model is also cheap (which essentially amounts to computing the dot product of the parameter vector and the feature vector for each realization). This means that we risk spending most of the time on just managing the data, unless we take care to implement this part efficiently. The improvements we have made in our code with respect to data management and model tuning is what we will be looking at for the remainder of this section. To be sure, [incr tsdb()] did already include rudimentary support for experimentation at the time of commencing work on this thesis. However, a single experiment with a few hundred training instances could easily take many hours to complete, and naïve memory management made it impossible to enlarge the data sets within the given architecture. A complete redesign and re-implementation was therefore necessary. To get a sense of scope, the distributed version of the experimentation environment of [incr tsdb()] currently comprises roughly 7000 lines of Common Lisp code and 400 lines of ANSI C (not counting any externally imported packages).[1]

6.2 Feature Caching

One of the most important changes we have made with respect to the experimentation environment in [incr tsdb()] during the work on this thesis, is the added support of *feature caching*. In short this means that features can be retrieved by

[1] In relation to the experiments reported in this thesis one could also count the collection of local Perl scripts, as used for auxiliary tasks such as preprocessing of corpus data, etc.

simple look-up at the time of running an experiment, instead of being extracted from the treebanks. In previous versions of the code, all the features were extracted freshly for each new experiment that was to be run. Given the grid-search context outlined above, and the extra overhead of cross-validation, this way of setting things up entails a large amount of redundant computation. The redundancy is of course caused by the fact there will usually be a great deal of overlap in feature sets between different experiments. Oftentimes the feature sets will even be identical, in cases where the only difference between consecutive experiments only involves parameters related to the learner. With the improved implementation that is available through the LOGON source tree at the time of writing, the use of feature caching helps us avoid this redundant computation. The cache facility relies on the use of an underlying database (DB) for storing the features. All features are thus extracted *once* in a single pass prior to training and then stored in the DB (i.e. the feature cache). The cache includes all relevant information about features and their values. At the time of running an experiment, the features are then simply looked up in the DB, which is far more efficient than actually extracting them from the treebank. Our first implementation of the feature cache was based on a beta-version of AllegroCache[2], a proprietary object database built on top of the Common Lisp Object System (CLOS). Moreover, the AllegroCache DB is also native to the particular Common Lisp implementation that [incr tsdb()] already makes heavy use of, namely AllegroCL.[3] However, for better efficiency and to avoid remaining challenges with Lisp memory management, we later migrated to the open-source database library Berkeley DB (BDB).[4] Although this requires the addition of a foreign function layer, we found that for our particular purposes it actually offers less cumbersome interfacing in terms of serialization/deserialization of keys and data pairs.

Before a given feature is stored in the cache for the first time, a two-way symbol table records a mapping from its symbolic representation (e.g. as seen in Table 5.3) to a unique numerical identifier. When caching the value of a particular extracted feature, a DB key is then generated on the basis of four things; the numeric identifier of the treebank item; the numeric identifier of the candidate realization within the item; the feature template id; and the template parameters. Indexed on this key then, the value of the given feature for a particular treebanked realization is stored in BDB. In most cases the features take on integer values, but some, such as the feature corresponding to the language model scores, are real-valued.

[2]See http://www.franz.com/products/allegrocache/ for more information.

[3]The developer and vendor of both AllegroCL and AllegroCache is Franz Inc. See http://www.franz.com/ for more information.

[4]See http://www.oracle.com/database/berkeley-db/db/ for further details on Oracle Berkeley DB.

For our purposes, the BDB ability to use blocks of memory corresponding to C structures, provides a direct way of serializing our cache keys and values. With AllegroCache, on the other hand, using its low-level BTree interface, we had to serialize manually, encoding a range of diverse data types as byte sequences.

Of course, not storing the actual symbolic representations makes the DB a lot more compact. But more importantly, this information is not needed by the learners, which use an input format where each feature appears only by a numerical index, together with the corresponding value. The symbolic representations of the features only need to be recovered in the case when a model is to be exported (e.g. to be applied on new data), which is then easily done by reference to the symbol table created during feature extraction.

The feature cache also records miscellaneous count statistics for the features, such as minimum and maximum observed values, the total number of items and realizations that a feature is active for, the number of items for which the feature had a discriminative potential (i.e. it had a different value for a gold and non-gold realization), and so forth. As we shall see later, this information is useful for defining different kinds of frequency cut-offs on features, normalizing feature values, and other operations that can now be done by looking up the required information in the cache instead of computing them from the treebank. The feature cache can also optionally record the similarity between each candidate realization and the gold reference, according to any number of different similarity measures. Some such similarity measures are described in Section 6.3.2 below. In addition to facilitating efficient evaluation, this information can be used for weighting each example prior to training. As described in more detail elsewhere, such similarity scores or weights can be used to induce a relative ranking between the training examples in SVM ranking (see Sections 2.4.2 and 7.3), and also to replace the counts of the frequency-based empirical distribution in MaxEnt (see Section 7.2.7).

The process of populating the feature cache is an expensive operation. For the *Tourist* treebank, using the settings for feature templates described in Section 5.5 above, creating the full feature cache takes roughly two and a half hours, running on a Linux box with a 64-bit 2.2 GHz (Opteron) AMD processor and 32 GB of RAM. This results in a cache that takes up 13.6 GB on disk. However, population of the feature cache is intended as a one-time operation, which ideally only needs to be repeated if there are changes in the treebank or in the definitions of the feature templates themselves. More importantly, the computational cost associated with the creation of the feature cache, is something that in itself should be taken as a direct indication of the savings we later earn at the time of running experiments.

6.2.1 Context Cache

We also implemented a *second layer* of caching, in addition to the feature cache. This layer is somewhat cryptically referred to as the *context cache*, due to some terminological import from one of the modeling toolkits.[5] The additional caching simply amounts to temporarily storing the features that are retrieved from the feature cache so that they can be reused between consecutive experiments if possible. After looking up features in the DB, each treebank item is associated with a file that stores the relevant features in the numerical format required by the learners. It is these intermediate files that constitute the context cache. Now, as noted previously, it is often the case that the only parameters that differ between two given runs are related to the estimation procedure itself. This will certainly always be the case also for the different fold combinations within an n-fold experiment. The context cache lets us take advantage of such cases where the differences between runs do not affect the specific features or feature values that are used.

To give an example, consider the case when we are tuning an estimation parameter such as the variance of the MaxEnt regularizer. If the value of the variance stays fixed across two (or more) consecutive experiments, but the feature configuration changes, we get a *cache miss* with respect to the context cache. We then need to fall back on our first level of caching, i.e. looking up features from the feature cache, recreating the context cache. However, in cases where the feature configuration stays fixed, and only the variance parameter changes, we can simply re-use the information in the context cache. Creating the actual input file to the learner is then simply a matter of concatenating files from the context cache, thus by-passing the need to read from the DB. A prerequisite for this arrangement to be of any use is, of course, that we also take care to set up the grid search in such a way as to always exhaust the specified settings for a given learner-specific parameter before we try to alter the settings that affect the features themselves. In the latter case, we cannot take advantage of this second layer of caching and the context cache needs to be replaced. We see that the sequencing of experiments is an important factor for minimizing the number of cache misses, and the code for carrying out batch-experiments in [incr tsdb()] is designed to automatically perform the search in such a way as to make maximally efficient use of the context cache.

Note that the two levels of caching come with two different levels of indexing. On the level of the feature cache, indexes are formed with respect to the relevant result id (within the item), item id (within the treebank), template id, and template parameters. This index allows us to retrieve the necessary information for a given realization and feature type as stored in the underlying data base. On the level of the context cache, indexes are formed with respect to item ids only. This index

[5] In the terminology of the TADM toolkit, the features that correspond to a given realization constitutes an *event*, and the set of events that correspond to a given semantics constitutes a *context*.

allows us to retrieve all the necessary feature information for a given item (i.e. all the feature vectors for its entire set of candidate realizations) as stored in the file system.

It should be emphasized that the improvements to the modeling facilities we have described here are, of course, not specific to the "direction" of realization ranking. The code has also been successfully applied for large-scale *parse-selection*, for example on the large Japanese Hinoki[6] treebank (Bond et al., 2004). As for June 2007, Hinoki contains 65,424 sentences annotated with the JACY grammar (Siegel & Bender, 2002), comprising a total of 5,255,925 candidate parses. After populating the feature cache for the purpose of parse selection,[7] the resulting symbol table contains 3,310,202 distinct features at a file size of 458 MB. The actual feature cache created[8] for Hinoki has a total size of 106 GB, and is spread over 68 files, one per profile. This last point warrants some further remarks. In order to operate on several smaller treebanks as if they were one, we have implemented support for what we call *virtual profiles*. Recall from Section 5.2 that a "profile" refers to the format used by [incr tsdb()] for representing a treebank internally. Furthermore, in many data sets such as Hinoki or the LOGON Tourist corpus, the treebanks are actually made up of several smaller treebanks, each stored and encoded in its own profile. The notion of a virtual profile means that several such smaller treebank profiles can be associated with each other within the system, allowing us to operate on them as if they were one (although the actual feature caches are stored "locally" as part of the individual profiles). Finally, note that, in addition to the grammatical annotations themselves, a profile can also record many types of performance statistics, all stored in the form of a relational DB. During the work on this thesis we also augmented the inventory of such performance statistics stored in the profiles. New additions include such things as item-level evaluation according to several different scoring metrics (more on those below), per-fold performance (e.g. iterations, running time, sub-sets of items per fold, various evaluation scores), and more.

6.2.2 Example Session

In order to make the description of the experimentation environment a bit more concrete, Figure 6.2 shows some examples of using the corresponding Lisp code

[6] Hinoki is a Redwoods-style treebank based on HPSG and MRS, developed by the NTT Natural Language Research Group. See http://www.kecl.ntt.co.jp/mtg/ for more information.

[7] The following figures refer to parse selection experiments on Hinoki carried out by Stephan Oepen in June 2007.

[8] On a machine with similar specifications as the one described above (64-bit dual-core CPU with 32 GB of RAM), this process took roughly 14 hours.

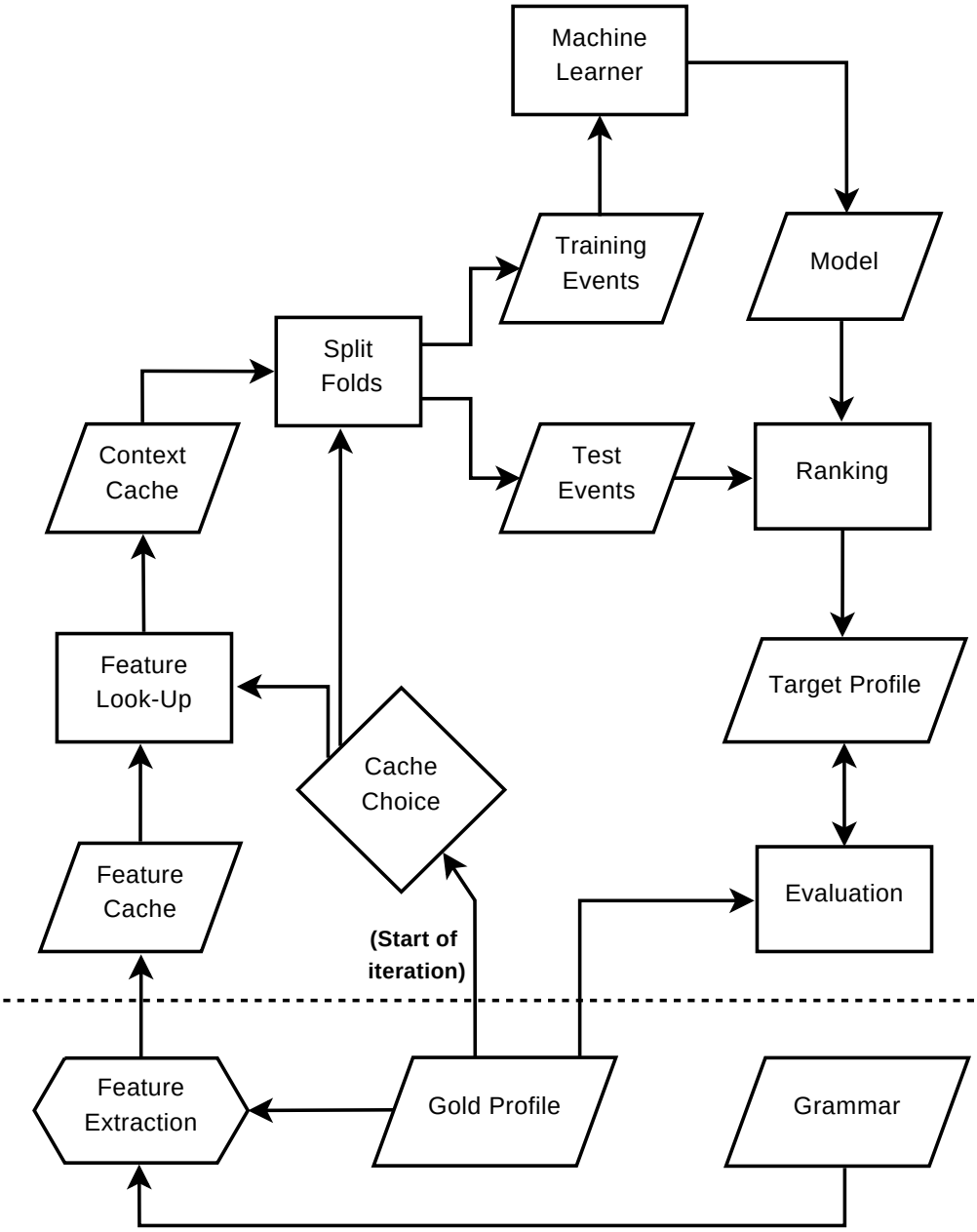


Figure 6.1: Flow of data in an experiment. The chart illustrates the processing pipeline for one iteration in an n-fold training and testing cycle. See the text for further explanation of the elements.

interface (many of the operations are also available via the [incr tsdb()] GUI). In addition, Figure 6.1 shows a flow-chart[9] illustrating the chain of events within a given experiment. We will run through the information provided in these two figures in the following paragraphs, summarizing the main points from the discussion above. Since much of the information in the two figures overlap—providing alternative views on the same processes—we will go through both of them in parallel, explaining each with reference to the other.

Figure 6.1 summarizes the flow of data and events in an experiment. The chart assumes an n-fold experiment, where the depicted cycle would be repeated in each of the n iterations. Note that the boxes below the dotted line represent data resources that remain constant across iterations and experiments. Here we find a gold profile, consisting of possibly several annotated reference treebanks, as well as the (ERG) grammar. The grammar is used at the point of feature extraction in order to re-create the HPSG signs for the derivation trees stored in the profiles. Note that feature extraction itself is also viewed as a preparatory step, resulting in the feature cache DB. Looking at the sample code in Figure 6.2, creation of the feature cache is carried out in step C), assuming that we have already loaded the system and the grammar in an initial step A) and specifying the feature parameters in step B). Note that C) specifies the Tourist generation treebank as the gold profile for which we want to create a feature cache (as further described in Sections 4.4.1 and 5.3.4, the development data used in this thesis is internally dubbed JHPSTG.G).

The goal is then to train and test a model, storing the results within a new target experiment profile. In the sample code in Figure 6.2, the function call in step D) sets off a full grid search over many different parameter configurations, varying the span of n-grams (`:ngram-size`) , the level of grandparenting (`:grandparenting`), and the variance of the prior (`:variance`). However, in Figure 6.1, now moving up above the dotted line, we see a snapshot of the events for a single fold-iteration[10] within one of these experiments. Now, when beginning a new experiment, the first thing that the system needs to decide on is the source of the feature information. As described above, if our feature configuration is unchanged from the previous experiment, we can reuse the information in the context

[9] Brief explanation of the relevant flow-chart semiotics: The straight rectangles denote processes, while the boxes with tilted sides denote data. The diamond square represents a choice point, and the hexagon represent a one-time preparation step.

[10] It is important not to confuse the many levels of iterations that are involved in experimentation. At the highest level, relevant to batch experimentation or grid search on different parameter configurations, we have the iterations that correspond to the completion of a full experiment. Furthermore, in the case of n-fold cross-validation, we sometimes talk about the n iterations across all fold divisions within each experiment. Yet another level corresponds the actual model estimation as carried out by the learner, where there will be an iterative search for the optimal parameters defining the model.

```
;; A) Load the system and the grammar.


;; B) Set the feature parameters.
(let ((*feature-grandparenting* 4)
      (*feature-active-edges-p* t)
      (*feature-ngram-size* 4)
      (*feature-lm-p* nil)
      (*feature-ngram-back-off-p* t)
      (*feature-frequency-threshold* nil))


;; C) Create feature cache for "jhpstg.g".
  (operate-on-profiles (list "jhpstg.g") :task :fc))


;; D) Run batch grid of 10-fold experiments on "jhpstg.g",
;; iterating through several configurations of parameters.
(batch-experiment :type :mem
                  :variance '(nil 1000 100 10 1 1.0e-1)
                  :absolute-tolerance 1.0e-10
                  :source "jhpstg.g"
                  :skeleton "jhpstg"
                  :random-sample-size nil
                  :ngram-size '(0 3)
                  :active-edges-p nil
                  :grandparenting '(0 3)
                  :counts-relevant 1
                  :nfold 10)


;; E) Estimate and export a model.
(let ((*feature-grandparenting* 3)
      (*feature-ngram-size* 3)
      (*feature-lm-p* nil)
      (*maxent-variance* 8e-4)
      (*feature-frequency-threshold* (make-counts :relevant 1)))
  (train "jhpstg.g" "jhpstg.g.mem" :fcp nil))
```

Figure 6.2: An example of using the experimentation code within the LOGON Lisp image, training a MaxEnt model on the Tourist generation treebank. The function calls are further described in the text.

```
;;;
;;; #[MODEL (3921 contexts; 255653 weights)]
;;; (erikve@ar.titan.uio.no; 15-oct-2007 (17:21 h))
;;;

:begin :model 3921.

*feature-grandparenting* := 3.
*feature-use-preterminal-types-p* := yes.
*feature-lexicalization-p* := no.
*feature-constituent-weight* := 0.
*feature-active-edges-p* := no.
*feature-ngram-size* := 3.
*feature-ngram-tag* := type.
*feature-ngram-back-off-p* := yes.
*feature-lm-p* := no.
*feature-frequency-threshold* := {0 0 0 1}.
*feature-random-sample-size* := no.
*maxent-method* := tao_lmvm.
*maxent-iterations* := 5000.
*maxent-relative-tolerance* := 1.0e-10.
*maxent-absolute-tolerance* := 1.0e-10.
*maxent-variance* := 8.0e-4.

:begin :features 255653.

(0) [1 (3) ŝubjh hspec d_-_the_le "the"] 0.181383 {22875 641 22875 81} [0 1]
(1) [1 (2) subjh hspec d_-_the_le "the"] 0.23911 {69967 1048 64305 163} [0 3]
(2) [1 (1) noptcomp mass_count_irule n_pp_mc-of_le] -0.0317733 {15618 333 14898 132} [0 2]
(3) [1 (2) nadj_rr noptcomp mass_count_irule n_pp_mc-of_le] 0.0237271 {4344 72 4344 4} [0 1]
(4) [1 (0) noptcomp mass_count_irule] -0.0237281 {15782 340 15052 132} [0 2]
(5) [1 (1) nadj_rr noptcomp mass_count_irule] 0.0237271 {4384 75 4384 4} [0 1]
(6) [1 (2) hspec nadj_rr noptcomp mass_count_irule] -3.11086e-4 {880 33 880 1} [0 1]
(7) [1 (3) hspec nadj_rr hcomp p_np_i_le "between"] 0.042414 {965 52 965 5} [0 1]
```

Figure 6.3: The first lines of the exported model file (‘*jhpstg.g.mem*’) produced in step D) of Figure 6.2. The header specifies the state of the relevant experiment variables during training. The rest of the file then lists information about the features. See the text for further explanation of the various fields.

cache (as will always be the case for folds within the same experiment, of course, beyond the first iteration). However, if anything has changed with respect to the features we want to use, we need to re-create the context cache and look up the relevant feature information from the feature cache. Then, based on the numerically encoded features in the context cache, we create the so-called *event files* that are passed on as input to the machine learner. In our case the learner will typically be TADM, resulting in a MaxEnt model. In the call to `batch-experiment` in step C) of Figure 6.2, this is indicated by the keyword option `:type :mem`. However, many other learners using a similar vector-based data representation could be plugged in instead. For instance, in the current implementation, specifying `:type :svm` instead will result in an SVM ranker, trained using the SVM^{light} toolkit. In any case, two such event files are created; one for the items in the training folds and one for the test fold. After we have estimated the parameters for the training events, the resulting model is applied for scoring the realizations represented by the test events. The corresponding ranking is then evaluated with respect to the reference data, and the results are stored within the experiment profile. In an n-fold cross-validation set-up, this cycle is repeated until we have tested on all n folds. Of course, the procedure is more or less the same if we want to train and export a model in a single pass, and then go on to test on a separate treebank. In that case the feature information must be extracted for the two profiles separately, and the test and training events will represent different profiles instead of different folds. The function call in step D) in Figure 6.2 initiates training of a MaxEnt model in one pass over the full Tourist generation treebank, exporting the resulting model to the file *'jhpstg.g.mem'*. Some of the modeling options that are specified here will become clearer later, such as the use of frequency-based relevance cutoffs, which we describe in Section 7.2.3. The first few lines of the exported model file are displayed in Figure 6.3 below. The top section of the file contains some meta-information, essentially specifying the complete state of the relevant variable environment which was in effect while training the model. We see that the model is trained for 3921 contexts (i.e. treebank items), with a total number of $255,653$ weights. Of course, this number also corresponds to the total number of active features in the model, as selected from the total pool of $1,182,587$ features in the underlying feature cache. The rest of the file lists information about the features themselves.

For a given line of feature information, the first number corresponds to its unique index in the model. The next field, enclosed by square brackets, gives the symbolic representation of the feature, as we have seen before in Section 5.3. It consists of the template id (e.g. '1' identifies a feature recording local derivational subtrees), the template parameters (e.g. '3', given feature type '1', indicates that we include three levels of grandparenting), and finally the elements of the symbolic instantiation of the feature as extracted from the treebank. The number

in the next field is the model parameter, i.e. the feature weight. Next, in curly brackets, we find four different types of feature counts, corresponding to the total sum of values for the feature, the number of distinct items (i.e. "contexts") it has occurred in, the number of realizations (i.e. "events"), and finally the number of so-called relevant items, i.e. the number of items for which the feature took on a different value for two or more distinct realizations. The last field on the line for each feature, again in square brackets, is the minimum and maximum observed value for the feature (where the value will typically be its frequency count). This information is useful, for instance, for the purpose of normalizing feature values, which can be relevant for certain learners.

As previously noted, most of the above discussion pertains to the extraction of data for training and testing of models, not the procedures for the actual model estimation itself. For model estimation we have integrated into [incr tsdb()] several freely available third-party software toolkits. For the estimation of n-gram language models we use the Carnegie Mellon Statistical Language Modeling (CMU SLM; v. 2.05) Toolkit[11] (Clarkson & Rosenfeld, 1997), and for building the SVM rankers we use the SVM^{light} toolkit[12] developed by Joachims (1999). However, the toolkit that has played the most important role in the development of our treebank models, in the directions of both parsing and generation, is the *Toolkit for Advanced Discriminative Modeling* (TADM; Malouf, 2002). TADM is the default discriminative learner in our experimentation environment, providing highly efficient tools for the training of conditional maximum entropy models. This is the modeling framework that we have used most extensively in relation to the kind of search problems discussed above, and the next section therefore provides a brief presentation of the TADM toolkit.

6.2.3 TADM

There exist a number of freely available software packages estimating maximum entropy models. However, our [incr tsdb()] experimentation environment is by now fairly well integrated with the *Toolkit for Advanced Discriminative Modeling* (TADM[13]) which is based on the open-source `estimate` package by Rob Malouf (Malouf, 2002). TADM, in turn, builds on the general-purpose and open-source software libraries for numerical optimization PETSc[14] and TAO.[15]

There are several attractive properties associated with Malouf's `C++` MaxEnt

[11] For more information, see `http://www.speech.cs.cmu.edu/SLM/toolkit.html`.

[12] See `http://svmlight.joachims.org/` for more information on SVM^{light}.

[13] See `http://tadm.sourceforge.net/` for more information on TADM.

[14] Portable Extensible Toolkit for Scientific Computation, see `http://www.mcs.anl.gov/petsc`.

[15] Toolkit for Advanced Optimization, see `http://www.mcs.anl.gov/tao`.

implementation which makes it a particularly suitable choice for parameter estimation for our purposes. While many MaxEnt implementations are limited to boolean or integer-valued features, TADM allows real-valued features. This is an important property if one wants to take full advantage of the flexibility of the MaxEnt framework, for example by integrating other models as features. While boolean feature values will only allow one to indicate whether a given feature is present or not, integer-valued features at least allow one to work with *counts* of feature occurrences. However, if one wants to integrate externally defined models as features (as we later do in Section 7.2.6), having only integer-valued features will often not be sufficient. Although an external model itself may represent all kinds of different things, like some metric indicating similarity or deviation, a statistical hypothesis test, or a probability or log-probability, its output will typically take the form of a real number. With TADM however, we can define feature values to correspond to, for example, the log-probabilities or perplexities computed by an n-gram language model.

As explained in Section 2.3.3, a well-known problem with MaxEnt estimation is the tendency to overtrain when working with a large feature space and only small amounts of training data—a typical scenario for NLP learning problems. It is therefore recommended to include some kind of regularization during training, and there are currently two such options available in TADM, by specifying either a Laplacian or Gaussian prior distribution on the feature weights (sometimes known as L_1 and L_2 regularization respectively).

The TADM toolkit also offers a range of algorithms for the actual parameter estimation, including *steepest ascent*, *conjugate gradient* (*positive Polak-Ribière*), *generalized iterative scaling*, *improved iterative scaling* and the *limited memory variable metric* algorithm. The various optimization methods are further described and compared by Malouf (2002), who furthermore shows empirical results in favor of using variable metric methods. The limited memory variable metric (LMVM) is the default in TADM and this is also the method used for all MaxEnt experiments described in this thesis. It is perhaps worth noting however, that the main differences between the alternative estimation techniques "only" concern performance characteristics during training (in terms of convergence properties, total running time, number of function evaluations, etc.), not the accuracy of the resulting model. Furthermore, not all algorithms are implemented to support real-valued features, but the default LMVM method does. Other than this, the inner workings and characteristics of the different optimization procedures are not of relevance to our current purposes, and the interested reader is referred to Malouf (2002) for further details.

In terms of extra functionality beyond the parameter estimation itself, the TADM toolkit is fairly bare-boned. Its input format is based on numeric feature indexes paired with their numeric values, and the output is a file listing the corre-

sponding λ-parameters defining the model. The actual feature extraction, indexing, and related tasks, are all done from our extensions to [incr tsdb()]. The same holds for the feature caching, organization of folds and cross-validation, tuning of the prior, and so on, as described above. Testing and evaluation of the resulting models, which is the topic of the sections that follow, is also done within our own [incr tsdb()] embedded experimentation environment.

This section has presented some of the nuts and bolts in our software environment for ML experimentation on treebank data. We have had a detailed look at several data management issues, including the implementation of feature caching—a facility enabling us to do large-scale experimentation with feature-based models on large data sets—and also presented the toolkit most extensively used for model estimation in our case, TADM. Many other improvements to the overall modeling facilities in [incr tsdb()] have also been carried out during the work on this thesis, some of which we describe when we shortly turn to the topic of evaluation and testing. Much of the associated code work has been based on periods of intensive peer-programming, representing a collaborative effort between the author and Stephan Oepen, the original developer of [incr tsdb()] and the supervisor of this thesis.

Before rounding off this section, it is perhaps worth noting that, so far, the rounds of "grid" search or parameter tuning described above have only been parallelized at the crudest possible level, viz. by simultaneously running rounds of experiments at several machines. However, the Linux servers of the Logic and Natural Languages research group (LNS; *Logikk og Naturlige Språk*) at the University of Oslo are now part of a larger cluster administered by the University's Scientific Computing Group. This means that proper parallelization of experimentation across many compute nodes is something that we plan to further explore in the future.

In the remaining part of this chapter will we still be occupied with additions to the experimentation environment, but we now move on to deal with the issue of *evaluation*. In the following section we look at several of the evaluation measures that we have integrated in the system in order to score and compare the performance of our different realization rankers. After that, we look at the use of hypothesis tests for assessing the statistical significance of any observed differences in the corresponding scores.

6.3 Evaluation Measures

When we turn to describe the outcome of the held-out testing in Chapter 8, we present test results that have been produced by a group of external and anonymous

human evaluators. With the help of Professor Emily M. Bender at the University of Washington (UW), seven master students in the computational linguistics programme were recruited to judge the relative quality of alternative generator outputs. As described in more detail later, the judges were given a questionnaire with a set of randomly selected test items, and instructed to assign a relative rank order to sets of candidate realizations as chosen by different models. Of course, this kind of manual evaluation is an invaluable help when it comes to assessing the relative performance of different systems. However, manual evaluation is obviously also a very time-consuming and labor-intensive process. Although it can be argued that it would be ideal to always have system output evaluated through manual human inspection, most of the time this is just not a realistic option in practice. Throughout the development cycles we typically need to perform many rounds of contrastive experiments and use relatively large data sets to make sure that our results are representative. It is common practice therefore to instead rely on various metrics for *automatic* evaluation. In addition to the reduced cost in terms of both time and labor, automatic evaluation metrics have the added benefit of being non-subjective and easier to standardize.

The automatic evaluation of our realization rankers is done with respect to two main types of measures; *exact match accuracy* and various notions of *string similarity*. In this section we discuss several such possible evaluation metrics, as well as some of the statistical significance test that we use when comparing the evaluation scores of different models. The issue of manual evaluation by human judges is deferred to Chapter 8, where we report test results for our models on held-out data.

6.3.1 Exact Match Accuracy

The exact match measure is simply the percentage of times that the top-ranked sentence (i.e. the output sentence) is identical with the reference or gold sentence in the test data. In other words, after a model has been applied to all possible paraphrases in a generation treebank, we count the number of times that the model assigns the best score to (one of) the string(s) marked as preferred. In the case of ties, i.e. if several realizations are given top rank by the model, the score is discounted proportionally. Sometimes we also report exact match accuracy for n-best lists, typically with $n = 5$. Other papers on statistical generation that include the exact match accuracy as an evaluation measure include Malouf (2000), Langkilde (2002), White (2004), and Cahill et al. (2007).

Note that, when highlighting the improvement of one ranker over another in terms of exact match accuracy, we will sometimes also refer to the relative *reduction in error rate*. With the accuracy defined as above, the error rate is of course naturally given as error $= 100 -$ accuracy. The relative percent-wise reduction in

error rate of system A with respect to system B is then given as $100 \times \frac{\text{error}^B - \text{error}^A}{\text{error}^B}$.

The simple measure of exact match accuracy offers a very intuitive and transparent evaluation metric. Still, it is arguably also an overly strict evaluation measure in the setting of natural language generation. In cases where a top-ranked candidate does not exactly match the reference, the candidate might still be judged to be appropriate or inappropriate to a varying degree. It seems both unfair and potentially uninformative to only give credit in cases of exact match, i.e. on an all or nothing basis. For this reason it is common to also include some kind of *similarity-based* evaluation measure. One prominent such measure is *BLEU*, an n-gram-based evaluation metric which is by now well-established in the field of MT. The evaluation settings for NLG and MT are quite similar in some respects, and metrics such as BLEU (which do not depend directly on the source but rather on a set of target reference sentences) are sometimes also adopted for evaluating generation systems. Below we look at several such measures based on various notions of string similarity, such as BLEU, NEVA, and *word accuracy* (WA).

6.3.2 String Similarity Measures

BLEU The similarity-based BLEU measure has gained a well-established role as an evaluation metric in MT, and is modeled after the *word error rate* measure which is commonly used in speech recognition. Assuming a set of one or more target references, the score is computed as a weighted average of the n-gram precision of the selected candidate realization with respect to the reference(s), for all $1 \leq n \leq 4$. For completeness the definition of BLEU[16] as given in (Papineni et al., 2002) is shown in Equation (6.1), where c and r are the number of words in the candidate and reference sentences respectively.

$$\text{BLEU} = \text{BP} \cdot \exp\left(\sum_{n=1}^{N} \frac{1}{N} \log p_n\right) \tag{6.1}$$

To avoid a bias towards shorter translations, the term BP imposes an exponential *brevity penality* when the candidate is shorter than the reference(s) and is computed as

$$\text{BP} = \begin{cases} 1 & \text{if } c > r \\ \exp(1 - \frac{r}{c}) & \text{if } c \leq r \end{cases} \tag{6.2}$$

[16]For computing the actual scores we use the implementation found in the *mteval toolkit* supplied by NIST.

p_n is a modified n-gram precision score computed for each $1 \leq n \leq N$ with $N = 4$ as

$$p_n = \frac{\sum_{C \in Candidates} \sum_{n\text{-}gram \in C} Count_{clip}(n\text{-}gram)}{\sum_{C \in Candidates} \sum_{n\text{-}gram \in C} Count(n\text{-}gram)} \tag{6.3}$$

where $Count$ simply counts the maximum number of times an n-gram occurs in a candidate, while $Count_{clip}$ limits this total count to the maximum count found for this n-gram in any single reference. BLEU is then defined as a weighted average of the logarithm of these n-gram precision scores, where each score is given equal weight ($1/N$). Papineni et al. (2002) note that this weighting, corresponding to the *geometric mean* of the precision scores, means that BLEU is more sensitive to longer n-grams. Although the metric has a constant range in $[0, 1]$, having more available reference sentences will generally mean a higher score.

One problem with the BLEU score is that it is intended to be computed at the level of a full corpus. However, working in the setting of single-sentence realization, we would like to have a similarity measure that allows us to directly asses the performance on the level of individual sentences. Instead of computing the brevity penalty and weighted average precision mean over an entire test corpus, the naïve solution would be to compute the same quantities instead at the sentence level, and use the corresponding average sentence scores for system level evaluation. As it turns out, however, the BLEU measure is not straightforwardly applicable as a sentence-level score. For instance, Forsbom (2003) notes some difficulties that arise in cases when the candidate sentence is shorter than 4 (or N) words or does not contain any matching 4-grams: The standard formulation of BLEU, as shown in Equation (6.1), is undefined (or, in practice, defined to be 0) in such cases (because of division by zero or log of zero respectively).

NEVA The context in which Forsbom (2003) addresses these problems is when using BLEU for evaluating sentence-level decisions in MT for diagnostic purposes and when only using a single reference sentence. However, the same issues arise in our setting. Now, Forsbom (2003) proposes the NEVA measure as an alternative reformulation of BLEU that avoids these problems. NEVA is computed as the arithmetic mean of the raw n-gram precision scores (i.e. leaving out the *exp* and *log*s in BLEU), and precision is only computed for n-grams of sizes lower than or equal to the length c of the candidate, i.e. $N = \min(c, 4)$. For the final rounds of held-out testing in this chapter, the evaluation will also include averaged sentence-level

NEVA scores for the different rankers.[17]

$$\text{NEVA} = \text{BP} \cdot \left(\sum_{n=1}^{N} \frac{1}{N} p_n \right) \tag{6.4}$$

where BP is defined as before and

$$N = \begin{cases} 4 & \text{if } c \geq 4 \\ c & \text{if } c < 4 \end{cases}$$

Although still within the same interval $[0, 1]$, Forsbom (2003) notes that NEVA generally gives a higher scoring level than the NIST implementation of BLEU. That tendency is also observed for our data sets, and it is particularly pronounced for the shorter sentences. Given the problems with applying BLEU on short sentences as pointed out above, this is not surprising. Of course, the scoring-level in itself is not important for evaluation purposes. What is of interest to us here is really only the relative difference in scores of the models that we want to contrast and compare. Indeed, the actual meaning of the scores returned by these precision-based similarity measures may often be somewhat opaque and difficult to intuitively interpret in isolation, at least when compared to the simple measure of exact match accuracy. Nonetheless, at least they allow us to incorporate a notion of *gradedness* when matching a generated realization and a reference.

As noted above, the original formulation of BLEU was modeled after the *word error rate* measure. Another variant of this measure, expressing *closeness* rather than *distance*, is *word accuracy*, which is what we turn to describe next.

WA The word accuracy measure is based on the so-called *Levensthein distance* between a candidate string and a reference, also known as *edit distance*. This is given by the minimum number of deletions, substitutions and insertions of words that are required to transform one string into another. If we let d, s and i represent the number of necessary deletions, substitutions and insertions respectively, and let l be the length of the reference, then WA is defined as

$$\text{WA} = 1 - \frac{d + s + i}{l} \tag{6.5}$$

As pointed out by Forsbom (2003), the underlying idea of the measure is to approximate the efforts of a post-editor, in that it expresses the minimum number of basic edit operations needed to rewrite the candidate translation or realization into the reference.

[17] Many thanks to Eva Forsbom for supplying the code for her implementation.

WA has long been a widely used evaluation measure within the field of automatic speech recognition, while Alshawi, Bangalore, and Douglas (1998) are often credited for introducing the metric for the evaluation of MT systems. More recently, WA (sometimes also referred to as *simple string accuracy*) has also gained a fairly wide-spread use within the field of NLG, and is for instance used in Bangalore and Rambow (2000), Corston-Oliver et al. (2002), Langkilde (2002), Belz (2005), and Cahill and Genabith (2006). WA will also figure as one of the main evaluation measures in this chapter. Note that, in cases where an item has several references, we always use the similarity score of the *closest* match. The same holds when we evaluate by NEVA, and also when computing scores for n-best lists. In other words, for both NEVA and WA, in case where more than one reference is available, we use the maximum score.

It is worth pointing out that WA is not entirely unproblematic: Bangalore, Rambow, and Whittaker (2000) note that it can be overly strict as an evaluation measure for NLG since movement operations will be doubly penalized as deletion and insertion. Furthermore, not all edit operations will be equally important. Nonetheless, for comparison of different rankers we believe that the simplicity and intuitive design of WA still makes it an appealing evaluation measure, complemented with exact match accuracy and NEVA. Finally, it is worth noting that, although we will not be using BLEU for evaluation in this thesis, our own preliminary experiments seem to agree with the observations of both Forsbom (2003) and Langkilde (2002), that the system-level rankings imposed by the WA and BLEU scores seem to closely agree with each other.

6.3.3 The Random Choice Baseline

In the next chapter we will describe and contrast the performance of several statistical realization rankers. However, we are interested in knowing not only how good the respective models perform relative to each other, but also how well they perform relative to the difficulty of the task itself. For this we need some way of assessing the difficulty of the ranking problem.

As described in Section 5.3.4 there are essentially two factors that determine the difficulty of the ranking task in quantitative terms. One is, of course, the average number of candidate realizations per item. This figure directly reflects the non-determinism of the generation process. But another factor is the number of candidates that are identified as gold during the labeling step, as described in Section 5.3.2. Of course, it would not matter how horrendously huge the average number of per-item realizations was, if two-thirds of them were labeled as gold anyway.

On the basis of the number of candidate realizations and references per-item, we can compute a *random choice baseline* for the exact match accuracy. This

is essentially the expected accuracy obtained if we were to just blindly pick a realization at random for each item in the data set. The baseline is computed as shown in Equation (6.6): For each item we divide the number of gold-labeled realizations (g) on the total number of available candidates (c). After summing these per-item gold-to-candidate ratios, we finally divide on the total number of items (n) to obtain a global random choice baseline.

$$\text{baseline} = \frac{1}{n} \sum_{i=1}^{n} \frac{g_i}{c_i} \tag{6.6}$$

Although there are ways of computing similar baseline figures also for the string similarity metrics, for example on the basis of repeated random sampling, we find that the random choice figure for the exact match accuracy is more intuitively meaningful and precise, and hence we choose to only compute baseline figures for this evaluation measure.

In the next section we turn to look at the issue of *significance testing* when comparing the evaluation results for two different rankers. However, before we leave the topic of evaluation measures, it is worth noting that we have not included a metric that takes into account the full rank order as such. Although we often refer to our task as that of "realization ranking", we might just as well have described it as "realization selection". In an actual NLG application setting, what we care about is the quality of the actual candidate that is chosen as the final output from the system. What we try to evaluate using the measures described above is exactly that. To some degree, we *do* actually take into account some of the other candidates further down the list when we compute evaluation scores with respect to n-best lists. Still, this is just to get some rough idea of "how far off" we were in terms of placing the reference candidate at the top of the list. This situation can be contrasted to a task such as information retrieval or document search. Here the rank order for the retrieved documents is indeed important, at least for the n first candidates on the list. The list of documents presented to the user will contain several candidates that are potentially relevant to the given query, and the rank order should ideally reflect the corresponding degree of relevance. However, for the purpose of statistical sentence selection for NLG, our interest is mainly tied to the *top-ranked* realization, not the full rank order of all the other competing realizations. This is typically also the situation when evaluating (re)ranking methods in relation to statistical parsing or machine translation.

Note that, another reason for why we also include evaluation in terms of n-best lists, is that this is actually relevant for subsequent of end-to-end reranking of translations when the generator is used within the LOGON MT system. The generator then delivers n-best lists of results (although with a fairly high value of n, e.g. $n = 50$) which are finally reranked using a global discriminative log-linear model (Oepen et al., 2007). This is further described in Chapter 9.

6.4 Hypothesis Testing

Whether the rankers are evaluated according to exact match accuracy, WA, or NEVA, we will occasionally want to check whether an observed difference in performance can be considered *statistically significant* or not by applying some kind of *statistical hypothesis tests*. While the previous section described various methods for quantifying the performance of our models, this section describes methods for quantifying the probability that an observed *difference* in this performance for a pair of models is due to chance. Although it is common-place in the literature on empirically oriented NLP to include significance testing of results, the details of how the tests are actually carried out is often left out of the discussion. This can make it difficult for others to make independent judgments about the reported results, and it also makes it hard to establish common practices. Instead of just citing test statistics and p-values, we think it is in order to here also devote some space to discuss how we actually set up the tests.

We will be considering two types of hypothesis tests, the *sign test* and the *Wilcoxon signed-rank test* (Wilcoxon, 1945). Before we delve into the details of the specific tests, let us give a general outline of how the tests are applied. When comparing the performance of two different rankers A and B, we will only work with so-called *paired* formulations of the tests, which means that we are looking at pairs of scores across the same sample of items. Since our rankers are tested on the same data sets, we can compute the differences in scores across matched pairs of test items, $\{(a_1, b_1), \ldots, (a_m, b_m)\}$. For example, a given pair a_i and b_i can correspond to the accuracy or WA scores on the ith test item for the two respective rankers A and B, and we then want to compute their difference $a_i - b_i$. The *null hypothesis* H_0 that we want to test is that any variation in this set of paired differences is due to pure chance alone. The alternative hypothesis would then be that an observed difference represents a real effect. As is always the case in hypothesis testing, some kind of test statistic is then computed to asses the correctness of the null hypothesis. Whether the given null hypothesis is rejected or not depends on the computed *p-value*, which can simply be seen as the probability that a random variable would take on a value greater than or equal to the observed value strictly by chance.[18] In all cases reported here, we reject the null hypothesis if the p-value is smaller than or equal to a pre-specified significance level of $\alpha = 0.05$.

Note also that we will only use *two-tailed* formulations of the tests. The name refers to the tails of the sampling distribution, and in a two-tailed test means that

[18] When computing the p-values for the various test statistics described here we use the open-source Common Lisp statistics package developed by Larry Hunter at the Computational Bioscience Program at the University of Colorado. For more information see http://compbio.uchsc.edu/Hunter_lab/Hunter/.

H_0 will be rejected when the value of the relevant test statistic is either sufficiently small *or* sufficiently large. This is in contrast to *one-tailed* tests, where some direction of deviance is already assumed before the test is applied. Furthermore, both the sign test and the Wilcoxon test are so-called *non-parametric* tests, which means that they do not make any distributional assumptions about the population from which the data has been sampled. This is in contrast to tests such as the t-test, which would require that the differences in scores would approximately follow the normal distribution. Using distribution-free tests, we do not need to commit to such assumptions, other than that the individual members of the sample are independent of one another (of course, the two samples themselves are not independent, as they they are taken to be paired).

Although there are many advantages with using non-parametric tests, one of the disadvantages is that they generally have less power or sensitivity (i.e. less chance of detecting a real effect) than their parametric counterparts, particularly when used with small samples. The sign test in particular is a rather crude and insensitive test, but at the same time one of the most convincing ones. Although the sign test and the Wilcoxon signed-rank test resemble each other in scope, the latter is much more sensitive. For large samples the signed-rank test has been shown to be almost as sensitive as the t-test, and for small samples with unknown distributions even more sensitive than the t-test. Note that, when applying these tests on paired samples of per-item scores, our samples will sometimes be large enough that the normality assumptions of the t-test would perhaps be warranted. However, we sometimes also want to apply the tests to much smaller samples of aggregated scores, such as per-fold averages across ten-fold cross-validation where it is safer to stick with the more robust alternative of non-parametric tests, albeit at the expense of some loss in sensitivity. If anything, this only means that we are more conservative in our judgments about significance.

6.4.1 The Sign Test

Lets us start by considering the evaluation scores computed according to the measure of exact match accuracy. Recall that, for this evaluation measure we simply check for each item whether or not the top ranked candidate exactly matches the reference or not. We then count the hits (=1) and misses (=0) and compute the overall percentage.

As said, when comparing the performance of two different rankers we will be looking at *paired differences* in scores. We will furthermore be disregarding cases where the difference is zero. In the case of the "hard" exact match measurements, the differences will then typically be either $+1$ or -1.[19] Intuitively it makes sense

[19] Strictly speaking, this is not necessarily always the case since, as noted in Section 6.3.1, the

to view these differences as a sequence of n *Bernoulli trials*, i.e. random variables that can have either of two possible outcomes; *success* or *failure*. Let us assume that we have some hypothesis for what the true probability p of success is for each n trial. The number of successes $K = k$ in such a sequence of n independent trials is then distributed according to the *binomial distribution*,[20] with mean $\mu = np$ and standard deviation $\sigma = \sqrt{np(1-p)}$.

Moreover, such cases with two possible outcomes provide exactly the setting where the so-called *binomial test* is suitable (see, for example, Abdi, 2007). The question that the binomial test answers is, given the hypothesized true probability of success, how likely are we to find results that deviate as far, or further, from this prediction. Now, the *sign test* is a special case of the binomial test where we hypothesize that the two outcomes have *equal* probabilities. If the null hypothesis holds, then the probability is $p = 1/2$ that either realization ranker will have better performance.

Let us now consider how we can apply the sign test to the scores of our realization rankers. Given two different rankers A and B, let $\{(a_1, b_1), \ldots, (a_m, b_m)\}$ represent pairs of per-item exact match scores across some given test set of m items. Let p represent the probability of A scoring higher than B. Our *null hypothesis* H_0 is that $p = 0.5$, meaning that there is an equal chance for each paired item that a_i is higher than b_i or the other way around. Note that we do not assume any directionality in our null hypothesis, and so a two-tailed version of the test is performed. Now, the first step is to compute the difference between the paired scores $a_i - b_i$. Cases where the difference between the two rankers is null are disregarded, resulting in a possibly reduced sample of n item pairs. Let n^+ correspond to the number of times that A gets a higher score than B, and n^- for the other way around (so that $n = n^+ + n^-$). The test statistic is then defined as the smaller of these to sums, $k = \min(n^+, n^-)$. Finally, to find the (two-tailed) p-value we need to compute the so-called *cumulative probability*, which according

accuracy scores are sometimes discounted in the event of ties. Nonetheless, when recording the difference for a given item pair we only note whether or not one score is larger than the other.

[20] Let the random variable K be distributed according to a binomial distribution with parameters p and n, denoting the probability of success and the number of trials respectively. Now, given n trials, the probability that the number of successes is (exactly) $K = k$, is computed as

$$f(k; n, p) = \binom{n}{k} p^k (1-p)^{n-k} \tag{6.7}$$

for $0 \leq k \leq n$. The so-called *binomial coefficient* $\binom{n}{k}$ is computed as $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ where $x!$ is the factorial of x (defined as the product of all positive integers less than or equal to x). The binomial coefficient $\binom{n}{k}$ is often read as n *choose* k, and is an expression for the number of ways of picking k unordered outcomes from n possibilities.

to the binomial distribution is defined as

$$f_c(k; n, p) = \sum_{i=0}^{k} \binom{n}{i} p^i (1-p)^{n-i} \tag{6.8}$$

In terms of the Bernoulli trials, the corresponding two-tailed p-value (given by simply doubling the one-tailed value; $2 \times f_c(k; n, p)$) expresses the chance of observing either n^+ or fewer successes, or n^- or more successes, in n trials. If the p-value is smaller than or equal to the significance level $\alpha = 0.05$, we reject the null hypothesis that each ranker has an equal chance of finding a match where the other does not.

Note that, in cases where n is sufficiently large, say $n > 30$, the binomial converges to the normal distribution. Moreover, computing the binomial coefficients can be expensive for large samples, and in such cases the p-value is therefore usually found by instead computing the *Z-score* as the test statistic and then finding the corresponding probability according to the standard normal distribution (Abdi, 2007). The Z-score of an observation $K = k$ is defined as

$$Z = \frac{k - \mu}{\sigma} \tag{6.9}$$

The Z-score, also known as the *standard score*, is expressed in the units of the standard deviation and it represents the distance between a value and the population mean. The population parameters are defined as described above, which according to the null hypothesis of the sign test (i.e. $p = 0.5$) amounts to $\mu = n/2$ and $\sigma = \sqrt{n/4}$.

This section has described the use of the sign test for assessing the statistical significance of differences in the exact match accuracy of different models. In the context of comparing different MT systems, Collins, Koehn, and Kučerová (2005) propose using the sign test for testing differences in BLEU scores. Recall from Section 6.3.2 that the BLEU score is defined relative to an entire corpus. This means that it can be tricky to apply statistical significance tests directly by comparing sentence-level scores. To get around this, Collins et al. (2005) try to approximate per-sentence scores by a method of leave-one-out testing, for each item in the test corpus computing the difference between the corpus-level scores with and without the item included. In our case, we do not have this problem. The string-similarity scores such as NEVA and WA are defined at the sentence-level, and we can apply significance tests straightforwardly using the set-up outlined above. However, although we chose to use the sign test for computing p-values for paired sets of exact match figures, we do not believe that this is necessarily the best way to test the differences in string-similarity values. When comparing the scores of two different rankers, the sign test can be said to focus on *how*

often one ranker does better than the other. This perspective works fine for the coarse exact match accuracy measure, which is based on dichotomous right-or-wrong judgments. Recall, however, that in the discussion of why we chose to also include similarity-based evaluation measures such as WA and NEVA, we argued that we would like to capture a notion of gradedness, since different candidate realizations can match a reference to varying degrees. In the context of these measures it makes sense to look not only at *how often* one ranker outperforms another, but by *how much*. The *Wilcoxon signed-rank test* that we present next is designed to be sensitive also to these types of effects.

6.4.2 Wilcoxon Signed-Rank Test

While the sign test only considers the *signs* of the differences, the *Wilcoxon matched-pairs signed-rank test* (Wilcoxon, 1945) factors in the *size* of the differences as well. As said, in the case of the exact match measure, the sizes of the differences are hardly an interesting aspect since the variables are effectively close to dichotomous. In the case of properly continuous valued measures such as WA and NEVA, it would seem like throwing away potentially useful information if the actual magnitudes of the paired differences are not taken into account.

Recall that, in the case of the sign test, the null hypothesis that we set out to test is the assumption that each paired score for any given item has an equal probability of being higher than the other. However, the sign test can equivalently be formulated as the hypothesis that the median difference is zero, since an equal number of cases are then expected to fall above and below this figure. Similarly, the Wilcoxon signed-rank test is used to test the median difference in paired data. The null hypothesis H_0 is that the differences ($d = a_i - b_i$) between the members of each pair has median value of zero. However, in contrast to the sign test, the Wilcoxon signed-rank test can also take the *magnitude* of the differences into account, and is therefore more meaningful for (paired) values that are measured on an ordinal or interval scale. In this way it is similar to the commonly used paired t-test, and indeed it is often regarded as the non-parametric equivalent of this test. While the t-test assumes that the differences are normally distributed, the Wilcoxon signed-rank test comes without this assumption since it only considers the rank order of the differences and not the actual values of the differences themselves. It only assumes that the differences themselves are symmetrically distributed.

The Wilcoxon signed-rank test is carried out as follows. First we compute all the pairwise differences between the m matched pairs $\{(a_1, b_1), \ldots, (a_m, b_m)\}$. The next step is then to rank all the differences according to their absolute value, $|a_i - b_i|$. In other words, each difference is assigned a rank according to its unsigned value. As for the sign test, any difference that corresponds to the assumed

center point (zero in our case) is simply ignored, resulting in a possible reduced sample of n differences. In the case of ties, the corresponding ranks are simply set to the average of the ranks that they would otherwise receive. We now sum the ranks of all the differences with the same sign. Let W^+ be the sum of all the ranks associated with the positive differences $(a_i - b_i > 0)$, and let W^- be the sum of all the ranks associated with the negative differences $(a_i - b_i < 0)$. If H_0 was indeed true, we would expect W^+ and W^- to be approximately equal in size. Let the test statistic W be defined as the smaller of these two sums, $W = \min(W^+, W^-)$. W can then be compared to a table of all possible distributions of ranks to check if the p-value is below α. Assuming that we are sampling data points from a population with a median value as that hypothesized by H_0, the p-value for the signed-rank test tells us the probability of randomly picking n points and finding a median as far, or further, away from this as the observed value.

For small values of n (typically $n \leq 30$), the critical values associated with this test statistic are tabulated. However, for larger samples the test statistic approximates the normal distribution with mean $\mu_W = N(N+1)/4$ and standard deviation $\sigma_W = \sqrt{n(n+1)(2n+1)/24}$, and the p-value can then be calculated using the Z-score in a similar fashion as for the sign test and as defined in Equation (6.9) (Siegel & Castellan, 1988).

In this chapter we have reviewed some of the challenges posed by working with complex feature sets in large-scale machine learning. We have also reviewed some of the changes we have implemented in our experimentation environment in order to efficiently deal with these issues, especially in relation to data management. Finally we discussed the various scoring schemes we use for evaluation, as well as some of the statistical tests we apply in order to asses the importance of any observed differences in these scores. In the next chapter we finally put all of this machinery to use in practice, as we move on to describe the development of the various models and evaluate their performance as realization rankers. Note that, while the next chapter only report preliminary evaluation results for the development set, Chapter 8 contains the evaluation results for the held-out test data.

Chapter 7

Developing the Models

In this chapter we will be concerned with the actual training and development of the different realization rankers, as well as their performance on held-out test data. We start out by describing the development of the n-gram LMs in Section 7.1. We then move on to describe the MaxEnt models in Section 7.2, including a model that incorporates the LM scores as a separate feature, before finally developing the SVM rankers in Section 7.3. Preliminary evaluation results on the development sets are presented for each model type in turn as we proceed, and the results for the development are summarized in Section 7.4. Evaluation results for the held-out test data are presented in Chapter 8.

7.1 Language Model Rankers

The first type of modeling framework that we explore for the ranking task is *n-gram-based language modeling*. The theoretical basis for this framework is described in more detail in Section 2.1. As mentioned in Section 3.2, n-gram-based language modeling is the most commonly used approach for statistical selection in NLG. Nonetheless, the literature does not have much to offer in terms of systematic evaluation of different modeling parameters, such as n-gram size, discounting, vocabulary size, cutoffs, etc. Below we try to assess some of the effects that these parameters have on the accuracy of resulting models.

The data used for training the LMs is an unannotated version of the 100 million word British National Corpus[1] (BNC). Before we turn to look at the results, the next section first describes some of the preprocessing procedures for preparing the data prior to training. The training procedure itself is carried out using the freely available Carnegie Mellon Statistical Language Modeling (CMU SLM; ver.

[1]For more information, see http://www.natcorp.ox.ac.uk/.

2.05) Toolkit[2] (Clarkson & Rosenfeld, 1997).

7.1.1 Preprocessing the Training Data

Although our BNC training data mainly consists of British English, the sentences generated by the ERG are American English. Granted, the distinction between British and American English is not always clear-cut, but there still exist some standard patterns that typically mark a difference between the two variants. Of course, it always makes sense to try to make our training data be as representative as possible for the actual application data. The first step of the data preprocessing is therefore to map some of the British forms to their American equivalents. Note that we are here only concerned with simple mappings between spelling variants (e.g. *standardise* → *standardize*, and *modelling* → *modeling*). We do not try to convert between differences in actual vocabulary (e.g. *vacation* versus *holiday*) as there is far too much ambiguity involved for this to be automated in a straightforward way. For converting between spellings we use the `translate` script of the *VarCon* package.[3]

Most punctuation such as periods, commas, hyphens, quotation marks, parentheses, etc., are split from the words they attach to, so as to make them separate tokens. We also tried to train models that disregarded punctuation by removing it completely from both the training and test data. However, we found that this led to inferior performance and that the punctuation marks seem to provide important cues that the model can take advantage of. Note that most contracted forms such as *haven't* are still treated as one word and with the punctuation intact. The form *'s* (corresponding to a possessive specifier or contracted auxiliary) is treated as a separate token though, as it can combine with an unlimited number of other forms. Periods are also split from the forms they attach to, except when in infix positions, as in abbreviations. Within numerical forms, infix punctuation such as commas and periods are swallowed by the numerical normalization: To align the numerical forms in the training data with those produced by the ERG when generating from MRSs, all numerical forms are mapped to a so-called *ersatz* token. Some examples of these conversions are listed in Table 7.1 below.

Examples (7.1) and (7.2) below show an example of a sentence before and after normalization.

(7.1) We hiked up to Peter's cabin once in the mid-90's.

(7.2) <s> we hiked up to peter 's cabin once in the mid - decadeersatzs . </s>

[2] For more information, see `http://www.speech.cs.cmu.edu/SLM/toolkit.html`.

[3] VarCon (Variant Conversion Info) is maintained by Kevin Atkinson. For further details, see `http://wordlist.sourceforge.net`.

Observed Form	Normalized Form
37,5 1.500.000 7 2/3	decimalErsatz
September 8	september dateErsatz
1860s 1980’s 70s 90’s	decadeErsatzs

Table 7.1: Examples of numerical normalization in the LM data.

The so-called *context cues* inserted at the beginning and end of the normalized string mark the boundaries of the sentence. Demarcating the boundaries of the sentence segments in this way intuitively makes sense, but it is also necessary from a technical point of view, as described in Section 2.1, because it ensures that we get a proper probability distribution over all strings independent of length.

In the following sections we are finally ready to describe the first results of our realization ranking experiments. On our way towards developing the final LM ranker we explore a range of different parameters, and we here include some of the preliminary results obtained from these initial models. As mentioned in the introduction above, this chapter deals with the training and tuning of the individual model types, and we here only report results for the development data. Then, after going through the development cycles of all the rankers in turn, we finally compare their performance on the held-out data sets in Chapter 8. Furthermore, recall from Section 5.3.4 that we use two parallel data sets for both the development stage and the held-out testing; one type generated with the inclusion of information structure (IS) markers in the input MRSs, and another type generated with this information underspecified. The items in the more underspecified data sets typically show a greater degree of indeterminacy and thereby offer a harder ranking task. The reader is referred to Tables 5.2 and 5.2 for a summary of the key properties of these data sets. As the data set generated with ISinformation is regarded the primary data set (being most representative for the ranking task as defined within the LOGON system), this will simply be referred to as the Tourist development data (i.e. without making any additional qualifications). On the other hand, whenever we need to make a reference to the secondary data set, we explic-

itly state that it is the version generated with underspecified IS marking.

7.1.2 Vocabulary

One of the prerequisites to training an LM is to decide on the *vocabulary* of the model. Of course, we have already taken some first steps towards this decision, since the normalization and preprocessing described in the previous section directly affect the inventory of words in the data. It is important to note that also the decisions made in the preprocessing stage were guided by empirical results. For example, we initially discarded all punctuation in the training data, considering only "proper" words when ranking the realizations. However, subsequent experimentation showed that including the punctuation marks in the data, and treating them as separate (i.e. non-attaching) tokens, actually benefited ranking performance significantly. Now, after settling on a final normalization scheme, we still need to determine the set of vocabulary items to include in the model. Of course, simply including everything in the training data would result in a far too big model with an overwhelming number of parameters. Also, given the previously mentioned Zipfian distribution that word occurrences typically display, it would probably not be very useful. Instead, the most common way of determining the vocabulary is to pick the k most frequently occurring word types in the training corpus.

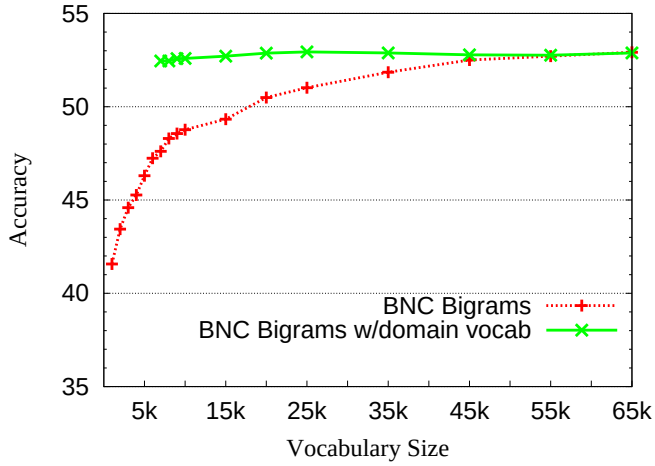
As the CMU toolkit imposes a maximum limit of 65,535 words in the vocabulary, this provides an upper bound on the number of words we can include. To asses the effect of vocabulary size on the model we trained a series of bigram models with an increasing number of words in the vocabulary, from one thousand to sixty-five thousand. The vocabulary items were selected according to the simple frequency criterion just described. Note also that this results in a model with a so-called *open vocabulary*. This is in contrast to a *closed vocabulary* model where all the words are known and specified in advance. However, such models require well-restricted settings where all the words in the test data are fully known in advance, and such ideal circumstances are rare in practise. Even when working within a circumscribed domain, such as the tourism domain in LOGON, it can prove impossible to have a fully specified and known vocabulary, not least due to the occurrences of proper names. In an open vocabulary model, on the other hand, all words in the training data and test data that fall outside of the specified vocabulary are mapped to a designated token <unk> (as described in Section 2.1.2).

The dotted lines of Figure 7.1, (titled *BNC bigrams* in the legend) show the accuracy of the various bigram models with respect to the increasing size of the vocabulary. As we can see, the performance of the models steadily increase as more words are added. The best performance is found in the models trained with the largest vocabulary, yielding 52.92% and 48.48% accuracy on the data sets

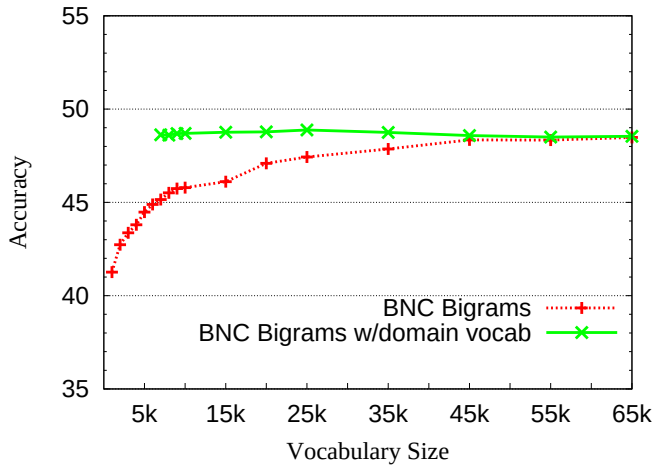
with and without IS information respectively. As expected, the slope of the curve is steepest for the first few thousand words that we include, and then the relative gain in accuracy gradually flattens out. This trend is perhaps not very surprising. Firstly, the first words that we include are also the most frequent ones, and are therefore likely to also be the most useful ones on new data. Secondly, the more words we include from the general domain BNC data, the more likely we are to cover the words that are actual in our domain, and in our domain-specific data sets. This latter point is important. We said earlier on (e.g. in relation to the mapping of British and American spelling variants) that we ideally wanted to train models using data that is as close as possible to the intended application setting. Of course, if we had access to a corpus of domain specific text that was large enough to train an n-gram LM, the task would be made a lot easier. But lacking such a luxury, we have to make the most out of the resources at hand. When defining our model vocabulary by blindly applying frequency thresholds to the general-domain BNC corpus, we run the risk of excluding many of the domain-related terms that actually do occur in the corpus, although maybe with a relatively low frequency of occurrence. On the other hand, we end up including many words that are not necessarily relevant for our particular domain, but that are nonetheless among the more frequent ones in the corpus. A better strategy would be to first make sure that the vocabulary we extract from the BNC includes as many words as possible that are representative of our domain. Then, after ensuring that these words are included, we can begin to extend the vocabulary based on the simple frequency principle.

To approximate such a solution, we first compile a complete vocabulary list from our Tourist development data. This gives us a list of $8,761$ word types (including both punctuation and word forms). We similarly compile a complete word list for the ("Americanized" and normalized) BNC corpus, resulting in a vocabulary list of $383,081$ types. Next we compute the intersection of these two vocabularies, resulting in a word list of $6,956$ types. This means that the degree of overlap between the two vocabularies turns out to be surprisingly good, with roughly 80% of the Tourist vocabulary being covered in the BNC. Of course, the remaining 20% of words from the Tourist data *not* occurring in the BNC, can safely be discarded from the model vocabulary, since we would have no statistics for them in our training data. In any case, this remaining set almost entirely consists of proper nouns, typically the names of mountain ranges and the like.

Of course, the vocabulary compiled from our Tourist development data does not constitute an exhaustive list of words relevant to our domain and the task of realization ranking. In order to ensure good performance also on new and unseen data, including the Tourist held-out test set, it may be a good idea to further extend the model vocabulary by including some of the additional words of the general-domain BNC, sorted by frequency. The ranking results for the models trained with



(a) Vocab. size vs. accuracy on the development data.



(b) Vocab. size vs. accuracy on the IS-underspecified dev. data.

Figure 7.1: The effect on accuracy when increasing the number of words in the vocabulary of bigram LMs. The graphs in (a) show results tested on the primary Tourist development treebank, and the graphs in (b) are for the IS-underspecified version. *BNC bigrams* shows the results of gradually extending the vocabulary with words from the general-domain BNC, sorted and selected according to frequency. *BNC bigrams w/domain vocab* shows the same, but starting from a domain-tuned vocabulary compiled from the intersection of words in the Tourist development data and the BNC.

this new domain-tuned vocabulary at its core, and incrementally adding in more of the frequency-sorted words remaining in the BNC, are denoted by the solid line

in Figure 7.1 (titled *BNC bigrams w/domain vocab* in the legend).

As can be seen, using the domain-tuned vocabulary as described above gives quite a positive boost to the ranking performance. Right from the onset, using a vocabulary consisting only of the words in the Tourist/BNC intersection ($\approx 7k$ words), we achieve an exact match accuracy that is on par with that of the model using the maximal BNC vocabulary ($= 65k$ words). At the same time we manage to keep model size at a minimum, given the reduced vocabulary. As said, making sure that the vocabulary of our general-domain language model overlaps as much as possible with the vocabulary of our in-domain development set, is a strategy that helps ensuring as good performance as possible for our intended application setting within LOGON. However, the results in Figure 7.1 are computed for the development set, which is the very same data that we used as a basis for compiling the word lists in the first place. In order to ensure equally good performance also on the held-out data, we want to make sure we have reasonably good coverage also of the other high-frequency word forms in the general-domain training data (i.e. words occurring with a high frequency in BNC but which happen to fall outside of the intersection with the development set). We therefore choose to also include some of the top stratas of the frequency-sorted BNC vocabulary. Admittedly, the precise choice of where to draw the line here is destined to be somewhat arbitrary. However, for both data sets we see that performance seems to peak for the model trained with a vocabulary of 25,000 words (at 48.88% and 52.94% accuracy on the data with and without IS information respectively). This is probably simply due to the back-off model striking a happy balance between the probability mass allocated to unknown words and the discounted probabilities used for the observed data. Guided by the peak in accuracy, and what seems like a reasonable compromise between model size and expected generalization capabilities on new data, we choose to stick with the domain-tuned vocabulary used in the 25k model for the rest of the n-gram LM experimentation.

As the inclusion of domain-adjusted vocabulary appears to immediately boost performance, a natural question to ask at this point is why we do not simply train the LM on the LOGON data instead. However, Velldal and Oepen (2006) report results where the general domain BNC LM is combined with an additional in-domain model trained on the texts that form the basis of the *Jotunheimen* corpus, a subset of the Tourist development corpus, containing a total of 5024 sentences. The optimal weights for a linear combination of the two models were calculated using the interpolation tool in the CMU toolkit (using the expectation maximization (EM) algorithm, minimizing the perplexity on a held out data set of 330 sentences from the *Hike* corpus). However, when applied for realization ranking on 634 items from the *Rondane* generation treebank, this interpolated model failed to improve on the results achieved by just using the larger general-domain model alone. This is probably due to the very limited amount of domain-specific data that we

have available for training. Note, however, that there is a continuation project of LOGON, called HandOn, which focuses on extending the coverage of the LOGON MT system in various respects. Although HandOn is still in progress at the time of writing, one of the ambitions of the project is to compile additional corpus material for the Tourist domain. Now, as there are still good reasons to assume that an in-domain LM can potentially perform better than the general-domain BNC model, if only given adequate amounts of training data, we plan to further pursue these experiments in the future, once more corpus material is available.

To sum up, in this section we have seen that, in the general case, using a larger vocabulary improves the accuracy on the realization ranking task, although obviously at the cost of creating models of a larger size. We see that the exact match accuracies of the bigram LMs seem to flatten out at a vocabulary size between forty and fifty thousand words. However, by using a more domain targeted vocabulary, extracted from the intersection of words in the development data and the BNC, we can achieve accuracy that is similar to the best-performing model with the larger frequency-selected BNC vocabulary, although by using a lot fewer words. As we shall see in the sections below, reducing the complexity and size of our n-gram model by having a smaller inventory of words in our vocabulary means that we can afford to make other parametric choices which also lead to an increased model size, such as training higher-order models with a larger value of n.

7.1.3 Perplexity vs. Log-Probability

We started out this section by saying that a candidate realization is scored and ranked according to the probability of its surface string as assigned by the language model. More precisely, we use the negative log-probability, as in

$$F(r) = -\log p_n(y(r)) = -\log p_n(w_1, \ldots, w_k) \tag{7.3}$$

where $y(r) = w_1, \ldots, w_k$ is the surface string corresponding to the yield of the generated realization r. However, the score that is actually computed by the `evallm` tool in the CMU toolkit which we use when applying the models, is the *perplexity* of each string, as formulated in Equation (2.14). More typically this score is computed over an entire test corpus and used for assessing the quality of a given LM q_n. As described in Section 2.2, the perplexity is then computed as $2^{H(x_1^N, q_n)}$, where H is the cross-entropy between a true distribution p and the model q_n, approximated with respect to some large sample sequence $x_1, \ldots, x_N$. Of course, our setting of scoring individual sentences is quite far from satisfying the asymptotic assumptions that hold on the corpus level. On the sentence level, the usual approximation to the perplexity essentially indicates the average

log-probability of the words in a string w_1^k:

$$2^{-\frac{1}{k}\log_2 p_n(w_1,\ldots,w_k)} \tag{7.4}$$

Note that the perplexity can also be expressed as the inverse of the geometric mean of the logarithms of the word probabilities:

$$\frac{1}{p_n(w_1,\ldots,w_k)^{(1/k)}} \tag{7.5}$$

This formulation makes it easier to see how the perplexity scores can be thought of as length-normalized sentence probabilities. Since longer strings will generally receive a lower likelihood from the generative language model, including some kind of mechanism for counter-acting this bias is generally held to be a good thing (see for instance Belz, 2005). However, for our data sets we actually found the inverse to be true. By undoing the normalization, and instead using the negative log-probabilities directly as in Equation (7.3), we found that the performance of the LM ranker improves drastically. On the Tourist development treebank the exact match accuracy of a trigram model increases from 42.90% to 53.62% when moving from perplexities to raw log-probabilities. (The models were otherwise trained using Witten-Bell discounting and the 25K intersected vocabulary described above.) For the data set with IS-underspecification the effect is even more pronounced, with the accuracy leaping all the way from 31.09% up to 49.27%. In other words, we achieve close to a twenty percent increase in accuracy, just by making a seemingly small change to the scoring function, undoing the length normalization. The difference in ranking performance is also very visible from the NEVA and WA string similarity metrics. Complete results for the contrastive evaluation of the two scoring schemes, for both versions of the development data and for all metrics, are summarized in Table 7.2 below. In the following paragraphs we look into some of the factors we found contributing to these relatively dramatic differences in ranking performance. We will refer to the LM using perplexity scores as the *perplexity ranker*, while the LM using non-normalized negative log-probabilities as the *log-prob ranker*.

Obviously, the differences in performance must in some way be related to length, as this is the only factor separating the two relevant scoring schemes. Given the performance gain of the log-prob ranker over the perplexity ranker, it seems that quite often it is the *shorter* candidates that are labeled as gold in the treebanks. This tendency would seem to be even more marked on the IS-underspecified profiles, where the relative difference in performance is even greater. Moreover, regardless of average sentence length, we know that the *variance* in length within individual data items (i.e. the differences in length for realizations of the same MRS) is greater for the IS-underspecified profiles. Additional

Performance of different LM scoring schemes				
	Primary		IS-underspecified	
	Perplexity	Log-prob	Perplexity	Log-prob
Accuracy	42.90	53.62	31.09	49.27
WA	0.871	0.903	0.682	0.819
NEVA	0.865	0.907	0.770	0.871

Table 7.2: Ranking performance improves dramatically when changing the **scoring function** from perplexity to raw negative log-probability. Results above are for a trigram model trained with Witten-Bell discounting, using the 25K intersected vocabulary described in Section 7.1.2. *Primary* refers to the standard Tourist development set, while *IS-underspecified* is the version constructed without MRS-marking of information structure (see Section 5.3.4 for details).

topicalized and passive constructions resulting from the increased level of underspecification in these profiles typically carry with them extra commas, complementizers, relative pronouns, auxiliaries, etc. As a consequence, these profiles are also more sensitive to changes in the scoring function that are related to length. It is not so surprising then, that the choice of what particular scoring scheme to use has a greater impact on this version of the development data. Now, while a topicalized or passive candidate is often longer, due to extra punctuation, by-phrases, etc., it turns out that it is typically also *not* labeled as preferred in the treebank. This means that normalizing for length, in order to even out the natural penalty for longer strings, can actually hurt performance. Table 7.3 shows a more detailed view of the accuracy scores of the perplexity ranker and the log-prob ranker on the IS-underspecified data, with test items broken down into bins according to reference length. This table thereby also gives an idea of the length distribution of items in our data sets.

It might be instructive at this point to look at some concrete examples of the errors actually made by the perplexity ranker. The following examples are all taken from the IS-underspecified generation treebanks for the Tourist corpus, and have been identified as part of an error-analysis conducted by Stephan Oepen. Since we are interested in bringing out the contrast between the two scoring schemes, we only look at errors that are unique to the perplexity ranker (i.e. items for which the choice of the log-prob ranker matches the reference). Following the item identifier from the treebank, each example first shows the candidate labeled as gold in the treebank, and then the 1-best or top-ranked candidate according the perplexity ranker.

LM ranking on the IS-underspecified Dev. Set

		Scoring Function	
Length	**Items**	**Perplexity**	**Log-prob**
$30 \leq l$	94	13.12	24.56
$25 \leq l < 30$	293	17.19	25.01
$20 \leq l < 25$	694	20.45	34.71
$15 \leq l < 20$	1095	27.11	39.69
$10 \leq l < 15$	1253	32.87	50.08
$5 \leq l < 10$	966	40.44	68.02
$1 \leq l < 5$	325	50.62	82.77
Total	**4720**	**31.09**	**49.27**

Table 7.3: Detailed view of the accuracy of a trigram model using different scoring functions, tested on the IS-underspecified Tourist development data. Test items are aggregated into bins according to their **reference length** l. *Log-prob* uses (negative logarithms of) the raw probabilities, while *perplexity* normalizes for length by dividing the score on the token count of the candidate. The table also shows the distribution of items according to length, with the second column listing the number of items in each bin.

(7.6) *Item 45511*
Gold: You follow Dummdalen to Svarttjørna.
Best: To Svarttjørna, Dummdalen is followed by you.

(7.7) *Item 61031*
Gold: The numerous tourists left their mark.
Best: By the numerous tourists, their mark was left.

In Examples (7.6) and (7.7) above, we see how the perplexity ranker incorrectly chooses variants that are both topicalized and passivized. Note that the language model seems to be very fond of commas, which of course are one of the most frequent tokens in the training data. The same seems to be true for many function words, which also appear with very high relative frequency and without showing any clear patterns of co-occurrence restrictions. As said, topicalization and passivization often carry with them an extra load of commas and function words, and with the length normalization inherent in the perplexity score, we often end up incorrectly preferring these longer candidates. Related observations seems to have been made during the development of the Nitrogen system. Langkilde and Knight (1998b) state that:

> We also notice that some kind of length heuristic is necessary; otherwise the straightforward bigrams prefer sequences of simple words like “was”, “the”, “of”, to more concise renditions of a meaning. (Langkilde & Knight, 1998b)

An additional example of an item for which the perplexity ranker incorrectly selects a topicalized variant with an extra auxiliary is given in Example (7.8) below.

(7.8) *Item 21032*
Gold: Many tourists were also brought here by rowboat.
Best: By rowboat, many tourists were also brought to be here.

Note that there are also many other factors that affect length, and which also happen to follow the same preference pattern in our treebanks, i.e. the shorter form is preferred over the longer one. Examples include such things as lexical variation or contracted forms, such as in the (tokenized) instances *weekend* vs. *week - end*, or *don't* vs. *do not*. Variation in length can also be due to differences in how adjectival phrases are constructed such as in

(7.9) *Item 41662*
Gold: Norway's longest fjord is Sognefjorden.
Best: Sognefjorden, Norway's most long fjord is.

In addition to incorrectly preferring the topicalized variant, Example (7.9) also shows how the perplexity ranker also prefers the (grammatically dubious) AP formed using *most long* instead of the superlative form *longest*. Similar incorrect preferences for longer constituents[4] are observed for noun phrases, such as the preference for *melting of snowfields* over *melting snowfields*.

Many of the examples shown above exhibit the distinctive pattern which by some is affectionately known as *Yoda speak* or *Yodish*, after the famous character from the Star Wars saga. The n-gram language models, and the perplexity ranker in particular, seem to be quite fond of this rather stilted mode of expression, preposing PPs, NP objects, adverbs, and even VPs.

Although we have seen several examples now of how the perplexity ranker makes many errors due to often preferring longer and thereby often topicalized candidates, a few of the errors made by the log-prob ranker are caused by the exact opposite behavior. For some of the items where the topicalized version is actually the one labeled as gold in the treebank, the log-prob ranker occasionally stumbles by incorrectly giving preference to a shorter and non-topicalized candidate. A simple example is the following item:

[4] Of course, the LM itself has no knowledge of constituents or phrases, and only considers sequences of surface text.

(7.10) *Item 43332*
Gold: Hopefully, I have succeeded.
Best: I have succeeded hopefully.

As should be self-evident given the results presented in this section, the log-prob scoring scheme will be the default for all LMs trained and tested in this thesis. In the next section we move on to take a closer look at how the choice of what particular discounting method to use can affect the ranking performance.

7.1.4 Discounting

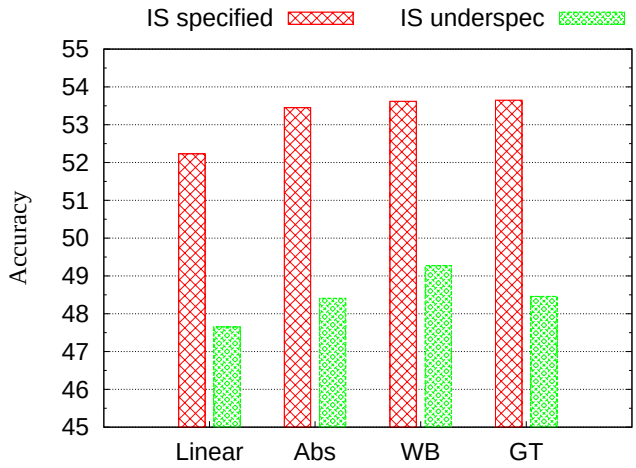
In Section 2.1.1 we reviewed the concept of discounting. The CMU SLM toolkit provides a choice of several discounting methods, and in this section we compare the ranking performance of various back-off LMs trained using Good-Turing, absolute, linear and Witten-Bell discounting.

Figure 7.2 shows the accuracy of a 3-gram model and a 4-gram model respectively, trained using the different available discounting schemes. For all configurations we see that the linear discounting strategy yields models with the lowest accuracy. For the 3-gram models, when computing the accuracy on the primary (i.e. IS-specified) test profiles, there is not much that sets the remaining discounting strategies apart. However, there seems to be a tendency that, while Good-Turing appears a good choice for lower-order models (e.g. $n = 2$), this changes when moving to models with a higher value of n, where instead both Witten-Bell and absolute discounting stand out as better choices. For the 4-gram models in Figure 7.2(b), we see that the Witten-Bell discounted model clearly outperforms the others. The differences in performance are even more marked when testing on the IS-underspecified profiles.

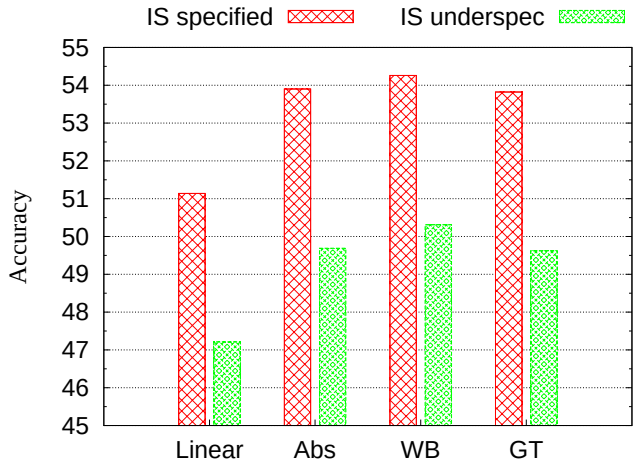
Given the relative superiority of the Witten-Bell method for the models we see here, especially for the 4-gram models, this will serve as the default discounting method in all LMs we train hereafter.

7.1.5 N-gram Order

In this section we investigate how the *order* of the n-gram model affects the ranking performance. When reviewing the literature on statistical realization ranking, we find that n-gram LM rankers are typically trained for $n = 2$. However, as we shall see, better ranking performance can be achieved by training models for larger values of n, carefully employing frequency cutoffs to avoid overly large models. It is worth noting here, perhaps, that this result is not necessarily a given. In fact, (Langkilde & Knight, 1998b) claim that using higher order models and going beyond bigrams may actually lower the performance on the ranking task.



(a) **3-gram** LM with various discounting strategies.



(b) **4-gram** LM with various discounting strategies.

Figure 7.2: Accuracy of various discounting strategies for a 3-gram and a 4-gram language model. The models are trained with a 25K vocabulary and no cutoffs, tested on both versions of the Tourist development set.

However, the arguments and the constructed examples given by (Langkilde & Knight, 1998b) seem to all hinge on the fact that one is not using back-off models, i.e. backing off to a lower-order model $n-1$ when encountering an unobserved n-gram. For more background on back-off LMs as used in our experiments, see Section 2.1.1.

Figure 7.3 below shows the effect on accuracy when increasing the order of n-grams recorded in the model. As we see, using a higher value of n clearly

improves performance, but perhaps only up to a certain point. While we see that there is a marked difference between models trained with n from 2 to 4, the gain in performance seems to flatten out for models where $n \geq 4$. When moving from a 4-gram model to a 5-gram model, the accuracy is more or less unchanged.

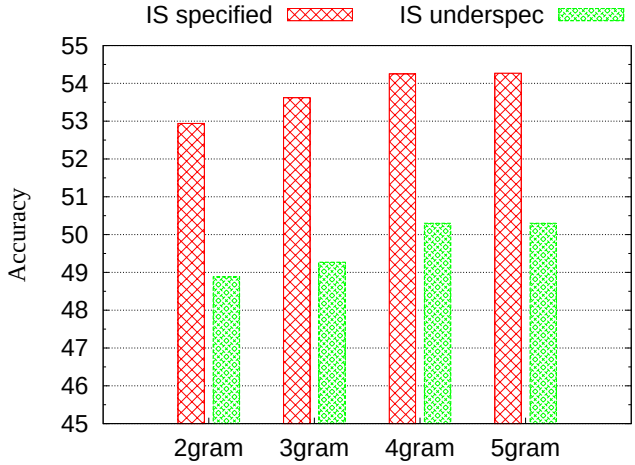


Figure 7.3: The effect on accuracy when increasing the order of n-grams in the LM (with a 25K vocabulary and no cutoffs).

As a curious fact, note that even a unigram model (not included in the plot) significantly improves on the random choice baseline by a good margin, and especially for the IS-underspecified profiles where the accuracy goes up from 16.62% (random) to 36.62% (unigram).

In sum, for realization ranking on our data sets, it seems desirable to use a rather high value of n, preferably a 4-gram model. However, as we increase the value of n, we also increase the physical (file) size of the model by a considerable amount. Although we may not care too much about the size of a model in terms of disk space and storage, we *do* care about the time it takes to load a model into memory before scoring a set of generated candidates in an actual application setting. And loading times are, of course, directly dependent on model size. However, the size of the LM can be brought down by employing *frequency cutoffs* on the n-grams to be recorded in the model. The use of cutoffs, and the corresponding tradeoff between model size and performance, is what we turn to next.

7.1.6 Frequency Cutoffs

In all the models we have applied so far, all n-grams observed for the specified vocabulary have been included in the model, irrespective of their frequency of occurrence. Even with our modest vocabulary size, collecting all this information

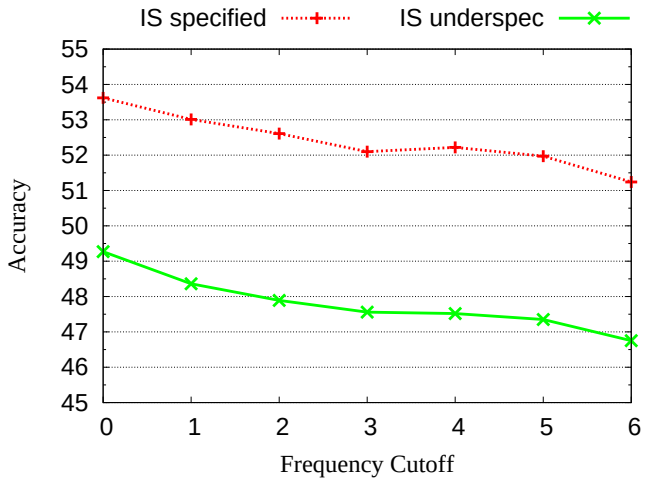
quickly leads to very large models, especially for models with a larger n. It is therefore common to apply some kind of frequency cutoff, discarding all n-grams that occur fewer than k times in the training data. Of course, although this quickly brings down the size of the model, both in terms of the total number of recorded n-grams and the number of bytes taken up in memory, it can also quickly reduce performance as we are discarding a lot of potentially useful information. Even though an event may be rare, it can still be informative, especially given the data sparseness typically faced in NLP. Based on experiments with both a 3-gram model and a 4-gram model, Figure 7.4 shows how the accuracy in the ranking task drops as the frequency cutoffs are raised. The relative loss in accuracy is largest for the first increment, i.e. when going from recording everything to only recording n-grams occurring more than once. We also see that this effect is strongest for the model with the larger n: For the 4-gram model the accuracy drops from 54.25% (no cutoff) to 52.81% (cutoff of 1) when tested on the standard data set, and from 50.30% to 49.40% on the IS-underspecified data. As we see, suppressing even the singletons has a negative impact on performance.

Figure 7.5 shows how the number of megabytes required to store an LM on disk drops as the frequency cutoffs on n-grams are raised. Again we see that the curve is dropping most precipitously for the first increment, and that the relative effect is more pronounced for the model with the higher order of n. Note that the trend in model size shown in Figure 7.5 is also proportional to the model *loading time*, i.e. the time it takes to read an LM into memory before scoring a set of candidate realizations.[5] This means that there is a clear trade-off here between performance and efficiency: While efficiency is negatively correlated with model size, performance is positively so, in the sense that the larger models typically do a better job at the ranking task. Although the CMU toolkit is able to write and read LMs in a binary form, the time it takes to load a model into memory can quickly become unacceptably long in an actual application setting if the size of the model is too big. To give a concrete example, performing some (very rudimentary) profiling,[6] it takes a good three seconds to load a full 4-gram model estimated without any cutoffs. However, already for the model trained with a flat frequency cutoff of 1, the loading time drops down to around 0.75s. When applying a cutoff of 2 to the 4-gram model, loading time is less than 0.45s.

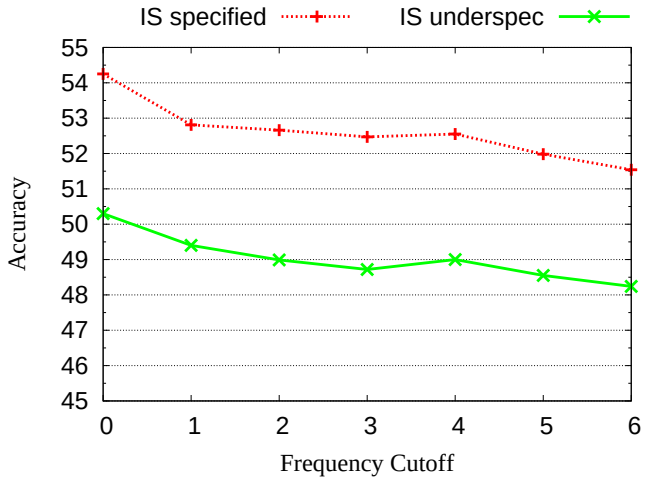
As we saw in Figure 7.3, the 4-gram model seems to provide the best choice of

[5] In our current run-time set-up for the LOGON prototype system, an LM has to be loaded fresh into memory for each item to be scored. All candidate realizations available for that item, however, can be scored within the same "session." In other words, we only have to load the model once for each set of realizations generated for a given input to the system.

[6] We here use the the `evallm` script provided by the the CMU toolkit for reading and evaluating language models. Loading times are then measured using the simple `time(1)` Linux command-line tool, computing averages of sums of the *usr* and *sys* output over repeated calls to `evallm`.



(a) 3-gram LM with various frequency cutoffs.



(b) 4-gram LM with various frequency cutoffs

Figure 7.4: Exact match accuracy of 3-gram and 4-gram models trained with various frequency cutoffs on the recorded n-grams. The LMs are trained using a 25K vocabulary and Witten-Bell discounting, and test results are shown for both versions of the Tourist development set.

model in terms of ranking accuracy. However, in its bare form, the 4-gram model also takes up quite a bit more space than the 3-gram model. As we have just seen, however, file size can quickly be brought down by imposing frequency cutoffs on

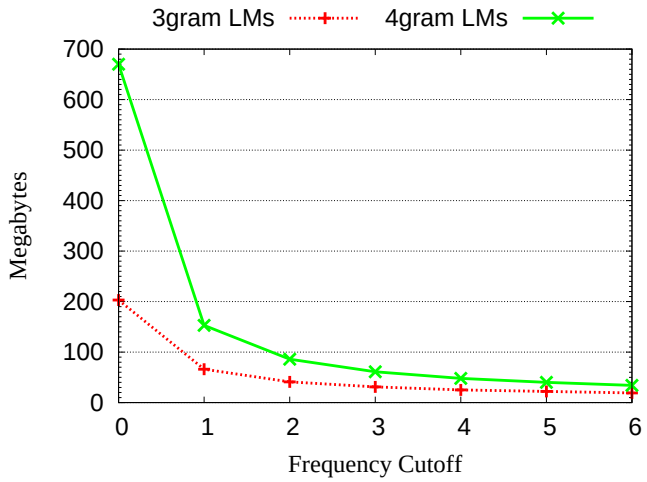


Figure 7.5: File sizes in megabytes for LMs with various frequency cutoffs on n-grams (using a 25K vocabulary).

the recorded n-grams, although at the cost of a significant[7] loss in accuracy (down by almost one and a half percentage points on the Tourist development data). However, in the models plotted above, the frequency cutoffs are applied rather indiscriminately; for ease of exposition the same cutoff-level is applied across the board for n-grams of all lengths (i.e. within the same model we apply the same cutoff for 2-grams as for 4-grams). A better strategy is to impose successively higher cutoffs for longer n-grams. For example, when we for a 4-gram model define the cutoffs 0, 1 and 2, for 2-grams, 3-grams and 4-grams, respectively, we are able to achieve roughly the same reduction in file size as seen for the uniform cutoff of 1 in Figure 7.4(b), but without sacrificing nearly as much in terms of accuracy. On the primary Tourist development set the accuracy drops from 54.25% (no cutoffs) to 53.75% (cutoffs 0–1–2). This difference is not detected as significant by the sign test, however. The same observation holds for the data set with underspecified IS information, and also when comparing the WA and NEVA scores instead of exact match accuracy and testing for significance using the signed-ranks test. Although the differences in performance are not found to be significant, the file size of the actual model is greatly reduced, dropping from 670MB to 170MB. In other words, we are able to reduce the footprint of the model to almost one fourth of its original size, while maintaining approximately the same level of ranking perfor-

[7] When tested on the Tourist development data, the difference in accuracy between a 4-gram model with no cutoffs and one with a uniform frequency cutoff of 1, has a p-value of 0.02 according to the sign test.

mance. This then, the 4-gram model using successive cutoffs as described above, will be our model of choice for the rest of our LM realization ranking experiments, both when moving on to the held-out testing and when including the LM scores as a feature in the discriminative models that we later move on to develop.

LM ranking results the Tourist development set				
	Primary		IS-underspecified	
	top	5-best	top	5-best
Accuracy	53.75%	82.64%	50.04%	80.08%
WA	0.907	0.986	0.825	0.967
NEVA	0.907	0.984	0.873	0.971

Table 7.4: Ranking performance of the 4-gram LM described above, tested on both versions of the Tourist development set. Section 5.3.4 provides more details about the data sets, and Section 6.3 describes the various evlauation measures.

7.1.7 Summary

In this section we have gone through a series of steps to finally arrive at the language model we will use for ultimately assessing the performance of n-gram-based realization rankers in this thesis. Before we round off this section and move on to the other model types we will be testing, let us sum up the specifications of our LM finalist. Using a raw text version of the BNC as input data, carefully normalized in a pre-processing step, we have estimated a 4-gram back-off language model using Witten-Bell discounting and a successive series of frequency cutoffs on n-grams. We have used a partly domain-tuned vocabulary of $25,000$ word forms based on word lists extracted from our domain-specific Tourist development data, and further extended with the most frequent words of the general-domain BNC. When applying the model, candidate realizations are scored and ranked according to (negative and non-normalized) log-likelihood. Table 7.4 summarizes the performance of the LM ranker, evaluated according to exact match accuracy, WA, and NEVA, and considering both 5-best lists and the top-ranked candidate only. Table 7.5 gives a more detailed view of performance in terms of exact match accuracy, grouping the data items into bins according to their number of available realizations (i.e. the degree of indeterminacy during generation). This latter table also includes the expected accuracy of the random choice baseline, from which we can clearly see the considerable improvement offered by the language model. In terms of reduction in error rate relative to the baseline, the LM ranker reduces the error by 35.72 percent for the standard Tourist development set, and by a dramatic

40.08% percent for the IS-underspecified version. As is clear from Table 7.5, the

LM ranking on the Tourist development data

Bin	Items	Words	Trees	Gold	Baseline	Accuracy
$100 \leq n$	369	27.5	360.0	8.0	3.33	25.07
$50 \leq n < 100$	230	24.7	73.5	4.9	6.64	31.50
$10 \leq n < 50$	1144	20.6	22.5	3.3	17.08	43.25
$5 \leq n < 10$	868	15.3	6.9	2.2	32.13	59.69
$1 < n < 5$	1310	13.9	3.2	1.4	45.64	70.99
Total	**3921**	**18.1**	**47.3**	**3.0**	**28.05**	**53.75**

LM ranking on the Tourist development data underspecified for IS

Bin	Items	Words	Trees	Gold	Baseline	Accuracy
$100 \leq n$	966	24.4	468.0	4.4	1.58	23.29
$50 \leq n < 100$	433	20.9	71.8	3.2	4.63	29.54
$10 \leq n < 50$	1607	17.2	23.8	2.2	10.83	45.42
$5 \leq n < 10$	842	12.1	6.9	1.7	25.02	64.58
$1 < n < 5$	872	11.3	3.3	1.3	41.82	84.35
Total	**4720**	**17.0**	**112.3**	**2.5**	**16.62**	**50.04**

Table 7.5: Detailed view of the accuracy of the LM ranker on both versions of the Tourist development data. Data items are binned relative to their number of realizations. The columns are, from left to right, the subdivision of the data, total number of items, average string length of the realizations, average number of realizations, average number of references, and finally the baseline for expected accuracy by random choice.

LM performs well above the random choice baseline. However, as the vocabulary of the model is partly determined by the words occurring in the development data itself (using only the words that are found to also occur in the BNC), it seems likely that all of these evaluation figures are overestimating the performance somewhat. We will have more to say on this when we turn to the held-out testing in Chapter 8, but first we move on to develop realization rankers within the two other modeling frameworks explored in this thesis; MaxEnt models and SVMs.

7.2 MaxEnt Rankers

In this section we develop realization rankers based on discriminative training using the framework of Maximum Entropy (MaxEnt) modeling, as described in

Section 2.3. While the n-gram language models we developed in the previous section were trained on raw text and purely oriented towards the surface strings, our MaxEnt models are instead trained on generation treebanks and based on structural properties of the realization as a whole. Recall that the generation treebanks pair up MRSs with all their possible realizations, both preferred and dispreferred, stored in the form of HPSG derivation trees. While the construction of the actual generation treebanks was described in Section 5.3, the feature templates that we use when extracting information from them were summarized in Section 5.5.

All the models we describe below are trained and tested through ten-fold cross-validation on our development data. As described in Section 6.1, this means that we first divide the data set into ten non-overlapping folds (or parts) of roughly equal size. We then repeatedly train a model on nine of the folds while reserving the one remaining fold for testing. After ten full rounds, always reserving a new fold for testing, we will have tested on the entire data set. We can then compute average performance scores across the ten folds. The reason for using such a set-up is, of course, to make the most out of our limited resources. Naturally, annotated and treebanked data, such as the Tourist data of the LOGON project, is a resource which is much more scarce than the raw text we could use for training n-gram language models. Note that, while we did use the Tourist development treebank in relation to the LM experiments, this was only for the purposes of vocabulary extraction and testing.

In the sections that follow we explore a range of different aspects related to training and tuning of the models: We start out, in Section 7.2.1, by describing the contribution of the different feature types, as well as different configurations of these features. In Section 7.2.2 we look into the relationship between model performance and the available amount of training data. Section 7.2.3 describes the use of so-called relevance cutoffs for reducing the total number of features. Then, in Section 7.2.4, we show the effect of tuning the variance parameter in the prior distribution imposed on feature weights. This regularization technique was described in Section 2.3.3. Finally, in Section 7.2.6, we define a combined model, adding the previously defined n-gram model as an additional feature in the best performing MaxEnt model, thereby combining the strengths of both model types. Recall that the TADM toolkit (Malouf, 2002) that we use for MaxEnt parameter estimation was presented in Section 6.2.3.

7.2.1 Feature Configurations

In Section 5.5 we defined a range of feature templates that we use for recording properties of the treebanked realizations. For each realization, the sequence of values that correspond to these feature functions is what constitutes its vectorial representation as used internally in the discriminative models. Each dimension in

a given feature vector f describes the data item with respect to a corresponding feature type. In this section we try to assess the usefulness of the individual feature templates described in Section 5.5, and try to arrive at the optimal combination of features to include when training the discriminative learners.

The starting point for our discussion will be the distribution in Figure 7.6, which plots the accuracy of models using different feature configurations, as applied to our two development data sets. Recall from Section 5.5 that, the "fundamental" feature type of our set-up, corresponds to the basic configurational features that throughout the derivation trees sample sub-trees of depth one. The first column for each data set in the histogram of Figure 7.6 corresponds to the accuracy of a model using only these features. Note that this is the same configuration as used for the MaxEnt model presented in Velldal et al. (2004). For the other columns in the chart we have added in new features to this "basic" model, as well as some combinations of features that are partly defined relative to each-other. We will discuss all of these variations in turn below, particularly with respect to how they improve (if at all) over the baseline MaxEnt model using only the basic feature type. Note that for all the model configurations shown in Figure 7.6, we have tuned the variance parameter of the prior individually. Still, we only perform a rather crude search at this point, casting the net widely, so to speak. As we shall see in Section 7.2.4 below, it is usually possible to push the performance even further if we also include an additional step of fine-tuning. After first finding roughly the best range of values, we can narrow in the search, testing different variance parameters with smaller intervals. Note that the total running time for a single experiment (i.e. the time it takes to complete a full ten-fold train-and-test cycle) is typically between one and two hours. For those familiar with TADM, it is perhaps also worth adding here that we have first performed some initial tuning of the convergence criterions. For all MaxEnt results reported here we "globally" fix the TADM parameters for both absolute tolerance and relative tolerance to 10^{-10} (the defaults being 10^{-10} and 10^{-7} respectively).

Now, we will have many things to say about Figure 7.6. The first thing to note is that even the basic MaxEnt model, using only sub-trees of depth one, clearly outperforms our final n-gram language model. Recall from Table 7.4, that the n-gram LM achieved an accuracy of 53.75% when tested on the Tourist development data. For the basic initial MaxEnt model we record an average of 64.05% across the ten folds in our cross-validated training cycle. In other words, the basic MaxEnt model manages to reduce the error rate relative to the LM by 22.27%. Looking at the similarity scores for the MaxEnt model on the same (i.e. primary) data set, the NEVA score is 0.931 (up from 0.907 for the LM) and WA is 0.924 (also up from 0.907). The difference in performance is just as clear when looking at the IS-underspecified profiles. The basic MaxEnt model here achieves a NEVA score of 0.904 (up from 0.873 for the LM) and a WA score of 0.858 (up from 0.825).

In terms of exact match accuracy, we recorded 50.04% for the n-gram LM on the underspecified data. For the basic MaxEnt model we record an average of 59.95%, corresponding to a relative error reduction of 19.84%. All of these differences in evaluation scores are detected as being strongly statistically significant by the two-tailed application of the paired sign-test and Wilcoxon test. These are encouraging figures, in that they clearly show us that the treebank training with "structural" features immediately pays off in terms of increased scores across all our evaluation metrics. The boost in performance probably also points to the importance of using data that is properly tuned to the specific domain, where it can be clearly delimited (as is the case in our application setting within LOGON). Even-though the amount of training data available for the discriminative MaxEnt models is relatively modest compared to the data we have for the linear language models, at least in terms of a simple word count, the fact that the treebanks are much more closely adapted to this specific domain seems to outweigh this disadvantage.

If we move on to the other columns of Figure 7.6, we see that the individual feature types that most clearly benefit the ranking accuracy are the lexical n-grams and the grandparenting (GP). For the models appearing in the histogram we use a span of $n = 4$ for the n-gram features and include 4 levels of upward chaining nodes in the GP features. Note that, these features are implemented using a kind of "back-off" scheme, meaning that for a given "maximal" feature we also include the less specific features that it subsumes. For example, a model that records 4-grams will also record 3-grams and 2-grams. The cascading GP features are defined in a similar way, so that for each 4-level GP relationship, we also record the 3-, 2-, and 1-level relationships. Furthermore, it is important that the n-gram features we talk about here are not confused with the n-gram language model. As described in Section 5.5, the n-gram features of the treebank model are defined over lexical types assigned by the grammar to the preterminal lexical identifiers of the derivation trees. Now, looking at the histograms for the standard and underspecified profiles in Figure 7.6, we find that the lexical n-gram features single-handedly boost the accuracy to 70.71% and 66.19%, respectively. In terms of reduction in error rate with respect to the basic model, these figures bring us down by 18.53% and 15.58% respectively. The grandparenting features offer an almost equally impressive increase in performance, with an accuracy of 69.74% (15.83% error reduction) and 65.70% (14.38%) respectively for the same data sets.

The two remaining feature types, active edges (AE) and constituent weights CW also offer a small but visible increase in performance. However, they do not contribute nearly as much as the n-grams and the GPs. When added to the basic model in isolation, the AE features achieve an accuracy of 64.13% on the primary data, and an accuracy of 60.07% for the underspecified data. Again, in terms of the error rate compared to the baseline MaxEnt model (i.e. the "basic" model), these figures correspond to a reduction of 0.22% and 0.3% respectively. Turning

to the CW features, their contribution is slightly more visible. When adding the CWs to the basic model we achieve an accuracy of 64.49% on the primary profiles (1.22% reduction in error rate), and 61.53% on the underspecified profiles (3.95% reduction in error rate). For both feature types, we see that the contribution is strongest for the data sets generated with IS-underspecification.

However, for both the AE and CW features, additional features are generated when we combine them with the grandparenting. For example, for each GP feature, up to the 4-level limit as used here, each daughter will spawn an additional AE feature corresponding to the non-branching path from the top-most dominating node down to the given daughter. Figure 7.6 therefore also includes separate columns for models where these feature types are used in combination. This reveals some interesting effects. Instead of observing a continued increase in performance when adding in features that in isolation give positive effects, we see that the performance flattens out, or even drops in some cases. For the combination of GP and AE features, accuracy on the primary and IS-underspecified development set is 69.39% and 65.97% respectively. For the GP and CW combo, the corresponding figures are 69.10% and 66.10%. In other words, when applied to the IS-underspecified profiles, both feature combinations lead to a minuscule increase in performance compared to the GP model in isolation. However, on the primary data, the equally tiny difference in performance is actually negative. In other words, for some of the data we actually do worse when adding the CWs and AEs to the models, given that it already contains the GP feature.

While Figure 7.6 to some extent allows us to see the contribution of each individual feature type in isolation, many of the features record overlapping information and may interact in ways that mean it can be difficult to predict what happens when we try to combine them in the same model. In addition to the view presented in Figure 7.6, i.e. incrementally adding new features to the basic model, we also experimented with the reverse approach, observing the effect of *leaving out* individual feature types from a model defaulting to including all features. Now, the total number of different feature configurations is, of course, much too large for it to be feasible to include all of them in a single diagram or tabulate all of the results. What is clear, however, is that we seem to be experiencing some kind of saturation effect. We reach a point where adding more features no longer adds to the accuracy, even though the relevant features may all prove useful when tested in isolation. We have already seen an example of this when we looked at the combination of AE, CW, and GP features above. We see the same effect when combining the n-gram features and the grandparenting features. Although the combination of these feature types does lead to better performance, the best performance is not found until after experimentally determining the best level of grandparenting and the best span of the n-grams, reducing the total number of features. These saturation effects may in part be due to the learner's (in)ability to handle a large number

of features relative to the modest size of our development data.

After extensive grid search across different feature combinations, we found that using 3-level grandparenting and 3-grams over lexical types, leaving out active edges and constituent weights, resulted in the model with the overall best performance. Admittedly, there were other feature configurations that resulted in similar ranking performance. However, these models include a lot more active features without being able to show statistically significant improvements. Given that other things are equal, it is good practice to decide by the principle of Occam's Razor. For our final configuration we therefore chose to use the simplest model (i.e. the one with fewer active features) of the ones with equivalent evaluation scores. The 3-gram and 3-level GP model achieved an accuracy of 71.85% on the primary development set. The corresponding NEVA and WA scores are 0.942 and 0.941. As mentioned above, an important estimation parameter in discriminative log-liner models is the variance of the prior distribution that we impose on the feature weights, and in this section we used a rather coarse-meshed net when searching for an appropriate value for this parameter. In Section 7.2.4 below we shall see how we can further improve ranking performance by additional fine-tuning of the prior. In the first section to follow, however, we take a look at how the performance of our MaxEnt realization rankers is affected by the size of the treebanked training data.

7.2.2 Learning Curves

The learning curves in Figure 7.7 shows the relationship between the size of training data and the accuracy of the resulting models. The graph plots the effect for two different configurations of MaxEnt models: One is the simpler set-up with only basic configurational features (i.e. *Basic*), and the other is a model using the final set of structural features we settled on in Section 7.2.1 above (i.e. *Extended*). The models are trained and tested using a somewhat modified version of the usual ten-fold cross-validation set-up on the Tourist development data. The x-axis of the graph in Figure 7.7 shows the percentage of items among the nine-tenth of the items in the training fold that were actually included for training in each iteration. Only for the last point on the axis are the models trained using the full set of available training items. Recall that the (primary) Tourist generation treebank that we use for development comprises a total of 3921 items. After first setting aside one tenth of the items for testing, this leaves a bit more than 3500 items in total for training (assuming an ordinary ten-fold set-up). Correspondingly, this also equals the number of training items used for the right-most models in Figure 7.7, i.e. the ones plotted at the 100% mark. When training the model corresponding to the first point (= 10%), on the other hand, only approximately 350 items are used.

The testing is carried out in the usual manner, scoring the held-out one-tenth

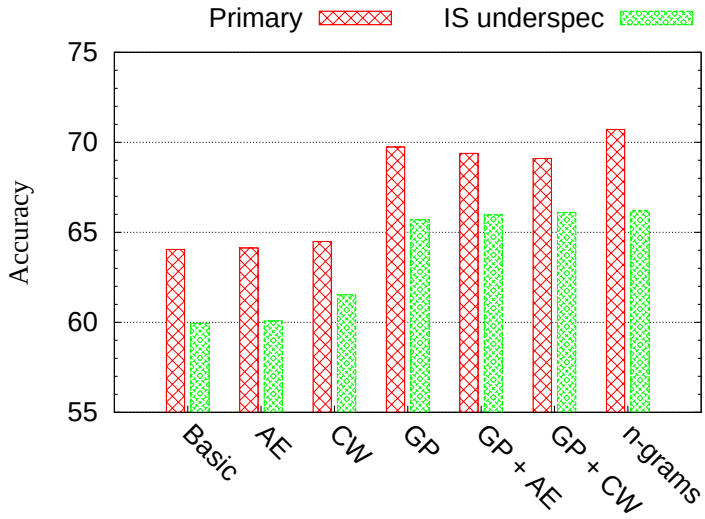


Figure 7.6: The effect of adding in various feature types. *AE* corresponds to active edges; *CW* to constituent weights (CW=2); *GP* to grandparenting (up to 4 levels); and *n-grams* are lexical type n-grams (with n=4). The histogram plots the best run for each feature configuration after individually tuning the prior. As usual, *Primary* corresponds to the standard data set with IS information intact in the input MRSs, while *IS underspec* refers to the data set generated without this information.

of items in each iteration over the data. The final accuracy scores are, as always, average scores across these folds. Note that, in order to ensure at least near-best performance for each size-bin, we also tuned the variance parameter of the prior individually.

As expected, for both models the graph shows that the benefit from additional data is greatest for the first increments. Still, even when only including 10% of the available training data the accuracies are quite respectable, giving 54.89% for the basic model and 57.68% for the extended model. We also observe that the relative difference in performance between the two feature configurations becomes clearer as we include more data. At the 50% point, the accuracies are 62.18% and 67.80% respectively, and at 100% of the data they are up to 64.05% and 71.85%. However, for both configurations we find that, as more data is included for training, the performance curve seems to flatten out somewhat. This trend is no doubt strongest for the basic model, however. For the richer configuration on the other hand, it still seems attractive to further extend the training data.

The trend shown in Figure 7.7 seems to fit in well with our earlier results on smaller data sets. For example, Velldal et al. (2004) presented preliminary results

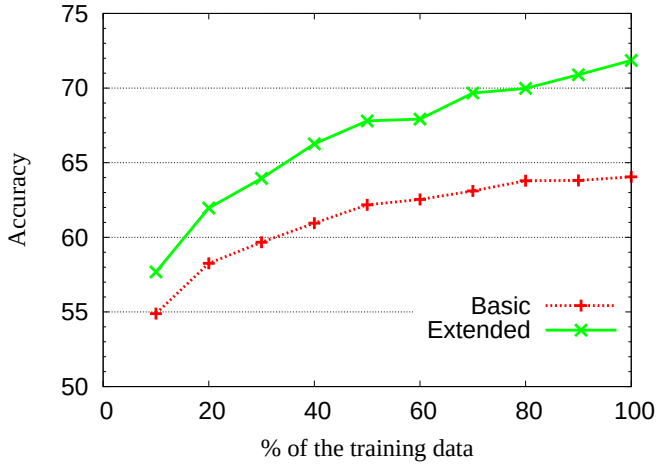


Figure 7.7: Learning curves for MaxEnt models with two different feature configurations. *Basic* is a model using only the basic feature types, i.e. sub-trees of depth one. *Extended* is the model which additionally uses three-level grandparenting and lexical type trigrams. The models are tested on the (primary) Tourist development data by ten-fold cross-validation. For the nine folds of training data in each round, we only use a random subset of n percent of the data (with n corresponding to the points on the x-axis).

on a smaller generation treebank[8] of only 261 items, where a model using only the "basic" features gave an exact match accuracy of 51.7%. When testing the same configuration on a sub-set[9] of the Tourist treebank, consisting of a total of 2190 items, Velldal and Oepen (2006) obtained an accuracy of 63.09%.

As said, for the model using the extended feature configuration, Figure 7.7 seems to suggest that using more training data than what we currently have available can be expected to further add to the accuracy. But in a sense, we actually *do* have more training data than what we have been able to use for the full model plotted here (or for any of the other MaxEnt models presented in this section for that matter). Of course, as the results are only based on ten-fold cross-validation, by averaging the accuracies of models iteratively trained over sub-sets of the data, we are never able to take full advantage of the entire set of available training items. In this sense, the use of n-fold cross-validation can be expected to slightly

[8]The data set is based on LOGON's *Hike* corpus, which (although significantly smaller) is similar to the Tourist treebank in scope and domain.

[9]More specifically, the sub-set corresponds to a previous version of the Jotunheimen treebank, which is one of several corpora that together make up the Tourist treebank, as described in Section 4.4.1.

underestimate performance. As described in Section 6.1, the purpose of the n-fold arrangement is only to help us make better use of our data during development. After settling for the best performing model configuration, a new model is trained on the entire data set, which will then be used for final evaluation on the held-out test data. Extrapolating on the basis of Figure 7.7, we can perhaps hope to gain almost a full percentage point in additional accuracy when training on the full data set. Of course, the flip-side to this story is exactly that it implies testing on held-out data. It seems reasonable to expect that our performance figures for the held-out data will drop somewhat compared to the development set for which we have carefully tuned our models. Still, even this effect should be counteracted to some degree by the cross-validation approach, where variation in the data is naturally taken into account even at the development stage by taking turns testing on different chunks of the data. At any rate, we will get a more accurate impression of the generalization capabilities once we get to Chapter 8, where all our realization rankers will be evaluated on the held-out test set. In the next section, however, we turn our attention from the number of *training items* in our models, to the number of *features*.

7.2.3 Frequency Cutoffs

For a reasonably sized treebank, the number of instantiated features produced by the feature templates easily grows quite large. For the generation treebanks that comprise the primary Tourist development data, consisting of 3921 items with a total of $195,242$ distinct realizations,[10] the feature templates described in Section 5.5 yield a total of $1,182,587$ instantiated feature types. For the IS-underspecified version of the data set, totaling 4720 items, the number of distinct feature types is $1,921,636$. Now, many of these features will never be relevant to the discriminative training, as they never appear in a way that can help us discriminate between the various realizations. If a given feature type happens to take on the same value for all of the candidate realizations available for a given MRS, it cannot aid the model in separating between the preferred and non-preferred candidates for that item. Moreover, since we in Section 7.2.1 decided to discard some of the feature templates from our MaxEnt ranker altogether, the space of candidate features for the development set is brought down to $336,613$.

In the experimentation environment in [incr tsdb()], we implement a simple frequency-based cutoff by removing features that are observed as *relevant* less than c times. We here follow the approach of Malouf and van Noord (2004) where *relevance* of a feature is simply defined as taking on a different value for any two

[10] Recall that each data item is taken to correspond to a given MRS together with its set of generated realizations.

competing candidates for the same input. A feature is only included in training if it is relevant for more than c items in the training data.

Cutoff	Features	Accuracy
0	336,613	71.85
1	255,648	71.85
2	104,905	71.15
3	59,629	70.45
4	41,414	70.06
5	31,563	69.77
10	13,908	67.20
25	4,614	63.41
50	1,933	59.11
100	694	54.16
250	125	40.73

Table 7.6: The effect on performance when changing the value of the **relevance cutoff**, tested on the Tourist development data. The column *Features* shows the average number of features included across the ten folds for each experiment, using the final feature configuration described in Section 7.2.1 (i.e. 3-level grandparenting and preterminal trigrams).

Table 7.6 shows the effect on the accuracy of the MaxEnt model when varying the cutoff level. The second column shows the average number of features included in the model throughout the ten-fold cross-validated experiments. Note that the first row, specifying a cutoff of $c = 0$, corresponds to including all extracted features without considering relevance at all. Note that the performance of this model is identical as for the model using a cutoff of $c = 1$. The reason is, of course, that the features in the complement set of these two models can never contribute to discriminate between competing realizations. Looking further down the list, we see that a model can be compacted quite aggressively without sacrificing much in performance. For the first increments we see that the model size can be cut roughly in half while only giving up a bit more than half a percentage point in accuracy. Even for the model with the cutoff set to $c = 100$, the MaxEnt ranker still outperforms the n-gram ranker we developed in Section 7.1.

As described in Section 7.2.1, we ended up not using the full set of features (i.e. omitting active edges, constituent weights, etc.) in our MaxEnt ranker, and the number of features is therefore not painfully high. In a model using the full set of templates defined in Section 5.5, the savings in terms of feature economy when using a cutoff of $c > 1$ would have been more important. However, in our current feature configuration, using 3-level grandparenting and LE type trigrams, the total

number of features is already at a manageable level, and we are not prepared to give up even the 0.7 percentage point in accuracy incurred by going from $c = 1$ to $c = 2$. The difference between the experiments using these two cutoffs are also detected as statistically significant for all our three evaluation metrics.[11] In other settings however, our priorities might be different. For instance, in a context where the model is to be exported using the full symbol table (see Section 6.2), it would potentially be more important to cut down on the number of features. This is the case, for example, for the standard MaxEnt models for realization and parse ranking included in the LOGON source tree distribution. These models are included as convenient default models intended for use during development, and not in a performance-critical application setting. Each time one of the many LOGON developers loads the source distribution, the full symbol table specifying the ranker needs to be read in order to import the model (along with all the other default settings and resources that are imported when loading the system). Given the results in Table 7.6, it seems reasonable to choose a more light-weight model for the distribution default, for example using $c = 3$ and thereby cutting down the number of features to almost one-sixth. This model takes up less space in the source repository and can be imported quickly, but it still provides near-optimal ranking performance.

7.2.4 Tuning the Prior

In section Section 2.3.3, we described the common practise of *regularization* in relation to log-linear models. The most commonly used method, which is also the method we will be using here, is to impose a prior Gaussian distribution on the λs, as shown in Equation (2.33). This has the effect of adding a quadratic penalty to each feature weight, thereby penalizing the likelihood function if the values of the weight parameters are too extreme. This reduces the risk of overfitting during training, which can lead to poor generalization abilities for the model. Unfortunately, it is usually not possible to know *a priori* or analytically what constitutes a reasonable value for the variance parameter of the prior, so this has to be empirically determined through experimentation.

In the initial phases of tuning a model, we usually perform a rather crude search, testing different values of σ^2 by large increments. We also noted this in Section 7.2.1 when we explored different feature configurations. In Tables 7.7 and

[11] For the model trained using a relevance cutoff of $c = 1$, the exact match accuracy on the Tourist development set is 71.85%. For the model using $c = 2$ the accuracy is 71.15%. The p-value for this difference according to the sign test is 0.0087. For the string similarity metrics the scores for $c = 1$ are NEVA = 0.942 and WA = 0.941, while for the model using $c = 2$ we find NEVA = 0.941 and WA = 0.938. When applying the Wilcoxon signed-rank test these differences receive p-values of 0.0026 and 0.0007 respectively. For all metrics we find that $p < \alpha = 0.05$.

Variance	Iterations	Accuracy
10^{2}	2.0	61.05
10^{1}	3.0	61.05
10^{0}	5.0	61.47
10^{-1}	9.5	64.96
10^{-2}	22.5	69.67
$\mathbf{10^{-3}}$	**55.2**	**71.85**
10^{-4}	127.4	71.38
10^{-5}	253.6	71.15

Table 7.7: The effect on performance when changing the value of the σ^2 parameter for the Gaussian prior on the MaxEnt parameter weights. The experiments are carried out by ten-fold cross-validation on the Tourist development data and the middle row shows the average number of iterations in each run.

Variance	Iterations	Accuracy
1×10^{-5}	55.2	71.85
9×10^{-4}	57.8	71.91
$\mathbf{8 \times 10^{-4}}$	**58.8**	**72.11**
7×10^{-4}	64.4	71.94
6×10^{-4}	66.0	72.02
5×10^{-4}	69.1	72.00
4×10^{-4}	74.7	71.84
3×10^{-4}	83.9	71.82

Table 7.8: Fine-tuning the σ^2 parameter of the prior by increments of 1×10^{-4}. Test results are for ten-fold cross-validation on the Tourist development data.

7.8 we see the results of tuning the variance of the prior for the final feature configuration we settled for in Section 7.2.1. In the first round of search we take large steps in order to determine what is roughly the right range of the value. Table 7.7 shows the effect on accuracy and number of iterations spent by TADM in estimation, when successively changing the variance by an order of magnitude. The figures are averages over a ten-fold cross-validation cycle. Then, having narrowed down the area of what seems to hold a good value (somewhere between 10^{-2} and 10^{-4}), we further fine-tune the parameter by increments of 1×10^{-4}. As shown in Table 7.8 we finally zoom in on a value of $\sigma^2 = 8 \times 10^{-4}$, resulting in an exact

match accuracy of 72.11%. Note that, the extra boost in ranking performance that we achieve by this final step of fine-tuning is indeed significant (at the 0.05-level) when compared to the best performer in the initial and more coarse-grained runs of Table 7.7. Note that we also find the ultimate peaks for the NEVA and WA scores at this same variance value, ticking in at 0.943 and 0.941 respectively. However, these peaks are more stable across the span of parameter values within the range found during the initial phase of the search.

MaxEnt ranking results the Tourist development set				
	Primary		IS-underspecified	
	top	5-best	top	5-best
Accuracy	72.11%	88.82%	68.05%	88.25%
WA	0.941	0.984	0.889	0.974
NEVA	0.943	0.985	0.919	0.980

Table 7.9: Results for the best performing MaxEnt model as described above, tested on both versions of the Tourist development set through ten-fold cross-validation. In addition to recording local subtrees of depth one, the model includes trigrams across the lexical types of the preterminal yields, as well as three levels of grandparenting (i.e. upward dominating nodes). The variance of the prior is globally set to 8×10^{-4}. The column for 5-best lists reports scores that are maximized over the 5 top-ranked candidates, as also described in Section 6.3.

As is clear from these tables, tuning the prior has a major impact on model behavior, both in terms of the number of iterations it takes to reach convergence and in terms of the ranking performance of the resulting model. It is also worth noting that what seems like the best value for the variance is usually also pretty stable across the different versions of our development set. For the feature configuration used here, the same value of $\sigma^2 = 8 \times 10^{-4}$ is also where we find the peak in performance on the IS-underspecified profiles, giving an exact match accuracy of 68.05% (NEVA = 0.919, WA = 0.889).

In Section 7.1.7 we included a table where the exact match accuracy of the n-gram LM ranker was broken down into bins according to the degree of indeterminacy, i.e. the number of candidate realizations generated per input MRS. Table 7.10 shows the same detailed view for our treebank-trained MaxEnt ranker. Side-by-side with the accuracy of the MaxEnt model, we also include the accuracy of the n-gram LM, as well as the random choice baseline, for reference. The last column of the table also indicates how much the MaxEnt model manages to reduce the relative error rate with respect to the LM. Across all bins we see that the MaxEnt model provides a rather dramatic improvement over the language model.

MaxEnt results on the Tourist development data

Bin	Items	Words	Trees	BL	LM	ME	RER
$100 \leq n$	369	27.5	360.0	3.33	25.07	37.83	17.03
$50 \leq n < 100$	230	24.7	73.5	6.64	31.50	48.19	24.37
$10 \leq n < 50$	1144	20.6	22.5	17.08	43.25	65.92	39.95
$5 \leq n < 10$	868	15.3	6.9	32.13	59.70	78.88	47.59
$1 < n < 5$	1310	13.9	3.2	45.64	70.99	86.89	54.81
Total	**3921**	**18.1**	**47.3**	**28.05**	**53.75**	**72.11**	**39.70**

MaxEnt results on the Tourist dev. data underspecified for IS

Bin	Items	Words	Trees	BL	LM	ME	RER
$100 \leq n$	966	24.4	468.0	1.58	23.29	44.91	28.18
$50 \leq n < 100$	433	20.9	71.8	4.63	29.54	52.16	32.10
$10 \leq n < 50$	1607	17.2	23.8	10.83	45.41	67.24	39.99
$5 \leq n < 10$	842	12.1	6.9	25.02	64.58	79.48	42.07
$1 < n < 5$	872	11.3	3.3	41.82	84.35	92.05	49.20
Total	**4720**	**17.0**	**112.3**	**16.62**	**50.04**	**68.05**	**36.05**

Table 7.10: Detailed view of the accuracy of the MaxEnt treebank model on both versions of the Tourist development data, computed after ten-fold training and testing. The data items are binned relative to their number of realizations, as specified by the first column. The next column, *Items*, lists the number of items within each bin, while *Words* gives corresponding the average string length. *Trees* is the average number of candidate realizations in the bin, and *BL* is the expected accuracy for the random choice baseline. *LM* is the accuracy of the n-gram language model, *ME* is the accuracy of the MaxEnt model, and finally *RER* is the relative reduction in error rate for the MaxEnt model with respect to the LM.

Looking at the right-most column in Table 7.10, considering data bins with increasing degree of generator indeterminacy, we see that the overall drop in the relative error reduction is less steep for the underspecified data than for the primary version. Moreover, when comparing the two different development sets, we see that the distribution of items with respect to average number of realizations is quite different. The underspecified version has quite a few more items sorting under the more "ambiguous" bins, which are also the bins that are associated with the lowest baseline. Recall from previous discussion that the items

for which we see a higher degree of indeterminacy during generation are typically also items for which the candidate realizations have a longer average string length. Of course, this correspondence is not very surprising. As the sentence length increases, so does typically also the number of choice points and the possibilities for non-determinism during generation. Now, when looking at the error rate for the bins with the higher degree of indeterminacy we see that the relative reduction is actually greater for the underspecified version. When discriminating between the candidates in these bins, information about non-local dependencies is likely to be more important, and the treebank features of the MaxEnt model seem to offer an advantage over the LM.

In the next section we will present some of the insights gained from a manual inspection of the errors made by both the MaxEnt model and the language model. When looking at the degree of overlap between these errors, we find that quite a few of them are unique to the individual models. For the 3921 items in the Tourist development data, more than 53% of the errors made by the LM are unique to that model (when compared to the MaxEnt ranker). Turning the tables, more than 38% of the errors made by the MaxEnt ranker are not made by the LM ranker. In order to shed light on the particular types of errors committed by the different models, a comparative error analysis was conducted. Some of the main results from this analysis, which represents a joint effort between Prof. Stephan Oepen and the author, are presented in Section 7.2.5 below.

7.2.5 Comparative Error Analysis

The main focus of the manual error analysis was to find out to what degree there are any substantial differences in the errors made by the LM ranker and MaxEnt ranker. Moreover, we wanted to see if there exist any systematic patterns with respect to the particular types of these errors. Just to be clear, the LM that we use is the 4-gram model defined in Section 7.1.7, trained on the BNC using Witten-Bell discounting and the 25k domain-adapted vocabulary. The realizations are then scored according to the log-probabilities of their surface strings. The MaxEnt ranker corresponds to the conditional model defined in the preceding sections, using a relevance cutoff of 1, and including 3-level grandparenting features and trigrams over lexical types for the preterminal yields, as described in Section 7.2.1.

The selection of items included for the comparative error analysis was extracted by taking a random sample of items in the development data for which the two models disagrees. furthermore, for each item we require that one of the models succeed in giving top rank to a candidate matching the reference while the other model fails. In other words, all errors in the sample are unique to one of the respective rankers, i.e. they can be thought of as selected using a logical

Language Model Error Types

Description	**%**	**Count**	**Sub-Types**
Reordering of modifiers to verbal projections	48%	38	PPs (15), lexical adverbs (12), subordinate clauses (6), PP and adverbial phrase (3), verb particles (2)
Lexical choice	28%	22	Orthographic variants (8), relative pronouns (6), realization of temporal preposition (2), numeric instead of literal number (3), other (3)
Reordering of modifiers to nominal projections	21.5%	17	Post-head positioned derived adjectives (7), reordering of simple adjectives (4), PPs (2), other (4)
Other	2.5%	2	
Total	100%	79	
Candidate sounds as good as the reference	16.5%	13	Orthographic variants (5), reordering of title construction (2), adverb reordering (1), literal vs. numeric form (1), reordered VP-modifying PPs (1), PP and AdvP reordering (1), subordinate clause reordering (1), overt temporal preposition (1)
Annotation errors	12.5%	10	post-VP PPs (7), other (3)

Table 7.11: Summary of the main error categories for test items where the error is unique to the LM ranker (when compared to the MaxEnt ranker). Based on manual error analysis of a random sample of items.

XOR (exclusive or) operation with respect to exact match accuracy. The sample includes a total of 122 items from the development treebank. Given the fact that the LM has a significantly lower accuracy than the MaxEnt model on this data, the sample naturally ended up comprising a higher number of LM errors. All in all, the sample includes 45 items where there is a mismatch for the candidate chosen by the MaxEnt model, and 77 items where there is a mismatch for the LM.

After manually inspecting all the items in this sample, the errors appear to be distributed over three main categories; *reordering of modifiers to nominal projections*; *reordering of modifiers to verbal projections*; and errors related to *lexical choice*. For each of these categories we further singled out several more spe-

cific sub-types of errors, such as PP reordering, adverb reordering, etc. The results of the analysis are summarized in Tables 7.11 and 7.12 below, enumerating the main error types observed for the respective models. For each of the main error categories we report the count of observed errors as well as the corresponding percentage. We also report the count (in parentheses) for each of the sub-types in the right-most column. Note that, because a given item might contain more than one type of error, the total count of observed errors is higher than the total count of items in the sample. Furthermore, the bottom part of each table lists some cases for which there is reason to further qualify the errors: For some of the items it can be argued that the top-ranked candidate provides an equally good output as the reference, even though the two do not match. For other items we found that there are errors in the annotation[12] in the treebank. It should be emphasized that, especially with respect to the latter two categories, both the counts and the percentages should be regarded as somewhat approximate, as the distinctions involved are often not clear-cut. Over the following paragraphs we present a range of examples of the different sub-types of errors listed in Tables 7.11 and 7.12.

MaxEnt Error Types

Description	**%**	**Count**	**Sub-Types**
Reordering of modifiers to verbal projections	32.5%	16	Reordering of subordinate clauses (9), lexical adverbs (4), PPs (3)
Lexical choice	30.5%	15	Relative pronouns (7), prepositions (2), other (6)
Reordering of modifiers to nominal projections	22.5%	11	Reordering of PPs (8), other (3)
Other	14.5%	7	
Total	100%	49	
Candidate sounds as good as the reference	14.5%	7	realization of relative pronoun (4), subordinate clause (2), choice of preposition (1)
Annotation errors	22.5%	11	post-nominal PPs (4), post-verbal PPs (1), other (6)

Table 7.12: Summary of the main error categories for test items where the error is unique to the MaxEnt ranker (when compared to the LM ranker). Based on manual error analysis of a random sample of items.

[12]Note that the sub-types listed for the annotation errors in Tables 7.11 and 7.12 do not necessarily coincide with the type of the annotation error itself, as it rather refers to the type of mismatch reported for the corresponding treebank item in the same table (above the line).

Looking at Table 7.11, we see that almost half of the errors committed by the LM are related to the reordering of modifiers to verbal projections. Within this loose category again, we find that reordering of *prepositional phrases* (PPs) is the most common form of error. Furthermore, the majority of these cases involve chains of *locatives* or *directionals* on the same head. Of course, this is something we find many examples of in our Tourist texts, which mostly take the form of area guides for mountain hiking. A typical problem with the candidates incorrectly chosen by the LM in these cases, is that they fail to respect general length preferences and attach the longer modifier phrase closer to the verb. An example of this type of error is shown in Example (7.11) below. Following the item identifier from the treebank, each example first shows the candidate labeled as gold in the treebank, and then the top-ranked candidate according to the model.

(7.11) *Item 271*
Gold: Buyers traveled around to villages in winter and bought animals.
LM: Buyers traveled to villages around in winter and bought animals.

We see that the candidate chosen by the LM ends up having an odd reordering of adverbial PPs. In this example, *around* is a lexical PP, so there are three modifiers on the verbal projection, and although the model gets the temporal PP correct in final position, it ends up preferring *to villages* before *around*. Taking a quick look at frequencies[13] in the BNC for some of the n-grams for which the two sentences differ, it is easy to see how the language model arrived at its preference. For example, while the bigram *traveled around* occurs only 18 times in the training corpus, the bigram *traveled to* occurs a total of 468 times. Similarly, while *villages in* occurs with a frequency of 204, *around in* has a frequency of 1106. It is plain to see how the frequency counts underlying the likelihood estimates of the language model are in favor of the reordered candidate in Example (7.11).

Note that the bottom row of Table 7.11 also lists some cases for which there appears to be *annotation errors* in the original parse treebank. For the LM mismatches, we find that roughly 10–15% of the items involve annotation errors. These are cases were the set of candidate sentences have been generated on the basis of a semantic analysis that is at least partly incorrect with respect to the sentence that we define as the reference. Naturally, this can greatly alter the odds of the ranker selecting a realization that matches the reference. Now, as shown in the sub-type column for the annotation errors, we find that the majority of these cases are related to reordering of post-verbal PP modifiers, i.e. exactly the type of LM error discussed above. A simple but fairly typical example that illustrates the

[13]Note that when we here talk about frequencies of occurrence in the BNC, we are actually referring to a normalized version of this corpus, as described in Section 7.1.1. Among other things, this normalization includes a conversion of standard spelling variants, such as *travelled* → *traveled*.

problem is item 12081 from the development data (we only display the relevant initial part of the sentences):

(7.12) *Item 12081*
Gold: The lodge is located in Lom Township in Oppland [...]
LM: The lodge is located in Oppland in Lom Township [...]

In this example we find two locatives on the same (verbal) head. Arguably, however, the second locative should actually have been attached to the first, given that the township is in Oppland. An analysis with this attachment would have reduced re-ordering variation, and thereby increased the chances of the model picking the correct realization. Although this is a rather innocent example, it nonetheless goes to illustrate the general case of how annotation errors in the original parse treebank are carried over into the generation treebank, altering the premises of ranking task. We see that the possible orderings that exist among the generated candidates are different from what they would have been given a different analysis, which means that this particular error would not have occurred. Of course, this is not to say that the ranker would automatically have been guaranteed to pick the reference if only the analysis had been corrected, only that it would have had another candidate set to choose from. Also for the items listed for the MaxEnt model in Table 7.12 have we noted several such annotation errors, and we will return to this issue below. It is worth noting, however, that while post-verbal PP reorderings comprise the single most common error type for the LM ranker (for the sample considered here at least), these are the same items that make out the majority of the cases for which we noted possible annotation errors.

Returning to Table 7.11 again, we find that the second most common subtype of the reordering errors for modifiers of verbal heads concerns adverbs. The following is a fairly typical example.

(7.13) *Item 63721*
Gold: So you should preferably try Kjerag further up the fjord.
LM: Preferably, so you should try Kjerag further up the fjord.

As we see, the LM prefers to have the lexical adverb in a disjunctive position at the beginning of the sentence. This error actually seems to form part of a general pattern that we also noted in Section 7.1.3, namely that the LM seems to be quite fond of placing the adverbs at the boundaries, either at the end or the beginning of the sentence. A similar case is shown in Example (7.14) below. This time the error concerns reordering of a PP and an adverbial phrase, but again we find the adverb in an outer position, this time sentence-final. For most of these errors it is clear how the corpus frequencies dictate such a preference. For example, in the case of the sentence-final *rapidly*, this word form is actually found in this position

in roughly ten percent of its occurrences in the training data. Naturally, this results in a relatively high probability for the trigram "*rapidly . </s>*", which is what the last sub-sequence of the top-ranked string in Example (7.14) actually looks like when presented to the LM.

(7.14) *Item 30362*
Gold: It is unwise to hike too rapidly through such surroundings.
LM: It is unwise to hike through such surroundings too rapidly.

During the analysis of the error sample, we often found that one great advantage of the n-gram-based language model, is that it is relatively easy to explain the rationale behind many of the model's decisions. They are by definition based on simple frequency counts from the training data, and this is something which is very easy to inspect. For the MaxEnt model, the underlying reasons for some of the errors are often much more opaque.

Staying within the category of verb modifier reorderings, we also find that the LM has a few errors related to the *reordering of subordinate clauses*, typically *if* or *as* clauses. This is also the sub-type of errors that we find to be one of the most common ones among the MaxEnt errors. Although the overall category of reordering of modifiers to verbal projections is the largest category also for the MaxEnt errors, the mismatches sampled for this learner seem to be more evenly spread out among our main error categories. Moreover, within this overall category, Table 7.12 shows that the sub-types of errors here have a quite different distribution than those of the LM. We find that reordering of lexical adverbs and PPs are much less prominent errors in the MaxEnt set (when looking at verb modifiers). Reordering of subordinate clauses, on the other hand, is a reoccurring error. It is perhaps worth reminding ourselves at this point that the error items we are looking at here are all *complementary* or *contrastive*, in the sense that we are only considering items for which one of the learners failed to select the reference while the other did not. Given that this set contains quite a few subordinate clause reorderings for both learners, we can expect this error type to stand out even more when also looking at the intersection of their errors. An example of one of the MaxEnt reordering errors is given in Example (7.15) below.

(7.15) *Item 12902*
Gold: Just after you have started out from Memurubu, the route crosses Muru via a solid bridge.
ME: The route crosses Muru via a solid bridge just after you have started out from Memurubu.

Recall that we are looking at errors sampled from the primary version of the development data here, not the IS-underspecified version as we used for the error

analysis in Section 7.1.3. However, the sentence-initial PP in the original (*Gold*) sentence of Example (7.15) is not analyzed as topicalized, hence there is no IS marking in the MRS, and hence both clause orderings are possible variants. It seems that the MaxEnt model in this case preferred the less marked of the variants, with the modifier following the VP. This item is also a good example of a case where the exact match criterion seems to be too stringent, as it can be argued that either variant seems equally acceptable. As can be seen in Tables 7.11 and 7.12, there are in fact several such cases in the sample, where the models select a candidate that we believe can be considered equally good as the realization that is labeled as gold. For the MaxEnt case, we see that the (minor) majority of these cases are items where the mismatch is related to how to best realize a *relative clause*. Item 33143 provides an example:

(7.16) *Item 33143*

Gold: In nice weather you will have a fantastic view of the area you are then descending into.

ME: In nice weather, you will have a fantastic view of the area into which you are then descending.

In Example (7.16) we see that there is a choice between using an overt relative pronoun or not, and correspondingly whether or not the preposition is stranded in the final position. In our view, this is another example where the candidate chosen by the MaxEnt ranker is at least as good as the reference. Yet another example is treebank item 120042, where the mismatch is related to the choice of preposition. Here the MaxEnt model prefers the sub-phrase *information on fishing spots*, while the reference uses *information about fishing spots*. Both variants have identical semantics, with *information* as a relational noun. In other words, the selection of preposition is here not determined by the semantics, and for this item we believe *on* and *about* to be equally good alternatives.

In the case of the language model—still looking at the items where we judge the top-ranked candidate to be just as good as the reference—we find the most common source of such mismatches to be *spelling variants* or *orthographic variants*. Mostly, these are trivial instances of lexical choice errors. For example, while the LM shows a consistent preference for the hyphenated forms of compass directions such as *south-east* and *north-west*, the reference data has an equally consistent preference for the concatenated forms *southeast* and *northwest*. This leads to several mismatches between the top-ranked LM candidates and the reference, penalizing the accuracy score. It seems that the language model is much more susceptible to commit such errors than the MaxEnt model. The reason, of course, is simply that the latter is trained on the same type of treebank data that we are testing on, making it much more sensitive or tuned to these preferences in the reference data.

To give another example of such an error that seems to reflect a rather spurious preference in the reference data, consider items 14852 and 22632 from the development treebank. Here the mismatch is incurred by a reordering of ERG's so-called title construction: While the LM wanted *lake Gjende*, the reference specified *Gjende Lake*. Again we find that the top-ranked LM candidate is as good as, if not better than, the reference.

It is worth noting that there are actually quite a few MaxEnt errors that came very close to being included in the count of "false" mismatches in Table 7.12, but which we in the end chose to treat as just plain errors. Many of these mismatches involve, for instance, adverbial reordering as in Example (7.17) below, but where the MaxEnt preference would have sounded fine had it only included some additional commas.

(7.17) *Item 21661*

Gold: The needs of the bipeds also changed as communications developed.
ME: As communications developed the needs of the bipeds also changed

A comma inserted after *developed* would have made the MaxEnt choice in Example (7.17) much better. As a general impression from looking at the MaxEnt errors, it seems that the top-ranked candidates are often somewhat short on punctuation. Recall from Section 5.3.2 that punctuation actually is ignored when labeling references in the generation treebanks. First and foremost this is due to how punctuation is currently implemented in the generation process. As of now, punctuation is controlled by the grammar, and we see that distinct realizations of the same semantics often end up differing only with respect to, for example, optional commas. In order to hold such cases of spurious indeterminacy outside of the proper ranking task, we currently disregard punctuation in the treebanking. This means that yields that differ only with respect to punctuation will be treated as equivalent when identifying the gold realizations. Given that this data also is the *training* material for our discriminative learners, this means that they end up being somewhat weak with respect to punctuation. Note that, given that this information is ignored when labeling references, this tendency does not negatively affect the exact match accuracy. However, in the rounds of manual error analysis, it does mean that we sometimes see a mismatch for a top-ranked candidate that we migth otherwise have judged to be equally good as the reference, were it not for the lack of proper punctuation. Note that the situation here is different for the language model. As described in Section 7.1.2, the LM was trained with punctuation intact and treated as separate tokens in the training data, because we found that it generally helped the model in making more accurate predictions. Given that commas (and other punctuation marks) are naturally very high-frequent token types in the training data, the LM is typically quite fond of having them around (even though we are not using any length penalty in the likelihood scores).

Example (7.16) above showed an item where a mismatch was incurred by how the MaxEnt model preferred to realize a relative clause, opting for an overt relative pronoun. As can be seen in Tables 7.11 and 7.12, under the main category of *lexical choice*, there is a relatively high number of similar cases contained in our random sample of errors. As a note on categories, although an incorrect choice of relative pronoun can be understood as an error related to nominal modification—relative clauses being post-modifiers of nominal projections—all the other errors bundled under that category concern *reordering*, and hence we count it as a lexical choice error instead. Now, most of these errors concern the actual choice of relative pronoun (not whether or not to make it overt), e.g. *who* instead of *that*, *that* instead of *who*, *that* instead of *which*, and *which* instead of *that*. Especially the latter is a recurring error for the LM. In general however, we see that incorrect choice of relative pronoun is actually one of the error types that occur relatively often for both models. For the MaxEnt model, including lexicalized treebank features should in theory be able to help address the problem, especially for the *who* vs. *that* errors. However, lexicalization is something that typically requires very large amounts of training material in order to be useful and not just increase problems with overfitting. It is also worth noting that there exists several works that specifically targets the problem of realization of relativizers. For example, Wasow, Jaeger, and Orr (2005) have studied frequency effects on the omission of relativizers, and have even experimented with MaxEnt classifiers for the prediction of *that*-less relative clauses.

On the other hand, it is also worth observing that the underlying grammar does not currently distinguish between *restrictive* and *non-restrictive* relative clauses. In many cases this is, of course, the underlying reason for why the choice of relative pronoun is left up to the ranker in the first place. This fact also goes to illustrate a more general point, which many of the errors mentioned in this section touch upon, namely the constant trade-off that exists between, on the one hand, making the grammar more specific, and on the other, delegating more of the responsibility to the statistical ranker. The last few examples above have all been taken from the category of lexical choice in Tables 7.11 and 7.12. Some of the remaining errors in this category among the LM items which we have not yet mentioned involve choices such as whether to use a numeric form or a literal number (e.g. *3* vs. *three*), whether to use an overt temporal preposition (e.g. *happen the same day* vs. *happen on the same day*), and abbreviations (*southeast* vs. *SE*). The next main error category in the tables is *reordering of modifiers to nominal projections*. For both the MaxEnt model and the LM ranker we see that roughly 20-25% of the errors fall within this category. However, the distribution of the specific sub-types of errors that we find within this category appears to be quite different for the two learners. For the LM we find that the most common cause of mismatches in this category is *reordering of adjectives*. A surprisingly frequent

error is exemplified by the noun–adjective reordering in item 63751 below:

(7.18) *Item 63751*
Gold: A marked trail shows the way to the plateau above Preikestolen itself.
LM: A trail marked shows the way to the plateau above Preikestolen, itself.

Now, syntactically a noun modifier can follow the head when it is sufficiently complex. Moreover, the ERG treats all derived adjectives as complex, and thereby as possible post-modifiers. It turns out the LM has an unfortunate preference with respect to this construction. The LM appears to persistently opt for having the derived adjectives in post-head position, resulting in odd sounding NPs like *excursions guided*, *woodcutting reduced*, *routes marked*, *stones cleared*, *mountain birch trees twisted*, etc. There are also other cases of adjectival modification where the LM seems to easily stumble, particularly the ordering among prenominal attributive adjectives. An example is found in treebank item 62792, where the LM prefers *stony smooth surfaces* instead of *smooth stony surfaces* as specified in the reference. This is a classic example of a problem where different variants are associated with different degrees of markedness, but where it is very difficult to define any hard constraints that singles out the variant that sounds most natural. In the context of NLG, Malouf (2000) presents a combination of ML methods that specifically targets the problem of adjective ordering, combining methods such as a bigram model, positional probabilities and memory based learning.

Judging from our random sample of errors from the development data, the MaxEnt model seems to fare much better when it comes to adjectives. Instead, the MaxEnt mismatches in the category of nominal head modifiers seem to most often be related to *reordering of PPs*. However, if we now look to the bottom row in Table 7.12, we see that the situation is similar as for the post-verbal PP reordering errors that we noted for the LM earlier. The items that appear to have an incorrectly treebanked parse are most often also the items where we observe a reordering error for post-nominal PPs. As an example, consider the following sequence of PPs taken from development item 393, describing the direction of a hiking trail:

(7.19) *Item 393*
Gold: [...] from Besseter in Sjodalen, over Besseggen, to Memurubu.
ME: [...] from Besseter over Besseggen in Sjodalen to Memurubu.

In the underlying parse treebank, *Besseter* is modified not only by *in Sjodalen*, but also (incorrectly) by *over Besseggen*. This is what opens the door to the reordering we see in the MaxEnt candidate above. However, this example also shows how

there is often a blurred line between what can be considered as annotation errors and what are practical solutions to limitations in the underlying grammar. For instance, the annotated structure in Example (7.19) might be related to the ERG analysis of *from / to* as a ditransitive preposition. Under that analysis, there is no room for an intervening adjunct, i.e. *over Besseggen*, hence it is forced into the first NP jointly with *in Sjodalen*.

Recall that the annotation errors presented above all have to do with errors in the parse for the reference string as it appears in the original (parse) treebank. As described in Section 5.3, our generation treebank is subsequently constructed on the basis of this treebank. Now, in relation to experiments with statistical *parse selection* on the Redwoods treebank, Toutanova et al. (2005) present an error analysis that contains several results that parallel those that we have presented here. Recall from Section 5.2 how the LOGON treebanks instantiate the same approach to treebanking as Redwoods, and the annotations underlying the experiments of Toutanova et al. (2005) are also based on (an earlier version of) the ERG grammar. Inspecting the errors made by their conditional log-linear parse selection model, Toutanova et al. (2005) conclude that roughly 62% of the cases are "real errors", while the others are associated with annotation errors in the treebank. They also noticed that their log-linear model seemed to be more susceptible to such annotation errors than the generative PCFG models that were applied to the same data. As can be seen from Tables 7.11 and 7.12, our discriminative MaxEnt model seems to be much more sensitive than the LM when it comes to annotation errors. We see that although roughly 22.5% of the errors made by the MaxEnt models concern items for which there is an annotation error, this is only the case for 12.5% of the LM errors. This trend is perhaps not so surprising, given that the MaxEnt learner is both trained and tested (through the ten-fold split routine) on material from the same treebank data, and can also easily adjust its feature weights to fit whatever noise and errors there are in the training data. As also mentioned in Section 2.3.3, the tendency of MaxEnt models to overfit is reinforced by the fact that we have a rather limited amount of training data, while simultaneously operating within a relatively large feature space. Moreover, as also noted by Toutanova et al. (2005), the annotation errors in the development data may also be a contributing factor to saturation effects as discussed in Section 7.2.1 where we experimented with adding more feature types to the model. Toutanova et al. (2005) report similar effects in relation to their MaxEnt models for parse selection.

Toutanova et al. (2005) report that the largest group of errors in relation to the parse selection experiments has to do with PP attachment. This is analogous to our finding that PP reordering, whether they concern verbal or nominal modification, seems to be one of the most common reordering errors committed by both the LM and the MaxEnt model. In the MaxEnt case, part of the motivation behind the constituent weight features was that they should be helpful for exactly such order-

ing choices, where there generally is a preference for placing "smaller" modifier phrases closer to the head that is modifed. However, in the end we decided to leave the constituent weight features out of the final feature configuration, since they did not seem to improve the overall results (as described in Section 7.2.1). However, in the course of the current error analysis we have discovered several annotation errors in the underlying treebank, of which the largest group has to do exactly with PP attachments (as reported in the bottom rows of Tables 7.11 and 7.12). This fact can perhaps partly explain why the constituent weight features failed to make any significant positive contribution.

In many ways, the presence of annotation errors gives reason to be optimistic with respect to the chances of improving the accuracy of the MaxEnt ranker even further. In Section 5.2 we highlighted the dynamic aspect of the Redwoods style treebanks, in the sense that the treebanks can easily be updated to reflect revisions in the grammar as it improves and develops over time, and we can re-construct the generation treebanks to reflect changes in the annotations of the parse treebanks. This was also discussed in Section 5.3 where we presented the notion of symmetric treebanks. Now, when any irregularities in the treebanking are reported, such as those discovered in the error analysis here, these can be corrected in the next revision of the treebank (and possibly also the grammar, depending on the type of error). When subsequently rerunning the MaxEnt experiments on an updated version of the treebank, there should be potential for seeing significant improvements.

In this section we have presented some of the insights gained from a manual error analysis of a random sample of errors made by the LM ranker and the MaxEnt ranker. All errors in the sample are unique to one of the respective models, thus allowing us to compare the strengths and weaknesses of the two modeling frameworks in a contrastive manner. We have seen that the specific items for which the models fail are often non-overlapping, and also that the errors tend to be of different kinds, as summarized in Tables 7.11 and 7.12. A natural consequence of this insight would be to attempt to "join forces" and apply the two models together in combination. Such an attempt is exactly what we turn to in the next section.

7.2.6 Combining Models

As is clear from the error analysis of the previous section, we have good reasons to believe that a *combination* of the LM and the MaxEnt model can lead to better ranking performance than what we are able to obtain with any of them on their own. Given the flexibility of the MaxEnt set-up, combining the two models is easy. In addition to the treebank features described in Sections 5.5 and 7.2.1, we can simply add the score of the 4-gram LM developed in Section 7.1 as a separate feature in the MaxEnt model. In other words, the value of the $d + 1$'th feature

Combined Ranking on the Tourist Development Set				
	Primary		IS-underspecified	
	top	5-best	top	5-best
Accuracy	74.25%	91.80%	70.47%	90.75%
WA	0.944	0.986	0.897	0.978
NEVA	0.946	0.987	0.925	0.983

Table 7.13: Ranking performance of the combined MaxEnt model described above (i.e. using the LM as an additional feature), tested on both versions of the Tourist development set. The variance of the prior is set to 3×10^{-4} for the model trained on the primary data, and 6×10^{-4} for the model trained on the underspecified data.

in the MaxEnt models is the log-probability of the string as given by the n-gram model p_n, i.e. $f_{d+1}(s, r) = \ln\ p_n(y(r))$, where $y(r)$ is the yield of r and $n = 4$, as before.

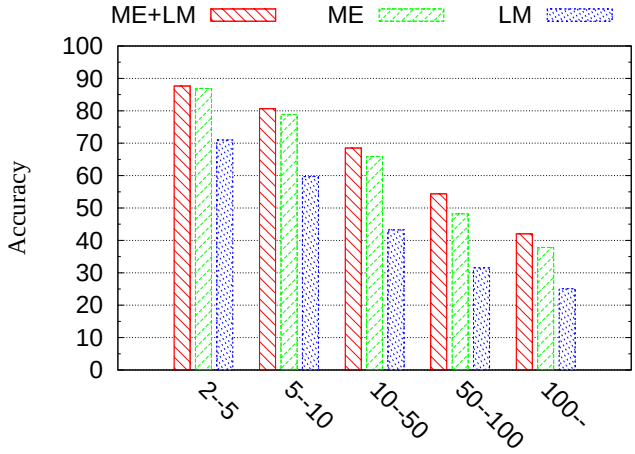
Recall from Section 2.3.1 that the formulation of the more general MEMD models makes explicit the so-called default distribution or reference distribution q_0, as in Equation (2.27). Johnson and Riezler (2000) show an interesting equivalence between using log-probabilities as features and using a weighted geometric mixture of the same probabilities for the default distribution q_0 (where the λ-parameters of the features would correspond to their weights in the mixture). Let us assume a set of k additional features f_{d+j} for $1 \geq j \geq k$, and that they are all of the form $\ln q_j(s, r)$, where each q_j is some probability distribution. In other words, we define $f_{d+j} = \ln q_j$. Johnson and Riezler (2000) show that if we let q_0 be a geometric mixture of the so-called auxiliary distributions $q_1, \ldots, q_k$, defined as $q_0(s, r) = \prod_{j=1}^{k} q_j(s, r)^{\lambda_{d+j}}$, then

$$q_\lambda(s, r) = \frac{1}{Z_\lambda} \exp\left(\sum_{i=1}^{d+k} f_i(s, r)\lambda_i\right) \tag{7.20}$$

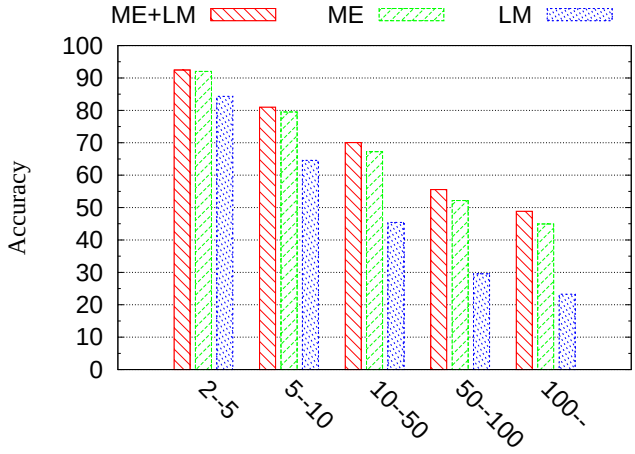
$$= \frac{1}{Z_\lambda} q_0(s, r) \exp\left(\sum_{i=1}^{d} f_i(s, r)\lambda_i\right) \tag{7.21}$$

As determined by the estimation procedure, the weight parameters govern the contributions of the individual auxiliary distributions in the final model. This means that a special case of the simple combined model we present here would be a MEMD model where the uniform distribution q_0 is replaced by the language model p_n. If $\lambda_{d+1} = 1$ then $\exp(f_{d+1}\lambda_{d+1}) = p_n$ and we would effectively have a

MEMD model as described above with $q_0 = p_n$.



(a) Accuracy relative to indeterminacy on the development data



(b) Accuracy relative to indeterminacy on the IS-underspecified development data

Figure 7.8: Accuracy of different models on the Tourist development data. The treebank items are binned according to indeterminacy. The bin '2–5' includes items for which the set of competing realizations is larger than or equal to two, but smaller than five; '5–10' includes items where the candidate set is larger than or equal to five, but smaller than ten; and so on. LM corresponds to the n-gram language model, ME is the MaxEnt model using only treebank features, and ME+LM is the combined model where LM scores are included as a feature in the MaxEnt model.

For our current purposes, however, we simply add the LM scores as a separate feature, and then re-estimate the model as described in Section 7.2.4. On the

Tourist development data, the best performing combined model (using a prior variance of 3×10^{-4} on the λ-weights) achieves a total of 74.25% in exact match accuracy. The NEVA and WA scores are 0.946 and 0.944 respectively. As we can see, integrating the LM as an additional feature in our treebank model provides a nice boost to the performance, resulting in an absolute increase of more than two percentage points in accuracy, up from 72.11%—a difference which is detected as strongly significant by the sign test ($p < 0.00005$). This corresponds to a relative reduction in error rate of 7.67% compared to the MaxEnt model that uses only treebank features. Relative to the language model alone, the reduction in error rate is 44.32%.

The differences in performance are not as visible for the string similarity metrics, although we observe a slight increase there too. However, when compared to the model using treebank-only features, only the difference in the NEVA score is detected as significant using the Wilcoxon signed-rank test ($p < 0.05$). Table 7.13 shows the full evaluation results for the combined model on both development sets, also including scores for the 5-best lists. We see that the results for the IS-underspecified version of the development data are also positive. Compared to the treebank-only model, the combined model offers an absolute increase in exact match accuracy of 2.42%, and a corresponding relative reduction in error rate by 7.57%. Here the increase in both the NEVA score and the WA score are also detected as being strongly statistically significantly. (Across all evaluation metrics the differences in scores receive $p < 0.000005$.) For both data sets we also observe an increase in all scores computed for the 5-best lists, as seen in Table 7.13.

Tables 7.5, 7.10, and 7.14 show the accuracies of the LM, the MaxEnt model and the combined MaxEnt+LM model, respectively, and with data items aggregated into bins according to the number of available realizations. In order to facilitate easier comparison of these results, the same results are plotted as histograms in Figure 7.8. As we can see, the gain in accuracy that we obtain with the combined ranker appears to be highest for the bins with a higher degree of indeterminacy, which typically also contain items with a longer average string length. In other words, the items for which the ranking task is most diffcult, from a baseline perspective at least, are also the items for which we obtain the greatest relative improvement in ranking performance.

7.2.7 Preference Weighted Training Data

Before we round off this section on MaxEnt rankers, we include a brief discussion of our attempts at estimating models from *preference weighted* training data. By this we mean that each candidate is given weight in the model according to its similarity towards the reference, where similarity can be measured in terms of, for example, WA or NEVA. This approach of scoring the different training examples

Combined MaxEnt+LM on the Tourist development data

Bin	Items	Words	Trees	Gold	Baseline	Accuracy
$100 \leq n$	369	27.5	360.0	8.0	3.33	42.01
$50 \leq n < 100$	230	24.7	73.5	4.9	6.64	54.35
$10 \leq n < 50$	1144	20.6	22.5	3.3	17.08	68.49
$5 \leq n < 10$	868	15.3	6.9	2.2	32.13	80.62
$1 < n < 5$	1310	13.9	3.2	1.4	45.64	87.63
Total	**3921**	**18.1**	**47.3**	**3.0**	**28.05**	**74.25**

Combined MaxEnt+LM ranking on the IS-underspecified version

Bin	Items	Words	Trees	Gold	Baseline	Accuracy
$100 \leq n$	966	24.4	468.0	4.4	1.58	48.86
$50 \leq n < 100$	433	20.9	71.8	3.2	4.63	55.54
$10 \leq n < 50$	1607	17.2	23.8	2.2	10.83	70.05
$5 \leq n < 10$	842	12.1	6.9	1.7	25.02	81.00
$1 < n < 5$	872	11.3	3.3	1.3	41.82	92.43
Total	**4720**	**17.0**	**112.3**	**2.5**	**16.62**	**70.47**

Table 7.14: Detailed view of the accuracy of the combined treebank model and language model, tested on both versions of the Tourist development data. The model is a MaxEnt model trained with the scores of the LM as a separate feature, in addition to the treebank features. The data items are binned relative to their number of realizations. The columns are, from left to right, the subdivision of the data, the corresponding number of items, average string length of the realizations, average number of realizations, average number of references, the random choice baseline for accuracy, and finally the accuracy of the combined MaxEnt model.

prior to estimation is also used for some of the SVM rankers that we turn to next in Section 7.3. But let us first take a step back and present some of the underlying motivations for this approach.

In the estimation of MaxEnt models as described above, the goal is to find parameters that maximize the probability of the reference sentences in the training data relative to all the other competing realizations that are generated for each input semantics. In Section 2.3.2, we described this as finding the parameters λ that maximize the conditional likelihood of our training data. Section 2.3 also gave an alternative but equivalent formulation of the estimation problem. When setting

up our training data, we assume that each reference realization has an observed frequency of one, while all the other "non-reference" candidates have frequency zero. These relative frequencies then define the empirical distribution $\tilde{p}$ for which we want to minimize the divergence of the model q_λ with the highest entropy. This means that our training data defines a rather crude division of the training examples. According to the frequency counts in the training data, each "correct" realization is given "weight" 1, while all other realizations are given weight 0. In other words, any generated derivations with a surface string not exactly matching the reference are all considered to be equally bad.

Let us for a moment revisit the issue of *evaluating* the generator output. As discussed in Section 6.3.1 above, one particularly simple measure of performance is to compute the *exact match accuracy*, as we have done many times throughout this chapter. As should be clear by now, this figure is simply the percentage of times that the top-ranked sentence (i.e. the output sentence) is identical with the reference or gold sentence in the treebank. However, as argued in the discussion of evaluation measures above, although the simple exact match measure offers a very intuitive and transparent mode of evaluation, it can also in many ways be considered an overly strict evaluation measure in the setting of natural language generation. In cases where a top-ranked candidate does not exactly match the reference, the candidate might still be appropriate or inappropriate to different *degrees*. As pointed out in Section 6.3, it can therefore be argued that it is potentially both unfair and uninformative to only give credit in cases of an exact match. Of course, this was an important part of the motivation for why we wanted to also include some kind of *similarity-based* measure in the evaluation, for which we here use NEVA and WA.

All of these evaluation measures are more closely described in Section 6.3. The point here, however, is to note how these observations about gradedness migth also apply to how we construct our *training data*. If, as we have just argued, several of the competing candidates might be more or less appropriate, it seems we might be throwing away potentially useful information if we do not take these relations into account already during training. If our training data in no way distinguishes the *really inappropriate* realizations from the *good-but-not-quite-perfect* realizations, then the model we train will perhaps do a less than optimal job at it also. The point is that our training data should represent what we want to learn as closely as possible. It can perhaps be argued that setting up our training data to dis-prefer all candidates except for the perfect match, does not go all the way in this sense.

We might hope to improve our results if we instead assign each training candidate a score representing its "quality" or its degree of appropriateness. Such an implementation is what we describe in this section. The approach we pursue here is similar to what Osborne (2000) and Malouf and van Noord (2004) do in

the context of *parse disambiguation*. In their experiments, the *quality* of each parse x is first measured in terms of its associated set of dependencies, and then this score is mapped to the probability $\tilde{p}(x|w)$ of that parse in the training data (for a given string w). The quality of each parse x is measured according to a (slightly modified) version of *concept accuracy*[14] (Boros et al., 1996) which indicates its similarity towards a corresponding treebanked dependency structure. Each parse is then added to the training set with a "frequency" proportional to this score. These frequencies are then normalized to give the target distributions $\tilde{p}(x|w)$ and $\tilde{p}(w)$ (where each sentence w is weighted proportionally to the sum of the weight of its associated parses x). To sum up, Osborne (2000) and Malouf and van Noord (2004) effectively use a target distribution that indicates *preference relations* rather than frequency or empirical likelihood.

In our case, we can mimic this approach by applying some kind of string similarity metric for scoring the examples prior to training, and then replacing the frequency-based empirical target distribution in Equation (2.30) with a target distribution based on this preference weighting. Of course, the WA and NEVA measures stand out as natural candidates for the job, given that we would probably also want to use the same measure for both weighting and evaluation. In other words, we can define a preference-based distribution $\tilde{p}(s, r)$ based on WA or NEVA, and then estimate the parameters of our model q_λ by maximizing the likelihood of the training data relative to this distribution (otherwise using the exact same approach as for the other MaxEnt models we have developed above). When computing the likelihood function, we then sum over *all* the competing (i.e. both optimal and sub-optimal) candidates for each item in the symmetric treebank. As pointed out by Osborne (2000), this means using a far less discontinuous target distribution where we assign a non-zero probability to all (or most) realizations, effectively weighting the importance of each of them.

During an earlier phase of the thesis work, extensive experiments[15] with the preference weighting approach were performed on somewhat smaller data sets (i.e. 265 items from the *Hike* treebank, as used in Velldal et al. (2004), and 634 items from the *Rondane* treebank, as used by Velldal and Oepen (2006)). Preference weighted distributions were defined using several different string metrics such as METEOR[16], WAFT (Forsbom, 2003), and BLEU (Papineni et al., 2002), in

[14] Defined as a generalization of *word accuracy* (WA) as often used in *speech recognition*, concept accuracy (CA) is used for evaluating speech *understanding* by Boros et al. (1996). Furthermore, the version of CA presented in (Malouf & van Noord, 2004) is analogous to the modified version of WA presented in (Forsbom, 2003), *word accuracy for translation* (WAFT).

[15] These experiments were carried out as part of the self-chosen research assignments in the course on Natural Language Generation offered by the Nordic Graduate School of Language Technology (NGSLT), as taught by Hercules Dalianis (spring 2005).

[16] METEOR is a recently proposed metric for evaluation of MT systems. In some respects

addition to NEVA and WA. We also experimented using different cutoffs on the scores (e.g. disregarding all but the highest scoring candidates), rounding, as well as additional transformations such as defining exponential fall-offs on the similarity scores in order to amplify the influence of the best candidates. We will be revisiting some of these points in the following section on SVM rankers.

In the end, however, we failed to improve our ranking scores using any of these variations, even when evaluating them using the same metric as whichever was used for the initial example weighting. None of the rankers trained on preference weighted data were able to outperform the rankers trained in the traditional way, i.e. using only a binary preferred / dispreferred distinction. As the results for this batch of experiments were rather conclusive, we have not attempted to replicate them for the larger Tourist treebank that we now have available. However, we will be extending parts of the overall approach when we explore the use of SVM rankers in the following section.

7.3 SVM Rankers

The previous section described a method for *preference weighting* the realization candidates prior to training. Using a string similarity measure such as WA, each generated candidate is assigned a quality score based on its similarity toward the reference. Of course, these scores can also be used as a basis for defining an *ordinal ranking* among the competing members within each candidate set. This means that we ignore the actual values of the similarity scores themselves, and instead focus only on the corresponding relative ranks that they define. In this way, each hypothesis is associated with a preference relation toward all the other hypotheses generated for the same semantics. As described in Section 2.4.2, such sets of preference relations are exactly what forms the basis for training *SVM rankers* as introduced by Joachims (2002). Sections 7.3.1 and 7.3.2 below look at various issues related to how such preference relations are defined in the training data. As we shall see, however, we quickly encountered problems due to the complexity of the optimization problem associated with learning SVM rankers for large data sets.

Note that all the SVM rankers that we develop in this section use exactly the same feature configuration as the combined MaxEnt model of Section 7.2.6. In other words, we use the final configuration of treebank features described in Section 7.2.1, including 3-level grandparenting and lexical type trigrams, in addition to the LM scores of the 4-gram model described in Section 7.1.7. In addition to potentially allowing us to learn ranking functions that take account of a richer set

this is rather more elaborate than the other metrics that we have considered, and it optionally includes both stemming and WordNet look-up. For more information on METEOR, see http://www-2.cs.cmu.edu/~banerjee/MT/METEOR/

of preference relations among the competing candidates, the SVM paradigm can also allow us to learn non-linear ranking functions that go beyond the capabilities of log-linear rankers such as MaxEnt (at least the way these models are standardly formulated, i.e. non-kernelized). After discussing a range of different development issues, Section 7.3.3 presents the final results for various SVM rankers on the Tourist generation treebank.

7.3.1 Preference Constraints for Ordinal and Binary Ranks

Let us start by refreshing some of the terminology we introduced in Section 2.4.2. For a given MRS s_i we have a set of generated realizations $\mathcal{Y}(s_i) = \{r_1, \ldots, r_m\}$. Let each realization r_j be associated with a score y_{ij} indicating its similarity toward the reference(s) defined for s_i. This score can for example be computed according to a string-similarity metric such as WA. Now, for each pair of candidates r_j and r_k in $\mathcal{Y}(s_i)$ where $y_{ij} > y_{ik}$, a preference constraint $r_j \prec_i^* r_k$ is added to the underlying optimization problem as defined in Equation (2.53). Each item in the training data will spawn a different set of such ordering constraints. As described in Section 2.4.2, the goal is then to learn an ordering relation $\prec$ which as closely as possible reflects this original ranking $\prec^*$. This is achieved by trying to estimate a weight vector w defining a maximum-margin hyperplane that correctly separates the examples in their corresponding feature space representation $\Phi(s_i, r_k)$. As described in Section 2.4.1, SVMs include a notion of slack variables that allows the learner to approximate a solution where it is not possible or computationally tractable to find a separating hyperplane. Furthermore, Section 2.4.2 described how the SVM ranking task can be formulated as classification task on the difference vectors $\Phi(s_k, r_i) - \Phi(s_k, r_j)$. This results in a ranking function where $w\,(\Phi(s_i, r_j) - \Phi(s_i, r_k)) > 1 - \xi_{i,j,k}$ means that $r_j \prec_i r_k$, i.e. the realization r_j is ranked higher than r_k for the input semantics s_i (Joachims, 2002).

We have tried learning several such ranking functions, using both WA and NEVA for computing the scores y_{ij} (which form the basis of the ordering relations in the training data). Unfortunately, what we discovered was that this defines an estimation problem that is simply too computationally expensive to be solved within the limits of our currently available computing resources. We found that we were not able to complete the SVM estimation on the full development set before the process size would grow too large to fit in the available memory. Recall from Chapter 6 that we are running these experiments on rather powerful machines, e.g. a 64-bit (2.4 GHz AMD Opteron) Linux machine with 32 GB of RAM. Courtesy of the Scientific Computing Group[17] at the University of Oslo, we were even granted

[17]The web pages of the Scientific Computing Group at UiO can be accessed at http://www.hpc.uio.no.

access to run experiments on a machine (*Dalco*, currently out of service) with a full 128 GB of RAM. As we shall see, in cases where we actually did manage to successfully complete estimation, running times would often be on the order of several days, and even weeks.

Let us backtrack for a moment to the preceding paragraph and again look at how we construct the ordering constraints. Joachims (2002) notes that the number of preference constraints added for the optimization problem, as defined in Equation (2.53), is what has the greatest influence on training time. Now, the factors governing how these constraints are constructed in the first place will, of course, be specific to the particular ranking problem and data set. Joachims (2002) and Radlinski and Joachims (2005) focus on optimizing search engines by learning ranking functions on the basis of query logs that describe how users have navigated through previous search results. The problem of ranking generator outputs provides a rather different setting. In our case, if a given input semantics s_i produces a total of n distinct candidates ($|\mathcal{Y}(s_i)| = n$), we will generally have n distinct scores or "labels" y_{ij}. This means that the total number of preference constraints imposed on the model for this given item can be defined as $n(n-1)/2$ (or, equivalently, as the sum of integers from 1 to $n-1$: $\sum_{i=1}^{n-1} i$). Now, in practise the total number of constraints might be somewhat lower, as occasionally we will have candidates where $y_{ij} = y_{ik}$ (for $i \neq j$). In other words, in cases where two candidates for a given semantics should happen to have the same score, no ordering constraint will be added to the optimization problem for this particular pair. How often this happens will depend both on the data and the "granularity" of the similarity metric we use when computing the scores y (e.g., in our experience, WA seems to be more "coarse" than NEVA). Nonetheless, we see that the number of constraints imposed on the optimization problem will grow very quickly when adding items of increasing indeterminacy. Let us for a moment revisit the basic characterizing statistics of our Tourist development data, as summarized in Table 5.1. We have a total of 3921 training items in the generation treebank. The average number of generated candidates per item is roughly 47. Now, if the corresponding variance was low, we might expect this average to give us a reasonable estimate of the number of ordering constraints we could expect to see generated, which would then roughly be on the order of one thousand per item, totaling to roughly $4,000,000$ for the full data set ($3921 \times (47 \times 46/2) \approx 4,000,000$). Of course, it does not take many items with a higher degree of indeterminacy before this number is dramatically increased. For example, for a semantics s_i with 2000 generated realizations, the number of ordering constraints added to the model for this item alone (assuming for the sake of the argument that all similarity scores are distinct, $y_{ij} \neq y_{ik}$) will be close to two million. As described in Section 5.3.4, there are many items in our development data that have hundreds and thousands of different realizations. Consequently, when trying to estimate an SVM ranker on the

full Tourist generation treebank, using WA scores as the basis for the preference relations, we generate a total of roughly $36,000,000$ rank constraints. Even given the substantial hardware resources listed above, this estimation problem simply proved to be computationally intractable.

Now, there are several ways we can cut down on the number of ranking constraints. One way is to use the method of *random sampling*. In this case the training data only includes a randomly chosen subset of the candidates that are available for each item. This approach can *dramatically* reduce the number of preference constraints added to the model, depending on how "aggressively" we choose to shrink the original candidate sets. For example, using a random sample of maximally ten candidates per item, the total number of WA-based rank constraints for the Tourist treebank is brought down from around $36,000,000$ to $115,000$. Using this approach we did manage to successfully train ordinal SVM rankers, but unfortunately without satisfactory results. For example, when using a linear kernel and a random sample of ten candidates per item, the best accuracy we obtained was 67.65% (NEVA $= 0.937$, WA $= 0.941$). Although this is well below the performance of the MaxEnt model using the same feature configuration (74.25%), it should be taken into account that we are using a significantly smaller subset of the candidates in the training data. We will give more details about the approach of random sampling in Section 7.3.2 below.

Another method for reducing the number of rank constraints, which we also tried in several variations, is to make the similarity scores more "coarse" through *rounding*, e.g. only preserving one decimal place. This has the effect of making more of the scores identical, which again means that there will be more candidate pairs for which we do not add any ranking constraints. This effect is also reasonable from a more principled point of view, as we will typically be less interested in preserving ranks separated by very fine-grained distinctions. The less difference in the scores separating two candidates, the more likely it is that the corresponding rank order will be somewhat arbitrary. If the score of r_j is much higher than then score of candidate r_k (i.e. $y_{ij} \gg y_{ik}$) then their corresponding rank order is something we would want to preserve in the model. If, however, the difference in scores is only marginal (i.e. $y_{ij} \approx y_{ik}$), it makes less sense to try and learn the corresponding ordering. Another and similar approach would be to define a threshold c that we require the difference in similarity scores to exceed before we consider the corresponding ordering relation to be interesting. In other words, we only add an ordering constraint $r_j \prec_i^* r_k$ if $|y_{ij} - y_{ij}| > c$.

The approaches sketched above are similar in spirit to how Shen, Sarkar, and Och (2004) define the ordering relations when training discriminative rerankers in the context of SMT. For the purpose of reranking n-best lists of translations from a baseline SMT system, Shen et al. (2004) apply a perceptron-like splitting algorithm and a large-margin modification of ordinal regression, methods that in many

ways are related to the SVM rankers we are working with here. As noted by Shen et al. (2004), the underlying assumption that consecutive ranks are separable may become problematic in cases where the defined ranking does not strongly distinguish between candidates. When using ranks defined in terms of, for example, the WA string-similarity between references and candidates, this will indeed often be the situation. This is another reason why operations such as those suggested in the previous paragraphs may be helpful.

As said, we have tried several variants of the approaches outlined above, but were still unable to obtain satisfactory results. In the cases where we did manage to complete full rounds of estimation, the running time was typically unacceptably long (e.g. several weeks), except for the runs using very small random samples. The accuracy of the resulting models (between 65% and 70% for the best performers) were also not competitive with the best performing MaxEnt models. Although additional parameter tuning would likely have improved the performance, this was often not feasible given the long running times.

Note that some of the attempts at training SVMs on the full sets of ordinal ranks were actually carried out on a smaller subset of the development treebank. Velldal and Oepen (2006) presented results for SVM rankers trained on an earlier version of the *Jotunheimen* subsection of the Tourist corpus. Now, although the version of the Jotunheimen treebank used by Velldal and Oepen (2006) only contained a total of 2190 items, it was generated with underspecified IS markers. As touched upon several times by now, generating from MRSs underspecified for IS typically means that we get a higher degree of indeterminacy. In other words, we generate a higher number of realizations per input semantics (averaging 85.7 for the data set of Velldal and Oepen (2006)). Naturally, the number of rank constraints that can then potentially be added to the model increases correspondingly.

In sum, *scalability* appears to be a big problem for the SVM ranking framework, when applied to large data sets with ordinal rank relations. In the end, we decided to instead try to train SVM rankers using a similar training set up as for the MaxEnt models. In other words, instead of trying to learn a ranking function from sets of preference relations defined in terms of similarity scores, we only try to learn a ranking function that reflects the binary distinction between preferred and non-preferred realizations in the treebank. For a given treebank item, all candidates that are labeled as preferred will rank above the candidates labeled as non-preferred, while all other preference relations are ignored. In other words, the ranker will only try to separate the references from the other candidates. All other distinctions are disregarded. This is similar to how we trained the MaxEnt rankers in Section 7.2 (except for the preference weighted models trained in Section 7.2.7). Of course, this dramatically reduces the number of rank constraints imposed on the learning problem. For each training item, the number of constraints will then simply be given by the number of preferred candidates times the

number of non-preferred candidates. Unless otherwise noted, the remainder of this section only deals with SVM rankers trained on such binary ranks.

In terms of similar work in relation to NLG, note that Nakatsu and White (2006) use SVM^{light} for reranking generated sentences in order to improve the of quality of synthetic speech. However, the data sets used by Nakatsu and White (2006) are substantially smaller than for the study we report here, meaning that scalability issues simply do not arise. First, OpenCCG (White, 2004) generates 25-best lists according to a trigram model trained on 750 sentences. An SVM ranker is then trained on 104 items using a random sample of 12 candidates from each n-best list. The preferences constraints for each set of 12 paraphrases are based on human judgments of synthetic voice quality, using a scale of 1-7 (Nakatsu & White, 2006).

Note that Toutanova, Markova, and Manning (2004) train SVM rankers for the purpose of parse selection on the Redwoods corpus. Toutanova et al. (2004) define tree kernels that are based on representations of so-called *leaf projections*, a notion defined as a path through a derivation tree from a leaf node to the root. Similarly to our case, Toutanova et al. (2004) only use preference constraints based on binary ranks. However, differences in the properties of the data sets means that the number of ordering constraints will be somewhat lower than in our case. Toutanova et al. (2004) train and test the rankers by ten-fold cross-validation using a set of 3829 ambiguous sentences from the Redwoods corpus. While the average sentence length is reported to be 7.8 words, the average number of parses per sentence is 10.8. Recall that for our two versions of the Tourist development data, comprising a total of 3921 and 4720 items respectively, the average number of realizations per item is 47.3 and 112.3.

7.3.2 Random Sampling

As explained in the preceding section, we ended up scaling down the scope of the learning problem by training SVM rankers on binary instead of ordinal ranks. In other words, the training data only distinguishes between references and non-references. Even so, when trying to train models on the full Tourist corpus, the amount of time spent on estimation can be considerable. This is not something that is particular to our problem, however. Powerful as they may be, SVMs are known to be notoriously slow to train. Moreover, the number of rank constraints generated for our treebank data is still of considerable size. This is especially true for the IS-underspecified version, as also touched upon above. When trying to estimate a binary ranks SVM on the full IS-underspecified Tourist treebank, we generate roughly $2,500,000$ preference constraints. This is still enough to often make the estimation process grow out of available memory (remember, the pysical memory available on our machines is 32 GB). Also, although we are able to com-

plete the estimation for the primary version of the development data, searching for appropriate values of various model parameters can still be quite time-consuming, especially if using a ten-fold cross-validation approach. Furthermore, in the discussion so far we have always been assuming linear kernels. When trying to use other kernels such as the polynomial kernel or RBF kernel, training time and memory requirements increase even further. All in all, we see that there is still need for further reducing the complexity of the estimation problem. As briefly mentioned above, one way to do this is to use the technique of *random sampling*. Instead of using the full set of candidate realizations within each training item, this means that we only use a random subset of some specified size. For example, the SVM rankers of Velldal and Oepen (2006) were trained on random samples of maximally 50 candidates. This implies that the training data includes, for each training item, a random pick of 50 non-references, in *addition* to all the available references. If the item has less than 50 candidates in the first place, all of them are included.

One potential problem with using random samples is that it introduces an element of *randomness* in the results. In other words, we might very well see diverging results for consecutive experiments that use the same parameter configuration but a different random sample. For this reason, one might want to run repeated experiments on various samples in order to compute averages and ensure that the results are representative. However, since we are here resorting to random sampling for the pragmatic reason of reducing estimation time, that alternative is not so attractive. We will see concrete examples of how random sampling affects estimation time in Section 7.3.3 below. However, it is worth pointing out that the "non-determinism" introduced by techniques such as random sampling can also be regarded as a desirable *feature*. Osborne (2000) use a similar sampling method when training log-linear models for large-scale parse selection. The suggestion made by Osborne (2000) is to iteratively estimate models using successively larger samples until one eventually finds the *most informative sample*, i.e. the sample that seems to yield the best performing model with the best generalization properties. Note that the samples need not necessarily be picked at random, but can instead be based on n-best lists from an underlying baseline system, etc. For the SVM rankers trained using random sampling in this thesis, we have not attempted to maximize performance by iteratively training on different samples, but this might be one way to improve the results, as also noted in Velldal and Oepen (2006).

Before we leave the topic of random sampling, it is worth noting that the element of non-determinism introduced by random sampling seems to be less marked for the SVM rankers than the MaxEnt models. We have experimented with random sampling also in the MaxEnt universe, and, although we have not made any systematic studies of this, our clear impression is that the variation in

the resulting model performance seems to be higher here than for the SVMs. One possible explanation for this is the fact that the SVM learner identifies a subset of *support vectors* that define the final model. The fact that the final solution is not dependent on the full set of data points might make the outcome more stable across different samples of non-preferred candidates.

7.3.3 Kernels and Tuning

Ultimately, despite many CPU weeks of experimentation, we did not manage to outperform the default linear kernel using any of the other available kernels, such as the polynomial kernel or the RBF kernel (as described in Section 2.4). For one thing, in order to successfully complete the estimating process with these kernels we could only use quite small random samples. Still even, when using the strategy of random sampling, the associated training time forbids any systematic experimentation with different parameter settings. For example, for a ten-fold cross-validation experiment using a polynomial kernel and a random sample of maximally 25 candidates, it takes between seven and ten CPU days to complete the full cycle. Using a random sample of 10, completing an experiment takes between two and three days. The best performing polynomial ranker we were able to train using a sample of 10 candidates per item (still using binary ranks) obtained an accuracy of 70.98%. The corresponding NEVA and WA scores were 0.936 and 0.935 respectively. Using a random sample of 25, the best performing polynomial kernel obtained an accuracy of 71.62%, NEVA of 0.937, and WA of 0.936. The training time when using the RBF kernel was even longer than for the polynomial kernel. Even when using a random sample of 10 it still typically takes around a week to complete an experimentation cycle. The best result we observed for the RBF ranker was 65.91% accuracy, 0.926 NEVA score, and 0.925 WA.

As said, the above results were produced using the approach of ten-fold cross-validation. The associated running times indicate that this is not a viable alternative for systematic experimentation with different SVM parameter settings during the initial development phase. The repeated training and testing on the nine to one folds prolong the running time of a single experiment significantly. Of course, one possibility is to instead run 5-fold or 2-fold experiments. However, as the effects of changing a particular parameter setting may partly depend on the size of the training data, we might then risk seeing results that do not transfer to a model trained on the full data set (which we will eventually want to do for the held-out testing). Instead, an alternative estimation strategy that we sometimes use during development is to terminate an experiment after it completes the first fold iteration. In other words, we train a model on nine folds, test on the one remaining fold, and then halt. Of course, this is similar to the set-up we use when ultimately evaluating our models in Chapter 8, where we have reserved a portion

of the data for held-out testing. However, the functionality in our experimentation environment as described in Section 6.1, allows us to complete any number of such fold iterations. We refer to this mode of experimentation as i-*out-of*-n-*fold* cross-validation. For example, we could choose to complete another two or three cycles before halting if we wanted to further ensure the representativeness of the results. This arrangement can be practical in cases where full ten-fold cross-validation is otherwise too expensive, as is often the case for the SVM rankers. Nonetheless, note that all the results that we actually report in this section have been produced through completing the full ten-fold cross-validation cycle.

C	Accuracy
10	65.31
5	71.00
1	72.22
0.5	72.30
0.1	72.72
0.05	72.97
0.01	**73.28**
0.005	73.25
0.001	72.42
nil	72.97

Table 7.15: The effect on ranking performance when changing the value of the constant C which governs the trade-off between maximizing the margin size and minimizing the training error. The column value *nil* indicates that we use the default value computed by SVM^{light} internally. Accuracies are averaged over ten-fold splits on the Tourist development data, using a random sample of (maximally) 25 candidates per training item.

As said, we have had most success with the SVM rankers using a linear kernel. The training times associated with these rankers were also much more reasonable, but still significantly longer than for the MaxEnt models (for which we rarely spend more than two hours for a full ten-fold cross-validation cycle, typically much less). As mentioned above, training a binary ranks SVM for the full IS-underspecified version of the Tourist treebank would imply adding roughly $2,500,000$ rank constraints to the optimization problem. As this turns out to be more than we can currently handle efficiently, we use a random sample of of 50 candidates when training on this version of the treebank. For the primary version of the Tourist treebank, a binary ranks set-up implies a total of almost $1,400,000$ preference constraints.[18] Completing a full run with the linear kernel on this data typically

[18]It might be worth trying to provide some further context for the figures we report for the

Summary of SVM Results for the Tourist Development Data

Scores for the Top-Ranked Candidates

	Primary				IS-underspecified			
	RS	Accuracy	WA	NEVA	RS	Accuracy	WA	NEVA
Lin. SVM	–	**73.84**	**0.942**	**0.944**	50	**68.99**	**0.892**	**0.920**
Poly SVM	25	71.62	0.936	0.937				
RBF SVM	10	65.91	0.925	0.926				

Scores for the 5-Best Lists

	Primary				IS-underspecified			
	RS	Accuracy	WA	NEVA	RS	Accuracy	WA	NEVA
Lin. SVM	–	91.56	0.985	**0.987**	50	**90.07**	**0.978**	**0.982**
Poly SVM	25	**91.71**	**0.986**	**0.987**				
RBF SVM	10	88.02	0.981	0.980				

Table 7.16: Test results for different types of SVM rankers, trained and tested through ten-fold cross-validation on the Tourist generation treebank. *Primary* refers to the standard Tourist held-out test data, while *IS-underspecified* is the version constructed without MRS-marking of information structure. Only the SVM ranker with the linear kernel was applied to the latter data set, due to the higher ranking complexity. The RS column lists the size of the random sample used during training, if any.

takes around twelve hours. When using a random sample size of 25, this can be reduced to just three hours. Note that this decrease in convergence time comes at the cost of roughly one percentage point in accuracy. In relation to the linear SVM rankers trained on the standard development corpus we therefore only use random sampling during the exploratory development phase, and not when training the models that we use for the final evaluations. Now, there are several estimation parameters that we tried to tune while developing the different rankers. Generally, there are fewer parameters that must be tuned when using the linear kernel

generated preference constraints. To put things in perspecitve, consider the experiments carried out by Radlinski and Joachims (2005) in the context of search engine optimization. Radlinski and Joachims (2005) estimate an SVM ranker on the basis of $120,134$ preference constraints generated from query logs describing the behaviour of search engine users. In relation to this learning task, Radlinski and Joachims (2005) comment that; *from a practical perspective our approach pushes the limit of problems that current SVM implementations can solve in reasonable time due to the number of constraints we generate*. On this background, it should not be surprising that estimating models on the basis of *several millions* preference constraints is not entirely straightforward.

than when using, for example, the RBF kernel or the polynomial kernel. As seen in Equation (2.47), the latter, non-linear kernels are associated with additional kernel parameters governing the transformations on the feature space. However, there are also many model parameters that are shared across the different kernel types.

Now, the most central such parameter is the regularization constant C which we see tuned for the linear SVM ranker in Table 7.15. As mentioned in Section 7.3.1 above, it is possible for SVMs to approximate a solution in cases where a separating hyperplane can not be found, by allowing for some degree of error. Now, as described in Section 2.4, the corresponding trade-off between maximizing the margin size and minimizing the training error is governed by the constant $C > 0$. Decreasing the value of C can be seen as "softening" the margin. From 7.15 we see that a value of $C = 0.01$ seems to yield the best ranking performance (when using random samples of 25 candidates). However, when including the full set of candidates for each item during training, i.e. not using random sampling, we actually find $C = 0.005$ to provide the best value. This is also the value we found to give the best results for the SVM rankers described in Velldal and Oepen (2006) (where we used random samples of 50). As SVMs can be sensitive to the numeric range of the features values, it is worth noting that we also experimented with different types of normalization of the feature values. For example, we tried normalizing all feature vectors to have unit Euclidean length, and also to have all feature values normalized to the interval $[-1, +1]$. However, we found that none of these strategies improved the performance of the estimated rankers. Rather, it contributed to increase the already severe memory requirements, due to the increased number of float-valued features (instead of the otherwise integer-valued counts).

The results obtained for the best-performing SVM rankers on the development data are summarized in Table 7.16. We see that it is the linear kernel that yields the SVM ranker with the best overall results. As pointed out, however, the linear SVM ranker is also the model for which we were able to perform the most extensive tuning. Even more importantly, when training the other kernelized rankers, we were not able to use the full set of available realization candidates . Instead we had to train on smaller subsets of the candidates as selected through random sampling. The (maximum) size of the respective samples is listed under the RS heading in Table 7.16. Despite using random samples of a relatively small size, we see that the polynomial SVM ranker achieves results that are quite close to those of the linear ranker. When evaluating the quality of the 5-best and not just the top-ranked candidate, we find that the polynomial kernel actually achieves slightly *better* results than the linear kernel. Note that Velldal and Oepen (2006) reported results for a polynomial ranker trained on only half the Jotunheimen generation treebank (comprising 2190 items). When testing on the remaining half, the

SVM results for the Tourist development data

Bin	Items	Words	Trees	MaxEnt	SVM
$100 \leq n$	369	27.5	360.0	42.01	39.57
$50 \leq n < 100$	230	24.7	73.5	54.35	54.35
$10 \leq n < 50$	1144	20.6	22.5	68.49	67.61
$5 \leq n < 10$	868	15.3	6.9	80.62	80.16
$1 < n < 5$	1310	13.9	3.2	87.63	88.17
Total	**3921**	**18.1**	**47.3**	**74.25**	**73.84**

SVM results for the IS-underspecified version

Bin	Items	Words	Trees	MaxEnt	SVM
$100 \leq n$	966	24.4	468.0	48.86	44.93
$50 \leq n < 100$	433	20.9	71.8	55.54	54.97
$10 \leq n < 50$	1607	17.2	23.8	70.05	69.09
$5 \leq n < 10$	842	12.1	25.02	81.00	79.10
$1 < n < 5$	872	11.3	41.82	92.43	92.66
Total	**4720**	**17.0**	**112.3**	**70.47**	**68.99**

Table 7.17: Detailed view of the accuracy of the linear SVM ranker, paired with the accuracy of the (combined) MaxEnt ranker for reference, and tested on both versions of the Tourist development data. The data items are binned relative to their number of realizations. The columns are, from left to right, the subdivision of the data, the corresponding number of items, average string length of the realizations, average number of realizations, the accuracy of the MaxEnt model, and finally the accuracy of the linear SVM ranker.

model obtained an exact match accuracy of 71.03%. This is just slightly below the 71.11% accuracy obtained by a linear SVM ranker trained and tested through full ten-fold cross-validation on the same data set (Velldal & Oepen, 2006). Although the two results are not directly comparable, given that the sets of test items are not identical, they still give reason to believe that we might be able to obtain better results with the polynomial ranker in future experiments, as we continue to see developments with respect to both hardware and algorithmic aspects. Note that the recently released SVM^{perf} toolkit (Joachims, 2006) implements algorithms that potentially can estimate SVM rankers much more efficiently than SVM^{light}. However, at the time of writing, this implementation currently only supports the use of binary (not ordinal) ranks and linear kernels. When it comes to the IS-underspecified

version of the development data, we were only able to apply the linear version of the SVM ranker, and even then only training on random samples of 50 candidates.

Comparing the SVM results in Table 7.16 to the MaxEnt results in Table 7.13, we see see that none of the SVM rankers obtains better results than the best-performing MaxEnt ranker. In the case of the IS-underspecified data, the fact that we had to resort to random sampling probably contributes to the reduced ranking performance of the SVM ranker compared to the corresponding MaxEnt ranker. All the differences in evaluation scores for this profile are detected as statistically significant at the 0.05-level. For the primary version, on the other hand, none of the differences between the linear SVM and MaxEnt are detected as significant. Table 7.17 shows a more detailed view of the accuracy of the linear SVM ranker on both versions of the development data. The results are broken down across groups of items with different degrees of generator indeterminacy. For reference we also include the corresponding accuracy of the (combined) MaxEnt ranker using the same feature configuration.

As touched upon in Section 7.3.2, the solution to the SVM optimization problem is typically based on just a small subset of the input vectors. These are the support vectors, defined as the examples lying within the margin area (or exactly on the margin in the case of hard margin models). This sparsity of the solution is an important property of SVMs. Recall that the preference constraints $r_j \prec_i^* r_k$ are reformulated to constraints on pairwise difference vectors, $w\left(\Phi(s_i, r_j) - \Phi(s_i, r_k)\right) > 1 - \xi_{i,j,k}$. In other words, SVM^{light} internally recasts the ranking task to a binary classification task for difference vectors $x = \Phi(s_i, r_j) - \Phi(s_i, r_k)$. It is these difference vectors x that actually constitute the training examples during estimation then. Now, when training a model on the entire (standard) development data, in order to export a final model for the held-out testing in Chapter 8, we end up generating a total of 1,392,785 such training examples. However, the final solution is indeed quite sparse, and only depends on a total of 25,455 support vectors. Of these, roughly half are within the margin area (so-called upper-bounded, meaning that $\alpha_i = C$ in terms of the dual formulation of Equation (2.43)), while the remaining half are exactly on the margin (i.e. $0 < \alpha_i < C$ in terms of the dual formulation).

7.4 Summary of Development Results

This chapter has described the development of a series of realization rankers, using several different modeling frameworks. We have reported quantitative evaluation results using measures such as exact match accuracy, NEVA, and WA, testing on two different versions of the Tourist development data. While the primary version of the data set includes *information structure* (IS) markers in the underlying MRSs

Performance of Realization Rankers on the Tourist Development Set						
	Primary			IS-underspecified		
	Accuracy	WA	NEVA	Accuracy	WA	NEVA
4-gram LM	53.75	0.907	0.907	50.04	0.825	0.873
MaxEnt	72.11	0.941	0.943	68.05	0.889	0.919
MaxEnt w/ LM	**74.25**	**0.944**	**0.946**	**70.47**	**0.897**	**0.925**
SVM w/ LM	73.84	0.942	0.944	68.99	0.892	0.920

Table 7.18: Summary of the performance for the different types of realization rankers on the development data. *Primary* refers to the standard Tourist generation treebank, while *IS-underspecified* is the version generated without MRS-marking of information structure (see Section 5.3.4 for details).

that we are generating from, this information is suppressed in the secondary IS-underspecified version of the data, which means that the latter generally has a higher degree of generator indeterminacy (i.e. a higher number of realizations per semantic input).

In Section 7.1 we developed a *4-gram language model* trained on a unannotated normalized version of the general-domain BNC, using a 25,000 word vocabulary partly extracted from the Tourist development corpus. Section 7.2 described several variants of *conditional maximum entropy models* trained on the generation treebanks we developed in Chapter 5. As seen in Table 7.18, the treebank-trained MaxEnt model outperforms the n-gram LM by a good margin on our ranking task. On the Tourist development data, the best performing MaxEnt ranker that only uses treebank features achieves an exact match accuracy of 72.11%, compared to 53.75% for the LM (although we do observe accuracies of up to 54.27% for the huge 5-gram LMs trained with no cutoffs). As described in Section 7.2.5, manual error analysis of a random sample of errors unique to each ranker reveals that large parts of the errors do not overlap. This lead us to try to join the two models as described in Section 7.2.6, resulting in a *combined ranker* where the LM is used as an additional feature in the MaxEnt treebank model. As summarized in Table 7.18, the combined model performs significantly better than either the LM or MaxEnt model on their own. Now, using the same feature configuration, we also trained a binary-ranks *support vector machine ranker*, as described in Section 7.3. The best-performing SVM ranker failed to improve on the MaxEnt model, however, but obtained comparative results nonetheless. For the standard development data, none of the differences in evaluation scores between the combined MaxEnt model and the SVM were detected as statistically significant. Note that the differences

in accuracy are tested using a two-tailed Sign-Test, while the similarity scores are tested using Wilcoxon Signed-Rank Test. For the IS-underspecified version, on the other hand, the differences in SVM and MaxEnt evaluation scores are indeed significant. This is most likely due to the fact that, for this data set, we were only able to train an SVM ranker using random samples of maximally 50 candidates per item. This was necessary in order to keep the number of preference constraints for the optimization problem down on a manageable level.

As discussed previously, the starting point for our treebank-based approach to realization ranking is provided by the Redwoods parse selection experiments reported by Toutanova et al. (2005). However, for obvious reasons, it is not possible to give any meaningful direct comparison of the results of the two studies. In addition to the fact that the actual ranking tasks are different, our experiments are based on different data sets and different versions of the ERG grammar. Furthermore, although the core feature sets are the same, many of the included feature types are also different. Despite all of these differences however, it may nonetheless be interesting to compare the rough regions of the accuracy, coupled with basic properties about the data sets such as ambiguity rate, as this can provide us with some indication about the appropriateness of adapting the methodology of parse selection for the task of realization ranking.

Now, the results for the conditional log-linear models developed by Toutanova et al. (2005) are based on a set of 5266 ambiguous items from the (3rd Growth of) the Redwoods treebank. These items have an average string length of 9.1, and the average degree of structural ambiguity per string is 57.8. The best performing parse selection model of Toutanova et al. (2005), representing a combination of all their reported feature types, achieves an exact match accuracy of 76.7%, computed as an average from ten-fold cross-validation. For comparison, the best ten-fold cross-validation results achieved for our log-linear realization model is 74.25% on the standard development set. Recall that the average number of realizations per item for this data set is 47.3, for a total of 3921 items. In other words, although the degree of indeterminacy is somewhat lower than in the experiments of Toutanova et al. (2005), the number of available training examples is quite a bit lower too. Moreover, given the trends we noted with respect to learning curves in Section 7.2.2, we can expect to still see better results when training on a larger data set. If we instead now turn to the underspecified version of our data set, the number of available training examples is closer to that of Toutanova et al. (2005), totaling 4720 items. However, the associated degree of indeterminacy is also substantially higher, with an average of 112.3 realizations per item. Still we are able to obtain 70.47% exact match accuracy. Finally, although the property of average string length will have somewhat different bearings for the two ranking tasks, it is still worth noting that the average string length in our Tourist generation treebanks is almost twice as long as for the data sets used by Toutanova et al. (2005). In sum

then, on the backdrop of various treebank properties such as the degree of indeterminacy and ambiguity, number of training examples, and average string length, and regardless of the differences in the respective ranking tasks, it seems justifiable to say that the relative performance of the parse rankers and the realization rankers are roughly within the same region.

In the next chapter we move on describe the evaluation of the realization rankers on the *held-out* Tourist test data. In addition to the automated evaluation with the metrics used above, the next chapter also includes a *manual* evaluation effort contrasting the LM ranker and the treebank-based MaxEnt ranker, as carried out by a panel of human judges.

Chapter 8

Held-Out Testing

In the course of the preceding chapter we have presented preliminary evaluation results for several different realization rankers on the Tourist development data. In this section we present the final evaluation results for the held-out Tourist test data. As described in Section 5.3.4, the held-out data consists of material that was set aside during the creation of the original Tourist corpus, and reserved for the purposes of final system evaluation of the LOGON MT system. Based on the parse treebanks created for this data upon the completion of LOGON, we subsequently produced a set of corresponding generation treebanks using the symmetric treebanking methodology introduced in Chapter 5. These generation treebanks then provide the held-out test data we use for the final evaluation of the realization rankers. In addition to the automatic quantitative evaluation in terms of accuracy and string-similarity, Section 8.1 reports results from a human evaluation effort using a panel of external anonymous judges. This manual evaluation is carried out as a contrastive comparison of the n-gram-based language model and the treebank-based MaxEnt model, and so does not include the other rankers.

As before, we will be testing the models on two different versions of the data set. In the primary or standard version, which is most representative for generation within the LOGON MT system, the realizations in the treebank have been generated from MRSs that include information-structural attributes encoding topicalization and passivization. As described in Section 5.3.4, we have also produced an underspecified version of the data, where the IS information is suppressed during generation. Naturally, this leads to a higher degree of generator indeterminacy, as the realizations will also include all grammatically legitimate topicalized and passivized constructions. This mode of presenting results, using parallel data sets with different degree of specificity, is also used by Langkilde (2002), although the particular dimensions of underspecification are not identical.

For convenience, Table 8.1 below repeats some of the core metrics for the held-out data reported in Section 5.3.4, where we discussed the creation of the

Tourist Held-Out Test Data

Aggregate	Items	Words	Trees	Gold	Baseline
$100 \leq n$	17	26.0	572.5	7.8	2.20
$50 \leq n < 100$	24	24.1	78.1	5.6	7.55
$10 \leq n < 50$	83	20.0	23.4	3.2	15.86
$5 \leq n < 10$	57	14.6	6.7	2.1	32.21
$1 < n < 5$	88	14.0	3.1	1.4	45.08
Total	**269**	**17.6**	**52.8**	**2.9**	**27.28**

Tourist Held-Out Test Data underspecified for IS

Aggregate	Items	Words	Trees	Gold	Baseline
$100 \leq n$	74	23.0	472.9	4.7	1.33
$50 \leq n < 100$	29	20.7	74.2	2.6	3.42
$10 \leq n < 50$	113	16.8	22.7	2.1	10.53
$5 \leq n < 10$	69	11.7	6.9	1.7	26.07
$1 < n < 5$	64	12.5	3.2	1.2	39.71
Total	**349**	**16.6**	**115.7**	**2.5**	**16.41**

Table 8.1: Some core metrics for the generation treebanks we use for held-out testing. The data items are aggregated relative to their number of realizations. The columns are, from left to right, the subdivision of the data according to the number of realizations, total number of items, average string length, average number of realizations, average number of references, and finally the baseline for expected accuracy by random coice.

generation treebanks. As we see, the random choice baselines for the two held-out data sets (27.28% and 16.41% for the primary and underspecified versions respectively) are very close to those of the corresponding development sets (28.05% and 16.62% respectively). The same holds for the other basic data statistics such as average number of realizations, average string length, and average number of associated references. For the 269 items in the standard held-out data we have an average of 52.8 realization candidates per item, and the maximum number of realizations generated for a single item is 2520. For the 349 items in the IS-underspecified version we have an average of 115.7 distinct realizations per item, and the maximum number of realizations generated for a single item is 3360. Furthermore, the average string lengths for the two versions of the data sets are 17.6 and 16.6, respectively. The distribution of test items according to average string

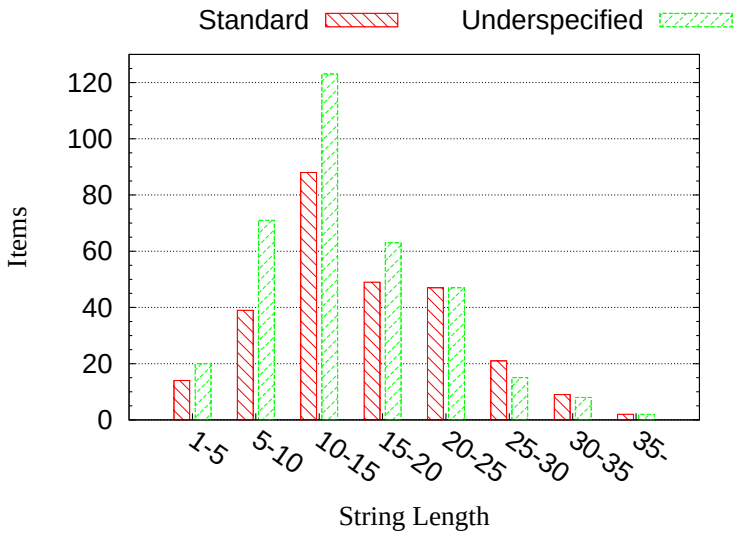


Figure 8.1: Average string length of items in the held-out test data. *Primary* shows the distribution of average string length for the 269 items in the standard test data, while *underspecified* shows the same for the 349 items in the treebank generated with IS-underspecification. A bin such as '5-10' comprises strings that contain more than or equal to five words, but less than ten.

length can be seen in the histogram of Figure 8.1.

Compared to the preliminary results observed for the development data, the results ultimately obtained for the held-out data will be a more reliable test with respect to the generalization capabilities of the models. After repeated cycles of testing on the development set, the models might be finely tuned to the particularities of this data, and the held-out testing provides a way to see how the models fare when faced with new and unseen data. Note that the held-out data also include corpus segments with unknown vocabulary (i.e. words not occurring in the development data).

Before presenting the actual results, it might be worth briefly reminding ourselves about the specifications of the particular models that we are testing. We will be presenting results for four different types of models. First we have the n-gram-based language model (LM) developed in Section 7.1. This is 4-gram back-off model, using Witten-Bell discounting, trained on a raw text version of the 100-million-word British National Corpus (BNC). The model uses a $25,000$ word vocabulary partly extracted from the Tourist development data. Second, we have the conditional *maximum entropy* (MaxEnt) model developed in Section 7.2. This model is trained using features extracted from the Tourist generation tree-

bank, as described in Table 7.2.1. Thirdly, Section 7.2.6 presented a *combined* model where the fluency scores of the LM were integrated as a separate feature in the MaxEnt model. Finally, using the same feature configuration as the combined MaxEnt model, we train a linear SVM ranker, as described in Section 7.3.3. The two combined models, i.e. the MaxEnt model and the SVM ranker, include a total of $255,654$ distinct features.

In the case of the conditional treebank models, we have followed a ten-fold cross-validation approach in order to maximally utilize our development data. This means that for each evaluation result presented for these models in the preceding chapter there are, not one, but ten different models. Therefore, before applying the rankers to the held-out data we have re-trained the models on the entire development treebank (using the same configurations of model parameters). Moreover, as before, we actually train two parallel versions of all of these discriminative models. The models we apply to the IS-underspecified test data have been estimated from the IS-underspecified training data. The models we apply to the standard test data have been estimated from the standard training data.

The held-out performance of the different rankers is summarized in Table 8.2. As before, we score the models according to three different evaluation measures; exact match accuracy, NEVA, and WA. The evaluation measures themselves are further described in Section 6.3. Now, comparing the scores of Table 8.2 to those of Table 7.18, we see that the evaluation results on the primary held-out test set closely mirror those of the development set. Not only are the relative ranks of the various models the same, but the actual evaluation scores themselves are very much in the same range. This means that, particularly for the discriminative learners, we find that the models exhibit very good generalization properties and overfitting does not appear to be a problem. For the MaxEnt model trained with treebank-only features, the accuracy drops by $72.11\% - 71.31\% = 0.80\%$. For the combined MaxEnt and SVM models (i.e. the models including the LM in addition to the treebank features), the drop in accuracy is 0.27% and 0.23% respectively. In other words, looking at exact match accuracy, the best generalization performance seems to be achieved by the SVM ranker. In terms of the string similarity metrics, on the other hand, it is the performance of the MaxEnt models that appear most stable. For both the treebank-only and the combined MaxEnt model, the held-out NEVA score remains the same as observed on the developments data (0.941 and 0.944, respectively).

Somewhat surprisingly, we find that the LM is actually the model that suffers the largest drop in performance when moving to the held-out data. The (absolute) difference in exact match accuracy for the two data sets is 1.15% ($53.75\% - 52.60\%$). As the LM was trained on the separate BNC, we had expected that there would be less difference between the development results and the held-out results for this ranker. For the previously reported held-out results

(albeit using different data sets) of Velldal and Oepen (2006), the LM performance appeared to be more stable with respect to the development set. However, as described in Section 7.1.2, the vocabulary of the LM trained for the current experiments is in part based on the word forms that occur in the development set, and that is likely an important factor for the observed difference.

The difference in LM accuracy is even larger when looking at the IS-underspecified data Table 8.2, dropping by 1.33%. For the discriminative treebank-based learners, on the other hand, the situation is quite different. Compared to the accuracy achieved on the development data, we find that all of them actually perform quite a bit *better* on the held-out data. The difference is most pronounced for the purely treebank-featured MaxEnt model, where accuracy is up by 3.15% (68.05% vs. 71.20%). But also for the accuracy of the combined MaxEnt model and the SVM ranker do we observe a rather large increase, going up by 1.74% and 2.07% respectively.

In sum, we see that the ranking performance of the discriminative learners on the held-out data is roughly *as good* or *better* than on the development data. Now, the main reason for this fortunate outcome probably has to do with the increased amounts of training data. Recall the learning curves that we computed for the MaxEnt model over the development data in Figure 7.7. The graph clearly indicated that the learner benefited from additional training data. Although the curve appeared to flatten out somewhat as more data was added, it was still rising at the point where all the available data was added, indicating that additional data would still be beneficial. Now, these learning curves were computed using a (somewhat modified) ten-fold cross-validation set-up. As we mentioned in Section 7.2.2, ten-fold cross-validation tends to underestimate performance. The reason is, of course, that it never takes advantage of the full set of training examples, but rather only nine-tenth of them at a time. As mentioned above, however, the models we here apply on the held-out data have been re-trained using the entire set of available development examples. Moreover, the results seem to indicate that the discriminative learners were able to make good use of this additional training data. For the IS-underspecified data we see that held-out results are in fact even better than the development results. This positive difference for the underspecified data can be explained by the reasoning as above. Recall from Section 5.3.4 that this instance of the generation treebank actually contains more items that are relevant as training data. While the primary development treebank comprises 3921 items, the underspecified version comprises 4720. We observe the same difference for the held-out data in Table 8.1. So, in absolute terms, the difference in the number of training examples for the ten-fold models and the final "all-inclusive" model is even larger for the IS-underspecified data, and hence the benefit of the final re-training is even greater.

The benefits of added data are also underlined when comparing to the results

Summary of Evaluation Results for the Tourist Held-Out Test Data

Scores for the Top-Ranked Candidates

	Primary			IS-underspecified		
	Accuracy	WA	NEVA	Accuracy	WA	NEVA
4-gram LM	52.60	0.904	0.887	48.71	0.819	0.857
MaxEnt	71.31	0.941	0.939	71.20	**0.887**	**0.923**
MaxEnt w/LM feat.	**73.98**	**0.944**	**0.941**	**72.21**	0.884	0.920
SVM w/LM feat.	73.61	0.939	0.938	71.06	0.878	0.918

Scores for the 5-Best Lists

	Primary			IS-underspecified		
	Accuracy	WA	NEVA	Accuracy	WA	NEVA
4-gram LM	81.61	0.981	0.977	78.33	0.957	0.963
MaxEnt	89.34	0.987	0.983	90.26	0.978	0.981
MaxEnt w/LM feat.	**92.94**	**0.992**	**0.988**	**92.55**	0.980	**0.984**
SVM w/LM feat.	91.82	0.990	0.987	91.40	**0.982**	0.982

Table 8.2: Held-out test results for the different types of realization rankers. *Primary* refers to the standard Tourist held-out test data, while *IS-underspecified* is the version constructed without MRS-marking of information structure (see Section 5.3.4 for details).

reported by Velldal and Oepen (2006). In this study, again using treebanks generated with IS-underspecification, we observed a marked drop in performance for the discriminative models when going from the development data to the held-out data. However, the models of Velldal and Oepen (2006) had been trained only on the Jotunheimen sub-corpus which contains roughly half the number of trainining examples that we now have treebanked in the full Tourist corpus.

The trends noted above are less clear when looking at the string-similarity metrics instead of the accuracy. We also find that the string-similarity measures appear less conclusive with regards to which of the treebank models is the best performer, especially when looking at the scores for the IS-underspecified data. Here we see that the best NEVA and WA scores are actually achieved by the MaxEnt model trained on treebank features alone. Maximizing scores over 5-best lists, we find NEVA leaves the SVM ranker as the best performer. Most consistently however, the combined MaxEnt model is still the model that ranks as the best performer. When evaluating the top-ranked candidates for the primary data, the

combined MaxEnt model receives the highest scores across all three evaluation metrics. The same also holds when evaluating the 5-best lists. For the underspecified version, it outperforms the second-best performer (which is also a MaxEnt model) by one percentage point in terms of exact match accuracy. As it turns out, however, only a few of the differences observed in Table 8.2 are actually detected as statistically significant. An important factor contributing to this is probably the relatively modest size of the held-out test set in itself. Recall from Section 6.4 that both the sign test and the signed-rank test are relatively crude and not very sensitive tests, as is typical for non-parametric tests On the other hand, they are typically very reliable when they actually do indicate significance. For all three evaluation metrics, only the differences between the 4-gram LM and the various treebank models are detected as statistically significant for $\alpha = 0.05$. Looking at the differences among the various treebank models, we would need to increase the significance level by an order of magnitude ($\alpha = 0.5$) before the aforementioned tests would render them significant.

8.1 Human Evaluation

So far in this thesis we have only relied on automated quantitative evaluation, using various objective test metrics. Although we have used manual inspection for the purpose of error analysis, the scoring and system-level ranking of the various model types have only been based on automatic evaluation. In this section we present test results that have been produced with the help of a group of external and anonymous human evaluators. With the kind assistance of Prof. Emily M. Bender, a group of seven MA students within the *the Professional Master's in Computational Linguistics Program* at the University of Washington were recruited to judge the relative quality of alternative generator outputs. The judges—all native speakers of English—were given a questionnaire with a set of test items from the held-out data, and asked to assign a relative rank order to lists of candidate realizations as chosen by different models. Below we first describe the process of compiling the questionnaire or evaluation form that we presented to judges. Note that the questionnaire itself is included in its entirety in Appendix A. The actual results of the evaluation are presented in Section 8.1.2.

8.1.1 Producing the Evaluation Form

First of all, in order to not make the evaluation task unmanageably large for the judges, we wanted to isolate the models that we believe will make for the most informative comparison. Now, the approaches that we are most interested in contrasting in this thesis are, on the one hand, that of surface oriented n-gram-based

language modeling, and, on the other, discriminative learning from generation treebanks. When constructing the evaluation set for the human judges then, we would obviously need to include the output of the language model. For the treebank models, we have had most success with those trained within the MaxEnt framework. Furthermore, to better isolate the differences in performance, we decided to use the MaxEnt model trained with treebank-only features, instead of the combined model which includes the LM as a feature. Given that these are the models we want to focus on, the actual evaluation set was constructed as follows.

We started by extracting all test items in the held-out data for which the LM and the MaxEnt model disagree with respect to their top-ranked candidate. In other words, we did not care whether any of them matched the reference. The only criterion was whether they made diverging decisions as to which is the best candidate. This gave us a preliminary evaluation set consisting of pairs of alternative realizations for a set of 73 test items (comprising slightly more than 27% of the 269 items in the held-out data). Then, for each of these items, we also included another realization that was randomly picked from the available candidates. This represents the behavior of the random choice baseline in practice. Finally, we also included the original reference string as it occurred in the (parse) treebank. Recall, as described in Section 5.3.2, that the naturally occurring strings in the underlying parse treebank provide the basis for which candidates we label as gold in our generation treebanks.

The outcome of the above procedure is an evaluation set where each element consists of a list of four alternative candidates, as chosen by means of four different selection strategies. Some additional preparation was done before the final evaluation set was presented to the judges, however. First we removed any duplicate entries in the lists, i.e. whenever the reference or random choice coincided with the candidates picked by the language model or the MaxEnt model, we only included one of them in the final evaluation set. The lists in the final set will therefore contain between two and four candidates. Note, however, that the rank assigned to a given candidate in these lists will automatically also be mapped to any other possible duplicate entries withheld from the final evaluation set. Furthermore, to make sure that the order of presentation will not affect the judgments of the evaluators, we also randomized the ordering of the candidates in the list for each test item. Of course, all of these operations are traced in a way that allows us to reconstruct the full original test set after evaluation.

Recall that the underlying generation task that we are working with in this thesis is defined as single-sentence generation. This means that no information about the overall discourse is taken into account when generating the surface realizations of a given semantics. However, in order to place the external evaluators in a better position to judge the quality of the alternative realizations, we do provide them with a minimum of context for each test item. Including some context from

the immediately preceding narrative will often make it easier to form an opinion about the relative quality of the candidates, compared to just seeing the candidates in isolation. (Of course, also presenting the evaluators with the underlying logical-form semantic representation that the sentences have been generated from, would be far too complex.) Note that, occasionally, the preceding test item will itself provide the necessary context, in cases where the extracted test items correspond to successive sequences of the original narrative (recall that the LOGON treebank data is constructed on the basis of running text in a corpus).

Now, for each list of alternative strings in the resulting evaluation form, the human judges were asked to rank them relative to each other according to which they considered to sound more natural. The judges where instructed to assign each candidate a rank position on a scale from 1 to 4 (or the number of candidates included in the particular list), also allowing for ties. In other words, the evaluators where not asked to make any absolute assessments of the quality of each realization. They were only asked to assign a relative rank order on the sets of alternatives. As mentioned above, a copy of the actual questionnaire can also be found in Appendix A.

8.1.2 Evaluation Results

We are now ready to look at the results we got back from the seven respondents.[1] Before we go on to present any quantitative results for the evaluation, note that all rank values are normalized to ensure consistency with respect to the following rule: If there is a tie between any candidates, these are assigned identical ranks corresponding to the next unused rank value. For example, given a list of alternatives $\{a, b, c, d\}$, with a judged to be the best candidate, followed by a tie between b and c, and finally d at the bottom, we will have the following distribution of rank values: $a = 1$, $b = 2$, $c = 2$, and $d = 4$.

For each of the 73 items in the test set, the judges assigned a relative rank order to the candidates picked by the different models. By summing all of these *per-item* rank values, we can, for each judge, obtain a *system-level* rank order on the four models themselves. By further summing the system-level rank values, we can obtain a *global* system-level rank order. This is what is shown in Table 8.3 below, where the models are sorted according to their rank values. For each model the data columns show, from left to right; the total sum of per-item rank values assigned by all judges; the average sum of rank values per judge; and finally the average per-item rank value. Recall that a lower score indicates higher rank (i.e. a higher relative quality according to the evaluators). As is clear from Table 8.3,

[1] In terms of age, the respondents range from 19–38 (avg.≈26), and with an even distribution of gender (four females and three males).

the reference sentences (*Gold*) were deemed to have the highest overall quality. Although this particular outcome was what we had anticipated, of course, it is a reassuring result nonetheless, as the reference sentences provide the basis for the automatic and quantitative evaluations that we otherwise rely on, and even provide the basis for how the training data itself is defined for the discriminative learners. In this sense, the fact that the reference sentences clearly stand out as the best candidates according to the human evaluators is something that further validates the assumptions underlying our symmetric treebanking methodology, as presented in Chapter 5.

Also as expected, we see that the random choice baseline receives the lowest average rank. Finally, the two middle rows of the Table 8.3 hold the most interesting results. We see that the candidates chosen by the MaxEnt treebank model are on average judged to be better than those of the n-gram language model. The candidates selected by the MaxEnt model receives an average rank position of 1.86, while the LM ranker receives an average rank position of 2.38.

Model	**Sum of Ranks**	**Average Sum**	**Average Rank**
Gold	674	96.3	1.319
MaxEnt	951	135.9	1.861
LM	1215	173.6	2.378
Baseline	1304	186.3	2.552

Table 8.3: Summary of rank values assigned by the human evaluators.

It is also worth noting that the relative rank order of the different systems as assigned by the human judges, is actually the same as the rank order assigned by the automatic metrics we use, such as edit distance and exact match accuracy. In this sense, the results of the manual evaluation effort seen in Table 8.3 are not only an evaluation of model performance, but indirectly also an assessment of the quality of the automatic evaluation metrics. If the relative ordering resulting from the human evaluation was different from the ordering obtained through the automatic evaluation methods, we would have good reasons to be suspicious of the latter. Fortunately, the outcome of the human evaluation provides us with no such reason for concern.

An important part of interpreting any human evaluation results is to measure the level of *inter-annotator agreement*. Now, the different rank scores shown in Table 8.3 seem to be separated by quite solid margins, already giving us good reasons to trust the significance of the result. However, we have also computed several independent measures that also indicate that the level of agreement among the evaluators is indeed quite high. For any candidate sentence given top rank by

some judge for a given item, the average number of judges who have given the same candidate top rank is 4.5 out of 7. For the candidate realization that was ranked top by the most judges for a given item, the average number of judges agreeing is 6.0 out of 7. Both of these quite intuitive measures seem to indicate that the evaluators tend to agree quite strongly on the judgments that they make with regards to their top-ranked sentences. However, in order to get a better impression of the inter-annotator agreement on the overall rankings, we also computed *Spearman's rank correlation coefficient*, denoted as ρ. Admittedly, having only 4 different ranking levels is a bit too few for correlation measures to really be meaningful. However, the coefficient still gives some impression of the level of agreement among the judges.

The ρ coefficient is a non-parametric rank statistic for variables that are measured at the ordinal level and is equivalent to Pearson's coefficient applied to ranks instead of raw scores. Let d_i be the difference between the ranks of each candidate for a given test item. Spearman's rank correlation coefficient is then computed as

$$\rho = 1 - \frac{6 \sum_{i=1}^{n} d_i^2}{n(n^2 - 1)} \tag{8.1}$$

where in our case we will have $n = 4$. Note that ties are handled in the same manner as for the Wilcoxon test described in Section 6.4.2, i.e. by using the arithmetic average of the corresponding ranks. To use the example from above, where we had $a = 1$, $b = 2$, $c = 2$, and $d = 4$, the normalized ranks would now be $a = 1$, $b = 2.5$, $c = 2.5$, and $d = 4$.

Although the correlation coefficient is based on a pairwise comparison of the judgments of two evaluators, we can easily extend this by computing, for each judge, the average correlation towards all the other judges. Finally, note that the correlation coefficient ranges from -1 for perfect negative correlation, through zero for totally independent judgments, to 1 for perfect positive correlation.

Table 8.4 shows Spearman's rank correlation coefficient computed for all the MA students participating in the evaluation. The first column simply enumerates the judges. The second column shows the corresponding ρ calculated as the average pairwise correlation for all items in the evaluation set. In the third column the coefficients are computed on the system-level. Using rank values that correspond to the relative rank ordering seen in Table 8.3, we here compute the average pairwise correlation coefficients with respect to overall judgments about the models themselves. The bottom row in the table shows the corresponding total averages.

The average Spearman correlation coefficient for all judges over the entire evaluation set is $\rho = 0.73$. Now, although this value would typically be taken to indicate a fairly high degree of agreement (recall the bounds $\rho \in [-1, 1]$), the fact that we only have four levels of ranks involved means that we still cannot say that

	Average Rank Correlation	
Judge	**Item Level**	**System Level**
1	0.687	1
2	0.781	1
3	0.727	1
4	0.701	1
5	0.695	1
6	0.747	1
7	0.748	1
All	0.727	1

Table 8.4: Average Spearman's rank correlation coefficient for each human evaluator.

this figure is statistically significant. However, we also computed ρ on the system-level. This was done by averaging all the pairwise correlations between all judges with respect to the relative rank order of the different models. This resulted in a correlation coefficient of $\rho = 1$. In other words, when looking at the system-level, the inter-annotator agreement could not possibly be any higher, and this time the figure is indeed statistically significant[2] at the 0.05 level.

Of course, the two selection strategies that we were most interested in getting a comparative evaluation for were the n-gram language model and MaxEnt treebank model. As seen in Table 8.3, the MaxEnt model seems to outrank the LM by a good margin according to the human judgments. However, we also tested the ranks associated with these two models in isolation. This is done in a way that

[2]Spearman's rank correlation coefficient can be understood as a hypothesis test where the null hypothesis is that the ranks assigned by one evaluator do not co-vary with the ranks assigned by the other. Given that we are operating with so few rank levels (we have four rank levels, given that we have four candidates that we wish to rank), we can easily compute an exact p-value for this statistic by using a *permutation test*. For a given value ρ, this can be done by simply computing ρ' for all possible combinations of rank assignments, and then reporting the ratio of times that ρ' is larger than or equal to ρ. Recall that ρ is always in the interval $[-1, 1]$. Under the null hypothesis of independence, any combination of ranks will be equally likely, and as expected, the sum and the average of all the permuted rank correlation coefficients ρ' is thereby zero. Note that we only need to compute the permutations for *one* set of ranks while keeping the other set fixed, leaving us with $4! = 24$ permutations instead of $4! \times 4! = 576$. Now, computing p-values for the coefficients in Table 8.4 reveals the problem of trying to measure correlation for so few ranks: Even for the system-level ranks where $\rho = 1$, we still only have $p = 0.042$. In other words, even in the case of perfect correlation, the associated p-value is just barely below $\alpha = 0.05$. In the case of average per-item correlation, where $\rho = 0.727$, we have $p = 0.167$.

resembles how we usually test for significance when dealing with the exact match accuracy of two different realization rankers. For each item in the questionnaire we assign a score of $+1$ or -1, depending on whether the average rank given to the MaxEnt candidate is higher or lower than that given to the LM candidate. This results in a sequence of 73 Bernoulli trials, one for each item in the evaluation set. We then apply a two-tailed sign test to this sequence, as described in Section 6.4.1. This results in a p-value of $p = 0.012$, showing that the differences in the human rankings for the language model and the MaxEnt model are statistically significant at $\alpha = 0.05$.

8.2 Summary

In this section we have applied the various realization rankers developed in Chapter 7 to the generation treebanks produced for the Tourist held-out data. The task of the rankers is to select the best surface realization for a given logical-form input semantics, generated in accordance with a linguistic precision grammar. Note that the rankers are tested on two different versions of the treebanks; one where the realizations are generated from MRSs that include encoding of information structure (IS), and another version where these attributes are treated as underspecified, leading to an even greater degree of generator indeterminacy. Table 8.5 shows a detailed view of the accuracy of the different rankers, breaking down the treebank items in bins according to generator indeterminacy in the same way as in Table 8.1. In Table 8.5 we summarize the overall ranking performance in terms of NEVA and WA in addition to exact match accuracy, also evaluating 5-best lists in addition to the top-ranked candidates in isolation. The weakest ranking performance is observed for the 4-gram LM. Still, with accuracies of 52.60% and 48.71% for the primary and IS-underspecified versions respectively, it still outperforms the corresponding random choice baselines of 27.28% and 16.41%. The discriminative MaxEnt model using treebank features performs significantly better, achieving an accuracy of 71.20% for the IS-underspecified data and 71.31% the primary data. The significance of the improved ranking performance of the discriminative treebank model over the generative n-gram model was also confirmed through a manual evaluation effort carried out by a panel of seven human judges. When asked to assign a relative rank to the quality of the candidates selected by different models, the judges significantly more often ranked the MaxEnt candidate higher than the LM candidate, and with a high degree of inter-judge agreement. However, the realization ranker that achieves the best overall performance in terms of all our automated evaluation metrics, is the *combined* MaxEnt ranker, which includes the LM scores as an additional feature among the treebank features. This model achieves an exact match accuracy of 72.21% and 73.98%

Tourist Held-Out Test Data

Indeterminacy	Items	Trees	BL	LM	ME	ME_{LM}	SVM_{LM}
$100 \leq n$	17	572.5	2.20	23.53	35.30	35.30	41.18
$50 \leq n < 100$	24	78.1	7.55	31.25	37.50	41.67	45.83
$10 \leq n < 50$	83	23.4	15.86	45.98	62.65	66.27	67.47
$5 \leq n < 10$	57	6.7	32.21	69.30	91.23	92.98	92.98
$1 < n < 5$	88	3.1	45.08	59.47	82.77	85.23	80.68
Total	**269**	**52.8**	**27.28**	**52.60**	**71.31**	**73.98**	**73.61**

Tourist Held-Out Test Data underspecified for IS

Indeterminacy	Items	Trees	BL	LM	ME	ME_{LM}	SVM_{LM}
$100 \leq n$	74	472.9	1.33	19.60	48.65	50.00	48.65
$50 \leq n < 100$	29	74.2	3.42	25.86	68.97	55.17	58.62
$10 \leq n < 50$	113	22.7	10.53	50.00	65.49	70.86	69.03
$5 \leq n < 10$	69	6.9	26.07	68.84	83.33	85.51	84.06
$1 < n < 5$	64	3.2	39.71	68.75	95.31	93.75	92.19
Total	**349**	**115.7**	**16.41**	**48.71**	**71.20**	**72.21**	**71.06**

Table 8.5: Detailed view of the ranking accuracy on the held-out test data. The data items are aggregated relative to their number of realizations, as shown in the first column. The second and third columns show, respectively, the total number of items within the bin, and their average number of realizations. The remaining columns lists the corresponding exact match accuracy for a range of different models: BL is the random choice baseline, LM is the 4-gram language model, ME is the maximum entropy model using only treebank features, ME_{LM} is the combined maximum entropy model which includes the LM scores as a separate feature, and finally SVM_{LM} is the linear support vector machine ranker using the same feature configuration as ME_{LM}.

when tested on the underspecified and primary treebanks respectively. Although we also trained an SVM ranker using the exact same feature set, this model did not improve on the MaxEnt model, although the differences are not detected as statistically significant at the 0.05 level. In the case of the IS-underspecified data however, we are still impressed by the fact that the SVM ranker obtained comparative results, as we were only able to train it using random samples of maximally 50 candidates per item.

For all the rankers, we see that the evaluation scores for the underspecified data

are slightly but consistently lower than for the primary data. However, the average number of realizations generated per semantic input for the IS-underspecified profiles (115.7) is more than twice as high as for the primary profiles (52.8). Correspondingly, at 16.41%, the random choice baseline for the underspecified version is much lower than for the primary, at 27.28%. In terms of relative reduction in error rate with respect to the baseline, the greatest improvements are actually achieved for the underspecified data. For example, as seen in Table 8.5, the accuracy of the LM on the underspecified and primary data is 48.71% and 52.60% respectively. Relative to the baseline, this correspond to an error reduction of 38.64% and 34.82%, respectively. Considering the same figures for the combined MaxEnt model, the "underspecified" accuracy of 72.21% and the "primary" accuracy 73.98% of correspond to a relative reduction of baseline error rate by 66.75% and 64.22% respectively.

Chapter 9

End-to-End Reranking

As mentioned in the introduction, the hybrid generator system described in this thesis is already embedded in the LOGON prototype[1] system for Norwegian-to-English machine translation. When discussing the details of the overall LOGON system in Chapter 4, we noted how all the three main components of the MT pipeline are associated with indeterminacy: The analysis of one Norwegian source sentence might produce several parses; for each input MRS, the semantic transfer stage might produce several transferred target MRSs; and, finally, as should be clear by now, the generator might produce several possible target realizations. As illustrated in Figure 4.5, at each stage in the pipeline the process may potentially branch out, giving rise to a downstream proliferation of indeterminacy. Now, the naïve approach would be to simply take all the hypothesis produced at each stage and pass them downstream to the next component. However, as each component may produce several hundreds or thousands distinct hypotheses, the associated combinatorics mean that exhausting the full fan-out of possible branches through the system will often be prohibitively expensive. As mentioned in Section 4.4, each component is therefore equipped with a statistical module for ranking the possible hypotheses. Given this ranking machinery, it might intuitively seem like a reasonable approach to only consider the top-ranked hypothesis at each stage for further processing. However, this approach presupposes the ideal case where each statistical module always chooses the correct hypothesis. If (for example) the source analysis fails to give top rank to the parse with the correct semantics, it will be impossible for other components to later recover the correct translation. Note that, in the course of the following sections, we will sometimes refer to this selection strategy as *top*.

[1] A publicly available online version of the LOGON demonstrator can be accessed at http://noen.emmtee.net/. Note that currently this online demonstrator does not yet include the discriminative reranker as described here, but rather relies on a simple linear interpolation of just the three per-component scores.

In this chapter we describe an alternative strategy for selecting the "best" translation, as presented by Oepen et al. (2007). This approach uses a discriminative log-linear model for end-to-end reranking, i.e. ranking the ordered list of final target sentences $\{e_1, \ldots, e_n\}$ conditioned on the input source sentence f. Moreover, the overall approach is similar to that described by Och and Ney (2002) in the context of SMT. As an alternative to the noisy channel approach, Och and Ney (2002) show how log-linear models can be used for defining translation models where the posterior probability $p_\lambda(e|f)$ is modeled directly. Various log-linear models are then used for extending a baseline SMT system by adding new feature functions. Although Och and Ney (2002) work in the setting of *statistical* machine translation, many of the same issues carry over to the ranking task for a hybrid "baseline" system.

In addition to showing how the realization ranking approach developed in the preceding chapters fits into to the overall LOGON translation machinery, this chapter will also show examples of alternative applications of the modeling techniques we described in Chapter 2. While the reranker itself is based on the same framework of maximum entropy modeling as used for the treebank-based generation models, the post-transfer MRS ranking is based on the framework of sequential n-gram modeling. In the sections below we first describe the three components of the LOGON baseline system and their respective models for statistical ranking. As the author has not personally been involved in the development of the parse selection models, we will devote somewhat more space to the two remaining components; transfer ranking and realization ranking. We then go on to describe the global reranking model and its feature types in Section 9.4. Finally, in Section 9.4, we report evaluation results for the quality of the translations produced by the LOGON system when using the discriminative reranker. The discussion in this chapter draws heavily on the presentation in Oepen et al. (2007), but also provides additional background and technical details in some places.

9.1 Parse Ranking

The parse selection employed in the LOGON system relies on the stochastic disambiguation facilities developed at PARC as part of the XLE system (Riezler et al., 2002; Riezler & Vasserman, 2004). This disambiguation scheme is based on discriminative log-linear modeling of the same form that we have described for our treebank-based realization rankers. The XLE environment provides a set of parametrized feature function templates to be instantiated in accordance with a given grammar and training corpus. In LOGON, the models are trained on a treebank of manually disambiguated LFG analyses for a subset of the Norwegian part of the Jotunheimen development corpus. The treebank itself was created as part

of the TREPIL project, a sister project to LOGON, which is currently also developing a dedicated toolkit for treebanking LFG analyses, called the LFG Parsebanker (Rosén, Smedt, & Meurer, 2006). The resulting treebank as used for training the log-linear model is relatively small, containing roughly 500 sentences, all with full parses, but chosen as to have relatively low ambiguity (fewer than 100 readings). Even with this limited amount of training data, the model performs at 53.8% exact match accuracy when tested on a held-out segment of 140 sentences from the same corpus (with a corresponding random choice baseline of 27.3%). When evaluating the exact match within 5-best lists, the model achieves 84.3% accuracy (with a corresponding baseline of 74.3%).

9.2 Transfer Ranking

As described in Section 4.1.1, the task of transfer is to map the source language MRS representations into target language representations suitable as input to the generator. This mapping is implemented as a resource-sensitive rewrite process, carried out through the application of a series of successive *MRS transfer rules* (MTRs).

Of the three ranking tasks, transfer ranking is the task for which there is the least amount existing related work in the field. Moreover, while the two other ranking modules for parsing and generation are based on discriminative log-linear models, the transfer ranker is based on a generative n-gram model. As we shall see, the model is trained in a similar fashion as for the n-gram LM that we developed as a baseline realization ranker in Section 7.1.

While MRS formulas are highly structured graphs, Oepen and Lønning (2006) suggest a reduction into a variable-free form that resembles elementary dependency structures. For the ranking of transfer outputs, the MRSs are broken down into basic dependency triples. Similarly, as an attempt at capturing co-occurrence constraints among relations that do not stand in a direct dependency relation, we also extract bigrams of all semantic predicates within the same MRS (assigning a canonical lexicographic ordering). After constructing a "corpus" of such tuples, their probabilities are estimated by adaptation of standard n-gram sequence modeling techniques. The actual training is done using the freely available CMU SLM toolkit (Clarkson & Rosenfeld, 1997), which we also used for training the n-gram LMs in Section 7.1.

Based on a training set of some 8,500 in-domain MRSs, viz. the treebanked version of the English translations of the (full) LOGON Tourist development corpus, our target language "semantic model" is defined as a Witten-Bell discounted trigram back-off model over the reduction of MRSs into dependency tuples. Figure 9.1 shows an example structure, corresponding to a total of ten triples, in-

```
{
  prpstn_m[MARG _recommend_v]
  _recommend_v[ARG1 pron, ARG2 _hike_n]
  _a_q[ARG0 _hike_n]
  _around_p[ARG1 _hike_n, ARG2 _source_n]
  implicit_q[ARG0 _source_n]
  poss[ARG1 _waterway_n, ARG2 _source_n]
  def_q[ARG0 _waterway_n]
}
```

Figure 9.1: Variable-free reduction of the MRS for the utterance 'We recommend a hike around the waterway's sources'.

cluding, for example, ⟨_around_p, ARG1, _hike_n⟩. The "vocabulary" of the model comprises some 4,400 distinct semantic predicates and role labels, for a total number of more than 51,000 distinct triples (distributed over a total of almost 600,000 examples), and almost 219,000 distinct (ordered) bigrams (distributed over almost 1,100,000 examples). When applying the model, post-transfer English MRSs are similarly broken down into segments of dependency tuples, before finally being ranked according to the perplexity scores assigned by the model.

Given that we do not have a transfer-level "treebank", evaluating the MRS ranking in isolation is not straightforward. However, lacking such data, we can contrast end-to-end system performance on the Jotunheimen test set with and without the ranker. When generating target strings on the basis of the ranked 5-best lists, success rate in generation is 86.5%. When instead using a non-ranked, random selection of five transfer outputs, the success rate in generation drops to 82.7%. In other words, contrasted with a random choice baseline, the transfer ranking leads to a relative reduction of generator failures by 22%. We also evaluated the impact of the MRS ranking when using the *top*[2] strategy for identifying the best translations (i.e. only using the top-ranked candidate at each stage through the pipeline, as described in the introduction). Restricting the comparison to the 109 test items that successfully translate in both configurations, our BLEU score over the final *top* translations drops from 37.41 to 30.29.[3]

[2]Note that this approach is referred to as *first* in the original discussion by Oepen et al. (2007), while an oracle approach employed for computing an upper bound is referred to as *top*.

[3]The BLEU measures in all our experiments are calculated using the freely available NIST Perl package, accessible at
ftp://jaguar.ncsl.nist.gov/mt/resources/mteval-v11b.pl.

9.3 Realization Ranking

As has been discussed in preceding chapters, for each target MRS that is passed on from transfer, the generator might produce several possible English realizations. The realization ranker that we use for ordering the generated sentences corresponds to a discriminative log-linear model as that developed in Section 7.2.6, trained on the in-domain Tourist generation treebank and including the BNC LM as a separate feature. However, for the results reported by Oepen et al. (2007), the integrated LM is only a trigram model (not a 4-gram model as used for the other generation experiments in this thesis).

Recall that, throughout the previous chapters, in addition to evaluating the realization rankers in terms of the top-ranked candidates, we have often also evaluated them in terms of 5-best lists. One of the reasons why this form of evaluation is relevant, as also mentioned in Section 6.3, is that the mode in which the generator is used within LOGON is for producing n-best lists of ranked realizations. In other words, similar as for the output from the analysis stage and the transfer stage, we only consider a subset of the available hypotheses, extracted in the form of ordered lists of the n top-ranked candidates. At the end of the pipeline, we then apply the global reranking model to find the most probable target candidate within this reduced space of hypotheses.

Note that, for the results reported in Oepen et al. (2007) and below, the fan-out branching factor is limited to a maximum of five output candidates from parsing and (within each branch) transfer. However, because there is no further downstream processing after generation, the indeterminacy of the generator will not further multiply in the same way, and we can therefore afford more candidate realizations per input MRS. For the current experiments, the 50 best candidates from generation are passed on to the reranker. This yields a total of up to $5 \times 5 \times 50 = 1250$ distinct fan-out outcomes. Note, however, that it is quite common for distinct fan-out paths to arrive at equivalent outputs, for example in cases where the same modifier attachment ambiguity is present in the source and target language.

Note that the 50-best lists of realizations for each input MRS can be extracted very efficiently using the technique of *selective unpacking* developed by Carroll and Oepen (2005). Recall that the LKB generator actually produces a compact representation of all the possible realizations in the form of a *forest structure*. As presented by Carroll and Oepen (2005), a dynamic programming algorithm is then applied for selective unpacking of the generation forest, obviating the need to enumerate all the possible candidates. By guiding the search through the forest using a discriminative realization ranker as developed in this thesis, Carroll and Oepen (2005) show that the selective unpacking procedure is guaranteed to extract the exact n-best list of optimal candidates according to the model.

9.4 End-to-End Reranking

After completing the actual translation process, branching out using the $5 \times 5 \times 50$ best candidates through the pipeline, the target translations are reranked using a global discriminative log-linear model. For a given source sentence f and translation e, the reranker directly models the posterior probability $P_\lambda(e|f)$. Although the reranker we describe here is built on top of a hybrid baseline system, the overall approach is similar to that described by Och and Ney (2002) in the context of SMT. As described in Section 2.3, one of the properties that make the framework of log-linear modeling attractive is the flexibility it allows in terms of combining disparate and overlapping sources of information in a single model, without running the risk of making unwarranted independence assumptions. Moreover, an advantage of working with a reranking setup is that the model can use global features that might not be available to the baseline system. Recall that a log-linear model is given in terms of a set of *specified features* that describe properties of the data, and an associated set of *learned weights* that determine the contribution of each feature. The information that the feature functions record can be arbitrarily complex, and a given feature can even itself be a separate statistical model. Of course, we have seen an example of this in relation to the realization ranker, where we include the score of the n-gram LM as a feature in the treebank model. In fact, all of the features of our reranking model are defined in terms of separate models in this way. While the model uses a small number of feature functions (11 in total), all of them can be seen to correspond to independent auxiliary models (or metrics). In this sense, the weight estimation can be seen as a search for the best way to combine the different models.

9.4.1 Features

The three most "fundamental" features in our log-linear reranker correspond to the three ranking modules of the baseline system, as described in the preceding sections above. In other words, these features record the scores of the parse ranker, the MRS ranker, and the realization ranker, respectively. Note that, for the log-linear parsing model and realization model, we define the features not in terms of the conditional probabilities, but rather use the non-normalized scores directly, i.e. summing the products of all weights and features. One reason why we do not normalize the scores to represent probabilities is that the scored candidates will often correspond to different branches through the pipeline, e.g. realizations generated for distinct MRSs. Moreover, as the models are *conditional*, normalization would result in the unwanted property that the score of a given candidate is dependent on the number of other alternative candidates in the same branch. For the transfer feature the value corresponds to the perplexity of the given set of dependency

tuples as computed by the n-gram model. In addition to these three core features, our reranking experiments so far have also taken into account another eight properties of the translation process. Based on the discussion in Oepen et al. (2007), we briefly present these features in turn below.

One important additional feature type corresponds to *lexical translation probabilities*. These are estimated on the basis of a small corpus of Norwegian–English parallel texts, comprising 22,356 pairs of aligned sentences. Out of these, 9,410 sentences are taken from the LOGON development data, while an additional 12,946 sentences are drawn from the English-Norwegian Parallel Corpus (Oksefjell, 1999). When including punctuation and sentence boundary markers, the token count for both languages is just above 300,000 (the English version contains a few thousand more words than the Norwegian text). The estimation of the lexical probabilities is carried out using the training scripts that are as distributed with the phrase-based SMT module *Pharaoh*, as implemented by Koehn (2004). The procedure is as follows: First, GIZA^{++} is used for producing word alignments in both directions, i.e. using both languages as source and target in turn. On the basis of these alignments, a maximum likelihood translation table is estimated, again in both directions. Finally, when defining the actual feature function for a given bi-directional sentence pair $\langle e, f \rangle$ and $\langle f, e \rangle$, we compute the length-normalized product of all pairwise word-to-word probabilities. It is worth noting that this feature can be thought of as a proxy for the sentence-level translation probabilities of the corresponding SMT model. Extracting these probabilities would require modifications to the decoder of the SMT engine, but this is something we are currently trying to solve in ongoing work. To be clear, we are not interested in performing the actual SMT decoding step in itself. We are only interested in computing the translation probabilities of the candidates produced by our own baseline system, for the purpose of including these as features in the reranker.

As described above, the log-linear realization ranker already includes a generative trigram LM as a separate feature. However, the LM is also included as a separate feature in the end-to-end reranker, as an independent indicator of output fluency. Another feature measures what we call *EP distortion*. The EPs (elementary predications) of the MRSs are linked to the corresponding surface elements by way of sub-string pointers. This information is preserved in transfer, so that post-generation we can compare the relative surface positions corresponding to the EPs in the input source and output target string. The EP distortion feature measures the degree of reordering of these surface pointers. Other features measure such properties as the ratio of word counts in the source and the target, the ratio of EP counts, and the total number of transfer rules that were invoked (as a measure of transfer granularity). A final feature in the model measures "semantic compatibility" between the MRS originally given as input to the generator and the finally generated realization. As previously mentioned in Section 4.3, a post-generation test which

is applied after all the possible realizations have been generated, sifts out any candidates whose MRS is not subsumed by the input MRS (Carroll & Oepen, 2005). Given an imperfect input (or error in the generation grammar), it is possible for none of the candidate outputs to fulfill the semantic compatibility test. If such a case arises during translation, the generator will gradually relax MRS comparison, going through seven pre-defined levels of semantic mismatch which we encode as an integer-valued feature in the reranking model.

Given that we are using are so many diverse feature types—probabilities, perplexity values, non-normalized log-linear scores, and of various other measured properties—the feature values are normalized into a comparable range, using min-max scaling. Then, just as for the realization rankers developed in Section 7.2, the model is estimated using TADM (Malouf, 2002), maximizing the regularized conditional likelihood of the training data as in Equation (2.33) (Johnson et al., 1999). In other words, for each input source sentence in the training data we seek to maximize the probability of its annotated reference translation relative to the other competing candidates. In the next section, we describe how the actual data set itself is defined.

9.4.2 Data Sets

As described in Section 4.4.1, the domain of LOGON is defined in terms of the texts that comprise the Tourist corpus. These texts consists of booklets on back-country activities in Norway, published in both Norwegian and English. In addition to the originally published translation, up to two additional translated versions are included on the English side. Furthermore, about ten per cent of the parallel corpus was reserved for held-out testing, of which one half only contains *known* vocabulary while another half additionally includes *unknown* vocabulary (i.e. words that do not occur in the development segment). For training and testing of the reranker, we only use the *Jotunheimen* subsection of the Tourist corpus. In addition to comprising the largest part of the corpus, the Jotunheimen section also come with three different translation references.

Some basic properties of the data sets (both the development set and the held-out test set) are summarized in Table 9.1. We see that for both versions the coverage of the system (i.e. the number of items that are actually translated) is just below 65%.

Although each source sentence in the Jotunheimen corpus is associated with three reference translations, we can not, of course, be certain that the MT system always produces a translation that exactly matches one of the references. Therefore, for creating the necessary training data, we mechanically "annotate" (i.e. label) candidate translations by means of the sentence-level NEVA (Forsbom, 2003) string similarity measure. For each source sentence, the translations with

	Items	Length	Coverage	Strings
Development set	2146	12.6	64.8	266.0
Held-out test set	182	11.7	63.2	114.6

Table 9.1: LOGON development and held-out corpora (for the *Jotunheimen* segment). Average string length and end-to-end coverage on the two sets are comparable, but the average number of candidate translations is higher on the development data.

the maximum NEVA score (among all candidate outputs for this input) are marked as preferred. In effect this means that the empirical distribution is defined in such a way that the parameters of the log-linear model are estimated to maximize the conditional probabilities of the candidates with the highest NEVA scores. Note that, initially, the way we here define the training data is similar to how we constructed the preference weighted training data for the realization ranker in Section 7.2.7. However, given the experience we reported there, we here only use the weights (i.e. the similarity scores) to define a binary distinction between references and non-references.

Note that, while we use NEVA for the automatic identification of references, we use BLEU for the corpus-level model evaluation. As described in Section 6.3.2, there are well-known problems associated with using BLEU at the sentence-level, which is why we use NEVA instead of BLEU for the automatic labeling of the training data.

9.5 Evaluation

For the evaluation in this section, we are only interested in the isolated performance of the reranker. Consequently, we only compute scores for the cases that are actually translated by LOGON (cf. the *Coverage* column in Table 9.1). Note that neither the reranker nor the statistical layers of the individual system components (parsing, transfer, and generation) are concerned with matters of *robustness*. The only objective of the statistical extensions is to handle the problems of indeterminacy and choice.

Table 9.2 summarizes end-to-end system performance (in terms of corpus-level BLEU scores and using three references) for various strategies of selecting hypotheses from the n-best lists obtained from the $5 \times 5 \times 50$ fan-out. The column labeled *reranker* corresponds to the discriminative log-linear model developed in Section 9.4 above. We give results for both the development data and the held-

out test data. The evaluation of the former is computed through ten-fold cross-validation, while the latter is scored using a model estimated from the complete development data.

			Baselines		Oracles	
	Items	**Reranker**	**Random**	**Top**	**Max**	**Human**
Development set	1391	44.10	34.18	40.95	49.89	–
Held-out test set	115	38.92	30.84	35.67	45.74	46.32

Table 9.2: BLEU scores for various end-to-end ranking schemes on the Jotunheimen data. Evaluation only includes items actually translated by LOGON (second column).

We also include two *baseline* measures in Table 9.2. The figure in the column *random* corresponds to using a random choice of one output in each context (averaged over twenty iterations), resulting in BLEU scores of 34.18 and 30.84 for the development and held-out set, respectively. The second baseline figure, listed under the column heading *top*, corresponds to always using only the top-ranked candidate from the ordered list for each component. In other words, this "informed baseline" corresponds to pursuing only a single path through the pipeline, finally resulting in a single translation (hence with no need for subsequent reranking).

As an *upper bound* on the reranking performance, Table 9.2 also provides two "oracle" scores: The first, labeled *max*, is obtained by selecting translations with maximal NEVA scores, i.e. using sentence-level NEVA as a proxy for corpus-level BLEU. The second, labeled *human*, reflects the annotations of a human judge on the held-out data: considering all available candidates, a native speaker of (American) English and near-native speaker of Norwegian, in each case, picked the translation judged most appropriate (or, in some cases, least awful). Oracle BLEU scores reach 49.89 and 46.32, for the development set and the held-out test set, respectively.

Now, we first note that the informed baseline, corresponding to always using the top-ranked candidate from each component (*top*), clearly improves over the random choice baseline. However, the log-linear reranker outperforms both of these baselines by a solid margin, yielding BLEU scores of 44.10 and 38.92 for the development set and the held-out set, respectively. Note that, although the performance seems to drop on the held-out data, associated baselines and oracle scores are also lower. Across all selection strategies we find that the BLEU scores on the training corpus are higher by about four points. Although we are pleased to see the clear improvement over the baselines, there is nonetheless also room for further improvements towards the oracle upper bound. We intend to continue experimentation with the reranking approach in the future, adding other feature types and performing additional fine-tuning. We also plan to perform a manual

error analysis, as well as a more accurate assessment of the contribution of the different feature functions. Finally, we also plan to explore other approaches for estimating the λ-parameters, for example by following the *Minimum Error Rate* approach of Och (2003), optimizing the scores of a given evaluation metric such as BLEU or NEVA directly.

Chapter 10

Summary and Concluding Remarks

This thesis has developed a new approach to the problem of indeterminacy in grammar-based tactical natural language generation. The problem itself concerns the fact that, for a given input semantic representation, the grammar might allow for several alternative paraphrases. For the specific generator we have worked with here—the LKB generator as used for producing English target language realizations within the LOGON MT system—we migth get several hundreds, and even thousands, of different surface strings generated for a single input semantics. In order for such a system to be of practical use for an end-user, we need a way of scoring and ordering the different candidates, and finally select a single output. This is the task we have here referred to has *realization ranking*.

Data-driven statistical methods provide us with a way to handle such cases of indeterminacy in a principled and systematic manner, and the traditional approach in statistically guided NLG has been to score and rank the paraphrases using n-*gram language models* (LMs). This is the approach pioneered in the Nitrogen and HALogen systems (Langkilde & Knight, 1998a; Langkilde, 2002). However, due to the many limitations inherent in such sequential and surface-oriented models, we have here developed a more linguistically informed approach, using models that are sensitive to the internal structure of competing realizations. The starting point of our approach is to recognize the symmetry in the ranking tasks for parsing and generation. By extending on the well-established methodology for statistical parsing and statistical unification-based grammars, and adapting this for the setting of generation, we have been able to successfully train *discriminative treebank models* for realization ranking. In particular our approach builds on the work by Toutanova et al. (2005) on statistical parse selection on the Redwoods treebank, using the LinGO ERG. This is a general-purpose wide-coverage lexicalist HPSG grammar, and it includes logical-form semantic representations based on MRS. As the grammar is specified within a completely declarative constraint-based formalism, it is also bidirectional, in the sense that it can be used for both parsing and

generation. Moreover, the same frameworks are used for generation within the LOGON MT system: Given an MRS produced in the semantic transfer step, the LKB generator produces English strings as licensed by the ERG.

A key step in our adaption of the parse selection approach for generation was to introduce the notion of a *generation treebank*. This is a treebank where semantic representations are paired with their set of corresponding linguistic realizations, as licensed by the underlying grammar, and with labels indicating the preferred realization(s) of each semantics. We have shown how such a generation treebank can be automatically created on the basis of an existing parse-oriented treebank, and for the experiments in this thesis we have successfully created several such generation treebanks on the basis of the Tourist corpus developed in LOGON. This procedure is based on the assumption that the preference relations encoded in the original treebank can be interpreted as *bidirectional* or *symmetric*. The procedure itself then requires mainly two steps; First we use the semantics associated with the originally treebanked strings as input to the generator, producing the full set of grammatically allowed paraphrases. Then we identify the references among these paraphrases by matching them against the strings in the original corpus. This provides us with the training data necessary for estimating a discriminative realization ranker, or what in a sense can be said to correspond to a statistical unification-based "generation grammar". In this thesis, this was done by training a *conditional maximum entropy model* that minimizes the divergence toward the empirical distribution given by the annotated generation treebank. Equivalently, the model can be seen to maximize the conditional log-likelihood of the training data, in a similar manner as described for statistical parse selection by Johnson et al. (1999). Moreover, as is common practise when estimating log-linear models, we used a regularized likelihood function by imposing a Gaussian prior distribution on the estimated feature weights (Chen & Rosenfeld, 1999; Johnson et al., 1999; Malouf & van Noord, 2004). The specific feature functions that we used were defined as an extension to the feature set used by Toutanova et al. (2005) for HPSG parse selection. The features are defined over grammatical derivation trees, recording various information about local sub-tree configurations as well as n-grams over abstract lexical types defined for the preterminal yields.

To contrast the performance of the treebank model with the more traditional approach of using n-gram language models, we have carried out extensive experimentation also with this modeling framework. The LM we ultimately used in the comparative evaluations was a 4-gram back-off model trained on the 100-million-word BNC. The estimation was done using the CMU SLM toolkit, with a vocabulary that was tuned to the relevant domain on the basis of the Tourist development data.

Before we go on to summarize the evaluation, note that we have reported test results for all our models on both the development data and the held-out data. Furthermore, both of these data sets come in two versions. The primary or standard

version, which is most representative of the practical ranking task within LOGON, includes aspects of information structure (IS) in the input MRS, governing phenomena such as topicalization and passivization. In a secondary version of these data sets, we treated the IS properties of the MRSs as underspecified during generation, leading to an even greater degree of indeterminacy. For the primary versions of the generation treebanks, there are 3921 items in the development data and 269 in the held-out data. The average number of realizations per input semantics is roughly 50, while the average string length of the realizations is roughly 18 tokens. For the underspecified versions, we had 4720 items for development and 349 items for held-out testing, averaging roughly 115 realizations per item, with an average string length of roughly 16.5. Finally, note that for a random choice baseline, the expected ranking accuracy on the IS-underspecified versions is roughly 18%, while it is closer to 28% for standard version.

After comprehensive testing we have seen that the treebank-based MaxEnt model performs substantially better then the n-gram language model for our realization ranking task. As summarized in Table 10, we have evaluated the performance according to exact match accuracy (i.e. the percentage of times that the top-ranked candidate exactly matches the reference), as well as the WA and NEVA string similarity metrics. Note that, on the development data, the treebank model was trained and tested using ten-fold cross-validation. On both the development data and the held-out data, the differences in scores are found to be statistically significant for $\alpha = 0.05$ (using a two-tailed matched-pairs version of the Wilcoxon signed-rank test and the sign-test).

In addition to evaluating the ranking performance in terms of the automatic metrics in Table 10, we also report results for a manual evaluation study, carried out with the help of seven anonymous judges. The judges are all students in the Professional Master's in Computational Linguistics Program at the University of Washington, and were recruited by Prof. Emily M. Bender who is the director of the program. Based on a random sample of items from the held-out data for which the treebank model and the LM disagreed on the top-ranked candidate, we constructed an evaluation form where the students were asked to judge the relative quality of the different surface realizations. For each item in the evaluation set we had also included one randomly chosen candidate, as well as the original string as it occurred in the underlying corpus. The results of this study showed that the evaluators consistently tended to assign a higher rank value to the sentence chosen by the MaxEnt model, as compared to the LM. Computing the Spearman rank correlation for the per-item average ranks also showed that the inter-judge agreement was relatively high. In fact, when considering the relative ordering of the models in terms of overall average rank-values, we found the judges to be in unanimous agreement. When considering the LM and the MaxEnt model in isolation, looking at the per-item rank values assigned to their respective sentences, we also found

the differences to be statistically significant according to the sign-test.

In order to shed light on the types of errors committed by the rankers, we also performed a manual error analysis, representing a joint effort by Prof. Stephan Oepen and the author. Based on a random sample of errors from the development data, we contrasted the errors made by the 4-gram LM and the MaxEnt treebank model, and found that a substantial portion of the errors were unique to the individual learners. Note that the random sample we considered was restricted to only contain items for which either of the models had a mismatch between its top-ranked candidate and the reference. Now, in the course of this manual error inspection we occasionally judged the "incorrect" top-ranked candidate to be of equally good quality as the reference, but where the mismatch was nonetheless penalized by to the (potentially overly) stringent measure of exact match accuracy. Although such mismatches caused by "spurious" preferences in the reference data will affect both the LM scores and the MaxEnt scores, we saw some evidence that the exact match measure penalizes the LM harder for these cases. This is because the MaxEnt model is always trained on material from the same underlying corpus, and thereby tends to be more sensitive to the preferences in the treebank. On the other hand, this is an observation that "cuts two ways", as it also means that the MaxEnt model is more sensitive to the presence of noise or annotation errors. This latter point is highly relevant since we did note a non-trivial number of cases where the mismatched items were associated with errors in the original parse treebank (from which the generation treebank is constructed). Naturally, for both the LM and the MaxEnt model, such annotation errors in the treebank can have a negative impact on ranking performance at the point of testing. However, for the MaxEnt model they can have a negative impact already at the point of training as the model is trained on material from the same treebank and can have its parameters fit according to the noise and errors. On the brighter side, the existence of such annotation errors means that we can expect to obtain even better ranking accuracy for our models as the these errors get corrected in the grammar and the treebanks.

The error analysis also revealed that the errors made by the two models tended to often not overlap. For the 3921 items in the Tourist development data, more than 53% of the mistakes made by the LM were unique to that model (when compared to the MaxEnt ranker). Similarly, we found that more than 38% of the mistakes made by the MaxEnt ranker were not made by the LM ranker. The fact that the different rankers seemed to have somewhat different strengths and weaknesses, motivated us to try to combine the two models into one. Given the flexibility of the MaxEnt approach, we simply added the LM scores as a separate feature in the MaxEnt model. Using this combined model, drawing on information from diverse knowledge sources, we obtained better ranking performance than for any of the individual component models on their own. The observed differences in

performance were found to be statistically significant for the development data, although the same was not clear for the differences observed for the smaller held-out set.

As an alternative to the framework of MaxEnt modeling, we have also carried out a large number of experiments with training *SVM rankers*, following the approach described by Joachims (2002) in the context of information retrieval. This framework could potentially have allowed us to learn non-linear ordinal ranking functions, reflecting a relative ordering of all alternative paraphrases for each item in the training data. The initial ordering in the training data was defined in terms of the similarity between the competing realizations and the respective references, e.g. as measured using WA. However, despite having access to quite substantial computing resources, we found that the full-fledged SVM ranking approach did not scale to the size of our ranking task. We did, however, manage to train *linear* SVM rankers using only the *binary* distinction between references and non-references in the treebank (i.e. defining the training data in the same way as for the MaxEnt models). For the IS-underspecified version, we furthermore had to limit the number of training examples by using random samples of maximally 50 paraphrases per item. In the end though, none of the SVM rankers achieved better results than the MaxEnt models, using exactly the same feature configuration. Still, as seen Table 10, the SVM performance was at a comparable level, and the differences were not detected as being statistically significant.

As said, the hybrid generation system we have been describing is currently used as part of the LOGON MT system. Nonetheless, we have stressed the fact that the main focus of this thesis is not on the problem of ranking *translation output*, but rather on the isolated and more general task of ranking *generation output*. It is worth noting that the ERG-based LKB generator is also in active use in other applications, e.g. speech prosthesis, paraphrasing, and grammar and controlled language checking. Still, complementing the core topic of this thesis, we have also described the training of a discriminative *end-to-end MT reranker*, as used for selecting the final output of the LOGON MT system. Recall that our log-linear realization rankers directly model the posterior probability $q_\lambda(e|s)$ of an English realization *e* given an input semantics s. The end-to-end reranker, on the other hand, directly models the posterior probability $q_\lambda(e|f)$ of an English target sentence *e* given a Norwegian source sentence *f*. In this sense the approach is similar in spirit to that of Och and Ney (2002) in the context of reranking for SMT.

Note that the reason why we refer to this as *re*ranking, is that the underlying hybrid "baseline" system already can be seen to deliver n-best lists of ranked hypotheses. This is based on the per-component statistical ranking through the MT pipeline. In other words, in addition to the realization ranker developed in this thesis, the LOGON system also employs a parse ranker and a transfer-level MRS ranker. While the parse ranking is based on existing technology within the

Summary of Test Results for the Tourist Development Data

	Primary			IS-underspecified		
	Accuracy	WA	NEVA	Accuracy	WA	NEVA
4-gram LM	53.75	0.907	0.907	50.04	0.825	0.873
MaxEnt	72.11	0.941	0.943	68.05	0.889	0.919
MaxEnt w/ LM	**74.25**	**0.944**	**0.946**	**70.47**	**0.897**	**0.925**
Lin. SVM w/ LM	73.84	0.942	0.944	68.99	0.892	0.920

Summary of Test Results for the Tourist Held-Out Data

	Primary			IS-underspecified		
	Accuracy	WA	NEVA	Accuracy	WA	NEVA
4-gram LM	52.60	0.904	0.887	48.71	0.819	0.857
MaxEnt	71.31	0.941	0.939	71.20	**0.887**	**0.923**
MaxEnt w/ LM	**73.98**	**0.944**	**0.941**	**72.21**	0.884	0.920
Lin. SVM w/ LM	73.61	0.939	0.938	71.06	0.878	0.918

Table 10.1: Summary of evaluation results for the different types of realization rankers on both the Tourist development data and the held-out test data. *Primary* refers to the standard Tourist held-out test data, while *IS-underspecified* is the version constructed without encoding of information structure attributes in the MRS. Note that, for the discriminative models, results on the development data are computed using ten-fold cross-validation.

XLE system, this thesis does provide some details on the development of our MRS ranker, which is based on n-gram statistics over dependency triples. Now, all of these ranking modules are included as feature functions in the end-to-end reranker. In addition, we have incorporated several other global reranker features such as lexical translation probabilities, source–target distortion measures, etc.

Naturally, there still remains many unopened doors in relation to the ranking approach developed in this thesis which we have not yet had time to explore. We hope to be able to pursue some of these possibilities in the near future however. To give an example, in relation to the end-to-end reranker we noted that an interesting venue for future work would be to train the model using a *Minimum Error Rate* approach as described Och (2003). Instead of maximizing the log-likelihood of the training data, this would mean optimizing the scores of a given evaluation metric such as BLEU, WA, or NEVA directly. Of course, this reranking approach could also be very interesting to pursue in relation to the treebank-based realization rankers. We briefly discuss some other such possibilities for further

experimentation below.

We have noted that the starting point of our approach is provided by the statistical parsing experiments of Toutanova et al. (2005). As noted in Section 7.4, it seems justified to say that the log-linear parsing models of Toutanova et al. (2005) and the log-linear generation models developed here, actually perform on a comparable level (although any direct comparison of the results is, of course, not possible). Despite differences in data sets, grammar versions, and the overall ranking task itself, the accuracy of the best performing models seem to at least lie within the same region. We take this as yet another indication that the transfer of methodology from the area of statistical parsing to the area of statistical generation was indeed well founded. Still, the fact that most of the model features were initially designed for the purpose of parse selection means that there is reason to believe that even better realization rankers can be built by further experimentation with additional feature functions specifically designed for this task. In relation to this it is also worth pointing out that, similarly to how we included the LM as a feature in the MaxEnt model, we could, of course, easily plug in additional auxiliary models in the same way. As an example, consider the models developed by Malouf (2000), specifically targeting the isolated problem of prenominal adjective ordering in generation. As further extensions to the treebank models developed in this thesis, it would be highly interesting to include several such specialized model as additional features.

In relation to the LM feature, we hope to be able to train models on larger data sets of in-domain material. As noted earlier, in Chapter 7 above, we have already tried to do this for the development data available in LOGON, but this data was not sufficiently large that we were able to improve on the general-domain BNC model. In the same vein, the preceding chapters have shown seen several examples of how the log-linear treebank models too seem to benefit from additional training data. This was indicated both by the learning curves that we computed for the development set, as well as the improvement in performance we witnessed for the IS-underspecified held-out data (compared to the underspecified development data, see Table 10). Note that as the development results were computed through ten-fold cross-validation, we never got to take full advantage of the available training data. The boost in accuracy on the held-out data then, was probably due to the increased amount of training data that was used when training the final treebank models on the complete development set. Recall that the reason why this effect might be more visible on the IS-underspecified version than the standard version is that the former actually includes more examples (due to technicalities in the treebanking procedure). All in all, these trends means that we can anticipate even better realization ranking results in the future as the amount of available treebank material continues to grow. In a similar vein, we also plan to create generation treebanks for other segments of the Redwoods treebank, in order to investigate

the performance of general-domain (and cross-domain) treebank models.

These last remarks point to a very important fact in relation to data-driven methods which is sometimes not fully appreciated. This is the importance of having access to large quantities of relevant data, preferably of high-quality (i.e. low noise ratio etc.), in annotated form, and tuned to the relevant domain. Although the reliance on data for data-driven methods may appear self-evident, and indeed it is, the creation of data resources is typically regarded as unrewarding work, not associated with the same prestige as for example the invention of yet another variation of a learning algorithm. Still, for all of the approaches investigated in this thesis it seems that additional data is the single most important factor when it comes to further improving the results.

To summarize, this thesis has introduced the novel notion of a generation treebank and shown how it can be used for training discriminative log-linear models for ranking the output of a grammar-based generator. The approach extends on the methodology previously used within the field of statistical parse selection, adapting it for the task of statistical realization ranking. Moreover, we have shown that, on our data sets, these treebank models substantially outperform the n-gram-based language models which have traditionally been applied for this task in statistical generation. We finally showed how even better ranking performance may be achieved by applying the two types of models in combination. Note that, although we have usually construed our generation task as following a *generate-and-select* approach, Carroll and Oepen (2005) show how the discriminative realization rankers we have developed in this thesis can be used for guiding the *selective unpacking* of realizations from a packed forest representation. In other words, Carroll and Oepen (2005) show how the discriminative rankers can be applied for on-demand enumeration of n-best lists, in globally correct rank order, with minimal search through the forest. Moreover, the hybrid generator described in this thesis, coupling a linguistic grammar with statistical ranking, is currently used for target language generation in the LOGON MT system for Norwegian–English translation of texts in the domain of Tourism. In this relation we have also shown how the realization ranker is integrated in the overall MT pipeline, and how it is used together with a discriminative log-linear model for end-to-end reranking of translations. As evidenced in emerging follow-up work such as that by Nakanishi et al. (2005) and Cahill et al. (2007), the realization ranking approach developed in this thesis is not restricted our particular system configuration, but also generalizes to other systems based on different grammar formalisms, target languages, or generators.

Appendix A

Human Evaluation Questionnaire

This appendix includes, in its entirety, the evalutaion form presented to the panel of external anonymous judges, as described in Section 8.1. The evaluation form was given to each judge as a plain ASCII file, in which they then edited the fields for rank values directly (e.g. |?| → |1|).

Following is a selection of outputs from a Natural Language Processing system. When given a meaning representation (as a logical expression) for a sentence, the system consults a computational grammar of English to find all strings that (according to the rules of its grammar) are (a) wellformed and (b) express the correct meaning. Often, the system will generate hundreds or more of candidate strings; to pick the most natural sounding output, the system further applies a ranking mechanism, based on a statistical model of English sentence structure.

To help us evaluate the performance of this ranking component, we kindly ask you to order alternate outputs according to how well they sound to you. We present blocks of between two and four candidates in each set, separated by '===' lines to mark the start and end of each block. Where necessary, blocks of candidates are preceded by one or more sentences of immediate context, i.e. sentences that occur right before the candidate block in the narrative. Note that some blocks only have [...] as their preceding context, indicating that a chunk of text has been omitted, and that this block can be judged in isolation. Other blocks may not be preceded by any context at all, in which case the immediately preceding block provides the relevant context. Finally, in a few cases multiple blocks essentially correspond to the same meaning; for these examples, please attempt to judge each block by itself, i.e. irrespective of another block with the same underlying meaning.

Here is an example:

[...] Sikkilsdalen is one of Norway's lushest mountain valleys and is much used for grazing.
===
|?| From Sikkilsdalsskaret, there are two trails marked. (590)
|?| From Sikkilsdalsskaret, there are two marked trails. (591)
|?| From Sikkilsdalsskaret, there are 2 marked trails. (592)
===

In judging a candidate block, please compare the various strings and work out (a) how they differ and (b) which of those variants you consider more or less natural. For this example, the ranking could be:

```
===
 |3|    From Sikkilsdalsskaret, there are two trails marked. (590)
 |1|    From Sikkilsdalsskaret, there are two marked trails. (591)
 |2|    From Sikkilsdalsskaret, there are 2 marked trails. (592)
===
```

Clearly, 'trails marked' is fairly odd. Furthermore, in written English, many style guides recommend using literal numerals ('two' instead of '2') for small numbers (up to twelve, say). Alternatively, the following would be possible:

```
===
 |3|    From Sikkilsdalsskaret, there are two trails marked. (590)
 |1|    From Sikkilsdalsskaret, there are two marked trails. (591)
 |1|    From Sikkilsdalsskaret, there are 2 marked trails. (592)
===
```

Here, the two bottom variants tie for the first rank, i.e. are judged equally good. Ties are perfectly legitimate where you find it impossible to rank two (or more) candiates with respect to each other. Accordingly, in our example, there is no second rank, and the candidate showing odd word order is ranked third.

Please work through this file in sequential order, replacing all |?| boxes with your judgments on the relative ranking of candidates. We plan to process these files automatically, so please return your results electronically, and please avoid making textual changes to this file, other than replacing our question marks with your numeric scores.

Native Language :
Age : ?? years
Gender :

[...] In terms of fish, there are only trout, and the stock varies a lot.

```
===
 |?|    The catch is generally less certain than it used to be. (430)
 |?|    The catch is generally less certain than it used to. (431)
 |?|    Generally, the catch is less certain than it used to be. (432)
===
```

[...] Come as You Please and Leave When You Want

===
|?| The unique thing about mountain tourist lodges is that you don't have to reserve a place in advance and you won't be turned away even when the lodge is full (440)
|?| The unique thing about mountain tourist lodges is that you do not have to reserve a place in advance, and you will not be turned away even when the lodge is full. (441)
|?| The unique thing about tourist mountain lodges is that you don't have to reserve a place in advance and you will not be turned away even when the lodge is full (442)
|?| The unique thing of mountain tourist lodges's that you do not have to reserve a place in advance, and you will be not turned away even when the lodge is full (443)
===

===
|?| If all beds are taken, you will be accommodated at least on a mattress in a corner anyway. (450)
|?| If all beds are taken you will be accommodated anyway at least on a mattress in a corner. (451)
|?| Anyway, you will be accommodated at least on a mattress in a corner if all beds are taken. (452)
===

[...] You have many options: Would you prefer the hut staffed, or do you prefer to do your own housekeeping? [...]
===
|?| Would you like to dance Norwegian folk dances in the evening or sit on the doorstep alone and let your imagination be colorred by the sunset? (460)
|?| Would you like to dance Norwegian folk dances in the evening, or sit alone on the doorstep and let your imagination be colorred by the sunset? (461)
|?| Would you like to dance folk Norwegian dances in the evening or sit alone on the doorstep and let the sunset color your imagination? (462)
|?| Would you like to dance Norwegian folk dances in the evening, or sit alone on the doorstep and let the sunset color your imagination? (463)
===

===
|?| Do you want to stay at an old summer dairy or in a newer building with all conveniences? (470)
|?| Do you want to stay at an old summer dairy or in a more new building with all conveniences? (471)
===

[...] In the mountain areas covered by this book, there are three different categories of tourist lodging.
===
|?| Staffed lodges have their own hosts who serve all meals. (480)
|?| Staffed lodges have their own hosts which serve all meals. (481)
|?| Lodges staffed have their own hosts which serve as all meals. (482)
|?| Staffed lodges have their own hosts, that serve all meals. (483)
===

===
|?| Huts, staffed, have a crew who provides all the meals. (490)
|?| Staffed huts have a crew, that provides all the meals. (491)
|?| Staffed huts have a crew which provides all the meals. (492)
===

[...] Self-service lodges are fully equipped with bed linen and kitchen utensils, and have supplies for sale. Hikers do their own housekeeping. [...]

===
|?| Unstaffed lodges don't have food supplies, but they are equipped and open on the same terms as self service lodges. (500)
|?| Unstaffed lodges do not have food supplies but they are equipped and open on the same terms as self service lodges. (501)
|?| Unstaffed lodges do not have food supplies, but they are equipped and open on the same terms as self-service lodges. (502)
===

[...]

===
|?| At permitting time, it is well worth a detour, to see Blåbreen. (510)
|?| Permitting time, it is well worth a detour to see Blåbreen. (511)
|?| Time permitting, it is well worth a detour to see Blåbreen. (512)
===

===
|?| The route crosses Glopåa on a bridge and meets the route from Gjendesheim before the trail goes steep downhill to Memurubu. (520)
|?| The route crosses Glopåa on a bridge and meets the route from Gjendesheim before the trail goes steep to Memurubu downhill. (521)
===

===
|?| Before the trail joins the Gjendesheim trail for a steep descent to Memurubu Glopåa is crossed on a bridge. (530)
|?| Glopåa is crossed on a bridge before the trail joins the Gjendesheim trail for a steep descent to Memurubu. (531)
===

[...] East of Gjendesheim, the bluish Gausdal Vestfjell mountains extend out on a blue horizon.

===
|?| In recent years, increasingly more people have become aware of this outstanding mountain area, that stretches all the way to Gudbrandsdalen. (540)
|?| In recent years, increasingly more people have become aware of this outstanding mountain area, who stretches all the way to Gudbrandsdalen. (541)
|?| In recent years, increasingly more people have become aware of this outstanding mountain area which stretches all the way to Gudbrandsdalen. (542)
===

[...] In both summer and winter there are marked routes from Gjendesheim in to Gausdal Vestfjell.

===
|?| On an easy days trip, one may also sample other attractions. Sikkilsdalshø, Gåpåpiggan, and Brurskarknappen are excellent places to visit. (550)
|?| On an easy day trip, one may also sample other attractions; Sikkilsdalshø, Gåpåpiggan and Brurskarknappen are excellent places to visit. (551)
|?| On an easy day trip, one may also sample other attractions. Sikkilsdalshø, Gåpåpiggan, and Brurskarknappen are excellent places to visit. (552)
===

The hike over to Sikkilsdalsseter is a splendid introduction to the area.

===
|?| Before you begin walking itself in hiking terrain from Gjendesheim, you must follow the road to Maurvangen. (560)
|?| From Gjendesheim, you must follow the road to Maurvangen before you begin walking in hiking terrain itself. (561)
|?| From Gjendesheim, you must follow the road to Maurvangen before you begin to be walking in hiking terrain itself. (562)
===

===
|?| When the hike down into Sjodalen through Sikkilsdalsskaret was one of the most popular routes in Jotunheimen the road wasn't there in the early days of tourism (570)
|?| The road was not there in the early days of tourism when the hike through Sikkilsdalsskaret into Sjodalen down was one of the most popular routes in Jotunheimen (571)
|?| The road was not there in the early days of tourism, when the hike through Sikkilsdalsskaret down into Sjodalen was one of the most popular routes in Jotunheimen. (572)
|?| When the hike into Sjodalen through Sikkilsdalsskaret down was one of the most popular routes in Jotunheimen, the road wasn't there in the early days of tourism (573)
===

From Maurvangen, it is an easy and pleasant trip up to the gorge, and merely the view from there and back toward Jotunheimen is worth the trip.

===
|?| The view the other way is not any less impressive. (580)
|?| The view the other way isn't any less impressive. (581)
===

Sikkilsdalen is one of Norway's lushest mountain valleys and is much used for grazing.

===
|?| From Sikkilsdalsskaret, there are two trails marked. (590)
|?| From Sikkilsdalsskaret, there are two marked trails. (591)
|?| From Sikkilsdalsskaret, there are 2 marked trails. (592)
===

===
|?| From Sikkilsdalsskaret, there are two marked routes onwards. (600)
|?| From Sikkilsdalsskaret, there are two routes marked onwards. (601)
===

===
|?| If the weather's good, you surely should choose the one over Sikkilsdalshø. (610)
|?| If the weather is good, you surely should choose the one over Sikkilsdalshø. (611)
|?| You should choose the one over Sikkilsdalshø surely if the weather is good. (612)
===

The marked ridge between Sikkilsdalshø and Sikkilsdalshornet is particularly impressive.

===
|?| The trail from the latter peak drops down along the west side of Skålbekken steeply to the hikers ' hut. (620)
|?| The trail from the latter peak drops steeply down along the west side of Skålbekken to the hikers ' hut. (621)
|?| The trail from the latter peak drops steeply to the hikers ' hut down along the west side of Skålbekken. (622)
===

It passes a series of lateral moraines that were deposited toward the end of the last ice age some 10,000 - 9,000 years ago.

===
|?| The alternative to the hike over the summit is a slightly shorter hike down in the valley along the 2 lakes. (630)
|?| The alternative to the hike over the summit is a slightly shorter hike along the two lakes down in the valley. (631)
|?| The alternative to the hike over the summit is a slightly shorter hike down in the valley along the two lakes. (632)
===

The route goes partly in rough talus. Just before the tourist lodge, it passes by Prinsehytta, the Royal Family's Easter residence for many years.

===
|?| On the way, you may also meet some of the horse herds the valley is known for. (640)
|?| On the way, you may also meet some of the horse herds, where the valley is known for. (641)
|?| On the way, you may also meet some of the horse herds why the valley is known. (642)
|?| On the way you may also meet some of the horse herds for which the valley is known. (643)
===

===
|?| Along the way, you may also encounter some of the horse herds the valley is famous for. (650)
|?| Along the way, you may also encounter some of the horse herds, who the valley is famous for. (651)
|?| Along the way, you may also encounter some of the horse herds, famous for which the valley is. (652)
===

Ever since 1868 Sikkilsdalsseter has been the state center for horse breeding.

===
|?| It is here that the government's selected stud horses have spent many a pleasant summer week in the company of their harem of mares. (660)
|?| It is here that the government's selected stud horses have spent many a pleasant summers week in the company of their harem of mares. (661)
|?| It is here where that the government's stud horses selected have spent many a pleasant summer week in the company of their harem of mares. (662)
===

===
|?| Here, the state's selected stallions spend enjoyable summer weeks with their harems of mares. (670)
|?| Here, the state's stallions selected spend enjoyable summer weeks with their harems of mares. (671)
===

===
|?| Much of the state's choice, breeding horses, has spent summers happy weeks here with their harem of females. (680)
|?| Many of the state's choice breeding horses have spent happy summers weeks here with their harem of females. (681)
|?| Many of the state's choice breeding horses have spent happy summer weeks here with their harem of females. (682)
===

===
|?| The horses are normally released at summer solstice. (690)
|?| The horses are normally released at summers solstice. (691)
|?| The horses are released at summers solstice normally. (692)
===

[...] Transportation Bus service to Gjendesheim, Maurvangen and Bessheim.
===
|?| Auto road to Sikkilsdalsseter. (700)
|?| Automobile road to Sikkilsdalsseter. (701)
===

[...] Steindalen is also an excellent starting point for the trip up Munken ridge and on through Kalvåhøgda. On the other side of the valley lie both Høgebrotet and Tjørnholstind, conveniently located for day hikes; and Vestre Leirungstind far up in the valley is also a peak that does not call for real climbing.
===
|?| There is so much here, that one can simply pick and choose. (710)
|?| There are so many here one can simply pick and choose. (711)
|?| There is so much one can simply pick and choose here. (712)
|?| There is so much, which one can pick and choose simply, here. (713)
===

[...] Transportation

===
|?| Automobile road and bus routes to Gjendesheim, and across Valdresflya. (720)
|?| Auto road and bus routes to Gjendesheim and across Valdresflya. (721)
===

[...] Gjendesheim is not only the gateway to Jotunheimen. From there it's also a good idea to start hikes in to Gausdal Vestfjell.

===
|?| The route over to Haugseter has been marked very recently, goes through impressive natural surroundings, and is a spectacular start in this area of the mountains. (730)
|?| The route over to Haugseter has very recently been marked, goes through impressive natural surroundings, and is a spectacular start in this area of the mountains. (731)
===

===
|?| The route to Haugseter is marked recently, runs through a wonderful landscape, and is a fine entry into these mountains. (740)
|?| The route to Haugseter is recently marked, runs through a wonderful landscape, and is a fine entry into these mountains. (741)
===

[...] You can ask the locals at Gjendesheim for transportation across the river.

===
|?| If you come from the other direction, there is a signal horn for signalling that you need transportation over the river, mounted on the river bank. (750)
|?| If you come from the other direction, there is a signal horn mounted on the river bank for signaling that you need transportation over the river. (751)
|?| If you come from the other direction there is a signal horn mounted on the river bank for signalling that you need transportation over the river. (752)
===

===
|?| After you pass the beautiful lower Leirungen lake, you will cross the road over Valdresflya. (760)
|?| You will cross the road over Valdresflya after you pass the lower Leirungen beautiful lake. (761)
|?| You will cross the road over Valdresflya after you pass the beautiful Lower Leirungen Lake. (762)
===

===
|?| The road over Valdresflya intersects just past the Nedre Leirungen beautiful lake. (770)
|?| The road over Valdresflya intersects just past the beautiful Nedre Leirungen lake. (771)
===

===

|?| After that, the trail runs up the hillside and past northern and southern Brurskarknappen, past Brurskartjørni, and down to the upper Heimdalsvatnet along Sandbakkbekken evenly. (780)

|?| After that, the trail runs evenly up the hillside and past northern and southern Brurskarknappen, past Brurskartjørni, and down to the upper Heimdalsvatnet along Sandbakkbekken. (781)

|?| After that, the trail even runs up the hillside, and past Brurskarknappen northern, and southern, past Brurskartjørni, and down to the upper Heimdalsvatnet along Sandbakkbekken. (782)

===

===

|?| Thereafter, the hike goes evenly up the hillside and past Nordre and Søndre Brurskarknappen, past Brurskartjørni, and down to Øvre Heimdalsvatnet along Sandbakkbekken. (790)

|?| The hike thereafter evenly goes up the hillside and past Nordre and Søndre Brurskarknappen, past Brurskartjørni, and down along Sandbakkbekken to Øvre Heimdalsvatnet. (791)

|?| The hike thereafter even goes up the hillside and past Nordre and Søndre Brurskarknappen, past Brurskartjørni, and down along Sandbakkbekken to Øvre Heimdalsvatnet. (792)

|?| Thereafter, the hike even goes up the hillside and past Nordre and Søndre Brurskarknappen, past Brurskartjørni, and down to Øvre Heimdalsvatnet along Sandbakkbekken. (793)

===

===

|?| The trail then climbs evenly past northern and southern Brurskarknappen, past Brurskartjørni, then down along Sandbakkbekken to Øvre Heimdalsvatnet. (800)

|?| The trail then climbs past Brurskarknappen northern and southern, past Brurskartjørni, then down along Sandbakkbekken to Øvre Heimdalsvatnet evenly. (801)

|?| The trail even then climbs past northern and southern Brurskarknappen, past Brurskartjørni, then down along Sandbakkbekken to Øvre Heimdalsvatnet. (802)

|?| The trail then climbs evenly past northern and southern Brurskarknappen, past Brurskartjørni, then down to Øvre Heimdalsvatnet along Sandbakkbekken. (803)

===

[...] Øvre Heimdalsvatnet is a research lake used by the university to study fish in an alpine lake.

===

|?| The route continues along the shore of the lake to the outlet at the east end it is usually easy to wade. (810)

|?| The route continues along the shore of the lake to the outlet at the east end which usually is easy to wade. (811)

|?| The route continues along the shore of the lake to the outlet at the east end that is usually easy to wade. (812)

===

Thereafter, it goes uphill again, over the east end of Valdresflya, and in between the prominent twin peaks, Østre and Vestre Gluptinden.

===
|?| From here, it's downhill all the way along Urekåni and down to Jotunheim road, where Haugseter is located on the other side of the road. (820)
|?| From here, it is downhill all the way along Urekåni, and down on the other side of the road to Jotunheim Road where Haugseter is located. (821)
|?| From here, it is downhill all the way along Urekåni and down to Jotunheim Road where Haugseter is located on the other side of the road. (822)
===

===
|?| From there, it is just downhill along Urekåni, and down where Haugseter is on the other side of the road to the Jotunheimveien. (830)
|?| From there, it's just downhill along Urekåni and down to the Jotunheimveien, where Haugseter is on the other side of the road. (831)
|?| From there, it is just downhill along Urekåni and down to the Jotunheimveien where Haugseter is on the other side of the road. (832)
===

===
|?| From here, it is all downhill with Haugseter across the road along Urekåna to the Jotunheimen road. (840)
|?| From here, it is all downhill along Urekåna to the Jotunheimen road with Haugseter across the road. (841)
===

===
|?| The limits of this booklet are such that the trip ends here, but this is only the start of hiking in Gausdal Vestfjell. (850)
|?| The limits of this booklet are such the trip ends here but this is only the start of hiking in Gausdal Vestfjell. (851)
===

[...] Transportation
===
|?| Automobile road to Leirvassbu and Fondsbu. (860)
|?| Auto road to Leirvassbu and Fondsbu. (861)
===

===
|?| Auto road and bus routes over Sognefjellet. (870)
|?| Automobile road and bus routes over Sognefjellet. (871)
===

[...] Hike 3d - 3 days - gg

===

|?| A round trip between farmsteads abandoned and mountain pastures in Utladalen's rich natural surroundings. (880)

|?| A round trip in Utladalen's rich natural surroundings between abandoned farmsteads and mountain pastures. (881)

|?| A round trip between abandoned farmsteads and mountain pastures in Utladalen's rich natural surroundings. (882)

===

===

|?| A round tour in Utladalens mighty scenery between abandoned farms and summer dairies. (890)

|?| A round tour between abandoned farms and summer dairies in Utladalens mighty scenery. (891)

|?| A round tour in Utladalens mighty scenery between farms abandoned and summer dairies. (892)

|?| A round tour in mighty Utladalens scenery between farms abandoned and summer dairies. (893)

===

1. Hjelle to Stølsmaradalen

===

|?| The trip starts with 5 kilometers along the public road to Vetti easily. (900)

|?| Easily, the trip starts with five kilometers along the public road to Vetti. (901)

|?| The trip starts easily with five kilometers along the public road to Vetti. (902)

===

===

|?| The hike starts easily with 5 kilometers to Vetti along Folkeveien. (910)

|?| The hike starts easily with five kilometers along Folkeveien to Vetti. (911)

|?| The hike starts with 5 kilometers along Folkeveien to Vetti easily. (912)

|?| The hike starts easily with five kilometers to Vetti along Folkeveien. (913)

===

===

|?| The hike gets an easy start along three miles of the road Folkeveien to Vetti. (920)

|?| The hike gets an easy start along 3 miles of the road Folkeveien to Vetti. (921)

===

[...]

===

|?| The trail continues further into the valley, and you should also allow yourself a side trip to the impressive Vettisfossen before you cross Utla on a bridge and set to the precipitous Brendeteigen. (930)

|?| The trail continues further into the valley and before you cross Utla on a bridge you should also allow yourself a side trip to the impressive Vettisfossen, and set to the precipitous Brendeteigen. (931)

===

===
|?| Fill your water bottle before the climb for you will need a few stops for air and water along the way. (940)
|?| Fill your water bottle before the climb for you will need a few stops along the way for air and water (941)
===

The consolution is that the elevation gain up the hillside is only about 500 meters, and that after a while you'll have a fantastic view toward Vettisfossen.

===
|?| Up at timberline, the trail meets the old cattle track from Avdalen and goes downhill slightly before you come to the idyllic old summers dairy, Stølsmaradalen. (950)
|?| Up at timberline, the trail meets the old cattle track from Avdalen, and goes downhill slightly before you come to the old idyllic summer dairy, Stølsmaradalen. (951)
|?| Up at timberline, the trail meets the old cattle track from Avdalen and goes slightly downhill before you come to the old idyllic summer dairy, Stølsmaradalen. (952)
|?| Up at timberline the trail meets the old cattle track from Avdalen, and goes slightly downhill before you come to the idyllic old summer dairy Stølsmaradalen. (953)
===

===
|?| At timberline, the trail meets the old tote road from Avdalen descending a bit as you reach the scenic summers old farm Stølsmaradalen (960)
|?| At timberline, the trail meets the old tote road from Avdalen descending a bit as you reach the scenic old summer farm, Stølsmaradalen. (961)
|?| At timberline, the trail meets the old tote road from Avdalen descending a bit as you reach the old scenic summers farm, Stølsmaradalen. (962)
===

[...] Hurrungane is in a Wilderness Area in Jotunheimen National Park with as few amenities as possible.

===
|?| DNT can provide more information about routes in this area and good guide books have been written for the area. (970)
|?| DNT can provide more information on routes in this area and good guide books have been written for the area. (971)
===

===
|?| But today's hike returns down along Utladalen gently hugging the hillside at timberline at the same elevation until it passes Fuglenosa. (980)
|?| But today's hike returns down along Utladalen gently hugging the hillside at the same elevation at timberline until it passes Fuglenosa. (981)
===

[...]

===

|?| Along the trail, there is also an unmarked path who leads up to the unstaffed Gravdalen lodge that is definitely worth a detour. (990)

|?| Along the trail, there is also an unmarked path, that leads up to the unstaffed Gravdalen lodge, that is definitely worth a detour. (991)

|?| Along the trail, there is also an unmarked path that leads up to the unstaffed Gravdalen lodge which is definitely worth a detour. (992)

===

===

|?| On the way, an excellent trail branches off up to the no-service Gravdalen cabin, which is absolutely worth a side trip. (1000)

|?| On the way, an excellent trail branches off up to the Gravdalen no-service cabin, that is absolutely worth a side trip. (1001)

|?| On the way, an excellent trail branches off up to the no-service Gravdalen cabin that absolutely is worth a side trip. (1002)

|?| On the way an excellent trail branches off up to the Gravdalen no-service cabin, which absolutely is worth a side trip. (1003)

===

===

|?| So that it is now operated as a staffed lodge during the summer volunteer workers from Årdal have also been at work, renovating the abandoned mountain farm. (1010)

|?| Volunteer workers from Årdal have also been at work renovating the abandoned mountain farm so that it is now operated as a staffed lodge during the summer. (1011)

|?| Volunteer workers from Årdal have also been at work renovating the mountain farm abandoned so that it is now operated as a lodge staffed during the summer (1012)

===

===

|?| Also at Avdalen, eager volunteer Årdalers have been around and refurbished the mountain farm abandoned, so it now operates as a lodge staffed in summer. (1020)

|?| Also at Avdalen, eager volunteer Årdalers have been around and have refurbished the mountain farm abandoned, so it now operates as a staffed lodge in summer. (1021)

|?| Also at Avdalen, eager volunteer Årdalers have been around and have refurbished the abandoned mountain farm, so it now operates as a staffed lodge in summer. (1022)

===

===

|?| At Avdalen, so that it now is operated in the summer as a hut staffed, more eager volunteers from Årdal have renovated the deserted farm (1030)

|?| At Avdalen, more eager volunteers from Årdal have renovated the deserted farm so that it is now operated as a staffed hut in the summer. (1031)

|?| At Avdalen, so that it is now operated in the summer as a hut staffed more eager volunteers from Årdal have renovated the deserted farm (1032)

===

[...] Transportation

===
|?| Auto road to Hjelle. (1040)
|?| Automobile road to Hjelle. (1041)
===

[...] 3. Nørdstedalseter to Trulsbu Although it is about 10 kilometers on the service road during the first part of this trip, don't let this frighten you.
===
|?| The road is easy to walk and the surroundings are splendid. (1050)
|?| It is easy to walk the road and the surroundings are splendid. (1051)
|?| To walk the road is easy, and the surroundings are splendid. (1052)
===

===
|?| You go to the end of the road and there, cross a stone dump, and continue along the route marked on the west side of Middalsvatnet and past the ruins of Medalsbu, an old DNT lodge. (1060)
|?| You go to the end of the road and there, cross a stone dump, and continue along the route on the west side of Middalsvatnet, marked, and past the ruins of Medalsbu, an old DNT lodge. (1061)
|?| You go to the end of the road and there, cross a stone dump, and continue along the marked route on the west side of Middalsvatnet and past the ruins of Medalsbu, an old DNT lodge. (1062)
===

[...] There are marked routes from there, both to Nørdstedalseter and to Sota Sæter.
===
|?| But you keep going to Trulsbu down into Vesledalen. (1070)
|?| But you keep going down into Vesledalen to Trulsbu. (1071)
===

[...]
===
|?| From Nørdstedalseter, there are two marked routes north to Sota Sæter. (1080)
|?| From Nørdstedalseter, there are two routes marked north to Sota Sæter. (1081)
===

===
|?| From Nørdstedalseter, there are two marked routes northwards to Sota Sæter. (1090)
|?| From Nørdstedalseter, there are two routes marked northwards to Sota Sæter. (1091)
|?| From Nørdstedalseter, there are 2 routes marked northwards to Sota Sæter. (1092)
===

[...] The marked trail continues along the northern side of the glacier.
===
|?| The other route to Sota Sæter from Nørdstedalseter goes along the eastern side of Illvatnet and is one hour longer. (1100)
|?| The other route from Nørdstedalseter to Sota Sæter goes along the eastern side of Illvatnet and is one hour longer. (1101)
===

===
|?| The other route from Nørdstedalseter to Sota Sæter runs on the east side of Illvatnet, and is in an hour longer. (1110)
|?| The other route to Sota Sæter from Nørdstedalseter runs on the east side of Illvatnet and is an hour longer. (1111)
|?| The other route from Nørdstedalseter to Sota Sæter runs on the east side of Illvatnet and is an hour longer. (1112)
===

[...] From this route, old marking runs over to Lundadalsbandet and Trulsbu.
===
|?| From Nørdstedalseter there's also a marked route westwards to Arentzbu uppermost in Mørkrisdalen. (1120)
|?| From Nørdstedalseter, there is also a route marked westwards to Arentzbu uppermost in Mørkrisdalen. (1121)
|?| From Nørdstedalseter, there is also a marked route westwards to Arentzbu uppermost in Mørkrisdalen. (1122)
|?| From Nørdstedalseter, there also is a route marked westwards to Arentzbu uppermost in Mørkrisdalen. (1123)
===

[...] The route goes along the east shore of Lundadalsvatnet and further out the valley to Heimste Lundadalssætri.
===
|?| From here there is a six-kilometer road down to the valley and the municipal center of Bismo. (1130)
|?| From here, there is a six kilometer road down to the valley and the municipal center of Bismo. (1131)
|?| From here, there is a 6 kilometer road down to the valley and the municipal center of Bismo. (1132)
===

===
|?| From there, it is 6 kilometers down the valley to the village center at Bismo. (1140)
|?| From there, it is six kilometers down the valley to the village center at Bismo. (1141)
|?| From there, it's six kilometers down the valley to the village center at Bismo. (1142)
===

[...] Transportation
===
|?| Automobile road and bus routes to Turtagrø. (1150)
|?| Auto road and bus routes to Turtagrø. (1151)
===

Bibliography

Abdi, H. (2007). Binomial distribution: Binomial and sign tests. In N. Salkind (Ed.), *Encyclopedia of measurement and statistics.* CA: Sage.

Abney, S. (1996). Statistical methods and linguistics. In J. Klavans & P. Resnik (Eds.), *The Balancing Act: Combining Symbolic and Statistical Approaches to Language.* Cambridge, MA: MIT Press.

Abney, S. (1997). Stochastic attribute-value grammars. *Computational Linguistics*, *23*(4), 597 – 618.

Alshawi, H., Bangalore, S., & Douglas, S. (1998). Automatic acquisition of hierarchical transduction models for machine translation. In *Proceedings of the 17th International Conference on Computational Linguistics and the 36th Annual Meeting of the Association for Computational Linguistics* (pp. 41 – 47). Montreal, Canada.

Alshawi, H., & Crouch, R. S. (1992). Monotonic semantic interpretation. In *Proceedings of the 30th Meeting of the Association for Computational Linguistics* (pp. 32 – 39). Newark, Delaware.

Bangalore, S., Chen, J., & Rambow, O. (2001). Impact of quality and quantity of corpora on stochastic generation. In L. Lee & D. Harman (Eds.), *Proceedings of the 2001 conference on Empirical Methods in Natural Language Processing* (pp. 159 – 166). Pennsylvania.

Bangalore, S., & Joshi, A. K. (1999). Supertagging: an approach to almost parsing. *Computaional Linguistics*, *25*(2), 237 – 265.

Bangalore, S., & Rambow, O. (2000). Exploiting a probabilistic hierarchical model for generation. In *Proceedings of the 18th International Conference on Computational Linguistics.* Saarbrücken, Germany.

Bangalore, S., Rambow, O., & Whittaker, S. (2000). Evaluation metrics for generation. In *Proceedings of the 1st International Conference on Natural Language Generation.* Mitzpe Ramon, Israel.

Belz, A. (2005). Statistical generation: Three methods compared and evaluated. In *Proceedings of the 10th European Workshop on Natural Language Generation (ENLG'05).* Aberdeen, Scotland.

Benson, S. J., & Moré, J. J. (2001). *A limited memory variable metric method for bound constrained minimization* (Tech. Rep. No. Preprint ANL/MCS-P909-0901). Argonne National Laboratory.

Berger, A., Della Pietra, S., & Della Pietra, V. (1996). A Maximum Entropy approach to Natural Language Processing. *Computational Linguistics*, *22*(1), 39 – 71.

Berger, A., & Printz, H. (1998). A comparison of criteria for Maximum Entropy / Minimum Divergence feature selection. In *Proceedings of the 3rd conference on Empirical Methods in Natural Language Processing* (pp. 97 – 106). Granada, Spain.

Besag, J. (1975). Statistical analysis of non-lattice data. *The Statistician*, *24*(3), 179 – 195.

Billot, S., & Lang, B. (1989). The structure of shared parse forests in ambiguous parsing. In *Proceedings of the 27th Meeting of the Association for Computational Linguistics* (pp. 143 – 151). Vancouver, Canada.

Blutner, R. (2000). Some aspects of optimality in natural language interpretation. *Journal of Semantics*, *17*(3), 189 – 216.

Bond, F., Fujita, S., Hashimoto, C., Kasahara, K., Nariyama, S., Nichols, E., Ohtani, A., Tanaka, T., & Amano, S. (2004). The Hinoki treebank: A treebank for text understanding. In *Proceedings of the first international joint conference on natural language processing (IJCNLP-04)* (pp. 554 – 559). Hainan, China.

Boros, M., Eckert, W., Gallwitz, F., Görz, G., Hanrieder, G., & Niemann, H. (1996). Towards understanding spontaneous speech: Word accuracy vs. concept accuracy. In *Proceedings of the fourth international conference on spoken language processing (icslp 96).* Philadelphia.

Bos, J. (1995). Predicate logic unplugged. In *Proceedings of the tenth amsterdam colloquium* (pp. 133 – 142). Amsterdam, Holland.

Boser, B. E., Guyon, I., & Vapnik, V. (1992). A training algorithm for optimal margin classifiers. In *Proceedings of the 5th annual workshop on Computational Learning Theory (COLT'92)* (pp. 144 – 152). Pittsburgh, Pennsylvania.

Brants, S., Dipper, S., Hansen, S., Lezius, W., & Smith, G. (2002). The TIGER treebank. In *Proceedings of the 1st workshop on Treebanks and Linguistic Theories.* Sozopol, Bulgaria.

Brown, P. F., Cocke, J., Della Pietra, S., Della Pietra, V. J., Jelinek, F., Lafferty, J., Mercer, R. L., & Roossin, P. S. (1990). A statistical approach to machine translation. *Computational Linguistics*, *16*(2), 79 – 85.

Busemann, S. (1995). Towards classification of generation subtasks. In W. Hoeppner & H. Horacek (Eds.), *Principles of natural language generation. papers from a dagstuhl-seminar* (pp. 25 – 32). Duisburg, Germany.

Cahill, A., Burke, M., O'Donovan, R., Genabith, J. van, & Way, A. (2004). Long-distance dependency resolution in automatically acquired wide-coverage PCFG-based LFG approximations. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics* (pp. 320 – 327). Barcelona, Spain.

Cahill, A., Forst, M., & Rohrer, C. (2007). Stochastic realisation ranking for a free word order language. In *Proceedings of the 11th European Workshop on Natural Language Generation (ENLG'07)* (pp. 17 – 24). Schloss Dagstuhl, Germany.

Cahill, A., & Genabith, J. van. (2006). Robust PCFG-based generation using automatically acquired LFG approximations. In *Proceedings of the 2006 conference on Empirical Methods in Natural Language Processing* (pp. 1033 – 1040). Sydney, Australia: Association for Computational Linguistics.

Carroll, J., Copestake, A., Flickinger, D., & Poznanski, V. (1999). An efficient chart generator for (semi-)lexicalist grammars. In *Proceedings of the 7th European Workshop on Natural Language Generation (ENLG'99).* Toulouse.

Carroll, J., & Oepen, S. (2005). High efficiency realization for a wide-coverage unification grammar. In R. Dale, K.-F. Wong, J. Su, & O. Y. Kwong (Eds.), *Proceedings of the 2nd International Joint Conference on Natural Language Processing* (pp. 165 – 176). Jeju, Republic of Korea.

Carter, D. (1997). The TreeBanker. A tool for supervised training of parsed corpora. In *Proceedings of the workshop on computational environments for grammar development and linguistic engineering.* Madrid, Spain.

Chen, S., & Rosenfeld, R. (2000). A survey of smoothing techniques for me models. *IEEE Trans. Speech and Audio Processing*, *8*(1), 37 – 50.

Chen, S. F., & Goodman, J. T. (1998). *An empirical study of smoothing techniques for language modeling* (Tech. Rep. No. Technical Report TR-10-98). Cambridge, Massachusetts: Computer Science Group, Harvard University.

Chen, S. F., & Rosenfeld, R. (1999). *A Gaussian prior for smoothing maximum entropy models* (Tech. Rep. No. CMUCS-CS-99-108). Carnegie Mellon University.

Chomsky, N. (1965). *Aspects of the theory of syntax.* Cambridge: MIT Press.

Chomsky, N. (1969). Quines empirical assumptions. In D. Davidson & J. Hintikka (Eds.), *Words and objections. Essays on the work of W. V. Quine* (pp. 53 – 68). Dordrecht: Reidel.

Clarkson, P., & Rosenfeld, R. (1997). Statistical language modeling using the CMU-Cambridge Toolkit. In *Proceedings of ESCA Eurospeech.*

Collins, M. (1997). Three generative, lexicalised models for statistical parsing. In *Proceedings of the 35th Meeting of the Association for Computational Linguistics and the 8th Conference of the European Chapter of the ACL* (pp. 16 – 23). Madrid, Spain.

Collins, M., Koehn, P., & Kučerová, I. (2005). Clause restructuring for statistical machine translation. In *Proceedings of the 43rd Meeting of the Association for Computational Linguistics* (pp. 531 – 540). Ann Arbor, Michigan.

Copestake, A. (2002). *Implementing typed feature structure grammars.* Stanford, CA: CSLI Publications.

Copestake, A., Flickinger, D., Malouf, R., Riehemann, S., & Sag, I. (1995). Translation using minimal recursion semantics. In *Proceedings of the sixth international conference on theoretical and methodological issues in machine translation.* Leuven, Belgium.

Copestake, A., Flickinger, D., Pollard, C., & Sag, I. A. (2006). Minimal Recursion Semantics: an introduction. *Research on Language and Computation*, *3*(4), 281 – 332.

Corston-Oliver, S., Gamon, M., Ringger, E., & Moore, R. (2002). An overview of amalgam: A machine-learned generation module. In *Proceedings of the 2nd International Conference on Natural Language Generation* (pp. 33 – 40). New York, USA.

Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning*, *20*(3), 273 – 297.

Cover, T. M., & Thomas, J. A. (1991). *Elements of information theory.* New York: Wiley & Sons.

Cristianini, N., & Shawe-Taylor, J. (2000). *An introduction to support vector machines and other kernel-based learning methods.* Cambridge University Press.

Crysmann, B. (2005). Relative clause extraposition in German. An efficient and portable implementation. *Research on Language and Computation*, *3*(1), 61 – 82.

Dalianis, H. (1996). *Concise natural language generation from formal specifications.* Unpublished doctoral dissertation, Department of Computer and Systems Sciences, Stockholm University, Stockholm, Sweden.

Darroch, J., & Ratcliff, D. (1972). Generalised Iterative Scaling for log-linear models. *Ann. Math. Statistics*(43), 1470 – 1480.

Della Pietra, S., Della Pietra, V., & Lafferty, J. (1997). Inducing features of random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *19*(4), 380 – 393.

Dyvik, H. (1999). The universality of f-structure: Discovery or stipulation? the case of modals. In *Proceedings of the 4th international lexical functional grammar conference.* Manchester, UK.

Flickinger, D. (2002). On building a more efficient grammar by exploiting types. In S. Oepen, D. Flickinger, J. Tsujii, , & H. Uszkoreit (Eds.), *Collaborative language engineering: A case study in efficient grammar-based processing* (pp. 1 – 17). CSLI Press.

Flickinger, D., Lønning, J. T., Dyvik, H., Oepen, S., & Bond, F. (2005). SEM-I rational MT. Enriching deep grammars with a semantic interface for scalable machine translation. In *Proceedings of the 10th Machine Translation Summit* (pp. 165 – 172). Phuket, Thailand.

Forsbom, E. (2003). Training a super model look-alike: Featuring edit distance, n-gram occurrence, and one reference translation. In *Proceedings of the workshop on machine translation evaluation: Towards systemizing MT evaluation, held in conjunction with MT Summit IX.* New Orleans, USA.

Gamon, M., Ringger, E., & Corston-Oliver, S. (2002). *Amalgam: A machine-learned generation module* (Tech. Rep. No. MSR-TR-2002-57). Microsoft Research.

Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R., & Sunderam, V. (1995). *PVM: Parallel Virtual Machine: a users' guide and tutorial for networked parallel computing.* Cambridge, MA: MIT Press.

Grice, P. H. (1975). Logic and conversation. *Syntax and Semantics, 3*(Speech Acts), 43 – 58.

Habash, N. (2003). Matador: A large scale Spanish-English GHMT system. In *Proceedings of the 9nt machine translation summit* (pp. 149 – 156). New Orleans, USA.

Habash, N. (2004). The use of a structural n-gram language model in generation-heavyhybrid machine translation. In A. Belz, R. Evans, & P. Piwek (Eds.), *Proceedings of the 3rd International Conference on Natural Language Generation* (pp. 61 – 69). Brockenhurst, UK: Springer.

Hagen, K., Johannessen, J. B., & Nøklestad, A. (2000). A constraint-based tagger for norwegian. In *17th scandinavian conference of linguistics.* Funen, Denmark.

Henriksen, P., Haslerud, V. C., & Eek Øystein (Eds.). (2001). *Engelsk stor ordbok – engelsk-norsk / norsk-engelsk* (1 ed.). Oslo: Kunnskapsforlaget.

Herbrich, R., Graepel, T., & Obermayer, K. (2000). Large margin rank boundaries for ordinal regression. In A. Smola, P. Bartlett, B. Schoelkopf, & D. Schuurmans (Eds.), *Advances in Large Margin Classifiers.* MIT Press, Cambridge, MA.

Hovy, E. (1997). Language generation: Overview. In R. Cole, J. Mariani, H. Uszkoreit, A. Zaenen, & V. Zue (Eds.), *Survey of the state of the art in human language technology (web edition).* Cambridge University Press and Giardini.

Humphreys, K., Calcagno, M., & Weise, D. (2001). Reusing a statistical language model for generation. In *Proceedings of the 39th Meeting of the Association for Computational Linguistics.* Toulouse, France.

Jaynes, E. T. (1957). Information theory and statistical mechanics. *Physical Review, 106*(4), 620 – 630.

Jelinek, F. (1998). *Statistical methods for speech recognition.* Cambridge, MA, USA: MIT Press.

Joachims, T. (1999). Making large-scale SVM learning practical. In B. Schölkopf, C. Burges, & A. Smola (Eds.), *Advances in kernel methods - support vector learning.* MIT-Press.

Joachims, T. (2002). Optimizing search engines using clickthrough data. In *Proceedings of the ACM conference on knowledge discovery and data mining (KDD).* ACM.

Joachims, T. (2006). Training linear SVMs in linear time. In *Proceedings of the ACM conference on knowledge discovery and data mining (KDD).* ACM.

Johnson, M., Geman, S., Canon, S., Chi, Z., & Riezler, S. (1999). Estimators for stochastic unification-based grammars. In *Proceedings of the 37th Meeting of the Association for Computational Linguistics* (pp. 535 – 541). College Park, Maryland.

Johnson, M., & Riezler, S. (2000). Exploiting auxiliary distributions in stochastic unification-based grammars. In *Proceedings of the 1st Conference of the North American Chapter of the ACL.* Seattle, Washington.

Kaplan, R. M., & Bresnan, J. (1982). Lexical-Functional Grammar: A formal system for grammatical representation. In J. Bresnan (Ed.), *The mental representation of grammatical relations* (pp. 173 – 281). Cambridge, MA: MIT Press.

Kay, M. (1996). Chart generation. In A. Joshi & M. Palmer (Eds.), *Proceedings of the 34th Meeting of the Association for Computational Linguistics* (pp. 200 – 204). San Francisco: Morgan Kaufmann Publishers.

Knight, K., Chander, I., Haines, M., Hatzivassiloglou, V., Hovy, E., Iida, M., Luk, S. K., Whitney, R., & Yamada, K. (1995). Filling knowledge gaps in a broad-coverage MT system. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence.*

Knight, K., & Hatzivassiloglou, V. (1995). Two-level, many paths generation. In *Proceedings of the 33rd Meeting of the Association for Computational Linguistics.* Cambridge, MA.

Knight, K., & Luk, S. (1994). Building a large knowledge base for machine translation. In *Proceedings of the american association of artificial intelligence conference (AAAI-94).* Seattle, WA.

Koehn, P. (2004). Pharaoh: a beam search decoder for phrase-based statistical machine translation models. In *The 6th conference of the Association for Machine Translation in the Americas* (pp. 115 – 124). Washington DC.

Langkilde, I. (2000). Forest-based statistical sentence generation. In *Proceedings of the 1st Conference of the North American Chapter of the ACL.* Seattle, Washington.

Langkilde, I. (2002). An empirical verification of coverage and correctness for a general-purpose sentence generator. In *Proceedings of the 2nd International Conference on Natural Language Generation.* New York, USA.

Langkilde, I., & Knight, K. (1998a). Generation that exploits corpus-based statistical knowledge. In *Proceedings of the 17th International Conference on Computational Linguistics and the 36th Annual Meeting of the Association for Computational Linguistics.* Montreal, Canada.

Langkilde, I., & Knight, K. (1998b). The practical value of n-grams in generation. In *Proceedings of the 9th International Workshop on Natural Language Generation* (pp. 248 – 255). Ontario, Canada.

Lønning, J. T., Oepen, S., Beermann, D., Hellan, L., Carroll, J., Dyvik, H., Flickinger, D., Johannessen, J. B., Meurer, P., Nordgård, T., Rosén, V., & Velldal, E. (2004). LOGON — A Norwegian MT effort. In *Proceedings of the Workshop in Recent Advances in Scandinavian Machine Translation.* Uppsala, Sweden.

Malouf, R. (2000). The order of prenominal adjectives in natural language generation. In *Proceedings of the 38th Meeting of the Association for Computational Linguistics* (pp. 85 – 92). Hong Kong.

Malouf, R. (2002). A comparison of algorithms for maximum entropy parameter estimation. In *Proceedings of the 6th Conference on Natural Language Learning* (pp. 49 – 55). Taipei, Taiwan.

Malouf, R., & van Noord, G. (2004). Wide coverage parsing with stochastic attribute value grammars. In *Proceedings of the IJCNLP workshop Beyond Shallow Analysis.* Hainan, China.

Mann, W. C., & Matthiessen, C. M. I. M. (1985). Nigel: A systemic grammar for text generation. In R. Benson & J. Greaves (Eds.), *Systemic perspectives on discourse: Selected papers papers from the 9th international systemics workshop.* London: Ablex.

Manning, C. D., & Schütze, H. (1999). *Foundation of statistical natural language processing.* The MIT Press.

Marcus, M. P., Santorini, B., & Marcinkiewicz, M. A. (1993). Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, *19*(2), 313 – 330.

Maxwell, J., & Kaplan, R. M. (1993). The interface between phrasal and functional constraints. *Computational Lingusitics*(19), 571 – 589.

van Noord, G., & Neumann, G. (1997). Syntactic generation. In R. Cole, J. Mariani, H. Uszkoreit, A. Zaenen, & V. Zue (Eds.), *Survey of the state of the art in human language technology (web edition)*. Cambridge University Press and Giardini.

Miller, G. A., Beckwith, R., Fellbaum, C., Gross, D., & Miller, K. J. (1990). Introduction to WordNet: an on-line lexical database. *International Journal of Lexicography*, *4*(3), 235 – 244.

Miyao, Y., Ninomiya, T., & Tsujii, J. (2004). Corpus-oriented grammar development for acquiring a Head-Driven Phrase Structure grammar from the Penn Treebank. In *Proceedings of the 1st International Joint Conference on Natural Language Processing*.

Miyao, Y., & Tsujii, J. (2002). Maximum entropy estimation for feature forests. In *Proceedings of the Human Language Technology Conference*. San Diego, CA.

Müller, S., & Kasper, W. (2000). HPSG analysis of German. In W. Wahlster (Ed.), *Verbmobil: Foundations of speech-to-speech translation* (pp. 238 – 253). Berlin: Springer.

Nakanishi, H., Miyao, Y., & Tsujii, J. (2005). Probabilistic models for disambiguation of an HPSG-based chart generator. In *Proceedings of the 9th International Workshop on Parsing Technologies* (pp. 93 – 102). Vancouver, Canada: Association for Computational Linguistics.

Nakatsu, C., & White, M. (2006). Learning to say it well: Reranking realizations by predicted synthesis quality. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics* (pp. 1113 – 1120). Sydney, Australia: Association for Computational Linguistics.

Nygaard, L., Lønning, J. T., Nordgård, T., & Oepen, S. (2006). Using a bi-lingual dictionary in lexical transfer. In *Proceedings of the 11th conference of the European Association for Machine Translation*. Oslo, Norway.

Och, F. J. (2003). Minimum error rate training in statistical machine translation. In *Proceedings of the 41st Meeting of the Association for Computational Linguistics* (pp. 160 – 167). Sapporo, Japan.

Och, F. J., Gildea, D., Khudanpur, S., Sarkar, A., Yamada, K., Fraser, A., Kumar, S., Shen, L., Smith, D., Eng, K., Jain, V., Jin, Z., & Radev, D. (2004). A smorgasbord of features for statistical machine translation. In *Proceedings of the 5th Conference of the North American Chapter of the ACL* (pp. 161 – 168). Boston.

Och, F. J., & Ney, H. (2002). Discriminative training and maximum entropy models for statistical machine translation. In *Proceedings of the 40th Meeting of the Association for Computational Linguistics* (pp. 295 – 302). Philadelphia.

Oepen, S., & Callmeier, U. (2004). Measure for measure. Towards increased component comparability and exchange. In H. Bunt, J. Carroll, & G. Satta (Eds.), *New developments in parsing technology.* Dordrecht, The Netherlands: Kluwer Academic Publishers.

Oepen, S., & Carroll, J. (2000). Ambiguity packing in constraint-based parsing — practical results. In *Proceedings of the 1st Conference of the North American Chapter of the ACL* (pp. 162 – 169). Seattle, WA.

Oepen, S., Dyvik, H., Lønning, J. T., Velldal, E., Beermann, D., Carroll, J., Flickinger, D., Hellan, L., Johannessen, J. B., Meurer, P., Nordgård, T., & Rosén, V. (2004). Som å kapp-ete med trollet? Towards MRS-based Norwegian – English Machine Translation. In *Proceedings of the 10th International Conference on Theoretical and Methodological Issues in Machine Translation.* Baltimore, MD.

Oepen, S., Flickinger, D., Toutanova, K., & Manning, C. D. (2004). LinGO Redwoods. A rich and dynamic treebank for HPSG. *Journal of Research on Language and Computation*, *2*(4), 575 – 596.

Oepen, S., & Flickinger, D. P. (1998). Towards systematic grammar profiling. Test suite technology ten years after. *Computer Speech and Language*, *12*(4), 411 – 435.

Oepen, S., & Lønning, J. T. (2006). Discriminant-based MRS banking. In *Proceedings of the 4th International Conference on Language Resources and Evaluation.* Genoa, Italy.

Oepen, S., Netter, K., & Klein, J. (1997). TSNLP — Test Suites for Natural Language Processing. In J. Nerbonne (Ed.), *Linguistic databases* (pp. 13 – 36). Stanford, CA: CSLI Publications.

Oepen, S., Toutanova, K., Shieber, S., Manning, C., Flickinger, D., & Brants, T. (2002). The LinGO Redwoods treebank. Motivation and preliminary applications. In *Proceedings of the 19th International Conference on Computational Linguistics.* Taipei, Taiwan.

Oepen, S., Velldal, E., Lønning, J. T., Meurer, P., Rosén, V., & Flickinger, D. (2007). Towards hybrid quality-oriented machine translation — on linguistics and probabilities in mt. In *Proceedings of the 11th International Conference on Theoretical and Methodological Issues in Machine Translation* (pp. 144 – 153). Skövde, Sweden.

Oksefjell, S. (1999). A description of the English-Norwegian Parallel Corpus: Compilation and further developments. *International Journal of Corpus Linguistics, 4*(2), 197 – 219.

Osborne, M. (2000). Estimation of stochastic attribute-value grammars using an informative sample. In *Proceedings of the 18th International Conference on Computational Linguistics.* Saarbrücken, Germany.

Papineni, K., Roukos, S., Ward, T., & Zhu, W.-J. (2002). Bleu. A method for automatic evaluation of Machine Translation. In *Proceedings of the 40th Meeting of the Association for Computational Linguistics* (pp. 311 – 318). Philadelphia.

Pollard, C., & Sag, I. A. (1987). *Information-based syntax and semantics. Volume 1: Fundamentals.* Chicago, IL and Stanford, CA: Center for the Study of Language and Information. (Distributed by The University of Chicago Press)

Pollard, C., & Sag, I. A. (1994). *Head-driven phrase structure grammar.* Chicago, Illinois: The University of Chicago Press and CSLI Publications.

Prince, A., & Smolensky, P. (1993). *Optimality Theory: Constraint interaction in generative grammar* (Tech. Rep.). Rutgers University Center for Cognitive Science.

Radlinski, F., & Joachims, T. (2005). Query chains: Learning to rank from implicit feedback. In *Proceedings of the ACM conference on knowledge discovery and data mining (KDD).* ACM.

Ratnaparkhi, A. (2000). Trainable methods for surface natural language generation. In *Proceedings of the 1st Conference of the North American Chapter of the ACL* (pp. 194 – 201). Seattle, Washington.

Reiter, E., & Dale, R. (2000). *Building natural language generation systems*. Cambridge University Press.

Reyle, U. (1993). Dealing with ambiguities by underspecification: Construction, representation and deduction. *Journal of Semantics*, *10*(2), 123 – 179.

Riezler, S., King, T. H., Kaplan, R. M., Crouch, R., Maxwell, J. T., & Johnson, M. (2002). Parsing the wall street journal using a lexical-functional grammar and discriminative estimation techniques. In *Proceedings of the 40th Meeting of the Association for Computational Linguistics.* Philadelphia.

Riezler, S., & Vasserman, A. (2004). Incremental feature selection and l1 regularization for relaxed maximum-entropy modeling. In *Proceedings of the 2004 conference on Empirical Methods in Natural Language Processing.* Barcelona, Spain.

Rosenfeld, R. (1995). The CMU statistical language modeling toolkit, and its use in the 1994 ARPA CSR evaluation. In *Proceedings of ARPA Spoken Language Technology Workshop.* Austin, TX.

Rosén, V., Smedt, K. de, Dyvik, H., & Meurer, P. (2005). TREPIL: Developing methods and tools for multilevel treebank construction. In *Proceedings of the 4th workshop on Treebanks and Linguistic Theories* (pp. 161 – 172). Barcelona, Spain.

Rosén, V., Smedt, K. D., & Meurer, P. (2006). Towards a toolkit linking treebanking and grammar development. In *Proceedings of the 5th workshop on Treebanks and Linguistic Theories* (pp. 55 – 66). Prague, Czech Republic.

Shannon, C. E. (1948). A mathematical theory of communication. *The Bell System Technical Journal*, *27*, 379 – 423 (pt.1) and 623 – 656 (pt.2).

Shaw, J., & Hatzivassiloglou, V. (1999). Ordering among premodifiers. In *Proceedings of the 37th Meeting of the Association for Computational Linguistics* (pp. 135 – 143). College Park, Maryland.

Shen, L., Sarkar, A., & Och, F. (2004). Discriminative reranking for machine translation. In *Proceedings of the Human Language Technology Conference and the 5th meeting of the North American Association for Computational Linguistics HLT-NAACL 2004.* Boston, USA.

Shieber, S. M. (1988). A uniform architecture for parsing and generation. In *Proceedings of the 12th International Conference on Computational Linguistics* (Vol. 2, pp. 614 – 619). Budapest, Hungry.

Siegel, M., & Bender, E. M. (2002). Efficient deep processing of Japanese. In *Proceedings of the 3rd workshop on Asian language resources and international standardization (at COLING-2002).* Taipei.

Siegel, S., & Castellan, N. J. (1988). *Non-parametric statistics for the behavioural sciences* (2 ed.). New York: McGraw-Hill.

Stolcke, A. (2002). SRILM – an extensible language modeling toolkit. In *Proceedings of the international conference on spoken language processing* (Vol. 2, pp. 901 – 904). Denver.

The Penman Project. (1988). *The penman primer.* (Unpublished documentation)

Toutanova, K., Manning, C. D., Flickinger, D., & Oepen, S. (2005). Stochastic HPSG parse disambiguation using the Redwoods corpus. *Research on Language and Computation, 3*(1), 83 – 105.

Toutanova, K., Manning, C. D., Shieber, S. M., Flickinger, D., & Oepen, S. (2002). Parse disambiguation for a rich hpsg grammar. In *First workshop on treebanks and linguistic theories* (pp. 253 – 263). Sozopol, Bulgaria.

Toutanova, K., Markova, P., & Manning, C. D. (2004). The leaf projection path view of parse trees: Exploring string kernels for HPSG parse selection. In *Proceedings of the 2004 conference on Empirical Methods in Natural Language Processing* (pp. 166 – 173). Barcelona.

Vapnik, V. (1998). *Statistical learning theory.* New York: Wiley.

Varges, S., & Mellish, C. (2001). Instance-based natural language generation. In *Proceedings of the 2nd Conference of the North American Chapter of the ACL* (pp. 1 – 8). Pittsburgh, Pennsylvania.

Velldal, E., & Oepen, S. (2005). Maximum entropy models for realization ranking. In *Proceedings of the 10th Machine Translation Summit.* Phuket, Thailand.

Velldal, E., & Oepen, S. (2006). Statistical ranking in tactical generation. In *Proceedings of the 2006 conference on Empirical Methods in Natural Language Processing* (pp. 517 – 525). Sydney, Australia.

Velldal, E., Oepen, S., & Flickinger, D. (2004). Paraphrasing treebanks for stochastic realization ranking. In *Proceedings of the 3rd workshop on Treebanks and Linguistic Theories.* Tübingen, Germany.

Wahlster, W. (1997). *VerbMobil — Erkennung, Analyse, Transfer, Generierung und Synthese von Spontansprache* (VerbMobil Report # 198). Saarbrücken, Germany: Deutsches Forschungszentrum für Künstliche Intelligenz GmbH (DFKI).

Wasow, T., Jaeger, T. F., & Orr, D. (2005). Lexical variation in relativizer frequency. In *Proceedings of the workshop "Expecting the unexpected: Exceptions in grammar", at the 27th Annual Meeting of the German Linguistic Association.* Cologne.

White, M. (2004). Reining in CCG chart realization. In A. Belz, R. Evans, & P. Piwek (Eds.), *Proceedings of the 3rd International Conference on Natural Language Generation* (pp. 182 – 191). Brockenhurst, UK: Springer.

White, M. (2005). Designing an extensible API for integrating language modeling and realization. In *Proceedings of the Workshop on Software at the 43rd annual meeting of the ACL.* Ann Arbor, USA.

Wilcoxon, F. (1945). Individual comparisons by ranking methods. *Biometrics Bulletin, 1,* 80 – 83.

Witten, I. H., & Bell, T. C. (1991). The zero-frequency problem: Estimating the probabilities of novel events in adaptive text compression. *IEEE Transactions on Information Theory, 37*(4), 1085 – 1094.

XTAG Research Group. (2001). *A lexicalized tree adjoining grammar for english* (Tech. Rep. No. IRCS-01-03). IRCS, University of Pennsylvania.

Zeevat, H. (2000). The asymmetry of optimality theoretic syntax and semantics. *Journal of Semantics, 17*(3), 243 – 262.

Zipf, G. K. (1935). *The psycho-biology of language.* Boston: Houghton Mifflin.