CONSTITUENT PARSING BY CLASSIFICATION

BY

JOSEPH TURIAN

A DISSERTATION SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
COMPUTER SCIENCE DEPARTMENT
COURANT INSTITUTE OF MATHEMATICAL SCIENCES
NEW YORK UNIVERSITY
SEPTEMBER 2007

I. DAN MELAMED

Pur un kaminiku pasi,
Kun un viezijiku 'skontri.
Ali vedri vistia,
Tres yavizikas tenia en su mano,
Una d'avrir, una di sirar, una di kitar todu il mal.
Ben pura Yusef, ben pura adalai.

As I walked along a narrow street,
I met an old man.
In resplendent green, he was dressed,
Three keys he had in his hand,
One to open, one to close, and one to remove all harm.
May the sacrifice of Joseph, Lord, be accepted instead of mine.

Prekante: A Ritual Prayer for Curing
Chapter 8 of *Ritual Medical Lore of Sephardic Women*

Acknowledgments

1. Thank you to I. Dan Melamed for teaching me to be a scientist.

2. Thank you to Michael Collins, Ralph Grishman, Mehryar Mohri, and Satoshi Sekine (my committee) for helping me make this dissertation stronger.

3. Thank you to Dan Bikel, Léon Bottou, Patrick Haffner, Yann LeCun, Adam Meyers, Chris Pike, Cynthia Rudin, Wei Wang, the anonymous reviewers of earlier versions of this work, and anyone whom I have forgotten for your helpful comments and constructive criticism.

4. Thank you Dave Ferguson, Maxim Likhachev, and Wheeler Ruml for providing pointers into the search literature.

5. Thank you to the United States National Science Foundation for sponsoring this research with NSF grants #0238406 and #0415933.

6. Thank you to Donald Knuth for TEX, Lesley Lamport for LATEX, and countless others who built the system used to typeset this dissertation.

7. Thank you to Ben Wellington for writing your thesis at the same as me, for griping with me, and for helping me stay sane.

8. Thank you to Jared Weinstein for never being too busy to answer my math questions.

9. Thank you to my parents for everything.

10. And thank you to Tiana. I love you.

Abstract

We present an approach to constituent parsing, which is driven by classifiers induced to minimize a single regularized objective. It is the first discriminatively-trained constituent parser to surpass the Collins (2003) parser without using a generative model. Our primary contribution is simplifying the human effort required for feature engineering. Our model can incorporate arbitrary features of the input and parse state. Feature selection and feature construction occur automatically, as part of learning. We define a set of fine-grained atomic features, and let the learner induce informative compound features. Our learning approach includes several novel approximations and optimizations which improve the efficiency of discriminative training. We introduce greedy completion, a new agenda-driven search strategy designed to find low-cost solutions given a limit on search effort. The inference evaluation function was learned accurately enough to guide the deterministic parsers to the optimal parse reasonably quickly without pruning, and thus without search errors. Experiments demonstrate the flexibility of our approach, which has also been applied to machine translation (Wellington et al., 2006; Turian et al., 2007).

Table of Contents

List of Figures

List of Tables

List of Listings

❧ First Chapter ❧

Introduction

> Oh, get ahold of yourself.
> Nobody's proposing that we parse English.
>
> Larry Wall

Natural language parsing is the task of transforming a sequence of tokens, which are typically words, into a structure that represents syntactic information, e.g. a parse tree or dependency graph. There are different kinds of parsing, including shallow parsing (a.k.a. chunking), deep parsing, dependency parsing, constituent parsing, semantic parsing, and discourse parsing. Parsing is a **structure prediction** task, a task in which the output space comprises structures. Examples of structure spaces include label sequences (e.g. for POS tagging), trees (e.g. for constituent parsing), and DAGs (e.g. for non-projective dependency parsing). What is structure, and how can one predict it? One clue comes from the etymology of the word: "structure" derives from the Latin *struere*, meaning "to build". A structure can be built by performing a sequence of **inference**s, each of which determines some part (**item**) in the eventual structure.[1] Inferences can take into account not just the input, but also previous decisions in the inference process (a.k.a. the parse **history**).

1.1 Importance of parsing

Parsing in natural language processing (NLP) is not an end-goal, but a means to an end. Lay-people rarely care about a parse as the final output of an NLP application. However, NLP practitioners may include parsing in the middle of a text processing pipeline. Inferred syntactic information can be leveraged to improve the accuracy of higher-level textual analysis. We provide some specific examples.

[1] According to the Oxford English Dictionary, the word "parse" is apparently from the Latin *pars*, meaning "part."

Figure 1.1: An example of a parse tree augmented for information extraction. Example from Miller et al. (2000).

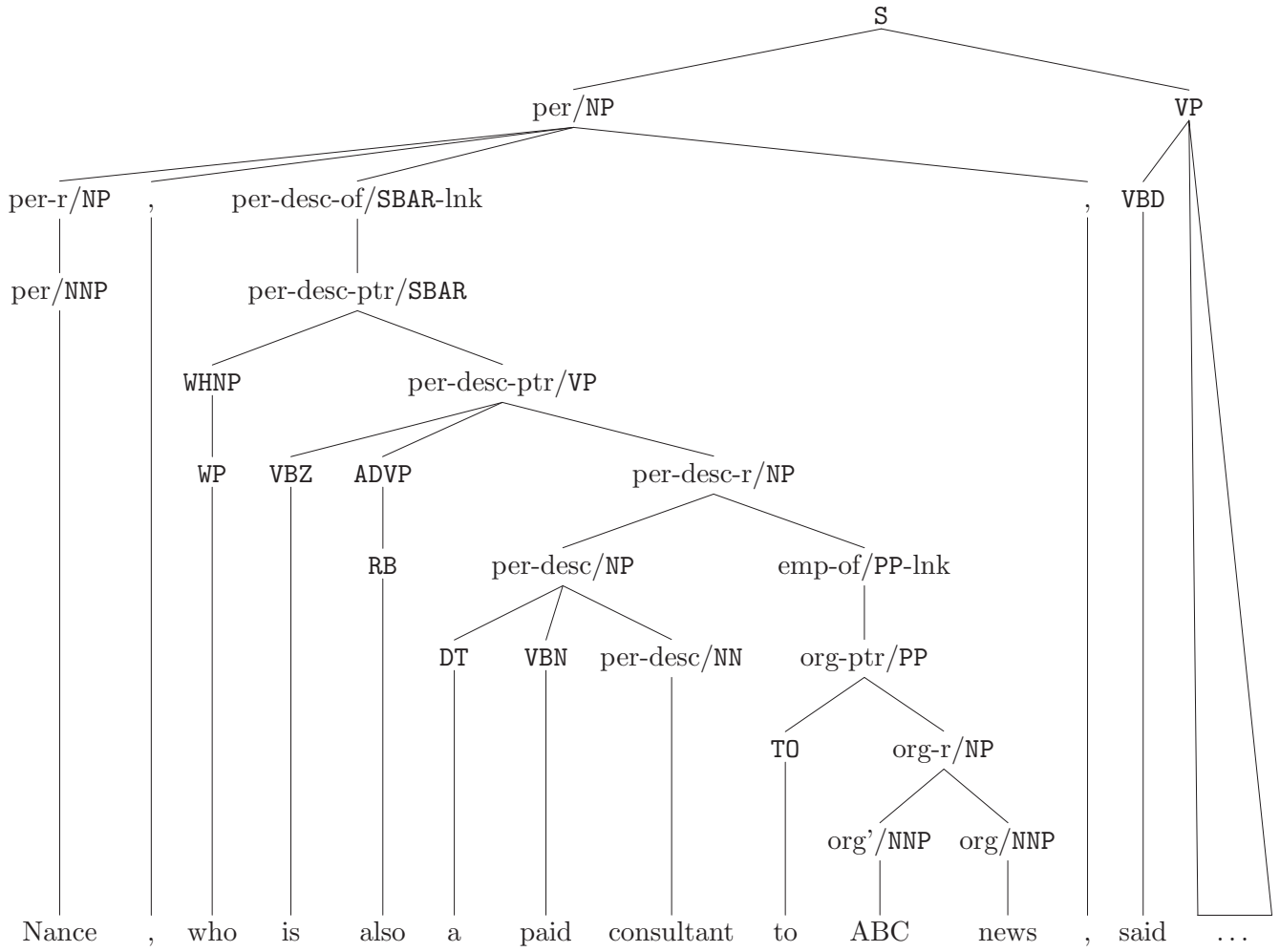
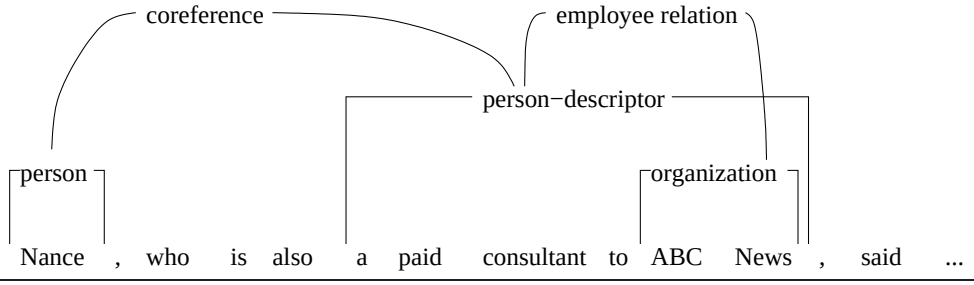


Figure 1.2: An example of a sentence semantically annotated for information extraction. Example from Miller et al. (2000).



Information extraction

Miller et al. (2000) present an information extraction approach which infers augmented parse trees that contain syntactic and semantic information. An example of an augmented parse tree is shown in Figure 1.1. This tree corresponds to the semantic annotation shown in Figure 1.2. In plain English (quoted from the third page of Miller et al. (2000)):

- "Nance" is the name of a person.
- "a paid consultant to ABC News" describes a person.
- "ABC News" is the name of an organization.
- The person described as "a paid consultant to ABC News" is employed by ABC News.
- The person named "Nance" and the person described as "a paid consultant to ABC News" are the same person.

Language modeling

Given some input Y, we wish to find the most likely output $\hat{W}$, i.e.:

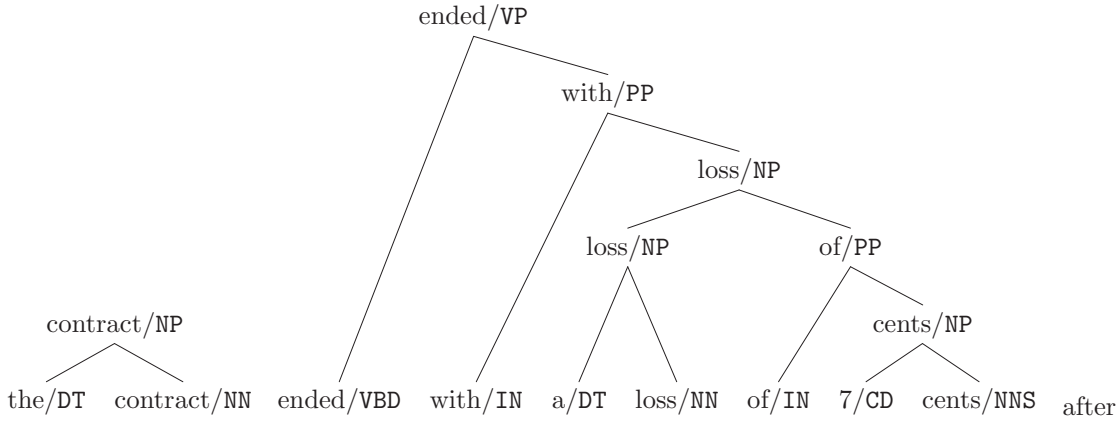
$$\hat{W} = \arg\max_W \Pr(W|Y) \tag{1.1}$$

By application of the Bayes rule:

$$= \arg\max_W \frac{\Pr(Y|W) \cdot \Pr(W)}{\Pr(Y)} \tag{1.2}$$

$$= \arg\max_W (\Pr(Y|W) \cdot \Pr(W)) \tag{1.3}$$

Figure 1.3: A left-to-right partial parse, to be used for language modeling in speech recognition. Constituents are annotated with their head-word. Example from Chelba and Jelinek (1998).



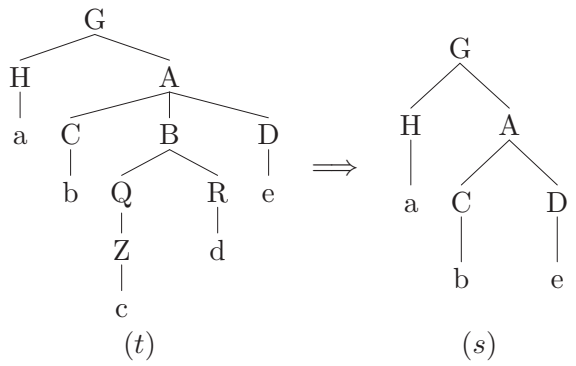
The last line follows because $\Pr(Y)$ is fixed. One intuition behind this formulation comes from communication theory, and is called the **noisy-channel** model (e.g. Brown et al., 1992; Daumé III and Marcu, 2002): We assume that the source model generates source W with likelihood $\Pr(W)$, which is then passed through a possibly noisy channel to generate Y with likelihood $\Pr(Y|W)$. We are given only the output Y of the channel, and must recover the most likely source W.

In many natural language tasks, given some input Y, the target output is a sequence of words W (e.g. a sentence). In this case, the source model is a **language model**, which assigns a probability to a sequence of words. Historically, language models have been based solely on n-grams (Brown et al., 1992). However, n-gram language models cannot capture long-distance dependencies, which motivates the use of syntactic structure in language modeling. For different tasks requiring language modeling, we present examples of approaches that incorporate syntax.

Language modeling for speech recognition

Chelba and Jelinek (1998) present a syntax-based language model that operates left-to-right, allowing it to be integrated into the decoding of word lattices for speech recognition. The authors present the example sentence: "the contract ended with a loss of 7 cents after trading as low as 89 cents" and consider the likelihood of recognizing the word "after." A trigram language model would be limited to predicting "after" given "7 cents." However, much more informative is the word "ended." Figure 1.3 shows the left-to-right partial parse of the sentence up to "after." This partial parse indicates that head-word "ended" is used

Figure 1.4: An example input tree t and output summary tree s. Example from Knight and Marcu (2000, 2002).



in predicting "after." The syntactic structure is able to capture long-range dependencies between words, and ignore interceding words that are less relevant.

Language modeling for summarization

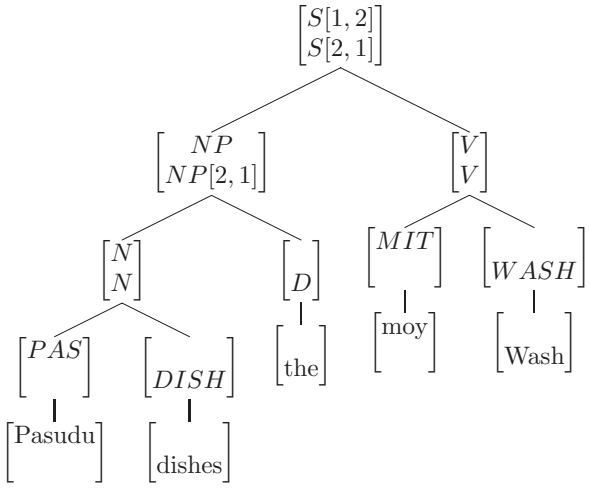
Summarization reduces the length of the input text, while maintaining as much information as possible and generating coherent grammatical output. Knight and Marcu (2000, 2002) propose two syntax-based approaches to sentence summarization. The first approach uses a noisy-channel model, where the source and channel models are unlexicalized PCFGs. Consider the example presented in Figure 1.4. t is a parsed version of the input sentence, and s is a proposed summary tree. The source model gives the language-model probability $\Pr(s)$, and it ensures that s is grammatical. Their language model uses unlexicalized PCFG productions and bigram statistics over the yield. The channel model gives the probability $\Pr(t|s)$ of transducing tree s into t, and it ensures coherence in expanding s into t. Their channel model uses unlexicalized context-free transductions, e.g. $\Pr(\text{A} \rightarrow \text{C B D}|\text{A} \rightarrow \text{C D})$. In the second approach of Knight and Marcu (2000, 2002), they propose a history-based method that extends a shift-reduce parser to include the "drop" operation. Daumé III and Marcu (2002) proposes a noisy-channel approach to document (multi-sentence) summarization. Their approach infers the syntactic structure and discourse structure over the input. Less important syntactic and discourse units are dropped from the structure.

Machine translation

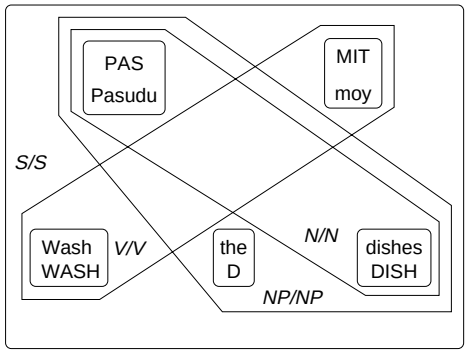
As stated earlier, Melamed (2004) and Melamed and Wang (2005) argue that machine translation can be viewed as parsing in two dimensions. Figure 1.5 gives an example of

Figure 1.5: A 2D multitree in English and transliterated Russian. Example from Melamed and Wang (2005). The three representations are equivalent: (a) Every internal node is annotated with the linear order of its children, in every component where there are two children. (b,c) Polygons are constituents.

(a) ordinary tree view



(b) parallel view



(c) 2D view

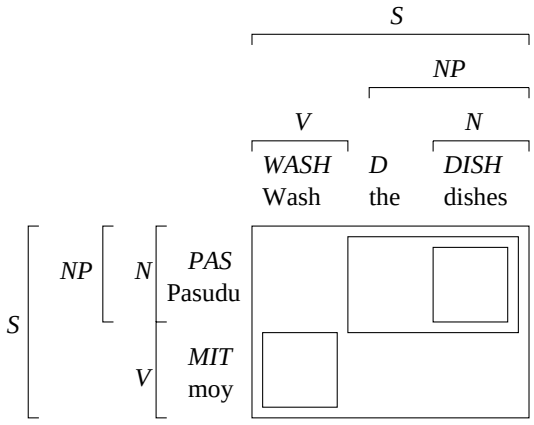


TABLE 1.1: Examples of translation rules from Marcu et al. (2006). The LHS is the "syntactified" target language phrase. The RHS is the source language phrase (Chinese, as English glosses).

r_1:	NP NNS astronauts	⟶	ASTRO- -NAUTS
r_2:	NP NP CC NP NNP NNP france and russia	⟶	FRANCE AND RUSSIA
r_3:	VP VBG PP IN coming from NP:x0	⟶	COMINGFROM x0
r_4:	NP NP:x0 VP:x1	⟶	x1 p-DE x0
r_5:	NNP france	⟶	FRANCE
r_6:	NP NP CC NNP france and NP:x0	⟶	FRANCE AND x0
r_7:	NNS astronauts	⟶	ASTRO- -NAUTS
r_8:	NNP russia	⟶	RUSSIA
r_9:	NP NNS:x0	⟶	x0
r_{10}:	PP IN:x0 NP:x1	⟶	x0 x1
r_{11}:	NP NP:x0 CC:x1 NP:x2	⟶	x0 x1 x2
r_{12}:	NP NNP:x0	⟶	x0
r_{13}:	CC and	⟶	AND
r_{14}:	NP CC NP:x0 and NP:x1	⟶	x0 AND x1
r_{15}:	NP VP VBG PP IN NP:x0 coming from NP:x1	⟶	x1 COMINGFROM x0

a 2-dimensional multitree in English and transliterated Russian. Syntax-based machine translation approaches currently achieve state-of-the-art accuracy (Marcu et al., 2006). Marcu et al. (2006) argue that purely phrase-based, i.e. non-syntactic, machine translation approaches can induce good translation lexicons, but are poor at rearranging translated phrases into grammatical output. They propose a probabilistic string-to-tree MT system which uses translation rules that include syntax on the target side. Examples of these rules are given in Table 1.1. r_1 indicates that the Chinese phrase "ASTRO- -NAUTS" may be translated into English as a noun phrase `NP` (`NNS` (astronauts)). r_3 indicates that the Chinese phrase "COMINGFROM" followed by some phrase x0 may be translated into a `VP` (verb phrase) with x0 as an `NP` (noun phrase).

1.2 Some recent trends in parsing research

How does one build an automatic parser? Historically, parsers were constructed manually based on grammarian-specified rules. Jelinek et al. (1994) explain:

> Parser development [was] generally viewed as a primarily linguistic enterprise. A grammarian examines sentences, skillfully extracts the linguistic generalizations evident in the data, and writes grammar rules which cover the language. The grammarian then evaluates the performance of the grammar, and upon analysis of the errors made by the grammar-based parser, carefully refines the rules, repeating this process, typically over a period of several years. (pg. 1 of 6)

Besides the enormous human effort required to craft such parsers, they generalize poorly to broader domains and different languages. Magerman (1994) and Collins (1999) review a number of these historical approaches.

The Penn Treebank

The Penn Treebank I was released mid-to-late 1991 (Collins, 1999, pg. 104), and it was improved and re-released as the Penn Treebank II (Marcus et al., 1993; Taylor et al., 2003). The development and public availability of these manually-parsed corpora (**treebanks**) ushered in a new era in the development of automatic parsers:

- The parsing task could be defined unambiguously: given an input sentence from the corpus, predict the tree in the treebank corresponding to this input sentence. Magerman (1994) called this the "treebank recognition problem" because the parser must predict the same tree that the treebank annotation scheme would generate.
- Treebanks allowed competing parsing techniques to be trained and evaluated under identical conditions. Black et al. (1991) proposed the so-called PARSEVAL family of measures for evaluating constituent parsers. They are based on computing the number

of constituents in the system's proposed parse tree that match the gold-standard parse tree in the treebank. These measures are labeled precision, labeled recall, and labeled F-measure, henceforth abbreviated as Prec., Rec., and F_1, respectively.

- Treebanks led to the decline of hand-crafted approaches and the rise of data-driven, statistical, corpus-based techniques.

Early work on treebank-induced statistical parsing began with unlexicalized PCFG-based approaches, with disappointing results (Collins, 1999, pg. 105). PCFGs were rejected as having overly restrictive independence assumptions that render them insufficiently powerful to model common linguistic phenomena. We examine subsequent trends in statistical parsing. All of these trends were, in one form or another, designed to improve modeling power and allow training with more powerful models. Because this dissertation is about the treebank recognition problem and more generally supervised learning techniques for structure prediction, rich but non-germane topics like unsupervised parsing methods and computational grammar formalisms are beyond the scope of this discussion. Terminology and more in-depth discussion are provided in Chapter 2 and Chapter 7.

History-based modeling and head annotation

Black et al. (1992, 1993) introduced two ideas to the parsing community that improve modeling power: head annotation and history-based modeling. For each constituent node, a certain subconstituent is determined to be representative in some sense. This subconstituent is the **head**. The head-tag and head-word of a node is determined by recursively selecting the head subconstituent until we reach a terminal node. History-based modeling uses the chain rule to rewrite the probability of a sequence of parsing decisions as the probability of the individual decisions, each conditioned on the previous decisions. In principle this allows arbitrary information to be used in the model, including head information. History-based modeling is described further in Section 2.4. It would be difficult to overstate the impact of these contributions, and they are individually or together present in many subsequent parsers.

Black et al. (1992, 1993) were restricted to using a history-based model with a hand-crafted grammar. Jelinek et al. (1994) and Magerman (1994) improved on the work of Black et al. (1992, 1993) by removing the hand-crafted grammar and instead using a process that directly predicted treebank trees. Magerman (1995) scaled the techniques in Jelinek et al. (1994) and Magerman (1994) to training and testing using the now-standard WSJ portion of the Penn Treebank II (Marcus et al., 1993; Taylor et al., 2003). The WSJ portion is the largest corpus in the treebank, with roughly 40,000 training sentences and 2,000 test sentences. Magerman (1995) presented the first results for parsing the WSJ that are significantly more accurate than PCFGs. For all these techniques, parameters were estimated using decision-trees with relative-frequency estimates at the leaves.

The Collins models

Collins (1996) determined the information needed by a conditional model to improve accuracy over the baseline established by Magerman (1995). These insights were later incorporated into the parsing models of Collins (1997, 1999, 2003), which surpassed the model of Collins (1996). We will describe the key aspects of these later models in depth.

Each model is a generative model estimated using relative frequencies. Features were based on head-word bigrams. There is special modeling for base noun phrases, which are the most common constituent in the treebank. Constituents are generated by the model top-down. If each inference were a complete top-down production, there would be an enormous number of potential inferences at each decision point. This would make search slow, and parameters estimated using relative-frequency would be unreliable because of data sparsity. Instead, each inference produces one constituent. Given a parent node, the first inference produces the head constituent. Subsequent inferences generate the sibling constituents to the right from the head outwards, until a dummy `STOP` item is generated, and then to the left from the head outwards, until a dummy `STOP` item is generated.

Collins presents three models, each with successively more conditioning information. Model 1 makes strong independence assumptions, using only a portion of the history information, because relative frequency estimation is very susceptible to sparse data problems. Model 1 includes the so-called "distance measure," which allows the model to prefer right-branching structures, as well as modification of the most recent verb. Model 1 is extended to Model 2 by modeling the complement/adjunct distinction, and subcategorization frames. Similarly, Model 2 is extended to Model 3 by modeling traces and *wh*-movement. To apply this approach to new languages and new domains, the re-parameterization process might require substantial intuition and human effort on the part of the experimenter.

Log-linear models

Relative frequency estimation is an unsophisticated technique for parameter estimation. It restricts the model to being a multinomial distribution over a discrete, finite set of conditioning events. Overlapping features cannot be used in the modeling. This limitation led to the rise of log-linear models estimated using maximum likelihood. The key advantage of this machine learning technique is its ability to use overlapping features.

We follow Jansche (2005) in using the term "log-linear" models. These models are sometimes also called "maximum-entropy" models in the NLP literature (e.g. Ratnaparkhi et al., 1994; Berger et al., 1996; Ratnaparkhi, 1997, 1999; McCallum et al., 2000). However, we avoid this term, because entropy can be maximized under a variety of constraints. For example, in unregularized maximum entropy approaches, one constraint is added for each feature to ensure that the expected values of the feature under the model and under the training data are equivalent. Other work (e.g. Dudík et al., 2007) shows how regularization can be introduced by maximizing entropy under relaxed constraints.

As far as we know, Ratnaparkhi et al. (1994) proposed the task of parse reranking as well as the first log-linear model in the parsing literature. They performed parse reranking using a global log-linear conditional model. The authors argue that the model should be globally normalized. However, it was impractical for them to compute the normalization term over all parses, so they instead normalized only over the top k parses.

Ratnaparkhi (1997, 1999) proposed a history-based parser that uses a locally-normalized conditional log-linear model. Ratnaparkhi (1999) pointed out that other machine learning techniques can be used for modeling the cost function in a history-based approach. Example generation was performed offline. Parameter estimates were unregularized, save for the use of frequency-based cutoffs on the features: features must occur at least 5 times in the training examples.

Like Ratnaparkhi et al. (1994), Johnson et al. (1999) proposed parse reranking that uses a globally-normalized conditional log-linear model. To avoid overfitting, their approach uses a **Gaussian prior** (ℓ_2-regularization) rather than frequency-based cutoffs. One limitation of the approach of Johnson et al. (1999) was that, to compute the partition function, they enumerated all possible parses for a given sentence. This was problematic because enumerating all such parses for a given sentence using a broad-coverage grammar can be prohibitive. Since they do not actually consider the problem of inference, their work is actually on parse reranking, and not full parsing. Geman and Johnson (2002), Miyao and Tsujii (2002), and Johnson (2003) subsequently proposed techniques for training globally-normalized conditional models for full parsing, which were used by Clark and Curran (2003, 2004, 2007)

Charniak (2000) proposed a local generative log-linear model. The model generates the tree top-down in a manner similar to the models of Collins. For each constituent, the parser first guesses the head-tag, and then the head-word, and then the expansion into further constituents. Expansion is done using a Markov grammar: first guess the head, then each left constituent sibling from the head out until a dummy `STOP` is generated, then each right constituent sibling from the head out until a dummy `STOP` is generated. Although in theory the model is locally normalized, in practice it was not. Because computation of the normalization term is expensive, Charniak (2000) merely skipped this step during parameter estimation. He used a variant of standard deleted interpolation for smoothing. The approach of Charniak (2000) achieved state-of-the-art accuracy, surpassing the results of Collins (1997). Like the approach of Collins, one limitation of the approach of Charniak (2000) is that it requires manual feature selection. For this reason, generalizing it to other tasks and languages may require significant human effort.

Large-margin methods

Collins (2004) analyzed various parameter estimation techniques through the lens of the statistical learning theory. The **margin** of a model on a particular instance is a measure of the distance between the score of the correct output and the score of all incorrect outputs.

A model that has large margins for a large percentage of training examples will tend to generalize accurately. For each training example, there can be an exponential number of incorrect outputs. A structure prediction model induced under the large-margin principle might have to satisfy exponential number of constraints. Subsequent work demonstrates various techniques for solving this exponential-sized optimization problem. Taskar et al. (2004a,b) and McDonald et al. (2005a,b, 2006) used independence assumptions to reduce the optimization problem to a polynomial size. Taskar et al. (2004b), Tsochantaridis et al. (2004, 2005), and Collins and Roark (2004) used inference to select subsets of constraints.

Using the entire history

In principle, sophisticated models can use information from the entire history when considering a parse decision. However, it is not obvious how to extract useful information from a variable-length history. All previous approaches we have discussed use only manually-defined feature sets that do not examine the entire unbounded history. Also, with the exception of the early decision-tree-based parsers, they all work over the feature set provided and do not combine features in useful ways. However, the model could have access to far more information were it to automatically combine fine-grained features in interesting ways.

Henderson (2003) studied methods for inducing history representations. This work culminated in the approach of Henderson (2004), which achieved state-of-the-art accuracy on constituent parsing. The parsers in this work use one of two different probability models, each of which are estimated using a recurrent neural network architecture called Simple Synchrony Networks (SSNs). The history and **lookahead** (remaining words in the input) are variable-length, but they are each compressed to a fixed-size **hidden** or **intermediate** representation. The hidden representation(s) and other conditioning information are then used to compute probabilities. SSN training simultaneously learns the mappings from variable-length representations to fixed-length representations and the mapping from the fixed-length representation to a probability estimate. Titov and Henderson (2007a,b) used a similar approach as Henderson (2003, 2004), but demonstrated an increase in accuracy by using incremental sigmoid belief networks instead of SSNs.

In these works, the hidden representation is opaque, as it is difficult for an experimenter to determine what information is present and what information is missing. The hidden representation for the lookahead is constructed right-to-left, i.e. with the inductive bias that words further from the current decision point are less important and are more likely to be lost during compression. For this reason, it might be difficult to learn to preserve the last few words of the sentence. Similarly, it might be difficult to learn cooccurrence features over unbounded lookahead ("is the word 'foo' anywhere in the entire right context?"). This example illustrates the potential difficulty in choosing an appropriate inductive bias for the hidden representation mappings. It might be difficult to determine the quality of and improve these mappings. In this dissertation, we demonstrate how to induce models

that can automatically use all available information from the history and input. One major contribution is that features accessing arbitrary information are represented directly without the need for an induced intermediate representation

Rerankers

Another interesting research direction has been parse reranking, a task originally proposed by Ratnaparkhi et al. (1994). A pre-existing parser is used to generate candidate parse trees for each input sentence. The parse reranker's task is to determine the best candidate parse. Parse rerankers do not need to perform inference to find the best solution, and instead focus on crafting accurate tree cost functions. Insights gleaned from reranking research might help improve the models of full parsers. The literature on parse reranking is quite extensive, and includes Johnson et al. (1999); Collins (2000); Collins and Koo (2005); Collins and Duffy (2002a); Shen et al. (2003); Charniak and Johnson (2005); and Kudo et al. (2005).

1.3 This dissertation

In this dissertation, we examine the task of **constituent parsing**, i.e. predicting a labeled tree over an unrestricted natural language string. Automatic parsing approaches have been an object of study for almost half a century (Joshi and Hopely, 1996),[2] yet state-of-the-art constituent parsers struggle to exceed 90% accuracy. We believe that the reason for this barrier is that parsing work has been too focused on taking approaches that work and making them more powerful (e.g. unlexicalized PCFGs → lexicalized PCFGs) rather than taking powerful approaches and making them work. Our overriding design principle is to avoid reducing the upper-bound on accuracy achievable. With this in mind, we attempt to automatically learn a cost function over all relevant information. In particular:

- We use the *entire* history in scoring parse decisions. As far as we know, all previous approaches, save those of Henderson (2003, 2004) and Titov and Henderson (2007a,b), have looked at information in a limited window. We use finer-grained information than all previous approaches, and this information is automatically combined in ways that improve the model's discriminatory power. This information is included in the model using a new machine learning technique that we develop.
- Most parsers employ local limits on search effort, e.g. beam pruning, whereas we impose a global limit on search effort. In our experiments, our best model finds the optimal solution for all sentences, i.e. the global limit is not exceeded and there are

[2] Joshi and Hopely (1996) cite the Transformations and Discourse Analysis Project Reports #15 through #19, 1959–60, University of Pennsylvania, which are available in the Library of the National Institute of Science and Technology (formerly known as the National Bureau of Standards), Bethesda, MD.

no search errors. As far as we know, no previous search-based parser has made this claim. Moreover, our cost function is so refined that, on 80% of sentences, the greedy solution is also the optimal one. A major contribution of our work is a cost-function that is accurate enough to guide search to the optimal solutions with very little perplexity.

Increasingly, emphasis has been placed on developing machine learning techniques to induce accurate parsers using as little manual effort as possible. Nonetheless, most current parsing approaches still require time-consuming feature engineering and task-specific approaches. To address these problems, we propose a flexible, end-to-end discriminative method for training parsers. The proposed parameter estimation technique is regularized without ad-hoc smoothing or frequency-based feature cutoffs. The training regime can use arbitrary information not only from the input, but also from the entire history. The learning algorithm projects the hand-provided features into a compound feature space and performs incremental feature selection over this large feature space. The core of the parser is a model learned to optimize a single regularized objective. The resulting parser achieves higher accuracy than a generative baseline, despite not using a generative model as a feature, not having extensive hand-crafted features, and not using much language-specific information.

The layout of this dissertation is as follows: Chapter 2 introduces terminology and our general approach to parsing, breaking down a parser into its major components: a **logic**, which structures the search space of possible parser decisions; a **cost function** (or **model**), which assigns weights to paths through the search space; and a **search strategy**, which is an algorithm that determines how the parser explores the search space. Chapter 3 describes the search strategy used by the parser to find the minimum cost output tree. Chapter 4 shows how we induce the inference cost function. Learning the cost function (a.k.a. **parameter estimation** or **training**) involves choosing a cost function that has the best generalization, i.e. that maximizes the expected value of the evaluation measure on unseen inputs. Chapter 5 describes optimizations and approximations used to speed-up training and to reduce memory consumption. Chapter 6 presents experiments and results. Chapter 7 discusses related work. Finally, Chapter 8 summarizes the contributions of this dissertation, as well as its limitations.

❧ Second Chapter ❧

General Approach

> For every problem there is one solution which is simple, neat, and wrong.
>
> H. L. Mencken

The proposed method employs the traditional AI technique of predicting a structure by searching over possible sequences of inferences, where each inference predicts a part of the eventual structure.

2.1 Terminology

The following terms will help to explain our work. A **span** (or **bracket**) is a range over contiguous words in the input sentence. Spans **cross** if they overlap but neither contains the other. A **labeled constituent item** (**item** or **constituent**, for short) is a (span, label) pair. An item is either a terminal or a non-terminal:

- Each word corresponds to exactly one **terminal** item. This terminal item spans that word, and is labeled by the part-of-speech (POS) tag of the word.
- A **non-terminal** item must have a non-terminal label, e.g. `NP` ("noun phrase").

When we say that an item crosses another item, we are referring to a property of their spans. A **state** is a set of items, none of whose spans may cross. The **initial** (or **start**) state contains only terminal items, one for every word in the input. A **final** (or **complete**) state is one that contains a non-terminal item labeled `TOP` whose span covers the input. When unambiguous, we will also refer to a final state as a **parse**. An **inference** is a transition between states, where the items in the **antecedent** (predecessor) state are a strict subset of items in the **consequent** (successor) state. An inference should not be confused with the inference process, which involves many inferences. The initial state has no incoming transitions, and the final state(s) have no outgoing transitions. The **search space** is a

directed acyclic graph containing as nodes the initial state and all states reachable from the initial state. Edges in this graph are inferences.[1] A **path** in this graph must begin at the initial state. A **complete** path is one leading to a final state. A state S is **correct** with respect to some **gold-standard** final state $\hat{S}$ iff $\hat{S}$ is reachable from S. A path is correct iff it leads to a correct state. An inference is correct iff the consequent state is correct. We view a parser as having three major components:

- a **logic**, which structures the search space of possible parser inferences,
- a **cost function** (or **model**), which assigns weights to paths through the search space,

  and
- a **search strategy**, which is an algorithm that determines how the parser explores the search space.

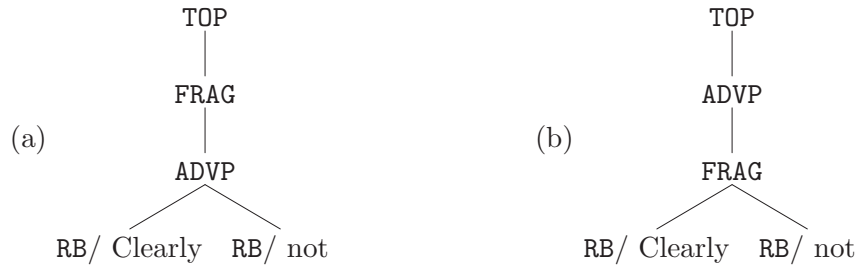
This is similar to the anatomy proposed by Melamed and Wang (2005), who decompose a parser into a grammar, a logic, a semiring, a search strategy, and a termination condition. We assume that our logic implicitly includes specifies the termination condition via final states. Our cost functions include the values produced by the grammar. In general, costs can be combined in arbitrary ways, not just using semiring operators.

2.2 Parsing logic

The parsing **logic** (Melamed and Wang, 2005) is a set of constraints that structures the search space for an arbitrary input sentence. A logic is **deterministic** (Marcus, 1980) if each state in the search space can be reached by only one path. Section 2.1 gave several task-specific constraints, e.g. a state cannot contain any two items with crossing brackets. There are several other constraints we impose throughout this work:

- Each inference adds a single item to the antecedent state to obtain the consequent state, so an inference is a $(\text{state}, \text{item})$ pair.
- A `TOP` item can be inferred only if the antecedent state contains some item spanning the entire sentence.
- `TOP` items must span the entire sentence.

Figure 2.1: Parses with different unary ordering.
(a) The gold standard parse. (b) Hypothetical parser output.



2.3 Unary ordering

In the previous sections, for clarity of exposition we have glossed over a problematic, but important detail: unary projections. A constituent item in a state is a **unary projection** (alternately, **unary rewrite**, or **unary** for short) if the state contains another item with the same span.[2] A **unary chain** is a sequence of unary projections, including the bottom-most item. Trees in the Penn Treebank (Marcus et al., 1993; Taylor et al., 2003) give a total vertical order over items with the same span. However, the standard PARSEVAL evaluation measures *ignore* any ordering over same-span items (Black et al., 1991). For example, if given the gold-standard parse in Figure 2.1(a) and the hypothetical parser output in Figure 2.1(b), PARSEVAL would give the parser a score of 100%, even though it inferred the `FRAG` and `ADVP` brackets in the wrong order.

We don't know whether unary orderings in the treebank provide useful information or whether they are a confounding variable. We opt to preserve the Penn Treebank information and update the definitions given in Section 2.1: our states keep track of the order of same-span items, and our inferences preserve the order of same-span item from the antecedent to the consequent. As far as we know, this design decision is the same as that of all the other parsers in the literature.[3] The reason unary order is typically preserved might be that, for most of these parsers, it is trickier to ignore unary orderings than to preserve them, but this tendency should not be taken as evidence that preserving unary orderings is useful.

We note two properties that follow from the constraint that inferences preserve unary ordering. First, a final state can be represented as a tree and a path can be represented

[1] This view of parsing is in contrast to one in which the search space is a directed *hyper*graph, nodes are *items*, and inferences are *hyperedges*, each of which have one *or more* antecedents (Gallo and Scutellà, 1999; Klein and Manning, 2001b).

[2] The first item inferred with a particular span added to a state is typically not considered a unary. Only the (subsequent) same-span items added thereafter are unaries.

[3] For example, Taskar et al. (2004b) flatten all unary chains to a single order-preserving compound label, e.g. `TOP/FRAG/ADVP` in Figure 2.1(a), thus ensuring that no two items in a state can have the same span.

as a sequence of trees. Second, our definition of "correctness" is stricter than that of PARSEVAL's: Figure 2.1(b) is an incorrect state, since it does not match Figure 2.1(a). Similarly, in the initial state of the sentence in Figure 2.1, the only correct inference would be to add an `ADVP` spanning "Clearly" through "not".

2.4 Modeling

Consider a sentence s and a parse p. More generally, s is an input and p is a complete output structure. We wish to model the conditional probability $\Pr(p|s)$. p comprises a sequence of individual items $d_1, \ldots, d_{|p|}$, where each item d_j belongs to a possibly infinite space $\mathcal{D}$. For notational convenience, define d_0 as s. We then have:

$$\Pr(p|s) = \Pr(d_1, \ldots, d_{|p|}|d_0) \tag{2.1}$$

$$= \prod_{j=1}^{|p|} \Pr(d_j|d_0, \ldots, d_{j-1}) \tag{2.2}$$

Equation 2.2 follows from the chain rule, and is the essence of history-based modeling (Black et al., 1992, 1993). Bikel (2004b, pp. 8–9) gives a more precise formal definition of probabilistic history-based models, albeit with a very different exposition from ours. Note that $\Pr(d_j|d_0, \ldots, d_{j-1}) = 0$ if the items in the state are contradictory or in some other way not licensed by the logic. We call $d_0, \ldots, d_{j-1}$ the **state** (a.k.a. **history**) at the j^{th} step in the path. To be precise, the state is a set of items whereas the history is a sequence of items. We assume that the order of the items in the history is not used in the modeling, and ignore the distinction between state and history. We will abbreviate the state $d_0, \ldots, d_{j-1}$ as S_j, where $S_j \in 2^{\mathcal{D}}$. Rewriting:

$$\Pr(d_j|d_0, \ldots, d_{j-1}) = \Pr(d_j|S_j) \tag{2.3}$$

We will use the abbreviation $\mathcal{D}(S_j) \subseteq \mathcal{D}$ to mean all items at state S_j that are permitted by the logic, i.e. $d \in \mathcal{D}(S_j)$ if and only if $\Pr(d|S_j) \neq 0$.

Independence assumptions can be introduced into the model. For example, we can make the Markov assumption that only the input and the last item affect the probability of the current item:

$$\Pr(d_j|S_j) \Rightarrow \Pr(d_j|d_0, d_{j-1}) \tag{2.4}$$

The two most common reasons for introducing independence assumptions are as follows:

- Unsophisticated parameter estimation techniques, such as relative frequency estimation, might not be able to use all the information in the history effectively.
- Independence assumptions might allow polynomial-time algorithms (e.g. dynamic

programming) to find the maximum probability parse for a given sentence, or at least to reduce the size of the search space.

However, we prefer not to restrict the information available to the model. In this dissertation, we develop automatic techniques for using all information in the history and show that our models are able to use this information to avoid exploring most of the search space.

If we want a true, non-deficient probability distribution over all complete output structures, we may impose the restriction that the sum of the probabilities of all items at a given state sum to 1:

$$\sum_{d \in \mathcal{D}(S_j)} \Pr(d|S_j) = 1 \tag{2.5}$$

This restriction corresponds to the assumption that one and only one item must be chosen at this state. Typically, this restriction is implemented by normalization, hence the name **locally-normalized conditional models**. Lafferty et al. (2001) argue that local normalization can cause **label bias**. Instead of imposing the restriction in Equation 2.5, it might be possible to avoid label bias in history-based models by estimating the likelihood of each inference independently of the other inferences at the state. This allows several good inferences to receive high probability at a good state, or perhaps all inferences to receive low probability at a bad state. Label bias will be discussed further in Chapters 4 and 7. To avoid label bias, we do not impose the restriction in Equation 2.5.

An **inference** is a transition between states. We write $i_j \in \mathcal{D} \times 2^{\mathcal{D}}$ to mean the j^{th} inference, which adds item d_j to state S_j. Rewriting:

$$\Pr(d_j|S_j) = \Pr(i_j) \tag{2.6}$$

The search space is a DAG, where each node is a state and each edge is an inference. We let $\mathcal{I}$ be the space of all inferences, and use the abbreviation $\mathcal{I}(S_j) \subseteq \mathcal{I}$ to mean all inferences at state S_j that are permitted by the logic. The difference between $\mathcal{I}(S_j)$ and $\mathcal{D}(S_j)$ is that the former consists of inferences, while the latter consists of items.

Each inference contains all state information. Since the information in an inference might be of variable length, one typically models the probability of an inference by first transforming the inference into a fixed-length real-valued feature vector using feature extraction function $X : \mathcal{D} \times 2^{\mathcal{D}} \rightarrow \mathbb{R}^{|F|}$. F is a finite, perhaps high-dimensional, feature space that indexes the entries of the feature vector. We will rewrite $\Pr(i_j)$ as $\Pr(X(i_j))$ and:

$$\Pr(p|s) = \prod_{j=1}^{|p|} \Pr(X(i_j)) \tag{2.7}$$

Instead of finding the parse with the highest conditional probability, we can find the parse with the lowest negative log-probability:

$$\arg\max_{p \in P(s)} \Pr(p|s) = \arg\min_{p \in P(s)} \left(-\log \Pr(p|s)\right) \tag{2.8}$$

$$= \arg\min_{p \in P(s)} \left(-\log \prod_{j=1}^{|p|} \Pr(X(i_j)) \right) \tag{2.9}$$

$$= \arg\min_{p \in P(s)} \left(-\sum_{j=1}^{|p|} \log \Pr(X(i_j)) \right) \tag{2.10}$$

where $P(s)$ are the complete output structures over input s that are licensed by the logic. We may even relax the restriction that our model is probabilistic, and instead use an arbitrary cost function. We use the notation that C is the cost of a complete output structure, and c is the cost of an individual inference. We assume that cost monotonically increases along a path, i.e. $c(i_j)$ is always positive. Generalizing, we write:

$$C(p|s) = \sum_{j=1}^{|p|} c(X(i_j)) \tag{2.11}$$

We might abbreviate $C(p|s)$ as $C(p)$ when confusion is unlikely. We might also omit X from the cost function for convenience. We let the cost function be parameterized by Θ:

$$C_\Theta(p|s) = \sum_{j=1}^{|p|} c_\Theta(X(i_j)) \tag{2.12}$$

Θ is a parameter vector which is chosen by training. We might refer to Θ as the model when confusion is unlikely.

We can express arbitrary tree costs with the decomposition in Equation 2.12. For example, all inferences can have cost zero until the last one. However, we usually prefer to push cost as early as possible in the inference chain, to make search faster. Unlike most approaches employed in NLP, the proposed method makes no independence assumptions: Inference cost function c_Θ can use arbitrary information not only from the input, but also from the entire state.

2.5 Search strategy

Given input sentence s, let $P(s)$ be the set of complete paths that are admitted by the logic. The parser uses a **search strategy** to find $\hat{p} \in P(s)$ with minimum cost $C_\Theta(p)$ under model Θ:

$$\hat{p} = \arg\min_{p \in P(s)} C_\Theta(p) \tag{2.13}$$

The parser then returns the parse (final state) of $\hat{p}$.

Chapter 3 describes our search strategy, Chapter 4 discusses how we induce the inference cost function, and Section 6.2 presents three different logics we use in our experiments.

❧ Third Chapter ❧

Search Strategy

A cage went in search of a bird.

Franz Kafka

Given an input sentence s, the parser uses the search strategy to find a solution to Equation 2.13. In general, the cost function c_Θ can consider arbitrary properties of the input and parse state. We do not know any tractable exact solution to Equation 2.13, such as dynamic programming. Our parser finds an approximate solution using agenda-driven search (e.g. Felzenszwalb and McAllester, 2007). The core data structure in this algorithm is an **agenda**, which is a priority queue. In general, the agenda stores entire search paths instead of single items. Since a deterministic logic defines a one-to-one correspondence between paths and states (see page 16), the agenda of a deterministic parser can store states instead of paths. We use the term **partial parse** to refer to the elements of the agenda, regardless of whether they are paths or states. We define $D_\Theta(p)$ as the agenda priority of partial parse p under model Θ, also known as a **figure-of-merit** (Caraballo and Charniak, 1998). The choice of D_Θ determines the **search strategy**, i.e. the way that the parser allocates search effort.

Parsers in the literature typically choose some local limit on the amount of search, such as a maximum beam width (e.g. Ratnaparkhi, 1999; Collins and Roark, 2004; Henderson, 2004; Titov and Henderson, 2007a,b). The problem with traditional beam-search is that it performs permanent pruning of nodes with an **inadmissible** technique (Zhou and Hansen, 2005). This leads to **incomplete** search, i.e. beam-search can have search errors and prune the optimal solution. With an accurate cost function, restricting the search space using a fixed beam width might be unnecessary. Instead, we impose a global limit on exploration of the search space. Termination is controlled by a limit γ on the maximum number of partial parses to score.

Listing 3.1 provides pseudo-code for agenda-driven search. The pseudo-code is for the special case of search over a directed graph rather than for the more general case of search over a directed hypergraph (a.k.a. and/or graph), in which consequent elements are generated by composing elements in the chart with the element most recently popped from the

agenda. For this pseudo-code, see Figure 5 in Felzenszwalb and McAllester (2007). In our algorithm, the parser maintains the best solution found thus far, initialized to no solution, which has cost $+\infty$. The agenda is initialized to contain the partial parse corresponding to the initial state for input sentence s. At each step in the search, the parser pops the highest priority partial parse from the agenda. As described on page 20, the cost of a path monotonically increases as it is extended. If the cost of the partial parse exceeds the cost of the current best solution, then no extension of this partial parse can beat the current best solution, and we can stop exploring this partial parse. Otherwise, we find all expansions that add a single item to this partial parse using $\mathcal{I}(p)$, as defined on on page 19. Expansions that lead to a final state are checked to see if they beat the current best solution. If they do, the current best solution is updated. Non-final expansions are added to the agenda, with priority determined by D_Θ. Search terminates either when the agenda is empty, in which case the parser has found the optimal solution, or when the limit on search effort is exceeded.

We experimented with two agenda-driven search strategies: standard best-first search, and greedy completion, a search strategy that is novel as far as we know.

3.1 Best-first search

In best-first search, the priority $D_\Theta(p)$ of a partial parse p is its negative cost $-C_\Theta(p)$, i.e. lowest cost partial parses are explored first. Because inference cost is non-negative, the partial parse cost is an under-estimate of the total tree cost. It is common for search strategies to employ of heuristic estimate of the cost of completing a partial parse. For example, in A* search the agenda priority of a partial parse is based on adding a heuristic completion cost under-estimate to the current cost of the partial parse thus far (Klein and Manning, 2003a). Typically, heuristic completion costs consider information from outside the item span. Since our costs can consider the entire context, they might already take into account some information about completion cost. For example, since we can consider arbitrary information from the state, including adjacent items, we can model context-summary estimates (Klein and Manning, 2003a). In other words, the priority can estimate the cost of completion, since the cost can be based on arbitrary information from the state and input. For example, consider the incorrect inference shown in Figure 3.1. The verb phrase (`VP`) should include the noun phrase (`NP`) and span "Go" through "home". However, if the model can examine only its descendants in evaluating an inference, then labeling "Go" as a `VP` would be plausible and would not receive high cost. With the restriction that it can examine only its descendants, the model would have insufficient information until it reached state (b). At state (b), it would consider inferring items that span "Go" through "home", i.e. items that have the `VP` and the `NP` items as its children. The model would finally have enough information to determine that it inferred the `VP` incorrectly and assign high cost to subsequent inferences at state (b). Having less information available forces

Listing 3.1: Pseudocode for agenda-driven search over a directed search graph.

```
    ▷ Given sentence s, model Θ, and a limit γ on the number of partial parses to score,
 1: procedure AgendaSearch(s, Θ, γ)                    ▷ approximately solve Equation 2.13
 2:     p̂ ← ∅, where C_Θ(∅) = ∞                        ▷ The current best solution is failure,
                                                                  ▷ which has infinite cost
 3:     p ← partial parse corresponding to the initial state of s
 4:     q.push(p, D_Θ(p))                  ▷ The agenda q contains only the initial partial parse
 5:     workdone ← 1                              ▷ We have scored only the initial partial parse
 6:     repeat
 7:         p ← q.pop()                ▷ Retrieve the partial parse p with highest priority D_Θ(p)
 8:         if C_Θ(p) < C_Θ(p̂) then                ▷ If p can lead to a solution that beats p̂,
 9:             Explore(p)                                                ▷ then explore p
10:     until Done(γ)
11:     return p̂                                          ▷ Return the best solution found

12: procedure Explore(p)                          ▷ Explore p by processing all its expansions
13:     for each p′ ∈ I(p) do              ▷ Find all expansions, i.e. inferences permitted at p
14:         if p′ does not lead to a final state then ▷ Non-complete paths can be explored,
15:             q.push(p′, D_Θ(p′))          ▷ so add p′ to the agenda q with priority D_Θ(p′)
16:         else                          ▷ Complete paths don't need to be put on the agenda,
17:             if C_Θ(p′) < C_Θ(p̂) then          ▷ so just check if p′ is the new best solution
18:                 p̂ ← p′                          ▷ If so, update the current best solution found
19:     workdone += |I(p)|                    ▷ Increase the number of partial parses scored

20: procedure Done(γ)                                              ▷ Are we done parsing?
21:     if q is empty then
22:         return true                                        ▷ We have the optimal solution
23:     else if workdone > γ then
24:         return true          ▷ We have exceed the maximum amount of search effort
25:     else
26:         return false
```

Figure 3.1: An incorrect parse inference. (a) The antecedent state. (b) The consequent state.

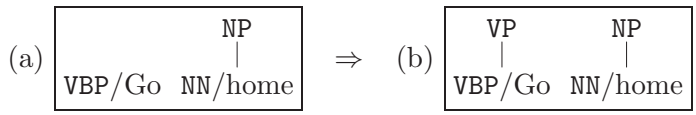
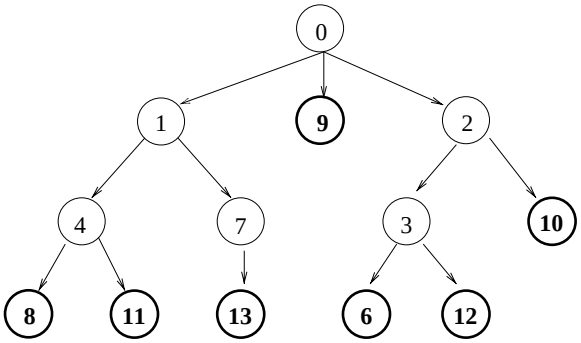


Figure 3.2: An example search space for a deterministic logic, depicted as a tree. (A non-deterministic logic's search space will be a DAG, not a tree.) The root of the tree is the initial state. Instead of labelling inferences (edges) with their cost, states (nodes) are labelled with the cost of the path from the root to that node. States are assigned unique cost values so they can be identified by their cost. Final states are indicated in bold.



Best-first search would explore the states in the following order:
0 (new best solution 9), 1, 2, 3 (new best solution 6), 4. Return 6.

Greedy completion search would explore the states in the following order:
0 (new best solution 9), 1, 4 (new best solution 8), 2, 3 (new best solution 6). Return 6.

the model to defer assigning cost until later in the search. Any decision to assign cost is irrevocable, so a model that has little information must be conservative in assigning cost. Deferring cost assignment until later means more branching early during search, i.e. more paths are considered plausible until more information is received later. In contrast, a model that can examine the entire state can assign high cost to the inference in Figure 3.1, since it can know that it is unlikely to have a `VP` followed by a sentence-final `NP`. A model with more information can assign cost earlier, and thus invest less search effort in wild goose chases. In this sense, the agenda priority can include information like the context-summary estimates of Klein and Manning (2003a), which can be automatically learned when the inference cost function is induced.

3.2 Greedy completion

Traditional greedy search pursues the lowest cost expansion at each step. After a solution is found, greedy search terminates. In **greedy completion**, traditional greedy search is the inner loop and all expansion partial parses considered but not greedily pursued go on the agenda. When traditional greedy search finds a solution or cannot proceed because it cannot beat the current best solution, the inner loop terminates and there is a frontier of expansion partial parses not explored during previous greedy searches. These frontier partial parses form the agenda. Greedy completion picks the lowest cost partial parse in the frontier and starts a new inner loop (iteration of greedy search) beginning at this partial parse. Another perspective is that greedy completion is agenda-based search with a particular priority: In greedy completion, the priority of a partial parse is its negative cost, except that we always prefer any partial parse that was a consequent of the last partial parse popped. Listing 3.2 provides more explicit pseudo-code for greedy completion. Figure 3.2 gives examples of best-first and greedy completion search.

Both greedy completion and best-first search are **complete**, i.e. guaranteed to return the optimal solution, under two assumptions: there is no limit on the number of partial parses scored, i.e. $\gamma = \infty$, and path cost is non-decreasing. The number of states in the search space is exponential in the size of the input, so in general exact search (with $\gamma = \infty$) is intractable. However, with a finite limit on search effort, best-first search may return poor solutions, or simply fail and return no complete path. At every point in time, greedy completion tries to find the best solution (complete path) as early as possible, in case it exceeds the limit on search effort. In this way, it attempts to progressively tighten the upper-bound on the optimal solution cost as quickly as possible, which makes it an **anytime** search algorithm. The observations made by Korf (1998) about depth-first branch-and-bound also hold for greedy completion:

> In [depth-first branch-and-bound], the cost of the best solution found so far is always an upper bound on the optimal solution cost, and decreases until it reaches the optimal cost. ... While [depth-first branch-and-bound] never expands any node more than once, its overhead consists of expanding some nodes whose cost exceed[s] the optimal cost. (pg. 9)

The intuition behind greedy completion is as follows: Since the problem solver is not rewarded partial credit for incomplete solutions, greedy completion's inner loop is motivated by the overriding imperative to complete quickly the work it is doing (Step 9 in Listing 3.2). Specifically, given a partial parse, the inner loop finds a solution from this partial parse as soon as possible, tiebreaking at each step in favor of lowest-cost consequent partial parse (Step 14 in Listing 3.2). In the outer loop (Steps 6–10 in Listing 3.2), greedy completion must pick a new point in the search space to restart the greedy search process. Harvey and Ginsberg (1995) write:

Listing 3.2: Pseudocode for greedy-completion search over a directed search graph. This algorithm is a special case of the agenda-based search algorithm (Listing 3.1 on page 25) for a particular choice of priority D_Θ. This pseudo-code explicitly describes how the search space is explored.

▷ Given sentence s, model Θ, and a limit γ on the number of partial parses to score
1: **procedure** GreedyCompletion(s, Θ, γ) ▷ approximately solve Equation 2.13
2: $\hat{p} \leftarrow \emptyset$, where $C_\Theta(\emptyset) = \infty$ ▷ The current best solution is failure, ▷ which has infinite cost
3: $p \leftarrow$ partial parse corresponding to the initial state of s
4: q.push(p, -$C_\Theta(p)$) ▷ The priority queue contains only the initial partial parse
5: workdone $\leftarrow 1$ ▷ We have scored only the initial partial parse
6: **repeat**
7: $p \leftarrow q$.pop() ▷ Retrieve the partial parse p with highest priority $-C_\Theta(p)$
8: **if** $C_\Theta(p) < C_\Theta(\hat{p})$ **then** ▷ If p can lead to a solution that beats $\hat{p}$,
9: GreedilyComplete(p) ▷ then greedily complete p
10: **until** Done(γ) ▷ Done is given in Listing 3.1, Step 20
11: **return** $\hat{p}$ ▷ Return the best solution found

▷ Greedily complete p, storing all non-explored states along the way
12: **procedure** GreedilyComplete(p)
13: **while** p does not lead to a final state and $C_\Theta(p) < C_\Theta(\hat{p})$ **do**
14: $\tilde{p} \leftarrow \arg\min_{p' \in \mathcal{I}(p)} C_\Theta(p')$ ▷ Find the lowest cost consequent inference,
15: **for each** $p' \in \mathcal{I}(p) \setminus \{\tilde{p}\}$ **do** ▷ and process every other consequent
16: **if** p' does not lead to a final state **then** ▷ Non-complete paths ▷ can be explored,
17: q.push($p', -C_\Theta(p')$) ▷ so add p' to the agenda q with priority $-C_\Theta(p')$
18: **else** ▷ Complete paths don't need to be put on the agenda,
19: **if** $C_\Theta(p') < C_\Theta(\hat{p})$ **then** ▷ so just check if p' is the new best solution
20: $\hat{p} \leftarrow p'$ ▷ If so, update the current best solution found
21: workdone += $|\mathcal{I}(p)|$ ▷ Increase the number of partial parses scored
22: $p \leftarrow \tilde{p}$ ▷ Greedily pursue the best choice
23: **if** p leads to a final state and $C_\Theta(p) < C_\Theta(\hat{p})$ **then** ▷ Check if p is the ▷ new best solution
24: $\hat{p} \leftarrow p$ ▷ If so, update the current best solution found

> Chronological backtracking [to the deepest unexplored partial parse] puts a tremendous burden on the heuristics [cost functions] early in the search and a relatively light burden on the heuristics deep in the search. Unfortunately, for many problems the heuristics are *least* reliable early in the search, before making decisions that reduce the problem to a size for which the heuristics become reliable. (pg. 2 of 7)

Greedy completion could backtrack **chronologically** to a deep but high-cost incomplete partial parse, as done by depth-first branch-and-bound, but it instead assumes that high-cost late-branching from a previous solution path is unlikely to produce a lower cost solution. Instead, greedy completion assumes that it has at least sufficient time remaining to greedily complete the incomplete partial parse of its choosing. So it backtracks to the lowest-cost incomplete partial parse (Step 7 in Listing 3.2), hoping that it will find a tighter upper-bound on the optimal solution cost, thus saving work in the long run.

3.3 Additional optimizations

We implemented two optimizations to the search algorithm in Listing 3.1:

- Assume the parser has a non-deterministic logic. As stated earlier, the agenda stores paths, and several paths might lead to the same state. So we keep a **chart** of popped paths. After we pop a path p (Line 7 in both listings), we check if the chart contains a path leading to the same state as p whose cost is no greater than that of p. In this case, exploring p would redo work at no less cost than before, so we skip p. This is standard chart parsing, which collapses states with identical signatures.

- As stated earlier, the space complexity of best-first search and greedy completion is, worst-case, exponential. If the limit on search effort γ is finite, space complexity is $O(\gamma)$. At any point during search, the agenda and chart need not store more than γ partial parses total. We can save more memory by using the "bound" operation of "branch-and-bound". We "bound" by removing from the chart and agenda any partial parse with cost higher than $C_\Theta(\hat{p})$, the cost of the current best solution.

It is conceivable that with the global limit on search effort we might not find the optimal solution. Nonetheless, we can determine if the solution returned by agenda-based search is optimal. We will perform this test in our experiments (Chapter 6) to measure the efficacy of our parsers' searches. We are guaranteed at termination that the solution returned is optimal if:

- cost is non-decreasing along the path to a complete structure,

- the cost of the solution is no greater than the cost of every partial parse on the agenda at termination, and

- the cost of the solution is no greater than the cost of every partial parse pruned. Derivations are pruned when they are not inserted into the agenda because it was full.

The proof is as follows: all solutions are either popped, led to by a partial parse in the agenda, or led to by a partial parse that was pruned. Any solution popped was either the solution returned, or it didn't have lower cost. From the conditions above, any solution led to by a partial parse in the agenda or by a partial parse that was pruned must have cost no less than that partial parse, which is no less than the cost of the solution returned. Hence, no solution has lower cost than the one returned.

3.4 Related work

It is common to see time-limited search tasks in which a complete solution is required, the setting for which **anytime** search algorithms are designed. Examples of anytime search algorithms include ARA* (Likhachev et al., 2004), ABULB (Furcy, 2004), and beam-stack search (Zhou and Hansen, 2005). However, we have not seen an approach identical to greedy completion proposed in the literature on search, speech, parsing, or machine translation. Greedy completion can be viewed as a modification to greedy search that adds cost-sensitive backtracking to make it a complete search algorithm.

Depth-first branch-and-bound is the closest approach we have found. Greedy completion is a variant of depth-first branch-and-bound (Korf, 1998) for cost-minimizing search. The difference is that after a solution is found or search cannot proceed, depth-first branch-and-bound backtracks chronologically, i.e. to the deepest partial parse in the agenda. Greedy completion backtracks to the minimum cost partial parse in the agenda. For this reason, depth-first branch-and-bound has worst-case linear space complexity, i.e. $O(\text{depth} \cdot \text{branching factor})$. Space-complexity is worst-case exponential for greedy completion, as well as for best-first search.

More distantly related is beam-stack search (Zhou and Hansen, 2005), an anytime search algorithm that adds backtracking to beam-search. This modification converts beam-search into a complete search algorithm, one that can recover the optimal solution given sufficient time. It is like breadth-first branch-and-bound, except that the beam width constrains how many partial parses are stored at each layer. Jelinek et al. (1994) and Magerman (1994, 1995) use a similar search technique, which they call multistack decoding. It too is a modification of beam-search to allow backtracking, which means that partial parses are not permanently pruned. This modification is included to avoid incompleteness.

Greedy completion is also similar to limited discrepancy search (Harvey and Ginsberg, 1995) generalized to arbitrary graphs (Furcy, 2004; Furcy and Koenig, 2005). A discrepancy occurs when, at some state, the search algorithm explores a non-minimum-cost expansion, i.e. is not greedy for a single decision. In limited discrepancy search, at the nth iteration, all paths with n discrepancies are explored. In greedy completion search, at the nth iteration,

a single path is pursued that has the smallest new discrepancy from any previous path explored.

Greedy completion is perhaps conceptually similar to the k-best parsing algorithm of Huang and Chiang (2005). Their algorithm first finds the 1-best solution, and then determines the remaining solutions using lazy backtracking.

Greedy completion should not be confused with "best-deepest best-first-search," which was proposed by Pemberton and Korf (1994a,b) for a restricted real-time search problem. In real-time search, after a pre-determined number of partial parses have been scored, a decision must be made: the problem solver must choose one of the children (consequents) of the current root node (initial partial parse) and make this child the new root node. The authors propose a real-time search strategy called "best-deepest best-first-search," in which agenda priority is negative cost, i.e. partial parses are explored according to a best-first strategy. At decision time, the problem solver moves towards the deepest partial parse explored, tiebreaking in favor of the lowest cost partial parse.

❧ Fourth Chapter ❧

Learning

> It especially annoys me when racists are accused of "discrimination." The ability to discriminate is a precious facility; by judging all members of one "race" to be the same, the racist precisely shows himself incapable of discrimination.
>
> Christopher Hitchens

4.1 Introduction

We introduce our basic approach to inducing the cost function. The **cost function** measures the compatibility between the input and some output. We define a family of cost functions parameterized by Θ, a real-valued **parameter** vector, which has one element for each feature $f \in F$. F is a finite, perhaps high-dimensional, feature space that indexes the entries of the feature vector. During training, our goal is to choose a cost function that has the best generalization, i.e. that maximizes the expected value of the evaluation measure (a.k.a. **true objective function**) on unseen inputs. This process of choosing Θ is also known as **parameter estimation**. Our evaluation measure for constituent parsing is the PARSEVAL F_1 (Black et al., 1991), which is based on the number of non-terminal items in the parser's output that match those in the gold-standard parse. We are given a training set I. We use that training set and our prior knowledge about the problem (the **prior**) to assess how well each cost function will generalize. This estimate of generalization is called the **expected risk** function (**risk**, for short, or **loss function**). Training is an optimization procedure for finding the cost function with minimum risk. The risk is also referred to as the **objective** function, because it is the objective of the optimization procedure used during training. In this chapter, we use the term "objective" when we do not need to distinguish between the true objective, e.g. 0-1 error in binary classification, and the surrogate objective, e.g. exponential loss. A **surrogate** is minimized during training

because it might be easier to optimize than the true objective and/or it might be more likely to generalize well because it can control for model complexity.

Expected risk

To ensure our risk function prefers cost functions that generalize well, it should balance the fit of the model over the training set against model complexity. The risk function R_Θ thus typically includes two terms: the **empirical** (or **unpenalized**) risk L_Θ, which is computed over the training set, and a **regularization** (or **penalty**) term Ω_Θ based on the prior:

$$R_\Theta(I) = L_\Theta(I) + \lambda\Omega_\Theta \tag{4.1}$$

For brevity, we may write $R_\Theta(I)$ and $L_\Theta(I)$ as R_Θ and L_Θ, respectively. The strength of the regularizer is controlled by λ, the regularization penalty factor. For a given choice of λ, the training procedure optimizes Θ to minimize the expected risk R_Θ over training set I. The regularization penalty factor is typically chosen by **cross-validation**, which maximizes some objective function over held-out development data. Ideally we should optimize the true objective during cross-validation, but if that is infeasible then we can optimize some surrogate objective.

Empirical risk

Ideally, the empirical risk is identical to the true objective function. In practice, minimizing the empirical risk might be difficult, so a common solution is to minimize a surrogate objective. For example, in the case of binary classification, minimizing the zero-one error is a combinatorial optimization problem that is known to be NP-hard. The exponential loss and logistic loss are smooth, convex loss functions that bound the zero-one error from above, and are commonly used surrogates for the zero-one error. Similarly, directly minimizing the PARSEVAL F_1 is quite tricky. Jansche (2005) shows how to maximize the expected F_1-measure of logistic regression models, but it is unknown to us how his method can be applied to entire parse trees. We instead choose a different surrogate to minimize, the statewise error. The **statewise error** is the likelihood that, at some correct state, the minimum cost inference is incorrect. Minimizing the statewise error is also NP-hard, but we could define a statewise margin and minimize the statewise loss. For example, let $\mathbf{S}$ be a sample of correct states to be used for training. Then we could define the samplewise unpenalized loss as:

$$L_\Theta(I) = \sum_{S \in \mathbf{S}} l(\mu_\Theta(\mathcal{I}(S))) \tag{4.2}$$

$\mathcal{I}(S)$ are all candidate inferences at S, as defined on on page 19. Under our logic (Section 2.2), $\mathcal{I}(S)$ are all candidate inferences that add a single item to state S. $\mu_\Theta(\mathcal{I}(S))$ is some margin over those inferences. For example, the statewise margin might be the highest

confidence assigned to any correct inference minus the highest confidence assigned to any incorrect inference. l transforms the margin into a loss.

There is a practical difficulty in optimizing the statewise loss in Equation 4.2. Namely, a state can contain all sorts of inferences, so we cannot partition the inferences and then parallelize training by optimizing each partition piece-wise. We will explain this optimization in Section 4.2, and later return to it in Section 5.1. We opt to approximate the statewise loss using a biased version of the **examplewise loss**:

$$L_\Theta(I) = \sum_{i \in I} b(i) \cdot l(\mu_\Theta(i)) \tag{4.3}$$

In this equation, l is a margin-based examplewise loss function, which allows us to optimize it piece-wise. Some training examples might be more important than others, so each is given a **bias** $b(i) \in \mathbb{R}^+$. This approximation is based on the intuition that, instead of learning to rank the inferences at each state as in Equation 4.2, we can learn the harder problem of classifying each inference in isolation into correct and incorrect, irrespective of the other inferences at the same state.

PRIORS

A probabilistic interpretation of the penalty term $\lambda \cdot \Omega_\Theta$ in Equation 4.1 is:

$$\lambda \cdot \Omega_\Theta = -\log \Pr(\Theta) \tag{4.4}$$

$$= -\log \left(\prod_{f \in F} \Pr(\Theta_f) \right) \tag{4.5}$$

$$= -\sum_{f \in F} \log \Pr(\Theta_f) \tag{4.6}$$

One common regularization approach is to apply a Gaussian prior distribution with mean zero and variance σ^2 to the parameters:

$$\Pr(\Theta_f) = \frac{1}{\sigma \cdot \sqrt{2\pi}} \cdot \exp\left(-\frac{{\Theta_f}^2}{2\sigma^2} \right) \tag{4.7}$$

which gives us:

$$\lambda \cdot \Omega_\Theta = -\log \Pr(\Theta) = -\sum_{f \in F} \log \left(\frac{1}{\sigma \cdot \sqrt{2\pi}} \cdot \exp\left(-\frac{{\Theta_f}^2}{2\sigma^2} \right) \right) \tag{4.8}$$

$$= \sum_{f \in F} \frac{{\Theta_f}^2}{2\sigma^2} + \text{constant} \tag{4.9}$$

We can apply this prior by defining $\Omega_\Theta = \sum_{f \in F} {\Theta_f}^2$ and $\lambda = 1/(2\sigma^2)$. Since this prior

corresponds to adding an ℓ_2-norm penalty term to the risk, it is often referred to as ℓ_2-regularization.

Another regularization approach is to apply a Laplacian (or double exponential) prior distribution with mean zero and variance $2\tau^2$ to the parameters:

$$\Pr(\Theta_f) = \frac{1}{2\tau} \exp\left(-\frac{|\Theta_f|}{\tau}\right) \tag{4.10}$$

which gives us:

$$\lambda \cdot \Omega_\Theta = -\log \Pr(\Theta) = -\sum_{f \in F} \log\left(\frac{1}{2\tau} \exp\left(-\frac{|\Theta_f|}{\tau}\right)\right) \tag{4.11}$$

$$= \sum_{f \in F} \frac{|\Theta_f|}{\tau} + \text{constant} \tag{4.12}$$

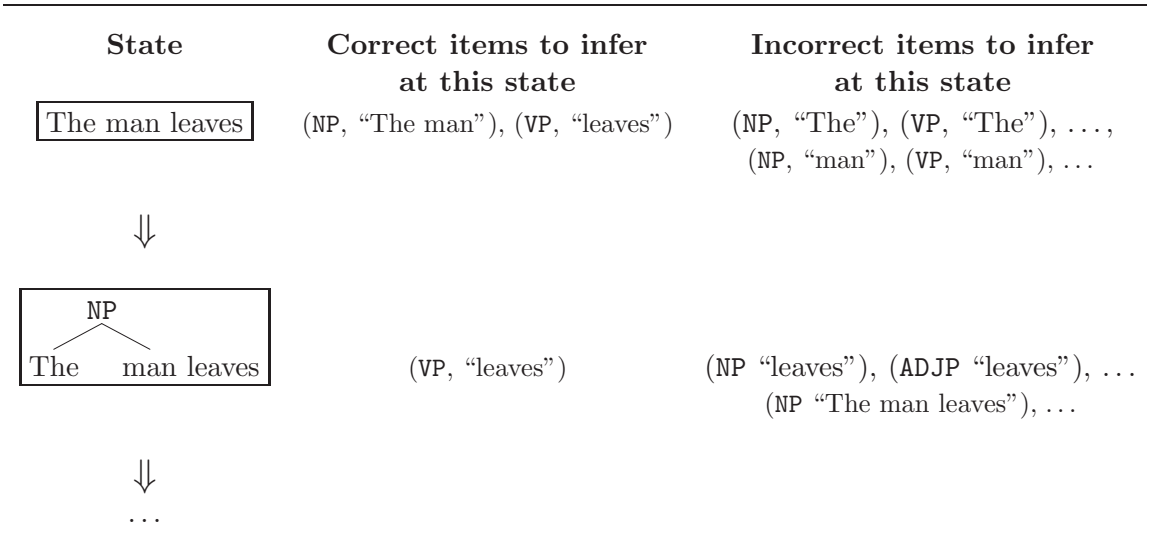
We can apply this prior by defining $\Omega_\Theta = \sum_{f \in F} |\Theta_f|$ and $\lambda = 1/\tau$. Since this prior corresponds to adding an ℓ_1-norm penalty term to the risk, it is often referred to as ℓ_1-regularization. Tibshirani (1996) uses this penalty in the context of least squares regression. Tibshirani (1996) calls his technique the "lasso" and notes that one of its advantages is that it leads to sparse solutions. We shall show why in Section 4.2.

The training set

Our training set I consists of candidate inferences from the parse trees in the training data. From each training inference $i \in I$ we generate the tuple $\langle X(i), y(i), b(i) \rangle$. $X(i)$ is a feature vector describing i, with each element in $\{0, 1\}$. We will use $X_f(i)$ to refer to the element of $X(i)$ that pertains to feature f. $y(i) = +1$ if i is correct, and $y(i) = -1$ if not. $b(i)$ is the **bias** of inference i.

The training data initially comes in the form of trees. These gold-standard trees are used to generate training examples, each of which is a candidate inference: Starting at the initial state, we randomly choose a complete path that leads to the (gold-standard) final state. In the deterministic setting there is only one correct path. If parsing proceeds non-deterministically then there might be multiple paths that lead to the same final parse, so we choose one randomly. All the candidate inferences that can possibly follow each state in this sequence become part of the training set. The vast majority of these inferences will lead to incorrect states, which makes them negative examples. Figure 4.1 illustrates example generation. The random correct path is shown in the left column. Each middle and right cell shows the items that can be correctly and incorrectly inferred at the state in the left cell, respectively. These inferences are the training examples generated by this tree. Observe that at the initial state, it is incorrect to infer any item spanning just "The" with a non-terminal label. In effect, there is an implicit *correct* inference spanning "The" with a dummy label meaning "not a constituent". This dummy label inference does not correspond

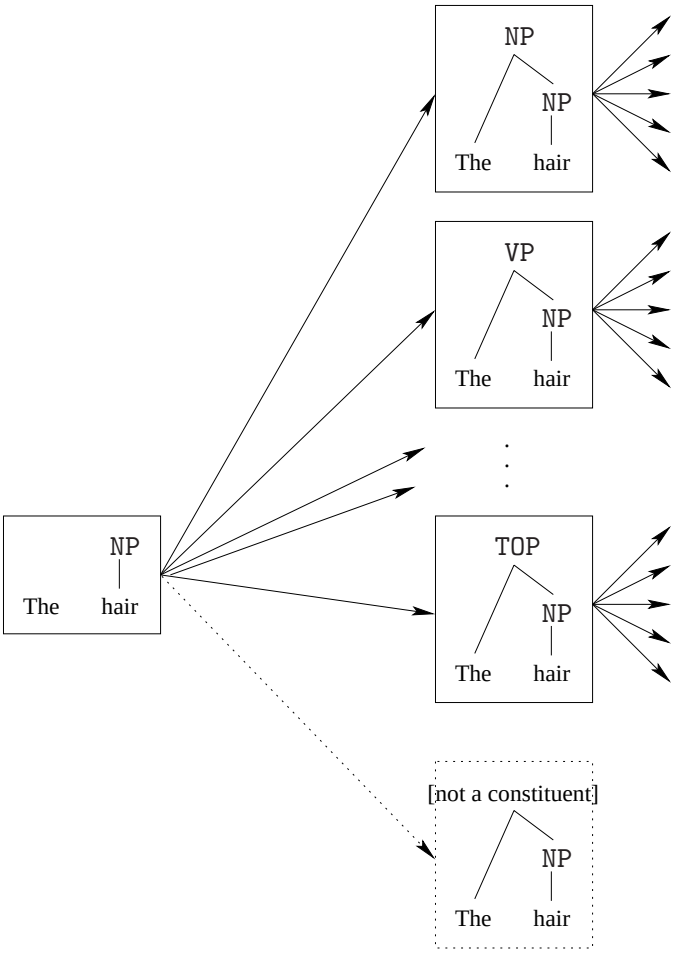
Figure 4.1: An example of example generation, where the logic requires that items are inferred non-deterministically bottom-up.



to an outgoing transition (inference) to a subsequent state, only constituent label inferences do. Predicting this dummy inference "dampens" the mass received by outgoing transitions, as recommended by Lafferty et al. (2001). So if the parser ever transitions to an incorrect state, which would normally be where label bias would arise, the model can predict "not a constituent" for every span and have very low outgoing transition probability. For this reason, we conjecture that our learning approach is not subject to label bias. Another example of where label bias might occur is given in Figure 4.2. In this example, we are at the incorrect state where the word "hair" has been labeled as a noun phrase (NP). This may have occurred with relatively high probability because the NP decision was made without considering any of the context. In isolation, "hair" could very well be a noun phrase. The parser now might have to decide between different constituent labels spanning "The hair." However, no decision is correct besides backtracking. A locally-normalized model would have to distribute the high probability mass among all subsequent (solid line) states, despite the fact that it now knows that they are all low probability. Our model considers each decision independently, so it would not have to assign mass to any of the subsequent states. Voting against all subsequent (solid line) states is equivalent to transferring mass to the dashed "not a constituent" state, which does not actually lead to any final state.

An advantage of this method of generating training examples is that it does not require a working inference engine and can be run prior to any training. A disadvantage of this approach is that it does not teach the model to recover from mistakes. Our training set is identical to the inferences that would be scored during search using an oracle cost function.

Figure 4.2: An example state at which label bias could occur. A locally-normalized model would have to distribute all probability mass from the state on the left to the subsequent (solid line) states on the right.



The oracle cost function $\hat{c}$ is:

$$\hat{c}(i) = \begin{cases} \infty & \text{if } y(i) = -1. \\ 0 & \text{if } y(i) = +1. \end{cases} \tag{4.13}$$

If we wanted to improve accuracy by teaching the parser to recover from its mistakes, we could use inference during example generation and iterate training. With a deterministic logic, using the oracle cost function to generate training examples is similar to the first iteration of SEARN (Daumé III et al., 2005, 2006). See Section 7.2 for more details.

Turian and Melamed (2005) observed that uniform example biases $b(i)$ produced lower accuracy as training progressed, because the induced classifiers minimized the error per *example*. To minimize the error per *state*, we assign every training state equal value and share half the value uniformly among the negative examples for the examples generated from that state and the other half uniformly among the positive examples.

Scoring function

We induce a hypothesis $h_\Theta(i)$, which is a real-valued inference scoring function. In the present work, h_Θ is a linear model (Collins, 2004) parameterized by vector Θ:

$$h_\Theta(i) = \Theta \cdot X(i) = \sum_{f \in F} \Theta_f \cdot X_f(i) \tag{4.14}$$

The sign of $h_\Theta(i)$ predicts the y-value of i and the magnitude gives the confidence in this prediction.

Equation 4.3 introduced the biased examplewise loss used during training. There are many possible choices for the margin-based per-sample loss function l. In the present work we use the log-loss (e.g Collins et al., 2002):

$$l(\mu_\Theta(i)) = \ln(1 + \exp(-\mu_\Theta(i))) \tag{4.15}$$

where $\mu_\Theta(i)$ is the **margin** of inference i:

$$\mu_\Theta(i) = y(i) \cdot h_\Theta(i) \tag{4.16}$$

Inference cost $c_\Theta(i)$ in Equation 2.12 is $l(\mu_\Theta(i))$ computed using $y(i) = +1$, i.e.:

$$c_\Theta(i) = l(h_\Theta(i)) = \ln(1 + \exp(-h_\Theta(i))) \tag{4.17}$$

So the inference cost $c_\Theta(i)$ is the risk of treating i as a correct inference.

We pause here to situate our model in the literature. Haffner et al. (2005) study maximum entropy models for multiclass prediction under a variety of conditions:

- estimating joint vs. conditional vs. class-conditional distributions

  and,

- treating output classes as exclusive vs. assuming independence between them.

If one makes an independence assumption between each class's conditional probability, and estimates them independently, then no normalization is necessary and estimators for different classes can be induced in parallel. This specific model is a multiclass conditional maxent model, reduced to binary using one-vs-all, and *assuming that each binary response is predicted independently*. This conditional maxent model for binary classification is equivalent to logistic regression. Jansche (2005) (citing Ratnaparkhi (1998, pg. 27)) also draws this connection, pointing out that log-linear models for binary classification are equivalent to logistic regression. Modulo choice of prior and our use of bias terms, we believe these models are equivalent to ours. In other words, all these models make the assumption that, for each span, each label's probability is estimated independently.[1] We speculate that this independence assumption allows conditional log-linear models to avoid label bias. Ratnaparkhi (1997, 1999) does not assume independence in his multi-class reduction, and hence normalizes over the inferences at each state. For this reason, label bias occurs because the mass that enters any state must be distributed among all outgoing states, with no opportunity for dampening.

4.2 Minimizing the risk

We minimize the risk R_Θ using a form of forward stagewise additive modeling, a procedure Guyon and Elisseeff (2003) refer to as embedded feature selection. At each iteration in training, we pick one or more parameters of the model to adjust (**feature selection**), then adjust these parameters (**model update**).

Related work

In this section, we discuss the most closely related works on fitting additive models, and on optimization using sequential updates. For a survey of the literature on boosting, see Meir and Rätsch (2003). When we use the term **base hypothesis** or **weak learner**, these can be arbitrary functions over the features in the inputs. In our setting, we impose the restriction that each base hypothesis is a single feature, and each combined hypothesis is a linear combination over the features (see Equation 4.14). Note that these features are not merely atomic features over the input attributes, as we will explain later in Section 4.2.

[1] This assumption is orthogonal to assuming that every feature is class-dependent. That assumption, in conjunction with lack of normalization, allows risk minimization to be parallelized according to class (Section 5.1).

Schapire and Singer (1999)

This work extends AdaBoost to combinations of real-valued functions. They cast AdaBoost as a method for minimizing the exponential loss over the training sample using a greedy coordinate-wise search.

They show how to choose the sequential parameter updates for different types of base hypotheses. In the general case, each parameter update can be found using a line-search. If the range of each base hypothesis is in $\{-1, 0, +1\}$, the parameter update can be derived analytically. Using this parameter update, they show how to induce decision trees with a new splitting criterion as opposed to, say, the Gini index or an entropic function. Using this new splitting criterion, greedily chosen splits directly minimize the exponential loss.

Because the models are unregularized, Schapire and Singer (1999) propose an ad-hoc technique for smoothing predictions that limits their magnitude. We note that, even though they normalize example weights at the beginning of every iteration, this only affects the strength of the smoothing term, and would otherwise not affect the induced model. We skip normalization in our approach because it is expensive and may interfere with the regularizer.

Schapire and Singer (1999) also provide some generalization bounds, as well as generalizing their training regime to multiclass, multilabel, and ranking scenarios.

Friedman et al. (2000)

Much of this work is devoted to understanding AdaBoost and situating it in statistics. This work examines the then relatively poorly understood boosting technique, and explains it in terms of well known statistical principles: additive modeling and maximum likelihood. It also provides a good history of the early work on boosting. The authors re-derive AdaBoost as a forward stagewise method for fitting an additive model using an approximation of the binomial log-likelihood. They provide several motivations for the exponential loss (**exp-loss**) used by AdaBoost, including:

- It is a differentiable upper-bound on the 0-1 loss (Schapire and Singer, 1999).
- The exponential loss is minimized at:

$$h_\Theta(i) = \frac{1}{2} \log \frac{\Pr(y(i) = +1|X(i))}{\Pr(y(i) = -1|X(i))} \tag{4.18}$$

  where:

$$\Pr(y(i) = +1|X(i)) = \frac{1}{1 + \exp(-2 \cdot h_\Theta(i))} \tag{4.19}$$

  and

$$\Pr(y(i) = -1|X(i)) = 1 - \Pr(y(i) = +1|X(i)) \tag{4.20}$$

> The binomial log-likelihood (which is the log-loss) has the same population minimizer as the exp-loss. The population minimizer is the choice of $h_\Theta(i)$ that minimizes the empirical risk. Also, around $h_\Theta(i) = 0$, the exp-loss and log-loss are equivalent up to the second order in the Taylor expansion.

They then propose a new boosting method, LogitBoost, for fitting additive logistic regression models by direct optimization of the binomial log-likelihood. This algorithm is a greedy forward stagewise approach, and it uses Newton steps. An interesting aspect of the algorithm is that, during the example reweighting stage at the beginning of each iteration, the example weight $w_\Theta(i)$ for inference i is:

$$w_\Theta(i) = \Pr(y(i) = +1|X(i)) \cdot \Pr(y(i) = -1|X(i)) \tag{4.21}$$

In other words, the weight of an example is independent of its y-value. Examples receive high weight if they are close to the decision boundary, i.e. have low magnitude confidence.

Mason et al. (1999, 2000)

These works propose a general class of gradient descent algorithms called AnyBoost. AnyBoost is a boosting technique for finding linear combinations of weak learners to minimize arbitrary risk functionals. The risk functional need not depend on the margin.

AnyBoost works by iteratively picking the weak learner that maximizes the downward decrease in the risk, i.e. its negative gradient. The authors present variants in which an ℓ_1 constraint is imposed on parameter values after each boosting iteration, as well as a variant with an ℓ_2 constraint. Little guidance is provided in constructing weak learners.[2]

MarginBoost is a special case of AnyBoost under a set of commonly made assumptions. In particular, MarginBoost assumes that the task is binary classification and that the risk function in MarginBoost must be the sample average of some risk function of the margin.[3] Under these assumptions, we can define a weight for each example based on the current model parameters. The next boosting iteration proceeds by finding the weak learner that minimizes the weighted error, which means that many off-the-shelf learner methods can be used for creating the weak learner. The authors show that, with a convex risk function such as the exp-loss and log-loss and a particular choice of parameter update, if the weak learner minimizes the weighted error then the MarginBoost algorithm converges to the global minimum of the risk. MarginBoost includes many popular voting methods, including AdaBoost (Schapire and Singer, 1999) and LogitBoost (Friedman et al., 2000).

[2] Note that their pseudo-code might incorrectly choose certain parameter values if the same base hypothesis is picked in more than one iteration.

[3] Modulo the bias term and not dividing through by the number of instances, Equation 4.3 is based on a risk function of the margin. However, using a risk function based on only the margin precludes including a model complexity penalty directly in the risk function (as we do in Equation 4.1).

By treating the margin-based risk function as an upper-bound on the 0-1 error, the authors derive a theoretically well-motivated risk function for MarginBoost:

$$l(\mu_\Theta(i)) = 1 - \tanh(\lambda \cdot \mu_\Theta(i)) \tag{4.22}$$

where λ is a parameter. This risk is non-convex and approaches the 0-1 risk for high values of λ. DOOM II is a variant of MarginBoost.ℓ_1 that uses this risk function. The authors find that this algorithm is more accurate than AdaBoost, especially when there is label noise present.

Perkins et al. (2003)

This work focuses on the problem of using gradient descent in function space for *regularized* risk minimization, i.e. formulations like ours (Equation 4.1) in which a model complexity term is included explicitly in the risk. They discuss the motivation for such risk functions, namely that minimizing an empirical, i.e. unregularized, risk can lead to overfitting. Like our work, they focus on the binomial negative log-likelihood (a.k.a. log-loss) empirical risk. They also discuss several common regularizers, including ℓ_0, ℓ_1, and ℓ_2, describing the history of these regularizers, as well as their various properties and motivations.

They propose the so-called grafting algorithm for minimizing regularized risks. At each iteration, a single feature is picked for parameter update, based on the gradient of the regularized risk with respect to this feature's parameter. This feature is added to the working set, and the parameters of all features in the working set are optimized using a parallel update technique. Optimization stops when the regularized risk canned be decreased any further. They describe grafting for linear models, as well as multi-layer perceptrons (MLPs). For MLP grafting, each iteration adds either a new hidden unit or a new input connection to an existing hidden unit.

For linear models, grafting is essentially a boosting algorithm that chooses a single parameter to update based on the gradient of the regularized risk. The main difference is that the default grafting procedure performs parallel updates over all working parameters, whereas boosting uses sequential updates, i.e. only updating the single parameter chosen.

Riezler and Vasserman (2004) propose a k-best variant of grafting for minimizing the ℓ_1-regularized log-loss. At each iteration, the k highest gradient features are added to the working set. Parallel optimization is performed using a conjugate gradient routine. The ℓ_1 penalty might drive certain parameters to zero, especially if the k-best features overlap or, worse yet, are redundant. For this reason, Riezler and Vasserman (2004) avoid numerical instability by explicitly pruning these features from the working set.

Feature selection

We want to choose parameters that allow us to decrease the risk quickly. We define the **gain** of a feature f as:

$$\begin{aligned} G_\Theta(I; f) = \max(&\text{decrease in risk as we increase } f\text{'s parameter value,} \\ &\text{decrease in risk as we decrease } f\text{'s parameter value,} \\ &\text{decrease in risk if we leave } f\text{'s parameter value unchanged}) \end{aligned} \tag{4.23}$$

Ideally, we could pick the parameters that will give the largest decrease in the risk when we adjust them. For example:

- Schapire and Singer (1999) provide a decision-tree splitting criterion that directly minimizes the unregularized exponential loss.
- Haffner et al. (2005) provide an analytic update that, for ℓ_1-regularized conditional maxent models with binary features, is close to the optimum.

However, with most risk functions, it is not possible to analytically determine the amount by which adjusting some parameter would reduce the risk. An alternative approach in this case is the pick the features with the steepest gradients, i.e. we perform gradient descent in function space (Mason et al., 1999, 2000; Perkins et al., 2003). To perform gradient descent in function space, we specifically define $G_\Theta(I; f)$ as follows:

$$\begin{aligned} G_\Theta(I; f) = \max(&\lim_{\epsilon \to 0^+} \frac{\partial\, -R_\Theta}{\partial \Theta_f} (\Theta_f + \epsilon), \\ &\lim_{\epsilon \to 0^-} \frac{\partial\, -R_\Theta}{\partial\, -\Theta_f} (\Theta_f + \epsilon), \\ &0) \end{aligned} \tag{4.24}$$

$$\begin{aligned} = \max(&-\lim_{\epsilon \to 0^+} \frac{\partial R_\Theta}{\partial \Theta_f} (\Theta_f + \epsilon), \\ &\lim_{\epsilon \to 0^-} \frac{\partial R_\Theta}{\partial \Theta_f} (\Theta_f + \epsilon), \\ &0). \end{aligned} \tag{4.25}$$

The limits are taken from above and below, respectively. Even though risk is continuous, it might not be continuously differentiable, so its gradient might not be defined for all values of Θ_f. For example, the ℓ_1-penalty is continuous but has a gradient discontinuity at $\Theta_f = 0$. Equations 4.23–4.25 determine the gain function using three separate cases. This analysis technique allows us to determine the gain for any continuous function, regardless of whether it is continuously differentiable or not. We are not aware of previous work on gradient descent in function space that has used this analysis technique. Mason et al.

(1999) and Mason et al. (2000), for example, work only with continuously differentiable cost functions. Perkins et al. (2003) and Riezler and Vasserman (2004) present ℓ_1-penalized gain values for zero-valued parameters, but without any derivation. Our analysis technique will be used to derive their gain values, as well as the gain for parameters that are non-zero. If the gradient is continuous, i.e. if:

$$\lim_{\epsilon \to 0^+} \frac{\partial R_\Theta}{\partial \Theta_f}(\Theta_f + \epsilon) = \lim_{\epsilon \to 0^-} \frac{\partial R_\Theta}{\partial \Theta_f}(\Theta_f + \epsilon) = \frac{\partial R_\Theta}{\partial \Theta_f}(\Theta_f), \tag{4.26}$$

then we have:

$$G_\Theta(I; f) = \max\left(-\frac{\partial R_\Theta}{\partial \Theta_f}(\Theta_f),\ \frac{\partial R_\Theta}{\partial \Theta_f}(\Theta_f),\ 0\right) \tag{4.27}$$

$$= \left|\frac{\partial R_\Theta}{\partial \Theta_f}(\Theta_f)\right|. \tag{4.28}$$

In other words, if the gradient of the risk is continuous and the gradient is not zero, then there must be some direction that decreases the risk. The gain of the feature f is the steepness of the gradient in the descent direction.

The gradient of the log-loss $\frac{\partial L_\Theta}{\partial \Theta_f}(\Theta_f)$ is continuous (proof omitted), so we can rewrite Equation 4.25 as:

$$\begin{aligned} G_\Theta(I; f) = \max(&-\lim_{\epsilon \to 0^+} \frac{\partial (L_\Theta + \lambda \cdot \Omega_\Theta)}{\partial \Theta_f}(\Theta_f + \epsilon), \\ &\lim_{\epsilon \to 0^-} \frac{\partial (L_\Theta + \lambda \cdot \Omega_\Theta)}{\partial \Theta_f}(\Theta_f + \epsilon), \\ &0). \end{aligned} \tag{4.29}$$

$$\begin{aligned} = \max(&-\frac{\partial L_\Theta}{\partial \Theta_f} - \lambda \cdot \lim_{\epsilon \to 0^+} \frac{\partial \Omega_\Theta}{\partial \Theta_f}(\Theta_f + \epsilon), \\ &\frac{\partial L_\Theta}{\partial \Theta_f} + \lambda \cdot \lim_{\epsilon \to 0^-} \frac{\partial \Omega_\Theta}{\partial \Theta_f}(\Theta_f + \epsilon), \\ &0). \end{aligned} \tag{4.30}$$

To determine $\frac{\partial L_\Theta}{\partial \Theta_f}$, the gradient of the unpenalized loss L_Θ with respect to the parameter Θ_f of feature f, we have:

$$\frac{\partial L_\Theta(I)}{\partial \Theta_f} = \sum_{i \in I} \frac{\partial l(i)}{\partial \mu_\Theta(i)} \cdot \frac{\partial \mu_\Theta(i)}{\partial \Theta_f} \tag{4.31}$$

where:

$$\frac{\partial \mu_\Theta(i)}{\partial \Theta_f} = y(i) \cdot X_f(i) \tag{4.32}$$

Using Equation 4.15, we define the **weight** of an example i under the current model as the rate at which the examplewise loss decreases as the margin of i increases:

$$w_\Theta(i) = \frac{\partial(-l(i))}{\partial \mu_\Theta(i)} = \frac{1}{1 + \exp(\mu_\Theta(i))} \tag{4.33}$$

Recall that $X_f(i)$ is either 0 or 1. Combining Equations 4.31–4.33 gives:

$$\frac{\partial L_\Theta(I)}{\partial \Theta_f} = -\sum_{\substack{i \in I \\ X_f(i)=1}} y(i) \cdot w_\Theta(i) \tag{4.34}$$

To determine the gradient of the penalty term, we consider several cases:

- If we are using the ℓ_1 penalty ($\Omega_\Theta = \sum_{f \in F} |\Theta_f|$) and $\Theta_f = 0$, then we are at the gradient discontinuity and we have:

$$\lim_{\epsilon \to 0^+} \frac{\partial \Omega_\Theta}{\partial \Theta_f} (\Theta_f + \epsilon) = +1 \tag{4.35}$$

$$\lim_{\epsilon \to 0^-} \frac{\partial \Omega_\Theta}{\partial \Theta_f} (\Theta_f + \epsilon) = -1 \tag{4.36}$$

  which gives us:

$$G_\Theta(I; f) = \max\left(-\frac{\partial L_\Theta}{\partial \Theta_f} - \lambda,\ \frac{\partial L_\Theta}{\partial \Theta_f} - \lambda,\ 0\right) \tag{4.37}$$

$$= \max\left(\left|\frac{\partial L_\Theta}{\partial \Theta_f}\right| - \lambda,\ 0\right). \tag{4.38}$$

  This equation is equivalent to the gain presented, without derivation, by Perkins et al. (2003) and Riezler and Vasserman (2004). To the best of our knowledge, the preceding derivation is new. Observe that unless the magnitude of the gradient of the empirical loss $|\partial L_\Theta(I)/\partial \Theta_f|$ exceeds the penalty term λ, the gain is zero and the risk increases as we adjust parameter Θ_f away from zero in either direction. In other words, the parameter value is trapped in a "corner" of the risk. In this manner the polyhedral structure of the ℓ_1 norm tends to keep the model sparse (Riezler and Vasserman, 2004). Dudík et al. (2007) offer another perspective, pointing out that ℓ_1 regularization is "truly sparse": if some feature's parameter value is zero when the risk is minimized, then the optimal parameter value will remain at zero even under

slight perturbations of the feature's expected value and of the regularization penalty. However, if the gain is non-zero, $G_\Theta(I; f)$ is the magnitude of the gradient of the risk as we adjust Θ_f in the direction that reduces R_Θ.

- If we are using the ℓ_1 penalty and $\Theta_f \neq 0$, then we have:

$$\frac{\partial \Omega_\Theta}{\partial \Theta_f}(\Theta_f) = \text{sign}(\Theta_f) \tag{4.39}$$

which gives us:

$$G_\Theta(I; f) = \left| \frac{\partial L_\Theta}{\partial \Theta_f} + \lambda \cdot \text{sign}(\Theta_f) \right| \tag{4.40}$$

- If we are using the ℓ_2 penalty ($\Omega_\Theta = \sum_{f \in F} {\Theta_f}^2$), then we have:

$$\frac{\partial \Omega_\Theta}{\partial \Theta_f}(\Theta_f) = 2 \cdot \Theta_f \tag{4.41}$$

which gives us:

$$G_\Theta(I; f) = \left| \frac{\partial L_\Theta}{\partial \Theta_f} + \lambda \cdot 2 \cdot \Theta_f \right| \tag{4.42}$$

The ℓ_2-regularization term disappears when $\Theta_f = 0$, and so—for unused features—ℓ_2-regularized feature selection is indistinguishable from unregularized feature selection. The only difference is that the ℓ_2 penalty term reduces the magnitude of the optimal parameter setting for each feature. This is why ℓ_2-regularization typically leads to models where many features are active (non-zero).

Compound feature selection

Although h_Θ is just a linear discriminant, it can nonetheless learn effectively if feature space F is high-dimensional. Features encode information about the inference in question. *A priori*, we define only a set A of simple atomic features (sometimes also called **attributes** or **primitive** features), specified in Section 6.3.

Feature **construction** or **induction** methods are learning methods in which we induce a more powerful machine than a linear discriminant over just the attributes, and the power of the machine can be data-dependent. This is in comparison to non-linear methods whose non-linearity is chosen *a priori*, such as is typical in choosing the network structure for neural networks or the kernel for SVMs. Choosing network structure or a kernel to optimize the objective typically requires significant human effort, and the goal of automatic feature induction techniques is to reduce this effort.

Our learner induces **compound** features, each of which is a conjunction of possibly negated atomic features. Each atomic feature can have one of three values (yes/no/don't

care), so the compound feature space F has size $3^{|A|}$, exponential in the number of atomic features. Each feature selection iteration selects a set of domain-partitioning features. The domain is the **inference space**, i.e. the space of all possible inferences. Specifically, a set of features $\tilde{F}$ **partition** the inference space iff:

- $\tilde{F}$ **covers** $\mathcal{I}$:

$$\forall i \in \mathcal{I}, \exists f \in \tilde{F} \text{ s.t. } X_f(i) = 1 \tag{4.43}$$

and

- all pairs of features in $\tilde{F}$ are **mutually-exclusive** (or **non-overlapping**):

$$\forall i \in \mathcal{I}, f, f' \in \tilde{F}, \text{ if } X_f(i) = 1 \text{ and } X_{f'}(i) = 1 \text{ then } f = f'. \tag{4.44}$$

In other words, for any inference i, there is a unique feature f in the partition $\tilde{F}$ such that $X_f(i) = 1$, and $X_{f'}(i) = 0$ for all other features. In this case, we say that i **falls** in f's partition, and we write I_f to indicate all inferences that fall in f.

One way to choose a set of domain-partitioning compound features is through greedy splitting of the inference space. This is the approach taken by decision trees, for some particular splitting criterion. Since we wish to pick splits that allow us to reduce risk, our splitting criterion uses the gain function. In particular, assume that we have two different two-feature partitions of some subspace of the inference space. The features f_1 and f_2 are one partition, and $f_1{}'$ and $f_2{}'$ are the other. Without loss of generality, we assume:

$$G_\Theta(f_1) \geq G_\Theta(f_2), \tag{4.45}$$

$$G_\Theta(f_1{}') \geq G_\Theta(f_2{}'), \tag{4.46}$$

and

$$G_\Theta(f_1) \geq G_\Theta(f_1{}'). \tag{4.47}$$

If $G_\Theta(f_2) \geq G_\Theta(f_2{}')$ then the first partition is preferable to the second because its gains are no worse. For example, if we have:

$$G_\Theta(f_1) = 5,\ G_\Theta(f_2) = 3 \tag{4.48}$$

and

$$G_\Theta(f_1{}') = 4,\ G_\Theta(f_2{}') = 2 \tag{4.49}$$

then we should prefer partitioning using f_1 and f_2. However, consider the possibility that $G_\Theta(f_2) < G_\Theta(f_2{}')$. The choice of partitioning is less clear if:

$$G_\Theta(f_1) = 5,\ G_\Theta(f_2) = 2 \tag{4.50}$$

and

$$G_\Theta(f_1') = 4,\ G_\Theta(f_2') = 3 \tag{4.51}$$

One possibility is to treat the goodness of each partition as, respectively, $\max(G_\Theta(f_1), G_\Theta(f_2))$ and $\max(G_\Theta(f_1'), G_\Theta(f_2'))$. Another possibility is to treat the goodness of each partition as $G_\Theta(f_1) + G_\Theta(f_2)$ and $G_\Theta(f_1') + G_\Theta(f_2')$, respectively. Addition worked better in preliminary experiments using ℓ_1 regularization, so we used it in our work. The work of Mason et al. (2000) and Mason et al. (1999), who studied a related problem, motivate this choice theoretically using a first-order Taylor expansion.

Model update

After we have selected one or more high-gain features, we update the model. **Parallel** update methods can adjust the parameter values of overlapping features to minimize the risk. **Stage-wise** or **sequential** update methods adjust the parameter values of *non*-overlapping features to minimize the risk, each of which can be optimized independently, e.g. using line search. Since domain-partitioning features are non-overlapping, we use sequential updates to choose parameter values.

Boosting regularized decision trees

To summarize, our approach to minimizing the risk is divided into feature selection and model update. Feature selection is performed using gradient descent in the compound feature space through a greedy splitting procedure. Model update is performed using a sequential update method.

Our specific implementation of this risk minimization approach is to boost an ensemble of confidence-rated decision trees. This boosted ensemble of confidence-rated decision trees represents Θ. We write $\Delta\Theta(t)$ to represent the parameter values chosen by tree t, and for ensemble T:

$$\Theta = \sum_{t \in T} \Delta\Theta(t) \tag{4.52}$$

Each internal node is split on an atomic feature. The path from the root to each node n in a decision tree corresponds to a *compound* feature f, and we write $\varphi(n) = f$. An example i percolates down to node n iff $X_{\varphi(n)} = 1$. Each leaf node n keeps track of delta-parameter value $\Delta\Theta_{\varphi(n)}(t)$. To score an example i using a decision tree, we percolate the example down to a leaf n and return confidence $\Delta\Theta_{\varphi(n)}(t)$. The score $h_\Theta(i)$ given to an example i by the whole ensemble is the sum of the confidences returned by all trees in the ensemble.

Listing 4.1 presents our training algorithm. At the beginning of training, the ensemble is empty, $\Theta = \mathbf{0}$, and λ is set to ∞. We grow the ensemble until the risk cannot be further reduced for the current choice of λ. So for some choice of penalty factor λ', our model is the ensemble up until when λ was decayed below λ'. We then relax the regularizer by

Listing 4.1: Outline of the training algorithm.

procedure TRAIN(I)
  ensemble $\leftarrow \emptyset$
  regularization parameter $\lambda \leftarrow \infty$
  while not converged **do**
    $g \leftarrow \max_{a \in A}(|\partial L_\Theta / \partial \Theta_\emptyset|, |\partial L_\Theta / \partial \Theta_a|, |\partial L_\Theta / \partial \Theta_{\neg a}|)$ ▷ Find the best unpenalized gain of any root split
    $\lambda \leftarrow \min(\lambda, \eta \cdot g)$ ▷ Maybe decay the penalty parameter so that training can progress
    $\Delta\Theta(t) \leftarrow \text{BUILDTREE}_\Theta(\lambda, I)$
    if $R_{\Theta + \Delta\Theta(t)} < R_\Theta + \epsilon$ **then** ▷ If we have reduced loss by some threshold
      $\Theta \leftarrow \Theta + \Delta\Theta(t)$ ▷ Then keep the tree and update the model
    else ▷ Otherwise, we have converged for this choice of λ
      $\lambda \leftarrow \eta \cdot \lambda$ ▷ Decay the penalty parameter

procedure BUILDTREE$_\Theta$(t, I)
  while some leaf in t can be split **do**
    split the leaf to maximize gain ▷ See Equation 4.54
  percolate every $i \in I$ to a leaf node
  for each leaf n in t **do**
    $\Delta\Theta_{\varphi(n)}(t) \leftarrow \arg\min_{\Delta\Theta_{\varphi(n)}(t)}(R_{\Theta + \Delta\Theta_{\varphi(n)}(t)}(I_{\varphi(n)}))$ ▷ Choose $\Delta\Theta_{\varphi(n)}(t)$ to minimize R_Θ using a line search
  return $\Delta\Theta(t)$

decreasing λ and continue training. We use the decay factor $\eta = 0.9$ as our **learning rate**. In this way, instead of choosing the best λ heuristically, we can optimize it during a single training run.

Each invocation of BUILDTREE has several steps. First, we choose some compound features that will allow us to decrease the risk function. We do this by building a decision tree, whose leaf node paths represent the chosen compound features. Second, we confidence-rate each leaf to minimize the risk over the examples that percolate down to that leaf. Finally, we append the decision tree to the ensemble and update parameter vector Θ accordingly. In this manner, compound feature selection is performed incrementally during training, as opposed to *a priori*.

The construction of each decision tree begins with a sole leaf node, the root node, which corresponds to a dummy "always true" feature $\emptyset$. By always true, we mean that $X_\emptyset(i) = 1$ for any example i. We recursively split leaf nodes by choosing the best atomic splitting

feature that will allow us to increase the gain. Specifically, we consider splitting each leaf node n using atomic feature $\hat{a}$, where

$$\hat{a} = \arg\max_{a \in A} \left[G_\Theta(I; f \wedge a) + G_\Theta(I; f \wedge \neg a) \right] \tag{4.53}$$

Splitting using $\hat{a}$ would create children nodes n_1 and n_2, with $\varphi(n_1) = f \wedge \hat{a}$ and $\varphi(n_2) = f \wedge \neg\hat{a}$. We split node n using $\hat{a}$ only if the total gain of these two children exceeds the gain of the unsplit node, i.e. if:

$$G_\Theta(I; f \wedge \hat{a}) + G_\Theta(I; f \wedge \neg\hat{a}) > G_\Theta(I; f) \tag{4.54}$$

Otherwise, n remains a leaf node of the decision tree, and $\Theta_{\varphi(n)}$ becomes one of the values to be optimized during the parameter update step.

Parameter updates are done sequentially on only the most recently added compound features, which correspond to the leaves of the new decision tree. After the entire tree is built, we percolate each example down to its appropriate leaf node. As indicated earlier, a convenient property of decision trees is that the leaves' compound features are mutually exclusive, so their parameters can be directly optimized independently of each other. We use a line search to choose for each leaf node n the parameter $\Theta_{\varphi(n)}$ that minimizes the risk over the examples in n.

❧ Fifth Chapter ❧

Optimizations and Approximations

> Premature optimization is the root of all evil.
>
> Donald E. Knuth

5.1 Parallelization

If we have some set of features that partition the inference space, the risk of each partition can be minimized independently, in isolation (as discussed in Section 4.2). For this reason, we can perform piece-wise optimization of the risk, asynchronously in parallel. We parallelized training by inducing 26 separate classifiers, one for each non-terminal label in the Penn Treebank. These classifiers each independently optimize the examples that infer items with a particular label. Consider the `NP`-classifier. It optimizes only those inferences for which the feature "the item being inferred is labeled `NP`" holds. Call this feature a. The `NP`-classifier only adjusts the parameters in Θ corresponding to compound features that include a. The drawback of this approach is that it introduces data fragmentation. Only parameters involving one of the aforementioned features can be adjusted. We cannot use features that detect trends across labels.

Ratnaparkhi (1999)'s parser had three phases: POS tagging, chunking, and constituent building. These were each trained in parallel, but Ratnaparkhi did not decompose the training of the multiclass inferences within each phase. Clark and Curran (2003) and Clark and Curran (2004) trained log-linear parsing models in parallel. However, their parallel implementations required message passing because their loss function includes a normalization term (partition function). Yamada and Matsumoto (2003) introduced the idea of asynchronous parallelization for training of discriminative parsers. They partitioned training examples by the POS tag of the leftmost child. Sagae and Lavie (2005) and Turian and Melamed (2005) simultaneously published results with parallelized discriminative training

for constituent parsing. Sagae and Lavie (2005) parallelized examples using the same splits as Yamada and Matsumoto (2003), whereas Turian and Melamed (2005) parallelized according to the label of the constituent being inferred. Wellington et al. (2006) and Turian et al. (2007) followed this approach and presented the first purely discriminative learning algorithm for translation with tree-structured models. One of the key factors in their success was parallelizing training into 40,000 different classifiers, one for each word in the source language.

Parallelization might not uniformly reduce training time because different classifiers train at different rates. However, parallelization uniformly reduces *memory* usage because each label's classifier trains only on inferences whose consequent item has that label.

Calibration

The previous section described how to train each label classifier in isolation. One key question remains: How do we **calibrate** the label classifiers with respect to F_1, to choose the overall parser?

In boosting an ensemble of decision trees, traditionally there is parameter T—the number of boosting iterations, i.e. the number of trees in the ensemble—which is typically chosen through cross-validation: T is chosen as the value that minimizes error on a held-out development set. One could imagine picking a single T, fixed across all labels. However, there is little reason to believe *a* priori that the a good T for one label will be good for every other label. Indeed, the different label classifiers train at different rates, so the relationship between generalization accuracy and T varies across labels.

Trying all possible combinations of stopping points for different classifiers would be prohibitive, as parsing is expensive. Instead, we tie the regularization penalty λ for each classifier. During cross-validation, we vary λ and parse all the different classifiers trained through this λ. We choose the value of λ that maximize F_1 on development data. The downside of this approach is that it implicitly assumes that each different label has the same noise level, and hence should have the same regularization penalty factor. This assumption is demonstrably false. Magerman (1995) found that annotation of `PRT` (particle) was inconsistent, and introduced the convention of converting `PRT` to `ADVP` (adverbial phrase) in preprocessing. The exploratory data analysis in Turian and Melamed (2006a) finds that, of all the common labels, annotation of `ADVP` and `ADJP` is particularly inconsistent. This discussion points towards another weakness of our approach: We apply the same strength prior to all features. Perkins et al. (2003) note that certain features might warrant stronger regularization than others. For example, they suggest that the constant offset term should be unregularized. This parameter corresponds to the root's "always true" feature. This feature is so common that its parameter value will be nearly identical regardless of whether we regularize it or not. Perkins et al. (2003) offer no more advice about how to choose different regularization terms for different features. It might be inappropriate to assume that compound features have the same weight prior. We could potentially penalize compound

features in proportion to their length. In this case, we would have no prior on the constant offset term.

There is another possible calibration technique that does not require tying the regularization penalty for different classifiers. Classifier penalty are chosen to optimize the statewise error, which is a surrogate of the PARSEVAL F_1. Computing the statewise error is significantly faster than computing the PARSEVAL F_1, which requires parsing.

To summarize, the reasons we can parallelize training are:

- Examples are generated prior to training, independent of the model. In comparison, generating training examples using a working parser is an inherently sequential step. However, parallel training could be occasionally punctuated by sequential example generation.

- During training itself, risk minimization can be asynchronously parallelized because the examplewise-loss can be decomposed and optimized piece-wise.

  and

- The regularization penalty factor λ is used at the end of training to calibrate the classifiers with respect to F_1. Since we tie using the regularization parameter, cross-validation cannot happen asynchronously.

We don't know how to parallelize training of ordinary boosting where the risk is not explicitly regularized and there is no penalty factor, because we don't know a good criterion on which to calibrate.

5.2 Sampling for faster feature selection

Building a decision tree using the entire example set I can be expensive. This is especially true if the training set and atomic feature space are large enough that the **atomic feature matrix** does not fit in memory. The atomic feature matrix caches the value of $X(i)$ for example training example i. If the atomic feature matrix is not stored in memory, then decision tree building will require repeated feature extraction, which is expensive. Parallelization uniformly reduces memory usage by the classifiers, but not enough so that the atomic feature matrix fits in memory.

Feature selection can be effective even if we don't examine every example. Since the weight of high-margin examples can be several orders of magnitude lower than that of low-margin examples (Equation 4.33), the contribution of the high-margin examples to the unpenalized loss gradient (Equation 4.34) will be insignificant. Therefore, we can ignore most examples during feature selection as long as we have a good estimate of the gradient of the unpenalized risk, which in turn gives a good estimate of the gain (Equation 4.30).

Before building each decision tree we use priority sampling (Duffield et al., 2005) to choose a small subset of the examples according to the example weights given by the current classifier, and the tree is built using only this subset. We make the sample small enough that its entire atomic feature matrix will fit in memory. To optimize decision tree building, we compute and cache the sample's atomic feature matrix in advance. Model update, in which the decision tree leaves are confidence-rated, is performed using the entire training set.

The priority sampling procedure to pick a sample S from I is as follows: Fix a sample size $|S| < |I|$. We have $|I|$ examples with weights $w_\Theta(i_1), \ldots, w_\Theta(i_{|I|})$, computed as in Equation 4.33. We independently draw $|I|$ values $\beta_1, \ldots, \beta_{|I|}$ uniformly at random from $(0, 1]$. The **priority** of the n^{th} example is $q(i_n) = w_\Theta(i_n)/\beta_n$, and the sample contains the $|S|$ examples with the highest priority. Let τ be the $|S| + 1^{\text{th}}$ highest priority. Example i_n in the sample has sample weight $\tilde{w_\Theta}(i_n) = \max(w_\Theta(i_n), \tau)$, and any example not in the sample has sample weight 0. Duffield et al. (2005) show that $\mathsf{E}[\tilde{w_\Theta}(i_n)] = w_\Theta(i_n)$. By linearity of expectation, $\mathsf{E}[G_\Theta(S, f)] = G_\Theta(I, f)$ for any feature f. In other words, gains computed over the sample are *unbiased* estimates of the gain computed over the entire training set.

Even if the sample is missing important information in one iteration, the training procedure is capable of recovering it from samples used in subsequent iterations. Moreover, even if a sample's gain estimates are inaccurate and the feature selection step chooses irrelevant compound features, confidence updates are based on the entire training set and the regularization penalty will prevent irrelevant features from having their parameters move away from zero.

Related work on sampling in boosting

In the boosting by filtering framework (Freund, 1995), examples are sampled by weight. Duffield et al. (2005, Section 1.3) examined classical sampling schemes, including over 50 schemes based on weight-sensitive sampling without replacement. They conclude that they do not provide unbiased estimates of subset sums. It follows that the gain estimates are biased in boosting by filtering approaches. Friedman et al. (2000, Section 9) propose sampling by "weight trimming", in which only the highest weight examples are used in each iteration. This approach too gives biased gradient estimates.

Our use of priority sampling is different from stochastic gradient boosting (Friedman, 2002) in several respects. The sample used by Friedman (2002) is chosen uniformly at random, without replacement, from the training set. The weights remain unchanged, so the sample's estimate of the risk gradient is biased. Friedman (2002) uses the sample not only for feature selection, but also for model updates. As discussed in Section 5 of Turian and Melamed (2006b), we cannot avoid using the entire example set for model updates. Otherwise, our example weights will be incorrect, and the subsequent sample will be biased. Friedman (2002) does not encounter this limitation because his sampling procedure does not need the example weights.

As far as we know, Turian and Melamed (2006b) was the first work to use provably unbiased estimates of the risk gradients for feature selection. Parallelizing training and using sampling gave a 100-fold increase in training speed over the naïve approach, as described by Turian and Melamed (2006b).

❧ Sixth Chapter ❧

Experiments

> Much of the excitement we get out of our work is that we don't really know what we are doing.
>
> Edsger W. Dijkstra

6.1 Data

We trained and tested on ≤ 15 word sentences from the English Penn Treebank (Marcus et al., 1993; Taylor et al., 2003). Our data splits are non-standard. Historically (Taskar et al., 2004b; Turian and Melamed, 2006a), sections 02–21 were used for training, section 22 was used for model selection (cross-validation), and section 23 was used for testing. However, this partitioning does not allow us to evaluate the generalization of cross-validated models to unseen data without re-using the test data. By convention, opening a new section of the Penn Treebank is discouraged. We used the following data splits:

- Training comprised sections 02–21 *minus 400 randomly chosen sentences*, a total of 9353 sentences.
- "Tuning" comprised these 400 sentences, and was used for model selection during cross-validation. We used 400 randomly sampled sentences rather than an entirely different section of the Penn Treebank to avoid topic shift.
- Section 22 was used as development data, to determine the generalization of the model selected over the tuning data. There were 421 sentences.
- Section 23 was used as the final test data, a total of 603 sentences.

We might be able to get higher accuracy by training on the entire training set. However, it would limit our ability to run controlled experiments that evaluate generalization accuracy. We have many models to pick between, one for each possible penalty factor λ. With many

Listing 6.1: Steps for preprocessing the data. Starred steps are performed only when parse trees are available in the data (e.g. not at test time).

1. * Strip functional tags and trace indices, and remove traces.
2. * Convert `PRT` to `ADVP`. (This convention was established by Magerman (1995).)
3. Remove quotation marks (i.e. terminal items tagged `` or '') (Bikel, 2004a). Annotation of quotation marks is "at the very bottom of the pecking order." (Bies et al., 1995, pg. 54)
4. * Raise punctuation[a] (Bikel, 2004a).
5. Remove outermost punctuation.[b]
6. * Remove unary projections to self (i.e. duplicate items with the same span and label).
7. POS tag the text using the tagger of Ratnaparkhi (1996).
8. Lowercase headwords.

[a] evalb ignores punctuation placement, so we might as well normalize it.

[b] As pointed out by an anonymous reviewer of Collins (2003), removing outermost punctuation might discard useful information. Collins and Roark (2004) saw a LFMS improvement of 0.8% over their baseline discriminative parser after adding punctuation features, one of which encoded the sentence-final punctuation.

models, model selection to maximize F_1 might "overfit" the cross-validation data (Ng, 1997). Turian and Melamed (2006a) observed a drop of 1.15% F_1 from cross-validation to test. This could be because of overfitting the cross-validation data, or because of topic shift.

We evaluated our parser using the standard PARSEVAL measures (Black et al., 1991): labeled precision, labeled recall, and labeled F-measure (Prec., Rec., and F_1, respectively), which are based on the number of non-terminal items in the parser's output that match those in the gold-standard parse. We computed the PARSEVAL measures using the 20060317 release of evalb, available at `http://nlp.cs.nyu.edu/evalb/`. Note that items labeled `TOP` are not scored.

The correctness of a stratified shuffling test (Noreen, 1989, Section 2.7) has been called into question (Michael Collins, p.c.), but we are not aware of any better significance tests for observed differences in PARSEVAL measures. So we used stratified shuffling with 100,000 trials to test whether the observed F_1 differences are significant at $p = 0.05$. We used an implementation of stratified shuffling written by Dan Bikel (`http://www.cis.upenn.edu/~dbikel/software.html`). We modified it to compute p-values for F_1 differences.

Preprocessing

Data were preprocessed as per Listing 6.1. Our parser does not automatically infer terminal constituents, i.e. part-of-speech tags. In principle, there is no reason we couldn't train the system to make terminal inferences as part of its parsing. However, that is beyond the scope of this work, so to obtain POS tags we use the tagger described in Ratnaparkhi (1996).

6.2 Logic

Parsing strategies

To demonstrate the flexibility of our learning procedure, we trained parsers with three different parsing strategies:

- left-to-right (**l2r**),
- right-to-left (**r2l**), and
- non-deterministic bottom-up (**b.u.**).

The non-deterministic parser was allowed to choose any bottom-up inference. The other two parsers were deterministic: bottom-up inferences had to be performed strictly left-to-right or right-to-left, respectively. An example of r2l parsing is shown in Figure 6.1. An example of b.u. parsing is shown in Figure 6.2. B.u. could have pursued several different paths to obtain the final tree, including the path shown in Figure 6.1. However, the r2l parser has only a unique path to the final tree. If it were following the path in Figure 6.2, it would have been prohibited from inferring "(`NP` the book)" because that inference is right of "(`NP` him)".

Additional validity restrictions on inferences

Besides validity restrictions for inferences imposed by the parsing task (Chapter 2) and the parsing strategy (Section 6.2), we place several additional restrictions on inferences:

- There are $l \cdot n^2$ possible (span, label) pairs over a frontier containing n items, where l is the number of different labels. We reduce this to the $O(l \cdot 5n)$ inferences that have at most five children. Only 0.57% of non-terminals in the preprocessed development data have more than five children.
- To ensure the parser does not enter an infinite loop, no two items in a state can have both the same span and the same label. Hence, if the inference is a unary projection, we disallow it if it projects to an item with the same label as any unary chain descendants. For example, if an `SBAR` was inferred as a parent of frontier item

Figure 6.1: An example of deterministic right-to-left bottom-up parsing. POS tags in the terminals are omitted.

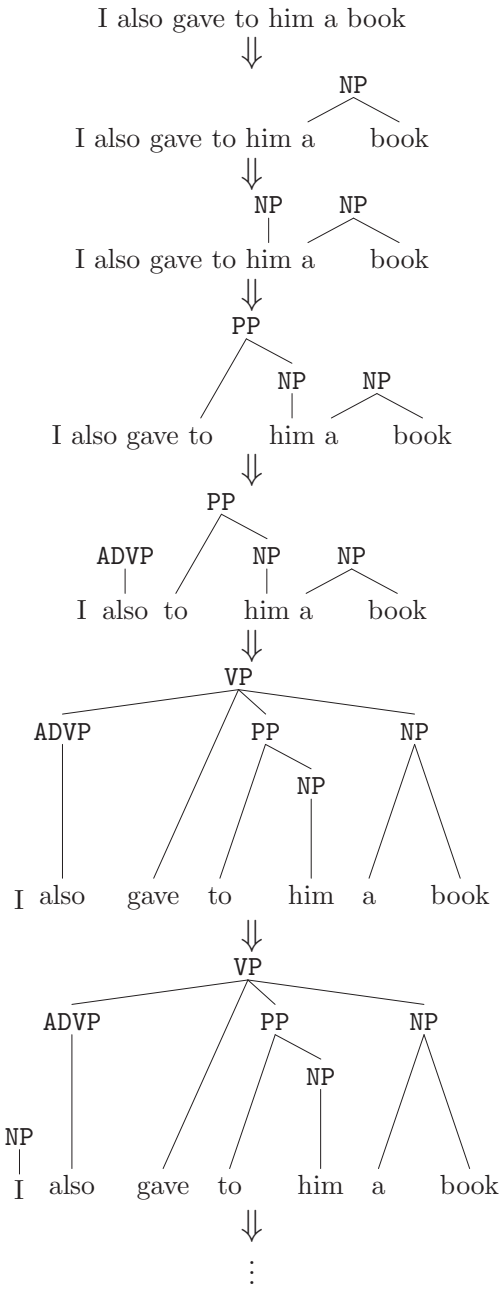


Figure 6.2: An example of non-deterministic bottom-up parsing. POS tags in the terminals are omitted. The final inference, TOP, is also omitted.

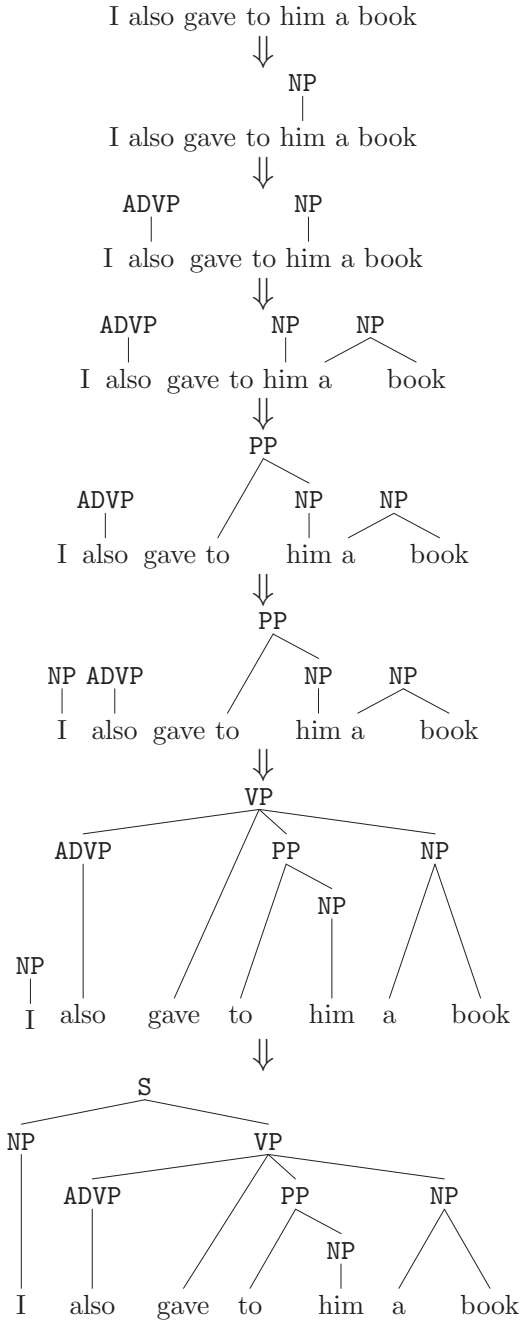
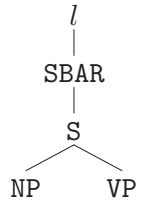


Figure 6.3: A candidate l-inference unary projection. l cannot be SBAR or S.



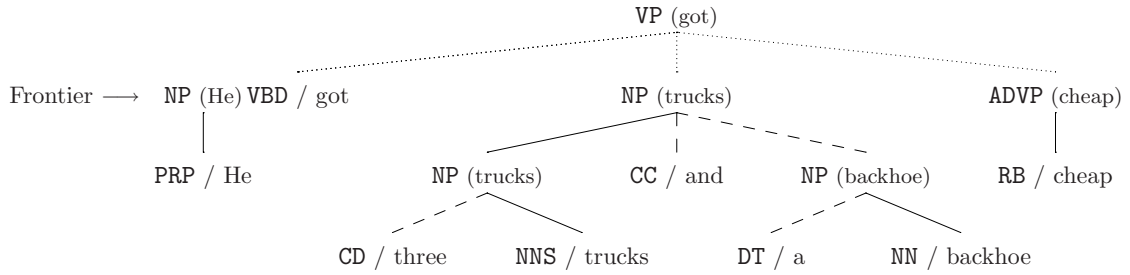
S, which in turn had been inferred as a parent of frontier items NP and VP, then this SBAR cannot become the sole child of an SBAR item or an S item. This example is depicted by Figure 6.3.

- Because we raise punctuation during preprocessing (Step 4 in Listing 6.1), no inference can have punctuation as an outermost item. So, the leftmost and rightmost frontier items spanned by the inference's item must not be punctuation items.

Given these restrictions on candidate inferences, the r2l parsing strategy generated 40.1 million training examples, b.u. generated 43.9 million training examples, and l2r generated 32.4 million training examples. Bottom-up generates the most training examples because all b.u. inferences at each state are permitted. l2r generates the fewest training examples because of the right-branching tendency of English: After performing a correct far-right inference, inferences to its left are subsequently prevented. In each of these training sets, 9598 are TOP-inferences. The rest are evenly distributed between the 25 different nonterminal labels.

The main limitation of our work is that we can do training reasonably quickly only on short sentences because a sentence with n words generates $O(n)$ inferences per state $\times$ $O(n)$ states per sentence $= O(n^2)$ training inferences total. Although generating training examples in advance without a working parser is much faster than using inference (Collins and Roark, 2004; Henderson, 2004; Taskar et al., 2004b), our training time can probably be decreased further by choosing a parsing strategy with a lower branching factor. Like our work, Ratnaparkhi (1999) and Sagae and Lavie (2005) generate examples off-line, but their parsing strategies are essentially shift-reduce so each sentence generates only $O(1)$ inferences per state $\times$ $O(n)$ states per sentence $= O(n)$ training examples. Sagae and Lavie (2005), for example, had only 1.5 million training examples total over the *entire* English Penn Treebank training set.

FIGURE 6.4: A candidate VP-inference, with head-children annotated.



6.3 FEATURES

We define the atomic feature tests that can be performed on some candidate inference. A note on terminology:

- The **frontier** of a state consists of the items with no parents yet.
- The **children** of a candidate inference are the frontier items below the item to be inferred i.e. those items that would become the children of the non-terminal item which we are considering adding to the state. For example, in Figure 6.4 the children are "VBD/ got", "NP (trucks)", and "ADVP (cheap)".
- The left and right **context** items are the frontier items to the left and right of the children of the candidate inference, respectively. For example, in Figure 6.4 the first item of left context (counting from the children out) is "NP (He)", and there are no right context items.
- The **head child** (sometimes also called the **heir** or the **governor**) of a candidate inference is the child item chosen by English head rules (Collins, 1999, pp. 238–240). Intuitively, the head child is the child item distinguished as being the most informative. Following the head-child path to a terminal gives us an item's **head word**. For example, in Figure 6.4 the head child is "VBD/ got" and the head word is "got".

Our atomic feature set A contained features of the form "is there an item in group J whose (label/headword/headtag/headtagclass) is X?". The predicate headtagclass is a supertype of the headtag. Given our compound features, these are not strictly necessary, but they accelerate training. An example is "proper noun," which contains the POS tags given to singular and plural proper nouns. See Table 6.2. Possible values of X for each predicate were collected from the training data. Possible values for J are given in Table 6.1.

Table 6.1: Item groups available in the default feature set. Length features only use the starred item groups.

- the first/last n child items, $1 \leq n \leq 4$
- the first n left/right context items, $1 \leq n \leq 4$
- the n children items left/right of the head, $1 \leq n \leq 4$
- the nth frontier item left/right of the leftmost/head/rightmost child item, $1 \leq n \leq 3$
- the nth terminal item left/right of the leftmost/head/rightmost terminal item dominated by the item being inferred, $1 \leq n \leq 3$
- the leftmost/head/rightmost child item of the leftmost/head/rightmost child item
- * the following groups of frontier items:
  - * all frontier items
  - * children items
  - * left/right context items
  - * non-leftmost/non-head/non-rightmost child items
  - * child items left/right of the head item, inclusive/exclusive
- * the terminal items dominated by one of the item groups in the indented list above

All features are of this form, except for length features (Eisner and Smith, 2005). Length features are of the form: "is the number of items in group J (equal to/greater than) n", with $0 \leq n \leq 15$. These feature templates gave rise to 1.13 million different atomic features. Our features are fine-grained compared to entire context-free production rules. The compound feature space, however, includes context-free rule features, as well as many of the features used by Collins and Koo (2005) and Klein and Manning (2003b), among others. In future work, we plan to try linguistically more sophisticated features (Charniak and Johnson, 2005) as well as sub-tree features (Bod, 2003; Kudo et al., 2005).

TABLE 6.2: POS tag classes, and the POS tags they include. Based on Melamed (1995).

Class	POS tags
CD	CD LS
CJ	CC , :
D	DT PDT PRP$ WDT
EOS	.
IN	IN RP TO
J	JJ JJR JJS
N	$ NN NNS
NP	NNP NNPS
P	EX POS PRP WP WP$
R	RB RBR RBS WRB
SCM	‘‘ ’’ -LRB- -RRB-
SYM	# SYM
UH	UH
UK	FW
VBG	VBG
VBN	VBN
V	MD VB VBD VBP VBZ

6.4 Results

To summarize, our default settings are as follows:

- Our search algorithm is greedy completion.
- The limit γ on the maximum number of paths to score during parsing is 100,000.
- We use ℓ_1-regularization.
- We perform feature selection over the compound feature space, not the atomic feature space. I.e. we induce entire decision trees, not decision stumps.
- We use all the features given in Section 6.3.
- The sample size used during decision tree building is 100,000 examples.
- The learning rate η is 0.9.

We perform experiments that vary each of these choices in turn.

Search strategy: Best-first vs. Greedy completion

We compare the two search strategies described in Chapter 3, greedy-completion and best-first search. We trained a b.u. model with ℓ_1-penalty parameter $\lambda = 1$. We use this model to parse the development data, varying the search strategy and γ (the limit on search effort). Figure 6.5 shows that, in the non-deterministic setting, best-first parsing does poorly compared to greedy completion. If they do the same amount of work, greedy completion finds a better solution on more than 60% of sentences. Moreover, best-first search frequently can't even find a solution in the time allotted. The non-deterministic parser can reach each parse state through many different paths, so it searches a larger space than a deterministic parser, with more redundancy. Best-first parsing flounders in the non-deterministic case because it is more susceptible to **spurious ambiguity** than greedy completion: It prefers to invest search effort finding different paths to low-cost states with few items, rather than finding complete solutions.

With a deterministic logic, greedy completion might do more work during search than best first. We trained l2r and r2l models with ℓ_1-penalty parameter $\lambda = 1$. For each of these models, we measured the amount of search done to find an optimal solution using greedy completion and best first. We compared the search effort of the two search strategies on a per-sentence basis. Figure 6.6 shows that, in the deterministic case, greedy completion requires only slightly more effort than best first search to find the optimal solution. Given that greedy completion was more robust than best-first search, we use greedy completion in the remaining experiments.

Figure 6.5: Parsing performance of the best-first parser as we vary the maximum number of partial parses scored. We parse the 421 sentences in the development data using the b.u. parser trained with ℓ_1-penalty parameter $\lambda = 1$.
Best-first "beats" greedy completion if it find a solution whose cost is no more than the solution found by greedy completion, scoring no more partial parses than greedy completion. Otherwise, greedy completion has found a better solution given the current limit γ.

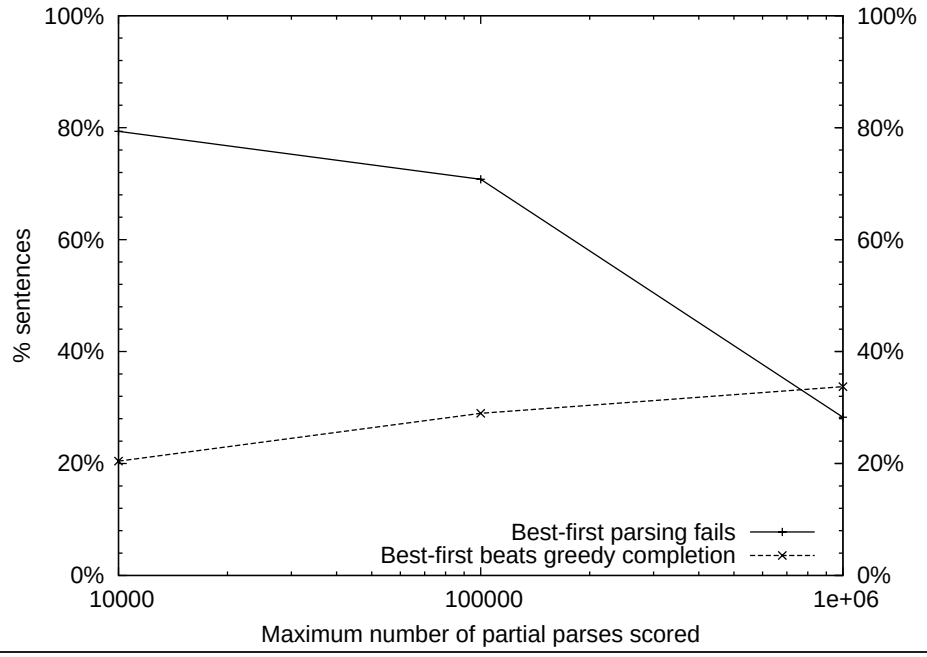
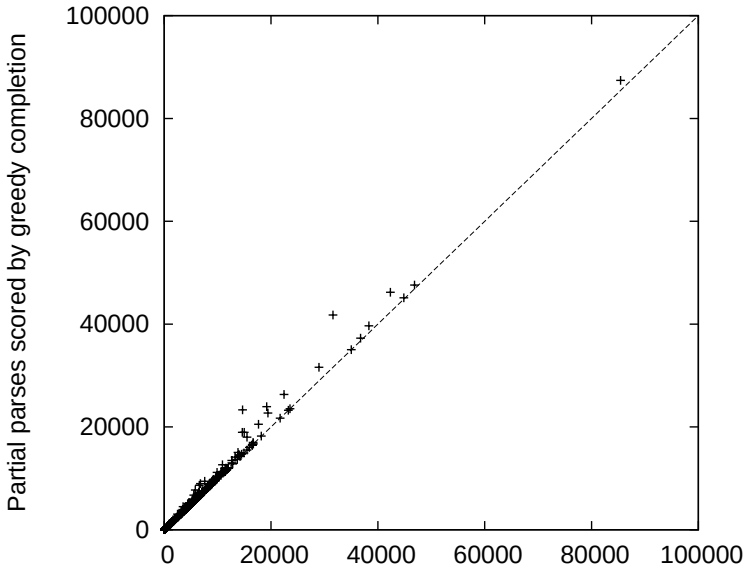
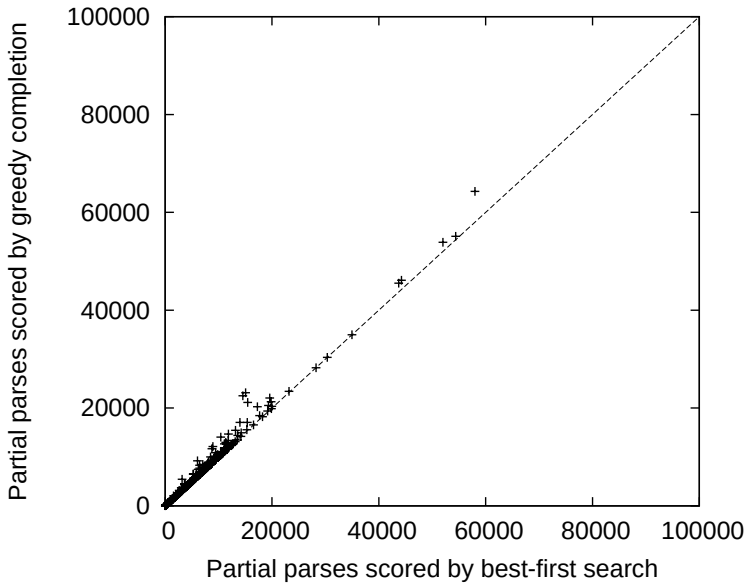


FIGURE 6.6: Number of partial parses scored to find the optimal solution, using best-first and greedy-completion parsing, on each sentence. We parse the 421 sentences in the development data using the (a) l2r and (b) r2l parsers trained with ℓ_1-penalty parameter $\lambda = 1$. The parsers never needed to score more than 100,000 partial parses to find the optimal solution, except for one sentence. This sentence was "For the moment, at least, euphoria has replaced anxiety on Wall Street." On this sentence, the best-first parser scored 323,497 partial parses, and greedy completion scored 323,703 (206 more). This sentence also required the most effort by the l2r parser, and is the point in the upper-right of (a).

(a)



(b)



Cross-validated runs

In the following experiments, we plot the F_1 of each parser on the tuning data as training progresses. For each model, we cross-validated the penalty term λ by picking the value that gave highest F_1 on the tuning data. To measure generalization, we evaluated each cross-validated model on our development set. These results are given in Table 6.3, as well as measurements of the model sizes. Model sizes are presented using three measures:

- the number of compound feature types (non-zero parameters in the model),
- the number of decision trees built,

  and
- the number of leaves over all trees (including those with zero delta-confidence).

Training time is roughly proportional to the number of leaves and all runs' training time are proportional to the number of leaves by the same constant factor, except where otherwise stated.

Table 6.3: PARSEVAL measures of the parsers with λ chosen to maximize F_1 on the tuning data. When not stated, the parsing strategy is r2l. A shaded cell means that the difference between this value and that of the r2l parser is statistically significant. The last three columns give measures of the size of the model: the number of compound feature types (non-zero parameters in the model), the number of decision trees built, and the number of leaves over all trees (including those with zero delta-confidence). The top section is given in exposition order, the bottom section is sorted by F_1 on the development set.

Run	λ	Tune Rec.	Tune Prec.	Tune F_1	Dev Rec.	Dev Prec.	Dev F_1	# compounds	# trees	# leaves
r2l	0.1090	91.84	91.92	91.88	90.39	90.08	90.24	3957	4011	154590
b.u.	0.0714	91.54	91.46	91.50	89.45	88.69	89.07	5015	5071	209920
l2r	0.1168	90.39	91.44	91.18	88.80	88.91	88.86	4675	4737	192619
r2l (stumps)	0.1012	86.05	87.07	86.56	83.91	85.01	84.46	19421	53238	105079
b.u. (stumps)	0.1093	84.85	86.81	85.82	83.21	83.75	83.48	19708	57367	113474
l2r (stumps)	0.1178	86.14	87.89	87.01	83.88	84.86	84.37	20231	60489	119638
ℓ_2	0.6265	90.19	90.51	90.35	89.45	89.69	89.57	360371	832	367253
unreg gain	0.1868	91.57	91.68	91.63	89.51	89.20	89.35	2084	2240	1481574
10K sample	0.0370	92.01	91.69	91.85	89.86	88.92	89.39	6221	5175	242202
decay factor η =0.1	0.0036	91.60	91.65	91.63	89.69	89.29	89.49	7345	666	110526
r2l	0.1090	91.84	91.92	91.88	90.39	90.08	90.24	3957	4011	154590
ℓ_2	0.6265	90.19	90.51	90.35	89.45	89.69	89.57	360371	832	367253
decay factor η =0.1	0.0036	91.60	91.65	91.63	89.69	89.29	89.49	7345	666	110526
10K sample	0.0370	92.01	91.69	91.85	89.86	88.92	89.39	6221	5175	242202
unreg gain	0.1868	91.57	91.68	91.63	89.51	89.20	89.35	2084	2240	1481574
b.u.	0.0714	91.54	91.46	91.50	89.45	88.69	89.07	5015	5071	209920
l2r	0.1168	90.39	91.44	91.18	88.80	88.91	88.86	4675	4737	192619
r2l (stumps)	0.1012	86.05	87.07	86.56	83.91	85.01	84.46	19421	53238	105079
l2r (stumps)	0.1178	86.14	87.89	87.01	83.88	84.86	84.37	20231	60489	119638
b.u. (stumps)	0.1093	84.85	86.81	85.82	83.21	83.75	83.48	19708	57367	113474

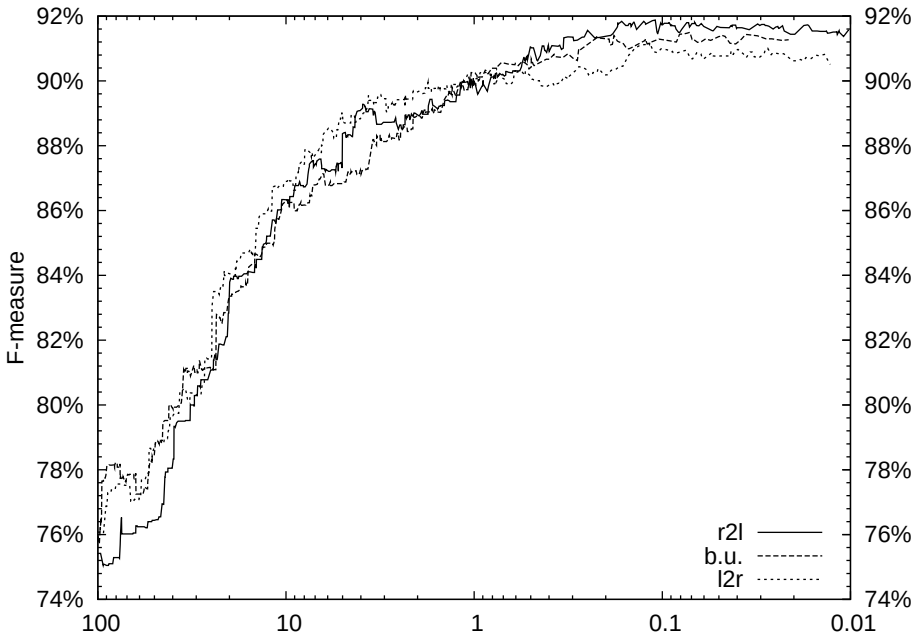
Parsing strategies: L2R vs. R2L vs. B.U.

We trained models with the default settings on page 69, but varying the parsing strategy. Figure 6.7 shows the learning curves of parsers using three different parsing strategies: l2r, r2l, and b.u. Of all three parser, l2r has the worst F_1 on both tuning and development. All its PARSEVAL measures are significantly lower than those of the r2l parser on the development set. l2r does poorly because its decisions were more difficult than those of the other parsers. If it inferred far-right items, it was more likely to prevent correct subsequent inferences that were to the left. But if it inferred far-left items, then it went against the right-branching tendency of English sentences. The l2r parser would likely improve if we were to use a left-corner transform (Collins and Roark, 2004). The hypothesis that learning a l2r model is more difficult than learning a r2l model is corroborated by the tree size measurements given in Table 6.3. The r2l model requires less effort to train further (lower λ).

Although agenda-driven search is not guaranteed to find the optimal solution to Equation 2.12 in polynomial time, in our experiments we found that it worked well in practice. The number of states in the search space is exponential in the size of the input, so one might worry about search errors. Nonetheless, in our experiments, the inference evaluation function was learned accurately enough to guide the deterministic parsers to the optimal parse reasonably quickly without pruning, and thus without search errors. For example, on the tuning data we observed that 80.5% of the greedy solutions of the r2l model are also optimal. By greedy solution, we mean the first solution found during greedy completion. Equivalently, the greedy solution is the one found by traditional greedy search (see Section 3.2). However, search using the best b.u. model exceeded the search limit of $\gamma = 100,000$ and returned a sub-optimal solution for 75% of the sentences on the tuning set. The best l2r and r2l parsers never exceed the limit and always returned the optimal solution. The best l2r, r2l, and b.u. parsers scored an average of 4614.8, 5453.0, and 51878 paths for each sentence in the tuning data, respectively. For this reason, the b.u. model took roughly an order of magnitude more time to parse. We also found that, as training progresses and the models become more refined, the l2r and r2l parsers found solutions *faster*. However, the b.u. parsing time remains relatively constant. This trend is shown in Figure 6.8.

FIGURE 6.7: PARSEVAL F_1 accuracy on the tuning data of parsers with different parsing strategies. (b) is a zoomed in version.

(a)



(b)

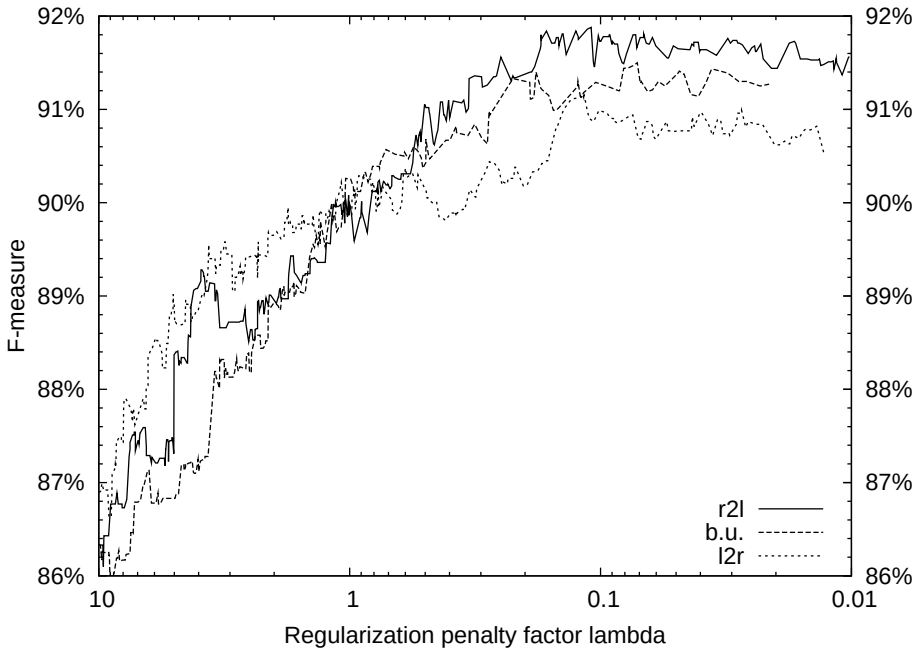
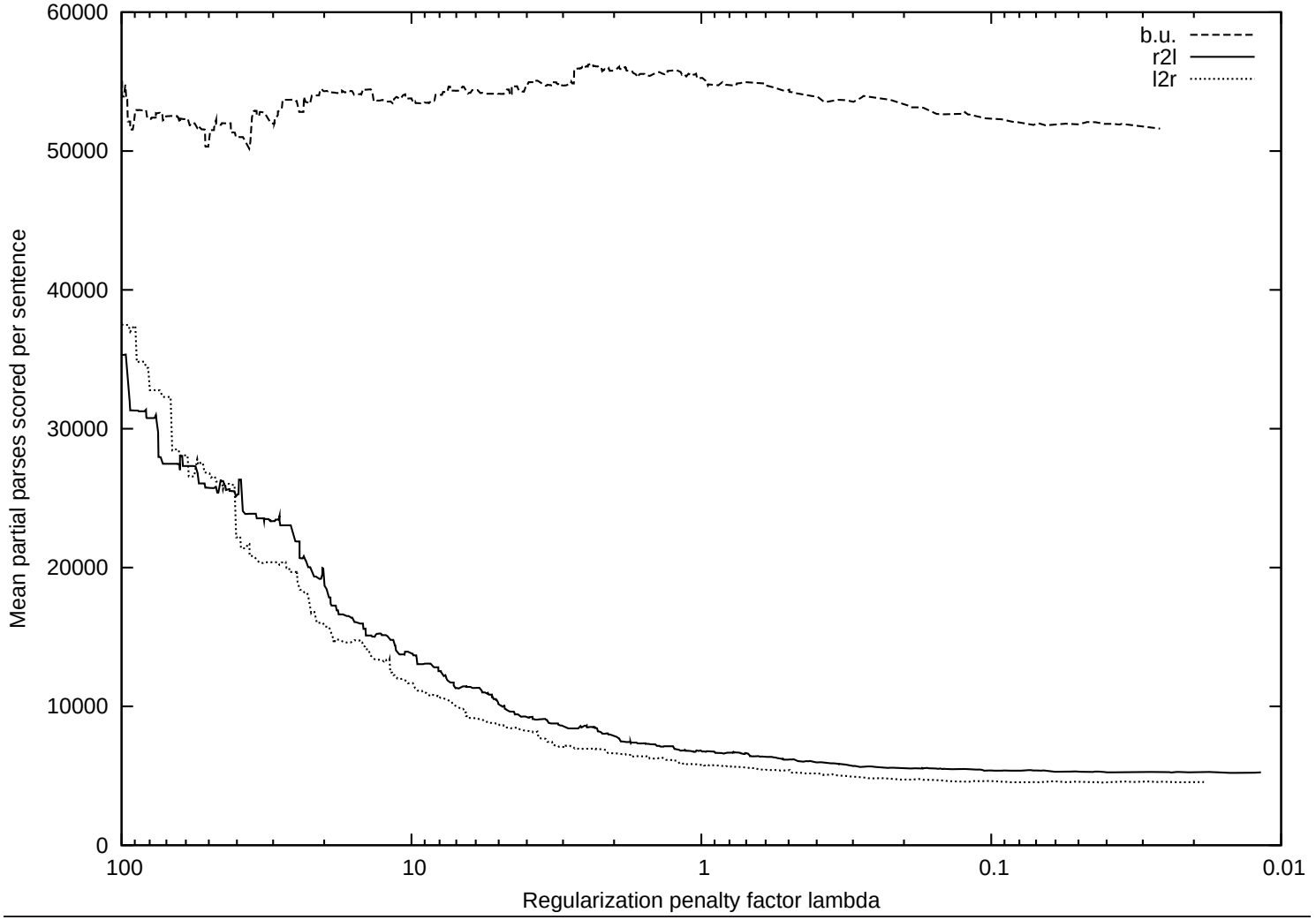


Figure 6.8: Mean search effort required to parse a sentence in the tuning data, as training progresses.



Decision trees vs. Decision stumps

In our next experiment, we compare models induced over the compound feature space with models induced over the *atomic* feature space. To learn the latter models, we induced decision **stumps**, i.e. decision trees that cannot split any node deeper than the root. Figure 6.9 shows the accuracy of these models on the tuning data as training progresses. The F_1 of each stumps model is at least 4% lower than that of the corresponding decision trees model. All three PARSEVAL measures were significantly higher for each model compared to its stumps equivalent, both on tuning and development. The stumps models even seem insufficiently powerful to model the training data, never surpassing 90% F_1. (See Figure 6.10.) This is not for want of training. Compared to building a full decision tree, building a decision stump will take a negligible amount of time. According to the profile in Table 6.4, each decision tree training iteration takes roughly $204.9/(204.9 - 127.6) = 2.65$ times as long as each decision stump training iteration. So the best r2l stumps run has trained roughly $53238/4011/2.65 = 5.01$ times as long as the best r2l decision tree run, $57367/5071/2.65 = 4.27$ times as long for l2r, and $60489/4737/2.65 = 4.82$ times as long for l2r. Our experiments show that our atomic features are powerful enough to learn the training data only when compounded.

Table 6.4: Profile given in Turian and Melamed (2006b) of a typical `NP` training iteration for the r2l model. The profile is given in seconds, and was measured using an AMD Opteron 242 (64-bit, 1.6Ghz).

Description	**mean**	**stddev**	**%**
Priority sample 100,00 examples from the training set	1.5s	0.07s	0.7%
Extract and cache the atomic features of the sample examples	38.2s	0.13s	18.6%
Build a decision tree using the sample	127.6s	27.60s	62.3%
Percolate every training example down to a decision tree leaf	31.4s	4.91s	15.3%
Choose the delta-confidence for each leaf node (line-search)	6.2s	1.75s	3.0%
Total	204.9s	32.6s	100.0%

FIGURE 6.9: PARSEVAL F_1 accuracy on the tuning data of parsers trained using decision trees and decision stumps.

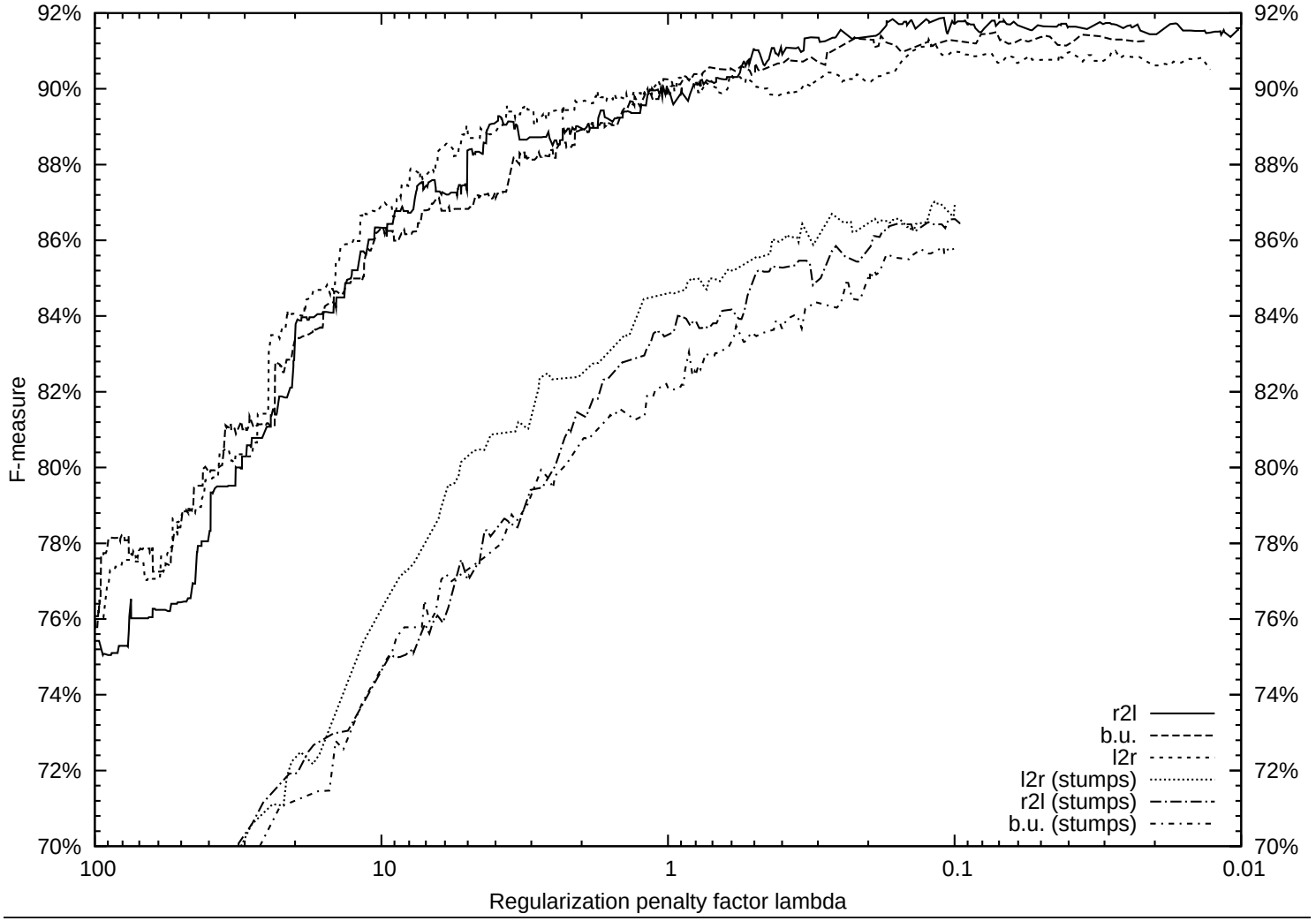
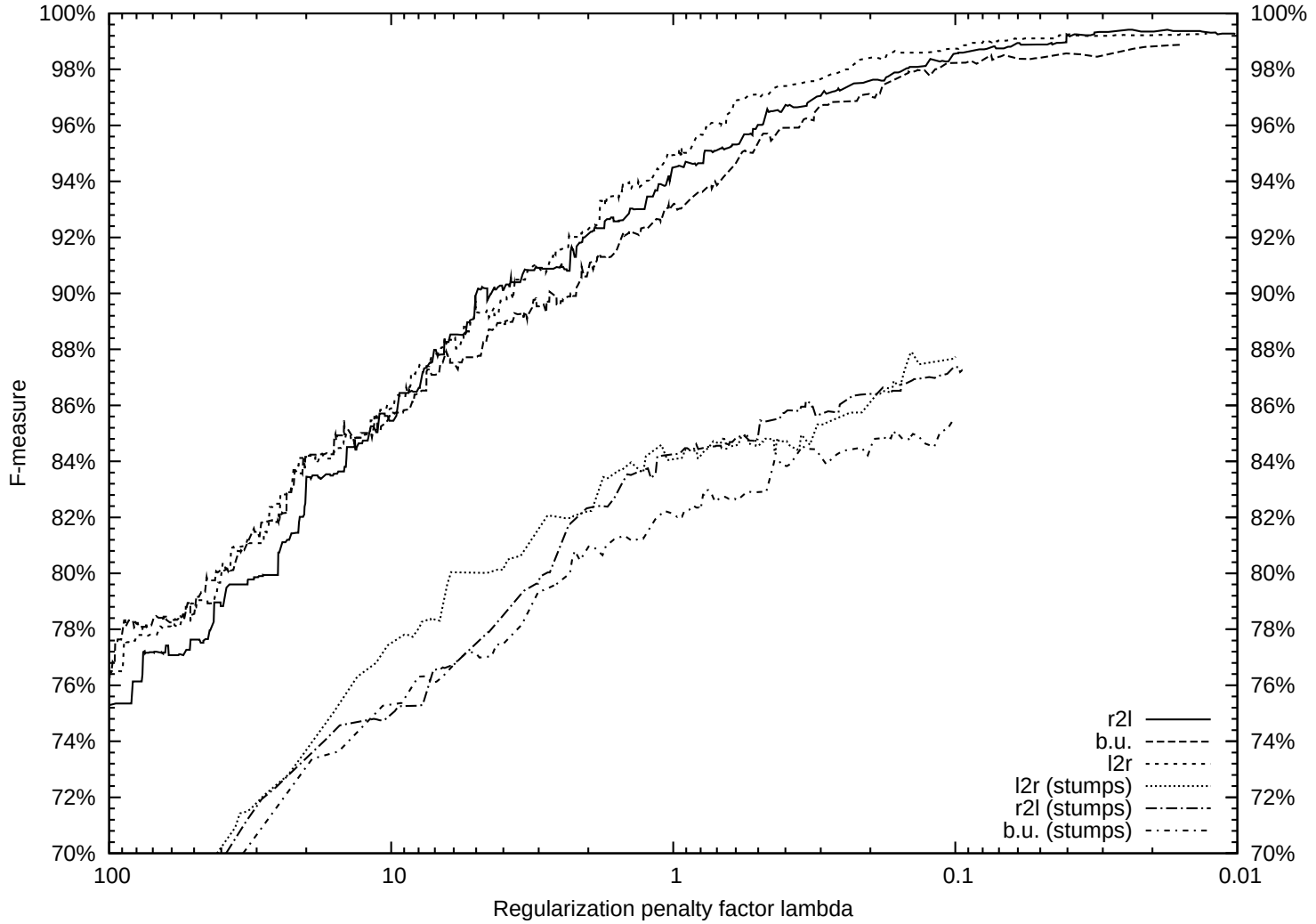


Figure 6.10: PARSEVAL F_1 accuracy on 400 *training* sentences of parsers trained using decision trees and decision stumps.



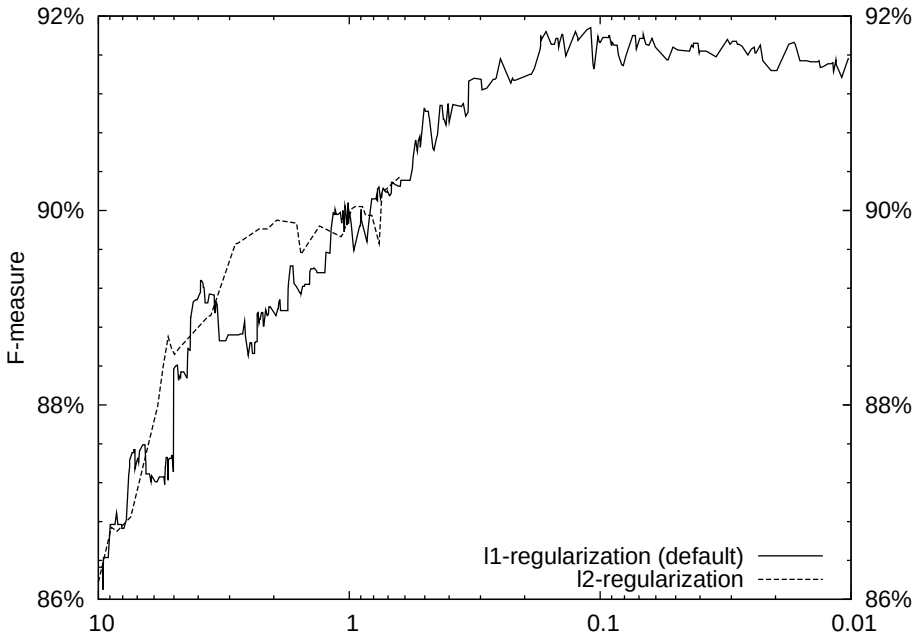
For the remaining experiments, we only use the r2l parsing strategy.

ℓ_1-regularization vs. ℓ_2 regularization

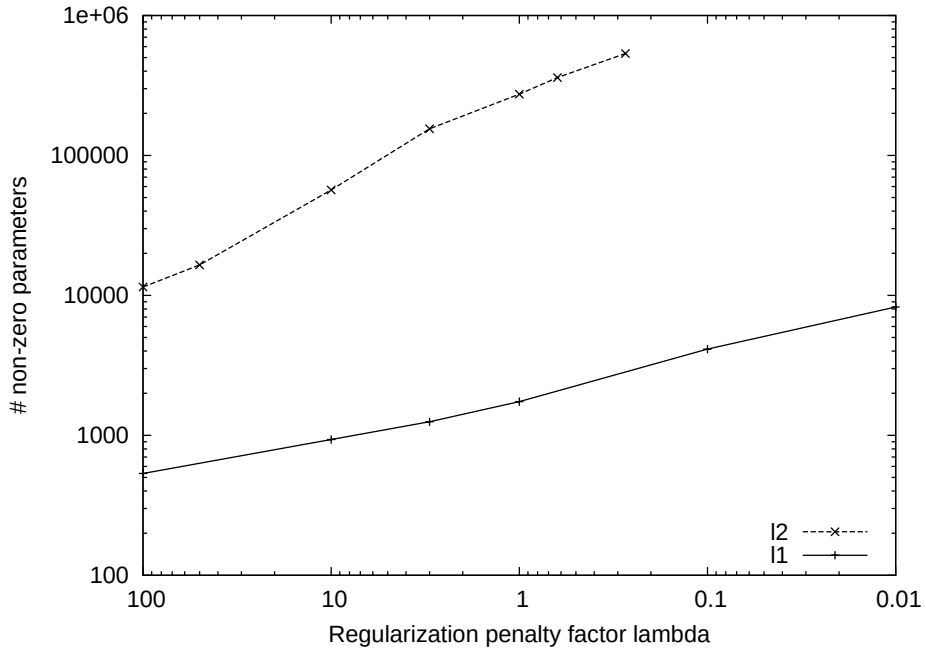
We induced ℓ_2-regularized decision trees. As noted on page 47, ℓ_2-regularized decision trees are identical to unregularized decision trees, except the confidences at the leaves are shrunk. The ℓ_1 model is quite sparse, averaging fewer than one non-zero parameter per decision tree. The ℓ_2 model, on the other hand, has on average $360371/832 = 433$ non-zero parameters per decision tree. Figure 6.11(a) compares the accuracy of the ℓ_2-regularized decision trees to the ℓ_1-regularized decision trees. We did not detect a significant difference in the PARSEVAL measures between the two learning methods at $\lambda = 0.6265$ (the penalty factor with the highest F_1 point on the ℓ_2 curve). However, we were unable to proceed further in testing with the ℓ_2-regularized models, as they did not fit in 2 GB of RAM. By the time the penalty parameter reaches 0.1, which is where most of the runs converged, the ℓ_2 model has over 535,000 non-zero compound features, over two orders of magnitude more than the ℓ_1 model. Figure 6.11(b) shows how, as training progresses, the ℓ_2 model grows more quickly in size than ℓ_1 model.

FIGURE 6.11: As training of the ℓ_1 and ℓ_2 regularized models progresses,
(a) PARSEVAL F_1 accuracy on the tuning data, and
(b) number of non-zero parameters (i.e. compound feature types).

(a)



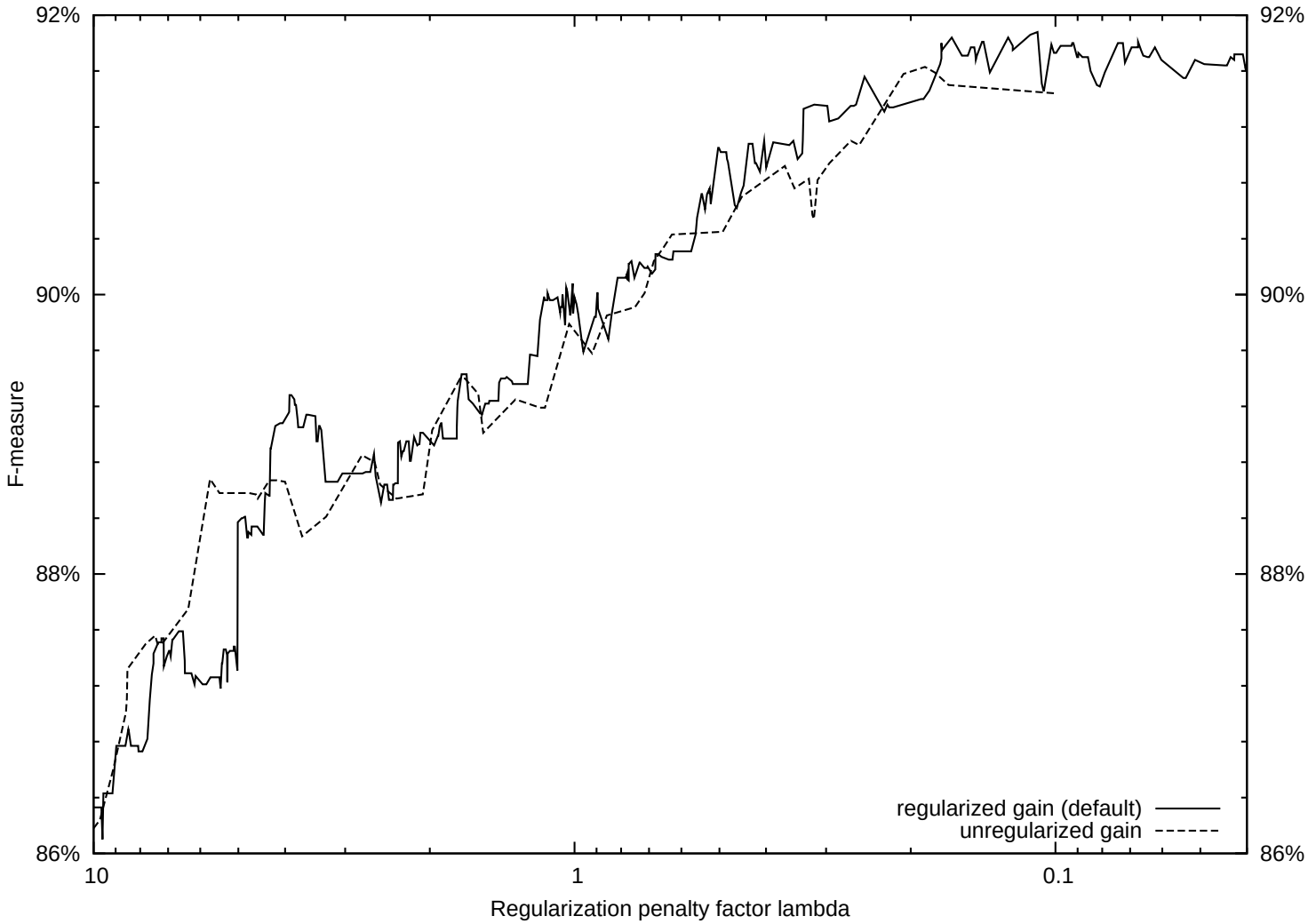
(b)



Regularized feature selection vs. Unregularized feature selection

Rosset et al. (2004) theoretically and empirically analyze boosting of the unregularized logistic and exponential losses. They find that, in the separable case, as training progresses the learned classifier maximizes the minimal ℓ_1-margin of the training data. Early stopping can thus be viewed as a method for picking the regularization penalty factor. When boosting these unregularized risks, care must be taken that model parameter values are not set too high. For example, if no incorrect example falls in a particular decision tree leaf, $+\infty$ is the confidence for this leaf that minimizes the unregularized risk. To prevent risky high-confidence parameter values, various forms of smoothing are applied to "slow" the learning process. For example, Schapire and Singer (1999, Section 4.2) add a small amount of example weight (in the sense of Equation 4.33) to both the correct and incorrect example weight totals. Mason et al. (1999) propose AnyBoost.ℓ_1 and AnyBoost.ℓ_2, which perform unregularized feature selection but impose an ℓ_1 and ℓ_2 constraint respectively on the parameters during model updates. To simulate this sort of approach in our framework, we boost *unregularized* decision trees by computing the leaf gain in Equation 4.30 using $\lambda = 0$. Leaf confidences are picked, as before, to minimize the regularized risk. The training curves are shown in Figure 6.12. Although we did not detect a significant difference of the PARSEVAL measures on the tuning data, training with an unregularized gain was slower than training with a regularized gain. As indicated by the leaf counts in Table 6.3, building unregularized decision trees and subsequently regularizing them slows the learning process by almost an order of magnitude compared to using an explicit regularizer throughout. The best unregularized gain parser didn't train to as low a λ as the default parser, but it nonetheless built $1481574/154590 = 9.6$ times as many leaves.

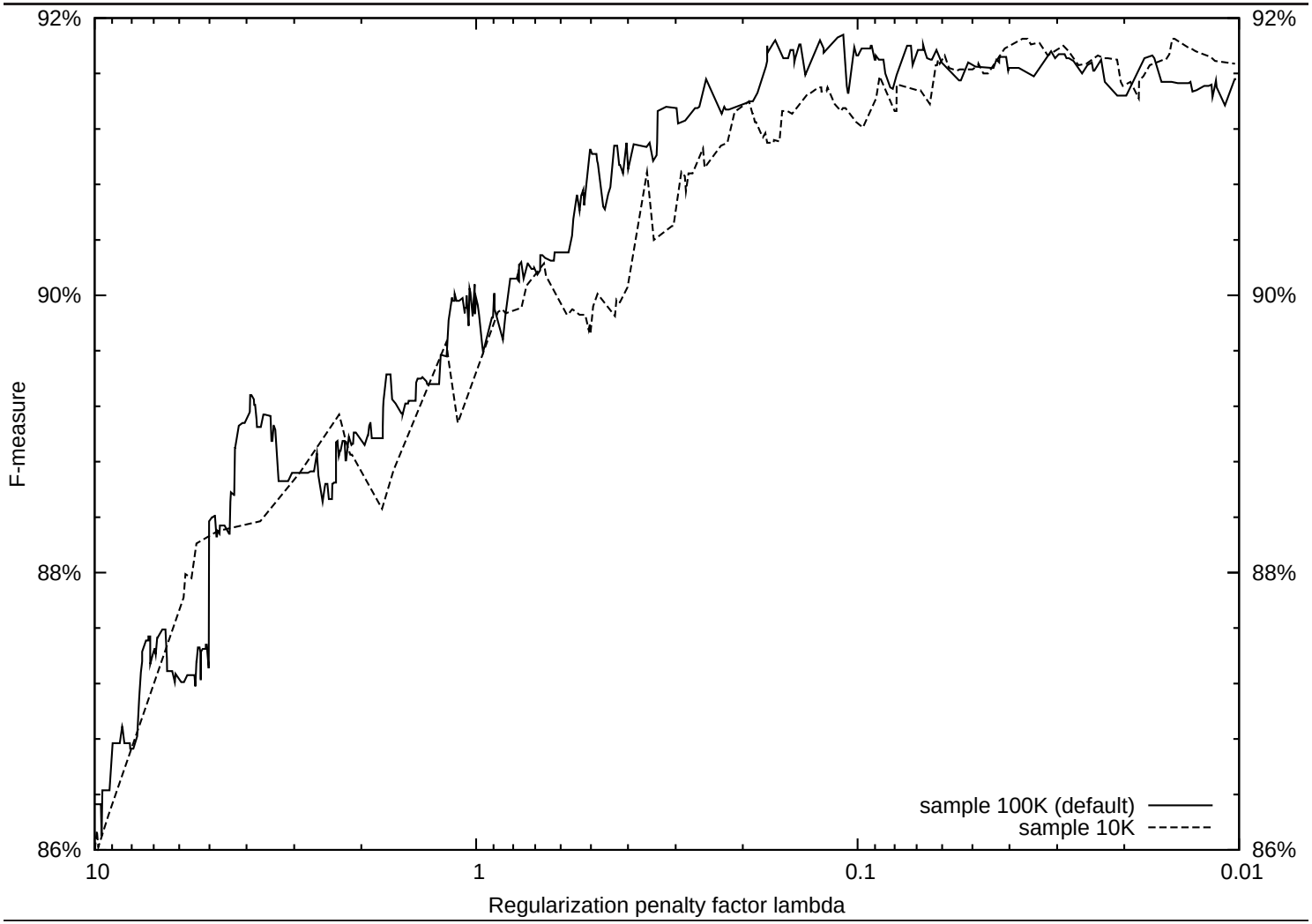
FIGURE 6.12: PARSEVAL F_1 accuracy on the tuning data of parsers trained using regularization and no regularization during feature selection.



100K sample size vs. 10K sample size

We experiment with using a sample size of 10,000 examples for decision tree building (0.63% of the training set), instead of the default sample size of 100,000 examples. As before, model updates are performed using the entire training set. Figure 6.13 shows the accuracy of this parser as training progresses. We did not detect a statistically significant difference in the PARSEVAL measures of the two runs on the tuning data. The 100K sample run had significantly higher F_1 on the development data than the 10K sample run. At a λ of 0.1090 (the best λ on tuning for the model built with a sample size of 100K), the model built with a sample size of 10K had 4349 compounds, 3672 trees, and 154630 leaves. Its decision trees were less sparse than the model built with a sample size of 100K, $4349/3672 = 1.18$ compounds per tree vs. $3957/4011 = .99$ compounds per tree, respectively. The number of leaves differed only by 40 in the two parsers at this λ. However, based on the profile in Table 6.4, we estimate that the 10K sample run builds trees 3.68 times faster than the 100K sample run. The 10K sample run will take one-tenth the time in the "Extract" and "Build" steps. $1.5 + 38.2/10 + 127.6/10 + 31.4 + 6.2 = 55.68$ seconds per tree, versus 204.9 seconds per tree with a sample of 100K examples.

FIGURE 6.13: PARSEVAL F_1 accuracy on the tuning data of parsers trained using sample sizes of 100K and 10K for decision-tree building.

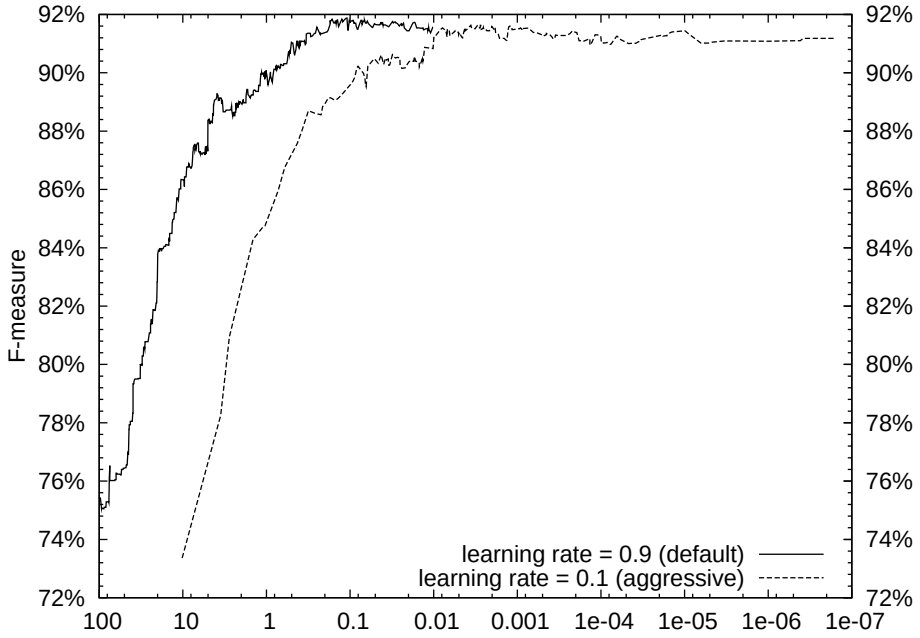


Learning rate $\eta = 0.9$ vs. $\eta = 0.1$

We experiment with a faster learning rate, $\eta = 0.1$. Figure 6.14(a) shows the accuracy of a parser trained with $\eta = 0.1$. We did not detect a significant difference between the $\eta = 0.1$ and $\eta = 0.9$ runs, either on tuning or development, for any of the PARSEVAL measures. The $\eta = 0.1$ model builds trees that are less sparse than those built with the slow learning rate, $7345/666 = 11.03$ compounds per tree vs. $3957/4011 = .99$ compounds per tree, respectively. As expected, with a faster learning rate the regularization penalty drops more quickly, i.e. training occurs more quickly. Nonetheless, the aggressive learning rate model's learning curve shows no evidence of overfitting. One possible explanation is that numerical issues mean that no progress is made in training once λ is very low, i.e. all model changes are *de minimis*. However, Figure 6.14(b) indicates that the model complexity is increasing as λ drops, so this explanation is refuted. Another possibility is that, even though progress is being made in training, the classifiers are just picking very rare features which do not actually occur in the held-out data. According to this argument, since the classifiers are giving essentially the same scores to the inferences in the held-out data, the F_1 curve does not show the effects of overtraining. However, this response does not hold water either: We compare the output sentences of the parsers on the tuning data at $\lambda = 1.6 \times 10^{-7}$ (the furthest reached during training) with $\lambda = 3.61 \times 10^{-3}$ (the best parser on tuning) and found that 14.50% percent of sentences were different. Even between $\lambda = 1.03 \times 10^{-6}$ and $\lambda = 1.6 \times 10^{-7}$, 2.25% percent of sentences were different. For two different models to find different solutions to Equation 2.13 implies that the individual inference costs were sufficiently different to affect the overall result. These experiments indicate that learning without overfitting is, in fact, occurring. Like the original work on boosting which couldn't detect overfitting, we are baffled by this phenomenon.

FIGURE 6.14: Experiments using learning rate $\eta = 0.9$ and 0.1.
(a) F_1 accuracy on the tuning data of parsers trained using learning rate $\eta = 0.9$ and 0.1.
(b) ℓ_1-norm (i.e. Ω_Θ *not* multiplied by λ) as training progresses of the parser trained using learning rate $\eta = 0.1$.

(a)



(b)

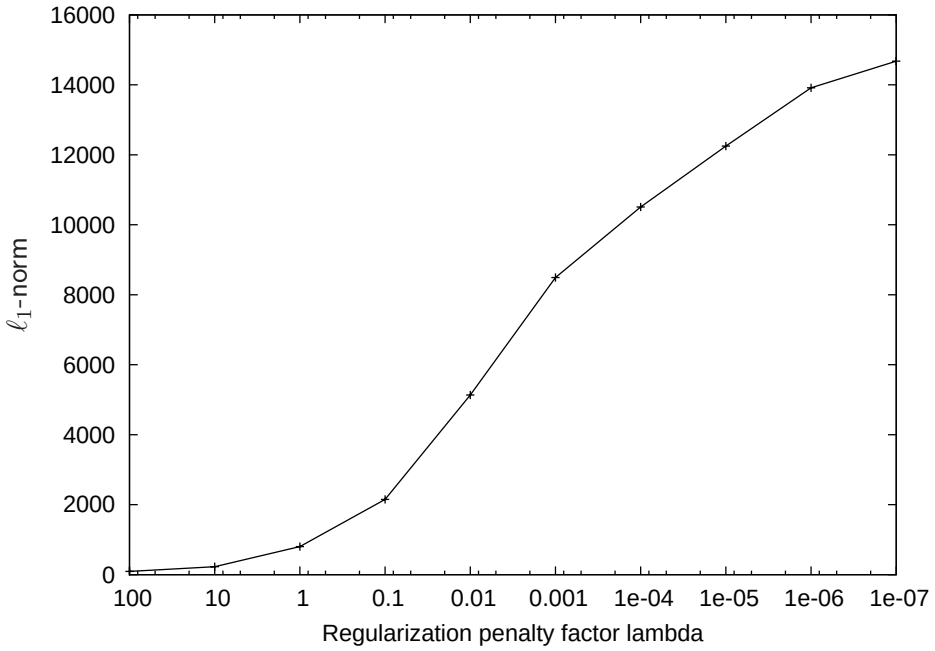


Table 6.5: Accuracy on test data (Section 23 of the English Penn Treebank), as reported by Turian and Melamed (2006a).

	% Recall	% Precision	F_1
Bikel (2004a)	87.85	88.75	88.30
Taskar et al. (2004b)	89.10	89.14	89.12
r2l	**89.26**	**89.55**	**89.40**

Test results

To situate our results in the literature, we compared our approach to those reported by Taskar et al. (2004b) for their discriminative parser, which were also trained and tested on ≤ 15 word sentences. The results reported by Taskar et al. (2004b) were not for a purely discriminative parser. Their parser beat the generative model of Bikel (2004a) only after using the output from a generative model as a feature. Unlike the previous experiments in this chapter, the parser of Taskar et al. (2004b) was trained on the entire training set (including the 400 sentences we call "tuning") and they used the development set for model selection, as described at the beginning of Section 6.1. Turian and Melamed (2006a) tested the r2l model with this data split, and we report their results. We also compare our parser to a representative non-discriminative parser (Bikel, 2004a), a "clean room" reimplementation of the Collins (1997, 1999, 2003) parser with comparable accuracy. (Bikel, 2004a) is the only non-discriminative parser that we were able to train and test under exactly the same experimental conditions, including the use of POS tags from Ratnaparkhi (1996). Table 6.5 shows the results of these four parsers on the test set. The accuracy of our parser is at least as high as that of comparable parsers in the literature.

❧ Seventh Chapter ❧

Related Work

A good composer does not imitate; he steals.

Igor Stravinsky

In this section, we summarize empirical parsing techniques and structure prediction techniques that have been applied to parsing. In Section 7.1, we describe different structure prediction models. In Section 7.2, we discuss how different types of models have been trained.

7.1 Modeling

In Equation 2.12, we gave the form of a history-based model with a generalized cost function parameterized by Θ. That equation was:

$$C_\Theta(p) = \sum_{j=1}^{|p|} c_\Theta(X(i_j)) \tag{7.1}$$

Parse p comprises the individual inferences $i_1, \ldots, i_{|p|}$. Feature extraction function X transforms an inference into a fixed-length real-valued feature vector. Typically, c_Θ non-linearly transforms the feature-vector into a non-negative cost. Equation 7.1 is a **local model**, in which cost is introduced at each decision step. Another common parsing approach is to use a **global model**, where:

$$C_\Theta(p) = C_\Theta(X(p)) = C_\Theta\left(\sum_{j=1}^{|p|} X(i_j)\right) \tag{7.2}$$

In other words, the global model accumulates feature counts over the entire parse, and only then transforms them into a cost. We stress that most—if not all—parsing models can be expressed using Equation 7.1, including global models and parse rerankers. We can express

global models using Equation 7.1 by choosing the feature extraction function appropriately and by assuming that only the final inference has non-zero cost. For parse rerankers, we can posit a single inference that produces the entire tree.

Nearly all parsing models use the restrictions that Θ is a real-valued parameter vector with the same dimensionality as the feature vectors and that:

$$c_\Theta(X(i_j)) = c(\Theta \cdot X(i_j)). \tag{7.3}$$

For a global model, this restriction is:

$$C_\Theta(X(p)) = C(\Theta \cdot X(p)) \tag{7.4}$$

A typical further restriction is that the model is log-linear, e.g.:

$$c(\Theta \cdot X(i_j)) = \exp(\Theta \cdot X(i_j)) \tag{7.5}$$
$$C(\Theta \cdot X(p)) = \exp(\Theta \cdot X(p)) \tag{7.6}$$

Certain works even assume that the model is linear:

$$C_\Theta(X(p)) = \Theta \cdot X(p) \tag{7.7}$$

For example, a PCFG can be represented as a linear model: Each feature f is a context-free production, $X_f(p)$ is the count of this production in the parse, and Θ_f is the negative log-probability of the production. Taskar et al. (2004b) induce a linear model for unlexicalized CFGs. Perceptron models, such as the averaged perceptron (Collins, 2002; Collins and Roark, 2004), are also linear models. McDonald et al. (2005a,b, 2006) and McDonald and Pereira (2006) describe linear models for dependency parsing.

There are some notable exceptions to the parsing model given in Equation 7.3:

- Henderson (2003, 2004) transforms a feature-vector into a cost using a neural network, and Θ corresponds to the weights of the connections in the neural net. More specifically, the history and lookahead are variable-length, but they are each compressed to a fixed-size **hidden** or **intermediate** representation. The hidden representation(s) and other conditioning information are then used to compute probabilities. SSN training simultaneously learns the mappings from variable-length representations to fixed-length representations and the mapping from the fixed-length representation to a probability estimate.

  Titov and Henderson (2007a,b) extend this approach by using an incremental sigmoid belief network (ISBN) to estimate the probabilistic model. The ISBN includes a latent Bayesian model of the history. These works were published after the experiments in Section 6.4 were conducted and published. Titov and Henderson (2007a,b) conduct an experiment in which they train and test their approach on the ≤ 15 words sen-

tences in the Penn Treebank using POS tags from the tagger of Ratnaparkhi (1996). They select hyperparameters using a different development set than was used by Taskar et al. (2004b) and us. Under these similar experimental conditions, Titov and Henderson (2007a,b) report higher test set results than our approach. The advantage of their approach is that it automatically induces the latent variable representation of the history. They do not need to define any atomic features. They simply determine the structure of their graphical model, which encodes their inductive bias. The disadvantage of their approach, which we discussed in Section 1.2, is that the latent representation is opaque. It can be difficult to determine what information is present and what information is missing. The choice of inductive bias can influence the latent representation in subtle ways. We aware of no work on constituent parsing besides these approaches and ours that attempt to use information from the *entire* state in scoring parse decisions.

- Kernel approaches transform primal formulations of linear models into a dual formulation and replace each dot product of feature vectors $X(p) \cdot X(p')$ with a kernel function $K(p, p')$. We have not seen kernelized local models in the literature. In principle, there is no reason why they cannot be used. The feature vector in a kernelized model has as many entries as there are training examples. Each entry in the feature feature is the kernel value of the entry's training example with respect to the structure being scored. For example, when scoring p, the feature entry corresponding to training example p' is $X_{p'}(p) = K(p, p')$. The model is linear in this kernelized feature vector. Many works that discuss approaches using linear models also present kernelized versions of their models. For example, Collins and Duffy (2002a,b) apply a subtrees-based tree kernel in the dual formulation of the perceptron algorithm. Taskar et al. (2004a) and Tsochantaridis et al. (2004, 2005) also discuss how their approaches can be kernelized.

Many parsers use probabilistic models. A **joint** (a.k.a. generative) model is one that estimates the joint probability $\Pr(s, p)$ of a sentence s and a parse tree p. Estimating a joint model using likelihood maximization can be quite expensive, since it requires computing expectations over *all* sentences and their licensed parses, not just the sentences present in the training data (Johnson et al., 1999). A **conditional** model is one that estimates the conditional probability $\Pr(p|s)$ of a parse tree p given a sentence s. To ensure the model estimates a true probability distribution, probabilistic models should be normalized. In particular:

- a **globally-normalized** conditional model is one that normalizes over all parse trees for a given sentence.

- a **locally-normalized** conditional model is one that normalizes over all inferences permitted at a given state. Lafferty et al. (2001) argue that local normalization in

structure prediction will lead to the so-called **label bias** problem. This is because per-state normalization implies that all the mass received by a particular state will be distributed among subsequent states, even if none of the subsequent states is promising. LeCun et al. (2007, Section 6.1) also argue that local normalization will incur label bias, writing that "every explicit normalization is another opportunity for mishandling unforeseen events". Ratnaparkhi (1997, 1999), McCallum et al. (2000), Henderson (2004), and Titov and Henderson (2007a,b) all use locally-normalized models, and hence should be subject to label bias. An example of label bias is given in the text on page 37 describing Figure 4.2.

The log-linear versions of the above two models are normalized as follows:

- Globally-normalized conditional log-linear models:

$$\Pr(p|s) = \frac{1}{Z(s)} \exp(\Theta \cdot X(p|s)) \tag{7.8}$$

  where the normalization term (a.k.a. **partition function**) is computed over all parses $P(s)$ for this sentence:

$$Z(s) = \sum_{p' \in P(s)} \exp(\Theta \cdot X(p'|s)) \tag{7.9}$$

- Locally-normalized conditional log-linear models:

$$\Pr(d_j|S_j) = \frac{1}{Z(S_j)} \exp(\Theta \cdot X(d_j|S_j)) \tag{7.10}$$

  where the normalization term is computed over all decisions $\mathcal{D}(S_j)$ allowed at this state:

$$Z(S_j) = \sum_{d \in \mathcal{D}(S_j)} \exp(\Theta \cdot X(d|S_j)) \tag{7.11}$$

Independence assumptions in the model

Feature extraction function X may encode independence assumptions by ignoring information from the inference. These assumptions allow the state space to be compressed by collapsing inferences with the same signature, i.e. the same X-value. If for all inferences i and i' where $i \neq i'$ we have $X(i) \neq X(i')$, then feature extraction retains sufficient information to discriminate between different inferences. In this case, no inferences are equivalent from a modeling perspective. The state space cannot be compressed.

The most common independence assumption is that the history factorizes, i.e. that the state can be decomposed into independent pieces. For example, we may assume that the score of an inference is based only its descendants, i.e. on the items below it in the state. Each inference rule that yields this item (the **consequent**) is based on combining a

set of inferred items that are its immediate children (its **antecedents**). These immediate children, in turn, preserve their respective descendant information. The inferred items are nodes in a directed acyclic hypergraph, and each inference rule that yields an item is a hyperedge from the antecedents to the consequent. This directed acyclic hypergraph is the search space (Gallo and Scutellà, 1999; Klein and Manning, 2001b). This data structure is alternately referred to as a **packed chart** (Geman and Johnson, 2002; Johnson, 2003), a **feature forest** (Miyao and Tsujii, 2002), and an **and/or graph** (Miyao and Tsujii, 2002; Felzenszwalb and McAllester, 2007). In an and/or DAG, the children of a conjunctive node are disjunctive nodes and the children of a disjunctive node are conjunctive nodes.

Nearly all statistical parsers and structure prediction approaches have used independence assumptions as a central design decision. In cases where very strong independence assumptions are made, dynamic-programming may guarantee exact solutions. For example, unlexicalized PCFG approaches (e.g. Taskar et al., 2004b) have worst-case time complexity cubic in the length of the sentence. McDonald et al. (2005a,b, 2006) explore dependency parsing approaches that use a first-order factorization, i.e. in which each edge's score is independent of the others. Under this assumption, non-projective dependency parsing is quadratic and projective dependency parsing is cubic. However, once independence assumptions are relaxed, parsing complexity increases and approximations must be made, e.g. by introducing pruning into search. For example, the Collins model (Collins, 1997, 1999, 2003; Bikel, 2004a) generates parse tree nodes top-down. All non-head children are conditioned on only the parent and the head child and a little more information, yet the search algorithm is $O(n^5)$ without pruning. Second-order non-projective dependency parsing is NP-hard, but even an approximate second-order approach is more accurate than the exact first-order approach (McDonald and Pereira, 2006).

Independence assumptions do not necessarily guarantee exactness, except when they are clearly implausible. For this reason, we believe that independence assumptions might be unnecessary, and that their importance has been overstated. There are several problems with independence assumptions:

- They may reduce the upper-bound on accuracy achievable.
- Crafting them appropriately might require substantial intuition and human effort.
- In complex search scenarios, powerful models might find better solutions with the same search effort as less powerful models. Alternately, having more information early on means more powerful models might require less search effort to find the same solutions as less powerful models. We provide an example of this argument in the text on page 24 describing Figure 3.1. Search effort is an even more important consideration in synchronous parsing for machine translation. A synchronous parser generalized from an ordinary parser based on dynamic-programming will have complexity $O(n^6)$ in the length of the longer sentence in the source or target language (Wu, 1997).

Models used in the literature

Many early statistical parsing approaches use local probabilistic models that are estimated using relative frequencies (Black et al., 1992, 1993; Jelinek et al., 1994; Magerman, 1994, 1995; Collins, 1997, 1999, 2003). Although in principle these models are supposed to be locally normalized, their probabilistic models may be deficient (sum to less than 1) due to smoothing. For example, Bikel (2004a) notes that the Collins models are deficient.

As far as we know, Ratnaparkhi et al. (1994) proposed the task of parse reranking as well as the first log-linear model in the parsing literature. They perform parse reranking using a global log-linear model. The authors argue that the model should be globally normalized. However, it is impractical for them to compute the normalization term over all parses, so they instead normalize only over the top k parses provided.

Ratnaparkhi subsequently proposed using locally-normalized conditional log-linear models for parsing (Ratnaparkhi, 1997, 1999). Charniak (2000) proposes a parsing approach that uses a local conditional log-linear model. He describes the normalized version of the model, but notes that normalization is the most expensive part of training. This sentiment is reiterated by LeCun et al. (2007, Section 6.1). To avoid this expense during training, Charniak (2000) merely omits computing the normalization constant. In this respect, the model of Charniak (2000) is the most similar model to ours.

Johnson et al. (1999) use globally-normalized conditional models for parse reranking. One limitation of their approach is that, to compute the partition function, they enumerate all possible parses for a given sentence. This is problematic because enumerating all such parses for a given sentence using a broad-coverage grammar can be prohibitively expensive. The authors experiment with small enough training sets that explicit enumeration is feasible, and candidate parses come from a packed chart that was provided to the authors by Ron Kaplan and Hadar Shemtov at Xerox PARC. Johnson et al. (1999) experiment with two corpora. The larger of the two has 980 sentences, 481 of which are ambiguous under the grammar, giving 3169 parses of the ambiguous sentences and 227 features.

Lafferty et al. (2001) was the first work to show how to use globally-normalized conditional models in structure prediction tasks that require inference. To overcome the presence of label bias in MEMMs (McCallum et al., 2000), Lafferty et al. (2001) propose conditional random fields (CRFs), a globally-normalized conditional model for sequence labeling. The authors claim that CRFs can be generalized to stochastic CFGs. CRFs can condition on arbitrary features of the input. One of the limitation of CRFs, however, is that they *cannot* condition on arbitrary previous decisions, i.e. the history. This restrictive independence assumption allows efficient training and decoding using dynamic-programming. Training uses a variant of the forward-backward algorithm and decoding is done using the Viterbi algorithm.

Geman and Johnson (2002), Miyao and Tsujii (2002), and Johnson (2003) subsequently proposed similar techniques for training globally-normalized conditional models for parsing. Their approaches crucially rely on independence assumptions that allow the search space

to be represented compactly as a directed acyclic hypergraph (Gallo and Scutellà, 1999; Klein and Manning, 2001b). The contribution of Geman and Johnson (2002), Miyao and Tsujii (2002), and Johnson (2003) is not this data structure, but rather an algorithm for efficient training of globally-normalized conditional models over this data structure. Like with CRFs, training uses a variant of the forward-backward algorithm and decoding is done using the Viterbi algorithm. Miyao and Tsujii (2002) argue that CRFs are a special case of this approach for and/or graphs in which each conjunctive node has only one daughter.

Geman and Johnson (2002), Miyao and Tsujii (2002), and Johnson (2003) note that the weaker the independence assumptions are, the larger the search hypergraph will be, and the less efficient dynamic-programming during training and decoding will be. In the extreme case where there are no independence assumptions, no inferences share the same signature and no compression of the search space can occur. The contributions of Geman and Johnson (2002) and Johnson (2003) are theoretical, and they do not report on experiments. Because their approach has significant memory requirements, Miyao and Tsujii (2002) present experimental results on a relatively small data set: 868 sentences and 5,715 features, with each sentence's forest having an average of 17,412 nodes. This experiment requires 1.5 GB of memory. Clark and Curran (2003, 2004, 2007) using the dynamic-programming based techniques of Geman and Johnson (2002), Johnson (2003), and Miyao and Tsujii (2002) for training globally normalized log-linear conditional models. By parallelizing training across many computers with synchronization, i.e. message-passing, the authors are able to train over larger data sets. Synchronization is needed to compute the normalization term. For example, Clark and Curran (2003) perform an experimenting use 36,400 sentences and 243,603 features, with each sentence's forest having an average of 52,000 nodes, requiring 30 GB of memory total.

7.2 Training techniques for different models

Training models for structure prediction typically involves optimizing loss functions that express desiderata for good structure predictors. We will discuss common loss functions below.

As far as we know, all globally normalized probabilistic models in the literature have been optimized to maximize the likelihood of the gold-standard parse. This expresses the desideratum that the correct parse receive high probability, and all other parses receive low probability. As discussed in the previous section, as far as we know all globally-normalized conditional log-linear structure prediction approaches impose restrictive independence assumption that allow efficient training and decoding using dynamic-programming. Training uses a variant of the forward-backward algorithm and decoding is done using the Viterbi algorithm. Two other approaches, Ratnaparkhi et al. (1994) and Henderson (2004), approximate the expectation using the k-best parses. Ratnaparkhi et al. (1994) use a globally-normalized log-linear conditional model for reranking. In one experiment, Henderson (2004)

train a globally-normalized joint model to maximize the conditional probability of each gold-standard parse given its sentence.

For local models, we noted that early probabilistic models were estimated using relative frequencies (Black et al., 1992, 1993; Jelinek et al., 1994; Magerman, 1994, 1995; Collins, 1997, 1999, 2003; Bikel, 2004a). These can be viewed as **offline** example generation techniques because they do not require inference. Only local constraints are enforced at each state. Most subsequent structure prediction approaches using local models including ours have been trained using local constraints generated offline (Ratnaparkhi, 1997, 1999; Charniak, 2000; Henderson, 2004; Sagae and Lavie, 2005; Titov and Henderson, 2007a,b). These approaches have all used deterministic logics. For a non-deterministic logic, we pick a single correct path randomly. The general form of these local constraints is to enforce the desideratum that at any correct state the local model will make the correct local decision and stay on the correct path. For probabilistic models this corresponds to maximizing the likelihood of the correct parse because the likelihood of each correct inference is maximized. An advantage of this method of generating training examples is that it does not require a working inference engine and can be run prior to any training. A disadvantage of this approach is that it does not teach the model to recover from mistakes. To more explicitly describe the relationship between offline training example generation and what occurs online, i.e. during inference at test time, observe that the parse decisions present in the training set are identical to the inferences that would be scored by our search algorithms using an oracle cost function.[1] An oracle cost function assigns cost 0 to all correct inferences and ∞ to all incorrect inferences.

There is a notable exception to the trend of local models being trained only under local constraints selected offline. Henderson (2004) uses locally-normalized joint and conditional models for parsing. He trains three types of parsers:

- A joint model trained under offline local constraints to maximize the joint probability of the parses and sentences in the treebank.

- A conditional model trained under offline local constraints to maximize the conditional probability of each gold-standard parse in the treebank given its sentence.

- A joint model trained to maximize the conditional probability of each gold-standard parse in the treebank given its sentence. As in the global conditional models, this requires computing an expectation over all parses for a given sentence. Henderson (2004) estimates this expectation using the probabilities of the k-best parses under the model and the probabilities of partial parses that were pruned during search for the k-best parses.

[1] For breadth-driven search algorithms, instead of cost-driven search algorithms like ours, this observation may not hold. In general, the parse decisions generated by this offline technique are identical to the inferences that would be scored by a greedy search algorithm using an oracle cost function.

As far as we know, the work of Henderson (2004) presents the only structure prediction experiments using both local and global training techniques for the same model. Unfortunately, because his global training technique is compromised for the sake of efficiency, it is difficult to make a direct comparison.

There is another family of approaches to training using inference, which has traditionally been used for training **distribution-free** structure prediction models. Collins (2004) analyzes various parameter estimation techniques through the lens of statistical learning theory. In this framework, we assume that some fixed unknown distribution generates both the training and test data, each point independently and identically distributed. The probabilistic models we have considered are **parametric** models that attempt to model this underlying distribution explicitly, either in a joint or conditional formulation. In the limit, as training data increases to infinity, maximum-likelihood estimation chooses a model that converges to the underlying distribution under the assumption the underlying probability distribution is in the family of distributions being modeled. Distribution-free methods do not make these assumptions. Principled distribution-free methods can nonetheless provide theoretical guarantees on the generalization accuracy of a model. These guarantees are based on the margin of the model. The **margin** of a model on a particular instance is a measure of the distance between the score of the correct output and the score of all incorrect outputs. A model that has large margins for a large percentage of training examples will tend to generalize accurately. For the 0-1 loss, Collins (2004) presents an optimization problem with an exponential number of constraints: For each of the exponential number of incorrect parses, it must be separated from the correct parse by a margin of at least μ. He goes on to show how the perceptron algorithm can estimate parameters under this optimization problem by using inference to choose only the single most-violating constraint at each step. Other approaches using large-margin formulations have optimization problems with an exponential number of constraints. Parameter update is typically intractable under all these constraints. However, inference can be used to select the most-violating constraints, which are used for parameter update. We will discuss the work of Taskar et al. (2004a,b) (Section 7.2), McDonald et al. (2005a,b, 2006) (Section 7.2), Tsochantaridis et al. (2004, 2005) (Section 7.2), Collins and Roark (2004) (Section 7.2), and Daumé III et al. (2005, 2006) (Section 7.2).

Taskar et al. (2004a,b)

Taskar et al. (2004a) describe a maximum margin method for estimating the parameters of a Markov network. They focus on conditional Markov networks, a.k.a. CRFs (Lafferty et al., 2001). Probabilistic graphical models compactly represent joint distributions over elements of an output structure by assuming that interactions are local but not completely independent. Maximum-margin methods have high accuracy, can be kernelized to use high-dimensional features, and offer theoretical generalization guarantees. Taskar et al. (2004a) propose so-called max-margin Markov networks (M^3Ns), which are designed to have the

advantages of both probabilistic graphical models and max-margin methods on structure prediction problems.

Max-margin estimation for a particular structure loss function can be performed by solving an optimization problem with a number of constraints based on the size of the output space, which is perhaps exponential. This estimate comes with a theoretical bound on generalization error. If we assume that elements in the output structure are not highly inter-dependent and that the loss factorizes according to the dependencies in the output structure, we can reformulate the primal problem as a dual formulation with a polynomial-size formulation. This problem might be too large for conventional quadratic program (QP) solvers, so instead Taskar et al. (2004a) performs coordinate descent using a structured analogue of sequential minimal optimization (SMO). Violated Karush-Kuhn-Tucker (KKT) conditions are found using inference.

Taskar et al. (2004b) extends the max-margin approach of Taskar et al. (2004a) to unlexicalized CFGs. The approach forgoes a probabilistic interpretation, and instead ensures that the highest-scoring parse is the gold-standard parse. Like approaches based on Markov models (McCallum et al., 2000; Lafferty et al., 2001; Taskar et al., 2004a), the model can condition on arbitrary features of the input but not the output. Because parse nodes are unlexicalized, a dynamic-programming solution is exact and requires cubic time. We have discussed how dynamic-programming-based globally-normalized conditional models use variants of the inside-outside algorithm to compute expectations over all parses. The parser of Taskar et al. (2004b) forgoes a probabilistic interpretation, is unnormalized, and requires calculations of Viterbi trees only. On the downside, unlexicalized CFGs are not powerful models. The globally-normalized conditional log-linear models are motivated by the desire to use more powerful features. Trading modeling power for better estimation can increase accuracy in the short-term, but might reduce the upper-bound on accuracy achievable.

Taskar et al. (2004b) define a quadratic program in which each incorrect parse is separated from the gold-standard parse by a margin that is proportional to the loss, i.e. incorrectness, of the incorrect parse. This program includes non-negative slack variables to account for outliers. There are as many variables and constraints in the optimization problem as there are possible parse trees. However, Taskar et al. (2004b) place restrictions on the features and the loss function that allow a clever factorization of the dual to polynomial size. Taskar et al. (2004a) used this decomposition for sequences and other Markov random fields, and Taskar et al. (2004b) extends this idea to CFGs. They assume that the feature vector and loss function factorize according the same independence assumptions as CFGs. The same decomposition can then be performed over the dual problem. The output structure is composed of "parts." There are two types of parts:

- Constituent items, i.e. labelled spans.
- Binarized context-free rule tuples, each the labels and spans of the parent and two child items.

These two types of parts are referred to as "edges" and "traversals," respectively, in the literature (Klein and Manning, 2001a). These parts are independent insofar as the features and loss function are concerned. This restriction on the features and the loss allows the dual to be factored to a polynomial size:

- The number of variables is cubic in the length of the sentence.
- The number of constraints is quadratic in the length of the sentence.
- The number of coefficients in the quadratic term of the objective is quadratic in the number of sentences and dependent on the sixth power of the length of the sentence.

As in Taskar et al. (2004a), the dual is optimized using an online coordinate descent method analogous to SMO. For each sentence, the best parse is found under the current model, and the weights are adjusted.

McDonald et al. (2005a,b, 2006)

These works study the problem of unlabeled dependency parsing. Their work proceeds from the desire for exact inference algorithms. Under the restrictive independence assumption that the score of a dependency tree factors into the sum of scores over edges, unlabeled dependency parsing reduces to the problem of finding the maximum spanning tree (MST) in a directed graph. Inference complexity is $O(n^2)$ for 1-best non-projective dependency parsing ($O(n^6)$ for k-best) and $O(n^3)$ for projective dependency parsing using a bottom-up dynamic-programming chart parsing algorithm ($+O(k \log k)$ for k-best).

Following Taskar et al. (2004a,b), who use a dynamic-programming-based factorization to decompose the optimization into a polynomial number of local constraints, these works factor the dependency parsing optimization into one constraint per edge. This generates a number of constraints quadratic in the length of the sentence. A quadratic number of constraints is few enough that using inference for further constraint subset selection is not necessary. At each iteration, for a given sentence the training algorithm performs an online update to the weight vector. The update minimizes the changes to the weight vector under the constraint that, for each node, the score for each correct incoming edge is separated by a certain fixed margin from the score of each incorrect incoming edge. This formulation assumes uniform loss for each incorrect edge. The final weight vector is an average of intermediate weight vectors, which reduces overfitting (Collins, 2002).

Later work (McDonald and Pereira, 2006) argues that first-order factorization is a very restrictive assumption. m^{th}-order projective MST parsing has complexity $O(n^{m+1})$ for $m > 1$. However, second-order non-projective MST parsing is NP-hard, so McDonald and Pereira (2006) devise a special-purpose approximate algorithm.

Tsochantaridis et al. (2004, 2005)

Like the work of Taskar et al. (2004a,b), this parameter estimation technique for structure prediction models uses the maximum-margin principle. The main contribution of this work is generalizing max-margin formulations to arbitrary loss functions. Whereas Taskar et al. (2004a,b) restricted the loss and features by assuming that it factorizes according to the dependencies in the output structure, Tsochantaridis et al. (2004, 2005) place no such restrictions on the loss function or features, allowing one to optimize the true objective directly.

Tsochantaridis et al. (2004, 2005) consider linear models, which can be kernelized in the dual formulation. They give three different types of margin maximization optimization criteria:

hard-margin SVMs Maximize the margin by which the correct solution is separated from every other solution. This can be represented as a quadratic program in which the ℓ_2-norm of the weight vector is minimized while, for each incorrect output, enforcing the linear constraint that the correct output and this incorrect output are separated by margin at least μ.

soft-margin SVMs Because the data might not be linearly separable, we can pay a penalty for each margin violation using a slack variable. The authors present two soft-margin formulations. In the first, margin violations are penalized using a linear penalty term. In the second, margin violations are penalized using a quadratic term.

loss-sensitive SVMs This general case subsumes the hard- and soft-margin formulations. The loss-sensitive SVMs come in two flavors:

slack rescaling Slack variables are rescaled by the loss incurred. This corresponds to the intuition that high-loss margin violations should incur higher penalty than low-loss margin violations.

margin rescaling Taskar et al. (2004b) proposes rescaling the margin to include the Hamming loss into a max-margin formulation. Tsochantaridis et al. (2004, 2005) generalize this principle to arbitrary loss functions.

As with soft-margins, the authors present variations on the loss-sensitive SVMs that penalize margin violations using a linear penalty term, as well as using a quadratic term.

In all of the formulations, the number of margin constraints is proportional to the number of incorrect outputs, which may be exponential. Although the optimization problem may have exponential size, the authors propose a training algorithm that, under certain conditions, finds arbitrarily good solutions to all the above SVM optimization problems in a polynomial number of steps. The idea is to take advantage of the max-margin structure of

the optimization problems. It can be shown that by explicitly satisfying an appropriately-chosen polynomial-sized subset of the constraints, the solution will be sufficiently accurate, i.e. all remaining constraints will be violated by no more than ϵ. A working set of constraints is maintained for each training instance. By doing so, the authors "construct a nested sequence of successively tighter relaxations of the original problem using a cutting plane method." (pg. 1462 of Tsochantaridis et al. (2005)) The relaxation of the problem is defined by the current working set of constraints selected. The training algorithm iterates over the training instances. For each training instance, it finds the most violated constraint under the current model. The cost of violating this constraint is sensitive to the choice of loss function and SVM formulation. If this constraint violation's cost is at least ϵ higher than the cost of violating each other constraint in the training instance's working set, then the new constraint is added to the training instance's working set. This corresponds to strengthening the primal problem by choosing a cutting plane that cuts off the current primal solution. The model is optimized under the new constraints. The algorithm stops when no constraints are added for any training instance, i.e. when no constraint is violated by more than ϵ.

Observe that training requires inference to generate a working set of constraints. The authors note that finding the most violating constraint might be the bottleneck in training, requiring more time even than solving the relaxed QP. Additionally, finding the most violating constraint might require modifying the inference algorithm. In the case of the zero-one loss, the most violating constraint is given by the highest scoring incorrect output. However, for other loss functions, the most violating constraint might not necessarily be highest scoring incorrect output, and the inference algorithm might need to be extended. The hidden difficulty in using arbitrary loss is that computing the most violating constraint for this loss and SVM formulation might be difficult. We might have algorithms to find the solution with maximum score, but it might be difficult to extend these algorithms to find solutions with maximum cost under some choice of loss function and SVM formulation. The authors present experiments where they estimate CFG parameters using SVMs that maximize margins using the F_1 loss. They use a variant of the Cocke-Younger-Kasami (CKY) algorithm (e.g. Aho and Ullman, 1972, pp. 314–320) to find the most violating constraint during training.

Collins and Roark (2004)

This parser's model is a linear discriminant whose parameters are estimated using the averaged perceptron algorithm (Collins, 2002). The parser attempts to find the Viterbi parse for each sentence. The logic is incremental, left-to-right, over a tree transformed using a selective left-corner transform and flattened. This logic is deterministic. Cost is non-monotonic, i.e. the cost of a partial parse might decrease as inferences are made during parsing. The parser uses breadth-first search with a beam to prune unlikely parse candidates.

The training algorithm integrates search and learning. Learning optimizes with respect

to the search strategy used at test time. Training requires inference for example generation. In the standard perceptron update (Collins, 2002), for each sentence the algorithm finds the highest-scoring solution under the current model and, if this solution is incorrect, updates the parameters using an additive perceptron update. Tsochantaridis et al. (2004, 2005) show that this constraint selection technique is valid for minimizing the 0-1 loss in a hard-margin SVM formulation. However, their theoretical guarantee might not apply when perceptron update is applied. Perceptron updates minimize the 0-1 error, i.e. all trees that do not match the gold-standard are treated as equally bad. The primary limitation of this approach is that, unless the true evaluation measure is complete tree match and not F_1, training optimizes a loose approximation of the true evaluation measure. In particular, many "gold-standard" trees in the treebank contain noise. For these trees, the perceptron algorithm cannot learn to prefer incorrect solutions with high F_1 to incorrect solutions with low F_1. The 0-1 loss is insensitive to the true loss.

Another limitation of the perceptron approach is that parameter estimates are unregularized. Collins and Roark (2004) use a small set of common features, so this might not pose a problem in their experiments. However, if there are rare features, model parameters could grow in magnitude until they overwhelm the influence of every other feature.

Because the inference step in training is very slow, Collins and Roark (2004) propose two ad-hoc improvements to the training procedure: "early update" and "repeated use of hypotheses."

early update Consider the possibility that, after j words in the sentence have been processed during search, a search error is made. We assume we mean the minimum such j. If the beam does not contain the partial analysis of the gold-standard tree involving the first j words because it has been pruned, a search error has occurred. At this point, search is terminated. The incorrect partial analyses in the beam are passed to the parameter estimation method. The parameter estimation method will then perform perceptron update with respect to the highest-scoring incorrect partial analysis.

The motivation for early update is as follows: If parameter update is performed over *complete* incorrect analyses, then penalty attribution is distributed over all incorrect decisions. However, given the specific search algorithm used in this work, correcting incorrect decisions after a search error occurs cannot help the parser to find the correct solution. Ideally, learning should attribute penalty as precisely as possible to the decision(s) that incur loss. Since the perceptron minimizes the 0-1 error, the earliest possible moment at which loss is definitely incurred is when there is a search error, so that is where update focuses. As mentioned, this optimization is designed specifically for the deterministic left-to-right logic and this particular search strategy.

Note that if noise in the gold-standard always prevents the model from advancing further right in the analysis, information further to the right can never be learned.

Early update makes an "enormous difference in the quality of the resulting model (pg. 6 of 8)", increasing F_1 by roughly 2.5%.

repeated use of hypotheses Similarly to Tsochantaridis et al. (2004, 2005), Collins and Roark (2004) cache the constraints generated for every sentence. Recall that early update might generate several partial analyses for each sentence, only the highest-scoring of which is used for each perceptron update step. The perceptron update is applied for five passes over each sentence in turn to the most-violating constraint, but no more than fifty times per sentence. These limits are imposed because the perceptron algorithm only converges on linearly separable data, and outliers might be linearly inseparable.

Daumé III et al. (2005, 2006)

This work proposes SEARN, a meta-algorithm that attempts to systematically improve local models for structure prediction that use a deterministic logic. Like the approach of Tsochantaridis et al. (2004, 2005), SEARN is applicable for any loss function and set of features. It requires only a cost-sensitive learning algorithm for binary classification.

The outer loop of the training algorithm is as follows: At each iteration the **policy** is a mixture of the optimal policy (the oracle cost function) and the learned policy (induced cost function). At the first iteration of training, the policy is solely the optimal policy. As training progresses, the influence of the optimal policy is degraded and the learned policy receives increasing importance. The policy mixture is used to choose an output for each training input. These outputs are used to generate cost-sensitive training examples. There is one training example for each inference in each output. Its cost is the lower-bound increase in loss due to choosing this inference. The learning algorithm is used to induce a classifier over these cost-sensitive training examples. This new classifier is mixed in to the policy. At the last iteration, the last classifier induced is returned.

The first iteration of SEARN is similar to the offline example generation technique used by local structure-prediction models such as ours. SEARN can be viewed as a method of iterating these training techniques to improve accuracy of local structure-prediction models and to teach them how not to compound error by making reasonable decisions in incorrect states.

❧ Eighth Chapter ❧

Conclusions

> Once the whole is divided, the parts need names.
> There are already enough names.
> One must know when to stop.
> Knowing when to stop averts trouble.
>
> Lao Tzu

This dissertation has presented a general approach to constituent parsing, which is driven by classification. The advantage of our approach is its flexibility:

- It is simple to substitute in different parsing strategies.
- We used little language-specific linguistic information: only the head rules and the POS tag classes.
- Our example generation technique does not require a working parser from which to bootstrap, and can be performed entirely offline.
- The regularization penalty factor λ is optimized as part of the training process, choosing the value that maximizes accuracy on a held-out tuning set. This technique stands in contrast to more ad-hoc methods, which might require prior knowledge or additional experimentation.
- The human effort is reduced for engineering powerful feature sets.
- Wellington et al. (2006) and Turian et al. (2007) applied our classification-driven learning approach to machine translation and presented the first purely discriminative learning algorithm for translation with tree-structured models. Our approach might also be useful for other structure prediction problems.

Our primary contribution is simplifying the human effort required for feature engineering. Because approaches that reduce the amount of feature-engineering required are simpler to apply to new tasks and new domains, we will discuss this contribution is more depth.

- Features can use arbitrary information from the state and input. Features that access arbitrary information are represented directly without the need for an induced intermediate representation (cf. Henderson, 2003, 2004; Titov and Henderson, 2007a,b).
- Feature selection and feature construction occur automatically, as part of learning. There is no need for crafting model parameterizations / independence assumptions that balance model power with the ability to estimate parameter values properly. We defined a set of fine-grained atomic features, and let the learner induce informative compound features. The atomic features need not be powerful in isolation.
- Even when building decision trees with many training examples and many splitting features, ℓ_1 regularization keeps the models sparse. For this reason, no frequency-based cutoffs are needed, and all words in the lexicon are used. On the other hand, ℓ_2-regularized decision trees were too large to fit in memory.
- In short, we spent little time on feature engineering. We defined a feature template and automatically extracted feature types from the training data that satisfied this template. Yet we observed no overfitting, even when the learning rate was increased. Our features used minimal linguistic cleverness. Nonetheless, our parser surpassed the generative baseline. As far as we know, it is the first discriminative parser to surpass the Collins (2003) model without using a generative model in any way.

Besides simplicity of feature engineering, our work provides some machine learning contributions that improve the efficiency of discriminative training:

- Classifiers for different non-terminal labels can be induced independently and hence training can be asynchronously parallelized. This is possible because the examplewise loss decomposes, and can be optimized piece-wise. We calibrate our classifiers by tying the regularization penalty factor λ across different classifiers.
- The regularization penalty factor λ is optimized it as part of the training process, choosing the value that maximizes accuracy on a held-out tuning set. This technique stands in contrast to more ad-hoc methods, which might require prior knowledge or additional experimentation.
- Risk functions like the ℓ_1 penalty are continuous, but not continuously differentiable. As far as we know, previous work on gradient descent in function space have either worked with continuously differentiable cost functions (e.g. Mason et al., 1999, 2000), or have not shown how to derive gain functions for risk functions of this type. We have presented an analysis technique for performing gradient descent in function space to minimize risk functions that are continuous but not continuously differentiable. We are not aware of previous work on gradient descent in function space that has used this analysis technique. We use this technique to derive the gain values given in Perkins et al. (2003) and Riezler and Vasserman (2004).

- We show how gradient descent in function space can be regularized. Feature selection is regularized by including the gradient of the penalty in the gain function. We show how the penalty's gradient can be included in the gain, even if the gradient is not continuous. We show that explicitly regularizing feature selection leads to faster training than performing unregularized feature selection and then using regularization during confidence-rating.

- Using our regularized gain function, we propose an algorithm for boosting decision trees that incorporates ℓ_1 regularization during feature selection. As far as we know, explicitly incorporating ℓ_1 regularization during *compound* feature selection is novel.

- The main engineering challenge in building this parser was fitting the training data into memory. We introduce gradient sampling, which—in conjunction with parallelization—increased training speed 100-fold. Our training method does feature selection more quickly by using principled sampling to estimate unbiased risk gradients (Section 5.2). As far as we know, performing feature selection using unbiased gradient estimates is novel.

The resulting model at the core of the parser is a machine learned to optimize a single regularized objective. It includes no generative model. The problems with using a generative model include:

- It might be difficult to account for certain features in a generative story.

- Constructing a generative model for new tasks and new domains might be time-consuming.

- Using a generative model introduces unregularized risk, thereby potentially decreasing the potential upper-bound on achievable accuracy. A discriminative model can use a generative model as a feature to achieve accuracy at least as high as the generative model (e.g. Collins and Duffy, 2002b; Collins and Roark, 2004). However, the model complexity of the generative component might increase risk, thus *reducing* the upper bound on achievable accuracy. Associating only a single parameter with the entire generative model gives only coarse-grained control to the optimization procedure. The generative model includes many features, each of which is associated with a generatively-tuned parameter. Taken as a whole, the parameter values of the *frequent* features determine the influence of the generative model. However, we speculate that inaccurately estimated and/or poorly smoothed parameters associated with *infrequent* features might introduce "hidden" model complexity.

Finally, we have introduced greedy completion, a new agenda-driven search strategy designed to find low-cost solutions given a limit on search effort. Although agenda-driven search is not guaranteed to find the optimal solution in polynomial time, we found that

it worked well in practice. The number of states in the search space is exponential in the size of the input, and one might worry about search errors. However, in our experiments, the inference evaluation function was learned accurately enough to guide the deterministic parsers to the optimal parse reasonably quickly without pruning, and thus without search errors. Our best model has such low perplexity that, on 80.5% of sentences, its completely greedy solution is optimal. (The completely greedy solution is the one found using breadth-first search with a beam width of 1.) This is in comparison to previous non-DP-based discriminative parsers, which all used best-first search with pruning (Ratnaparkhi, 1999; Henderson, 2004; Collins and Roark, 2004; Titov and Henderson, 2007a,b) or no search at all (Sagae and Lavie, 2005). In the non-deterministic setting, greedy completion found better solutions that best-first search in the allotted time.

The main limitation of our work is that we can do training reasonably quickly only on short sentences because a sentence with n words generates $O(n^2)$ training inferences. Although generating training examples in advance without a working parser is faster than using inference (Collins and Roark, 2004; Henderson, 2004; Taskar et al., 2004b), our training time can probably be decreased further by choosing a parsing strategy with a lower branching factor. Like our work, Ratnaparkhi (1999) and Sagae and Lavie (2005) generate examples off-line, but their parsing strategies are essentially shift-reduce so each sentence generates only $O(n)$ training examples.

Bibliography

Alfred J. Aho and Jeffrey D. Ullman. *The Theory of Parsing, Translation and Compiling*, volume 1. Prentice-Hall, Englewood Cliffs, NJ, 1972. [103]

Adam L. Berger, Stephen A. Della Pietra, and Vincent J. Della Pietra. A Maximum Entropy approach to Natural Language Processing. *Computational Linguistics*, 22(1): 39–71, 1996. [10]

Ann Bies, Mark Ferguson, Karen Katz, Robert MacIntyre, Victoria Tredinnick, Grace Kim, Mary Ann Marcinkiewicz, and Britta Schasberger. Bracketing guidelines for Treebank II style Penn treebank project. Technical report, University of Pennsylvania, 1995. URL `citeseer.ist.psu.edu/bies95bracketing.html`. [60]

Daniel M. Bikel. Intricacies of Collins' parsing model. *Computational Linguistics*, 30(4): 479–511, 2004a. [60, 90, 95, 96, 98]

Daniel M. Bikel. *On the parameter space of generative lexicalized statistical parsing models*. PhD thesis, University of Pennsylvania, Philadelphia, PA, USA, 2004b. [18]

Ezra Black, Steven Abney, Dan Flickenger, Claudia Gdaniec, Ralph Grishman, Philip Harrison, Donald Hindle, Robert Ingria Fred Jelinek, Judith Klavans, Mark Liberman, and Tomek Strzalkowski. A procedure for quantitatively comparing the syntactic coverage of English grammars. In *Speech and Natural Language*, 1991. [8, 17, 33, 60]

Ezra Black, Fred Jelinek, John Lafferty, David M. Magerman, Robert Mercer, and Salim Roukos. Towards history-based grammars: using richer models for probabilistic parsing. In *Proceedings of the DARPA workshop on Speech and Natural Language*, pages 134–139, Morristown, NJ, USA, February 1992. Association for Computational Linguistics. [9, 18, 96, 98]

Ezra Black, Fred Jelinek, John Lafferty, David M. Magerman, Robert Mercer, and Salim Roukos. Towards history-based grammars: Using richer models for probabilistic parsing. In *Proceedings of the Thirty-first Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 31–37, Morristown, NJ, USA, June 1993. Association for Computational Linguistics. [9, 18, 96, 98]

Rens Bod. An efficient implementation of a new DOP model. In *Proceedings of the Eleventh European Chapter of the Association for Computational Linguistics (EACL)*, 2003. [66]

Peter F. Brown, Peter V. deSouza, Robert L. Mercer, Vincent J. Della Pietra, and Jenifer C. Lai. Class-based n-gram models of natural language. *Computational Linguistics*, 18(4): 467–479, 1992. [4]

Sharon Caraballo and Eugene Charniak. New figures of merit for best-first probabilistic chart parsing. *Computational Linguistics*, 24(2):275–298, 1998. [23]

Eugene Charniak. A maximum-entropy-inspired parser. In *Proceedings of the Applied Natural Language Processing (ANLP) of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pages 132–139, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc. [11, 96, 98]

Eugene Charniak and Mark Johnson. Coarse-to-fine n-best parsing and MaxEnt discriminative reranking. In *Proceedings of the Forty-third Annual Meeting of the Association for Computational Linguistics (ACL)*, Morristown, NJ, USA, 2005. Association for Computational Linguistics. [13, 66]

Ciprian Chelba and Frederick Jelinek. Exploiting syntactic structure for language modeling. In *Proceedings of the Thirty-sixth Annual Meeting of the Association for Computational Linguistics (ACL) and Seventeenth International Conference on Computational Linguistics (COLING) 1998*, pages 225–231, August 1998. [4]

Stephen Clark and James R. Curran. Log-linear models for wide-coverage CCG parsing. In *Proceedings of the Eighth Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 97–104, Morristown, NJ, USA, July 2003. Association for Computational Linguistics. [11, 53, 97]

Stephen Clark and James R. Curran. Parsing the WSJ using CCG and log-linear models. In *Proceedings of the Forty-second Annual Meeting of the Association for Computational Linguistics (ACL)*, 2004. [11, 53, 97]

Stephen Clark and James R. Curran. Wide-coverage efficient statistical parsing with CCG and log-linear models. *Computational Linguistics*, 2007. To appear. [11, 97]

Michael Collins. Discriminative reranking for natural language parsing. In Pat Langley, editor, *Proceedings of the Seventeenth International Conference on Machine Learning (ICML)*, pages 175–182, San Francisco, CA, USA, June–July 2000. Morgan Kaufmann Publishers Inc. [13]

Michael Collins. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the Seventh Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2002. [92, 101, 103, 104]

Michael Collins. Head-driven statistical models for natural language parsing. *Computational Linguistics*, 29(4):589–637, 2003. [iv, 10, 60, 90, 95, 96, 98, 108]

Michael Collins. Parameter estimation for statistical parsing models: Theory and practice of distribution-free methods. In Harry Bunt, John Carroll, and Giorgio Satta, editors, *New Developments in Parsing Technology*, volume 23 of *Text, Speech and Language Technology*, chapter 2. Springer, 2004. [11, 39, 99]

Michael Collins. A new statistical parser based on bigram lexical dependencies. In *Proceedings of the Thirty-fourth Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 184–191, 1996. [10]

Michael Collins. Three generative, lexicalized models for statistical parsing. In *Proceedings of the Thirty-fifth Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 16–23, July 1997. [10, 11, 90, 95, 96, 98]

Michael Collins. *Head-Driven Statistical Models for Natural Language Parsing*. PhD thesis, University of Pennsylvania, 1999. [8, 9, 10, 65, 90, 95, 96, 98]

Michael Collins and Nigel Duffy. Convolution kernels for natural language. In Thomas G. Dietterich, Suzanna Becker, and Zoubin Ghahramani, editors, *Proceedings of the Fifteenth Annual Conference on Neural Information Processing Systems (NIPS 14)*, Cambridge, MA, 2002a. MIT Press. [13, 93]

Michael Collins and Nigel Duffy. New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron. In *Proceedings of the Fortieth Annual Meeting of the Association for Computational Linguistics (ACL)*, July 2002b. [93, 109]

Michael Collins and Terry Koo. Discriminative reranking for natural language parsing. *Computational Linguistics*, 31(1):25–69, 2005. [13, 66]

Michael Collins and Brian Roark. Incremental parsing with the perceptron algorithm. In *Proceedings of the Forty-second Annual Meeting of the Association for Computational Linguistics (ACL)*, 2004. [12, 23, 60, 64, 74, 92, 99, 103, 104, 105, 109, 110]

Michael Collins, Robert E. Schapire, and Yoram Singer. Logistic regression, AdaBoost and Bregman distances. *Machine Learning*, 48(1–3):253–285, 2002. [39]

Hal Daumé III and Daniel Marcu. A noisy-channel model for document compression. In *Proceedings of the Fortieth Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 449–456, July 2002. [4, 5]

Hal Daumé III, John Langford, and Daniel Marcu. Search-based structured prediction as classification. In *NIPS Workshop on Advances in Structured Learning for Text and Speech Processing*, 2005. [39, 99, 105]

Hal Daumé III, John Langford, and Daniel Marcu. Search-based structured prediction. *Machine Learning*, 2006. In submission. Available at `http://pub.hal3.name/`. [39, 99, 105]

Miroslav Dudík, Steven J. Phillips, and Robert E. Schapire. Maximum entropy density estimation with generalized regularization and an application to species distribution modeling. *Journal of Machine Learning Research*, 8:1217–1260, June 2007. [10, 46]

Nick Duffield, Carsten Lund, and Mikkel Thorup. Sampling to estimate arbitrary subset sums. Available at `http://arxiv.org/abs/cs.DS/0509026`, 2005. [56, 57]

Jason Eisner and Noah A. Smith. Parsing with soft and hard constraints on dependency length. In *Proceedings of the Ninth International Workshop on Parsing Technologies (IWPT)*, 2005. [66]

Pedro F. Felzenszwalb and David McAllester. The generalized A* architecture. *The Journal of Artificial Intelligence Research*, 29:153–190, June 2007. [23, 24, 95]

Yoav Freund. Boosting a weak learning algorithm by majority. *Information and Computation*, 121(2):256–285, 1995. [57]

Jerome H. Friedman. Stochastic gradient boosting. *Computational Statistics and Data Analysis*, 38(4):367–378, February 2002. [57]

Jerome H. Friedman, Trevor Hastie, and Robert Tibshirani. Additive logistic regression: A statistical view of boosting. *The Annals of Statistics*, 28(2):337–374, 2000. [41, 42, 57]

David Furcy and Sven Koenig. Limited discrepancy beam search. In Leslie Pack Kaelbling and Alessandro Saffiotti, editors, *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 125–131. Professional Book Center, July–August 2005. [30]

David A. Furcy. *Speeding Up the Convergence of Online Heuristic Search and Scaling Up Offline Heuristic Search*. PhD thesis, Georgia Institute of Technology, December 2004. [30]

Giorgio Gallo and Maria Grazia Scutellà. Directed hypergraphs as a modelling paradigm. Technical Report TR-99-02, Università di Pisa, February 1999. [17, 95, 97]

Stuart Geman and Mark Johnson. Dynamic programming for parsing and estimation of stochastic unification-based grammars. In *Proceedings of the Fortieth Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 279–286, July 2002. [11, 95, 96, 97]

Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, 2003. [40]

Patrick Haffner, Steven Phillips, and Rob Schapire. Efficient multiclass implementations of L1-regularized maximum entropy. Available at `http://arxiv.org/abs/cs.LG/0506101/`, 2005. [39, 44]

William D. Harvey and Matthew L. Ginsberg. Limited discrepancy search. In Chris S. Mellish, editor, *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 607–615. Morgan Kaufmann, August 1995. [27, 30]

James Henderson. Inducing history representations for broad coverage statistical parsing. In *Proceedings of the Human Language Technology Conference (HLT) of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pages 24–31, Morristown, NJ, USA, 2003. Association for Computational Linguistics. [12, 13, 92, 108]

James Henderson. Discriminative training of a neural network statistical parser. In *Proceedings of the Forty-second Annual Meeting of the Association for Computational Linguistics (ACL)*, 2004. [12, 13, 23, 64, 92, 94, 97, 98, 99, 108, 110]

Liang Huang and David Chiang. Better k-best parsing. In *Proceedings of the Ninth International Workshop on Parsing Technologies (IWPT)*, 2005. [31]

Martin Jansche. Maximum expected F-measure training of logistic regression models. In *Proceedings of the Human Language Technology Conference (HLT) and Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 692–699, Morristown, NJ, USA, 2005. Association for Computational Linguistics. [10, 34, 40]

Frederick Jelinek, John Lafferty, David Magerman, Robert Mercer, Adwait Ratnaparkhi, and Salim Roukos. Decision tree parsing using a hidden derivation model. In *Proceedings of the ARPA Workshop on Human Language Technology*, pages 272–277, Morristown, NJ, USA, March 1994. Morgan Kaufmann. [8, 9, 30, 96, 98]

Mark Johnson. Learning and parsing stochastic unification-based grammars. In Bernhard Schölkopf and Manfred K. Warmuth, editors, *Proceedings of the Sixteenth Conference*

on Computational Learning Theory (COLT) and the Seventh Kernel Workshop, volume 2777 of *Lecture Notes in Computer Science*, pages 671–683. Springer, August 2003. [11, 95, 96, 97]

Mark Johnson, Stuart Geman, Stephen Canon, Zhiyi Chi, and Stefan Riezler. Estimators for stochastic "unification-based" grammars. In *Proceedings of the Thirty-seventh Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 535–541, June 1999. [11, 13, 93, 96]

Aravind K. Joshi and Phil Hopely. A parser from antiquity. *Natural Language Engineering*, 2(4):291–294, 1996. [13]

Dan Klein and Christopher D. Manning. Parsing with treebank grammars: empirical bounds, theoretical models, and the structure of the Penn Treebank. In *Proceedings of the Thirty-ninth Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 338–345, Morristown, NJ, USA, July 2001a. Association for Computational Linguistics. [101]

Dan Klein and Christopher D. Manning. A* parsing: Fast exact Viterbi parse selection. In *Proceedings of the Human Language Technology Conference (HLT) of the North American Chapter of the Association for Computational Linguistics (NAACL)*, Morristown, NJ, USA, 2003a. Association for Computational Linguistics. [24, 26]

Dan Klein and Christopher D. Manning. Accurate unlexicalized parsing. In *Proceedings of the Forty-first Annual Meeting of the Association for Computational Linguistics (ACL)*, July 2003b. [66]

Dan Klein and Christopher D. Manning. Parsing and hypergraphs. In *Proceedings of the Seventh International Workshop on Parsing Technologies (IWPT)*. Tsinghua University Press, October 2001b. [17, 95, 97]

Kevin Knight and Daniel Marcu. Statistics-based summarization — step one: Sentence compression. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI) and Twelfth Conference on Innovative Applications of Artificial Intelligence (IAAI)*, pages 703–710. AAAI Press / The MIT Press, 2000. [5]

Kevin Knight and Daniel Marcu. Summarization beyond sentence extraction: a probabilistic approach to sentence compression. *Artificial Intelligence*, 139(1):91–107, 2002. [5]

Richard E. Korf. Artificial intelligence search algorithms. In Mikhail J. Atallah, editor, *Algorithms and Theory of Computation Handbook*, chapter 36. CRC Press, Boca Raton, FL, 1998. [27, 30]

Taku Kudo, Jun Suzuki, and Hideki Isozaki. Boosting-based parse reranking with subtree features. In *Proceedings of the Forty-third Annual Meeting of the Association for Computational Linguistics (ACL)*, Morristown, NJ, USA, 2005. Association for Computational Linguistics. [13, 66]

John Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In Carla E. Brodley and Andrea Pohoreckyj Danyluk, editors, *Proceedings of the Eighteenth International Conference on Machine Learning (ICML)*. Morgan Kaufmann, June 2001. [19, 37, 93, 96, 99, 100]

Yann LeCun, Sumit Chopra, Raia Hadsell, Marc'Aurelio Ranzato, and Fu Jie Huang. *A Tutorial on Energy-Based Learning*. MIT Press, September 2007. [94, 96]

Maxim Likhachev, Geoffrey J. Gordon, and Sebastian Thrun. ARA*: Anytime A* with provable bounds on sub-optimality. In Sebastian Thrun, Lawrence Saul, and Bernhard Schölkopf, editors, *Proceedings of the Seventeenth Annual Conference on Neural Information Processing Systems (NIPS 16)*, Cambridge, MA, 2004. MIT Press. [30]

David M. Magerman. *Natural language parsing as statistical pattern recognition*. PhD thesis, Stanford University, Stanford, CA, USA, February 1994. [8, 9, 30, 96, 98]

David M. Magerman. Statistical decision-tree models for parsing. In *Proceedings of the Thirty-third Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 276–283, June 1995. [9, 10, 30, 54, 60, 96, 98]

Daniel Marcu, Wei Wang, Abdessamad Echihabi, and Kevin Knight. SPMT: Statistical machine translation with syntactified target language phrases. In *Proceedings of the Eleventh Conference on Empirical Methods in Natural Language Processing (EMNLP)*, July 2006. [viii, 7, 8]

Mitchell P. Marcus. *Theory of Syntactic Recognition for Natural Languages*. MIT Press, Cambridge, MA, USA, 1980. [16]

Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics*, 19(2): 313–330, 1993. [8, 9, 17, 59]

Llew Mason, Jonathan Baxter, Peter Bartlett, and Marcus R. Frean. Functional gradient techniques for combining hypotheses. In Alexander J. Smola, Peter Bartlett, Bernhard Schökopf, and Dale Schuurman, editors, *Advances in Large Margin Classifiers*, chapter 12. MIT Press, Cambridge, MA, USA, 1999. [42, 44, 49, 84, 108]

Llew Mason, Jonathan Baxter, Peter L. Bartlett, and Marcus R. Frean. Boosting algorithms as gradient descent. In Sara A. Solla, Todd K. Leen, and Klaus-Robert Müller, editors, *Advances in Neural Information Processing Systems 12 (NIPS 1999)*, pages 512–518. The MIT Press, 2000. [42, 44, 45, 49, 108]

Andrew McCallum, Dayne Freitag, and Fernando C. N. Pereira. Maximum entropy Markov models for information extraction and segmentation. In Pat Langley, editor, *Proceedings of the Seventeenth International Conference on Machine Learning (ICML)*, pages 591–598, San Francisco, CA, USA, June–July 2000. Morgan Kaufmann Publishers Inc. [10, 94, 96, 100]

Ryan McDonald and Fernando C. N. Pereira. Online learning of approximate dependency parsing algorithms. In *Proceedings of the Eleventh European Chapter of the Association for Computational Linguistics (EACL)*, pages 81–88, April 2006. [92, 95, 101]

Ryan McDonald, Koby Crammer, and Fernando C. N. Pereira. Online large-margin training of dependency parsers. In *Proceedings of the Forty-third Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 91–98, Morristown, NJ, USA, 2005a. Association for Computational Linguistics. [12, 92, 95, 99, 101]

Ryan McDonald, Fernando C. N. Pereira, Kiril Ribarov, and Jan Hajič. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of the Human Language Technology Conference (HLT) and Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 523–530, Morristown, NJ, USA, 2005b. Association for Computational Linguistics. [12, 92, 95, 99, 101]

Ryan McDonald, Koby Crammer, and Fernando C. N. Pereira. Spanning tree methods for discriminative training of dependency parsers. Technical Report MS-CIS-06-11, University of Pennsylvania, Department of Computer and Information Science, October 2006. [12, 92, 95, 99, 101]

Ron Meir and Gunnar Rätsch. *An introduction to boosting and leveraging*, pages 118–183. Springer-Verlag New York, Inc., New York, NY, USA, 2003. [40]

I. Dan Melamed. Statistical machine translation by parsing. In *Proceedings of the Forty-second Annual Meeting of the Association for Computational Linguistics (ACL)*, 2004. [5]

I. Dan Melamed. Automatic evaluation and uniform filter cascades for inducing N-best translation lexicons. In *Proceedings of the Third ACL Workshop on Very Large Corpora (WVLC)*, 1995. [67]

I. Dan Melamed and Wei Wang. Generalized parsers for machine translation. Technical Report 05-001, Proteus Project, New York University, 2005. `http://nlp.cs.nyu.edu/pubs/`. [5, 6, 16]

Scott Miller, Heidi Fox, Lance Ramshaw, and Ralph Weischedel. A novel use of statistical parsing to extract information from text. In *Proceedings of the Applied Natural Language Processing (ANLP) of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pages 226–233, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc. [2, 3]

Yusuke Miyao and Jun'ichi Tsujii. Maximum entropy estimation for feature forests. In *Proceedings of the Human Language Technology Conference (HLT)*, March 2002. [11, 95, 96, 97]

Andrew Y. Ng. Preventing "overfitting" of cross-validation data. In Douglas H. Fisher, editor, *Proceedings of the Fourteenth International Conference on Machine Learning (ICML)*, pages 245–253. Morgan Kaufmann, July 1997. [60]

Eric W. Noreen. *Computer-Intensive Methods for Testing Hypotheses : An Introduction*. Wiley-Interscience, April 1989. [60]

Joseph C. Pemberton and Richard E. Korf. An incremental search approach to real-time decision making. In *Proceedings of the AAAI Spring Symposium on Decision Theoretic Planning*, pages 218–224, Menlo Park, California, March 1994a. AAAI Press. [31]

Joseph C. Pemberton and Richard E. Korf. Incremental search algorithms for real-time decision making. In Kristian J. Hammond, editor, *Proceedings of the Second International Conference on AI Planning Systems (AIPS-94)*, pages 140–145. AAAI, June 1994b. [31]

Simon Perkins, Kevin Lacker, and James Theiler. Grafting: Fast, incremental feature selection by gradient descent in function space. *Journal of Machine Learning Research*, 3:1333–1356, 2003. [43, 44, 45, 46, 54, 108]

Adwait Ratnaparkhi. A maximum entropy model for part-of-speech tagging. In *Proceedings of the First Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 133–142, 1996. [60, 61, 90, 93]

Adwait Ratnaparkhi. A linear observed time statistical parser based on maximum entropy models. In *Proceedings of the Second Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 1997. [10, 11, 40, 94, 96, 98]

Adwait Ratnaparkhi. *Maximum Entropy Models for Natural Language Ambiguity Resolution*. PhD thesis, University of Pennsylvania, Philadelphia, PA, 1998. [40]

Adwait Ratnaparkhi. Learning to parse natural language with maximum entropy models. *Machine Learning*, 34(1-3), 1999. [10, 11, 23, 40, 53, 64, 94, 96, 98, 110]

Adwait Ratnaparkhi, Salim Roukos, and R. Todd Ward. A maximum entropy model for parsing. In *Proceedings of the Third International Conference on Spoken Language Processing (ICSLP) 1994*, pages 803–806, September 1994. [10, 11, 13, 96, 97]

Stefan Riezler and Alexander Vasserman. Incremental feature selection of ℓ_1 regularization for relaxed maximum-entropy modeling. In *Proceedings of the Ninth Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2004. [43, 45, 46, 108]

Saharon Rosset, Ji Zhu, and Trevor Hastie. Boosting as a regularized path to a maximum margin classifier. *Journal of Machine Learning Research*, 5:941–973, 2004. [84]

Kenji Sagae and Alon Lavie. A classifier-based parser with linear run-time complexity. In *Proceedings of the Ninth International Workshop on Parsing Technologies (IWPT)*, 2005. [53, 54, 64, 98, 110]

Robert E. Schapire and Yoram Singer. Improved boosting using confidence-rated predictions. *Machine Learning*, 37(3):297–336, 1999. [41, 42, 44, 84]

Libin Shen, Anoop Sarkar, and Aravind K. Joshi. Using LTAG based features in parse reranking. In *Proceedings of the Eighth Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 89–96, Morristown, NJ, USA, July 2003. Association for Computational Linguistics. [13]

Ben Taskar, Carlos Guestrin, and Daphne Koller. Max-margin markov networks. In Sebastian Thrun, Lawrence Saul, and Bernhard Schölkopf, editors, *Proceedings of the Seventeenth Annual Conference on Neural Information Processing Systems (NIPS 16)*, Cambridge, MA, 2004a. MIT Press. [12, 93, 99, 100, 101, 102]

Ben Taskar, Dan Klein, Michael Collins, Daphne Koller, and Chris Manning. Max-margin parsing. In *Proceedings of the Ninth Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2004b. [12, 17, 59, 64, 90, 92, 93, 95, 99, 100, 101, 102, 110]

Ann Taylor, Mitchell Marcus, and Beatrice Santorini. The Penn Treebank: An overview. In Anne Abeillé, editor, *Treebanks: Building and Using Parsed Corpora*, chapter 1, pages 5–22. Kluwer Academic Publishers, 2003. [8, 9, 17, 59]

Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society*, 58:267–288, 1996. [36]

Ivan Titov and James Henderson. Constituent parsing with incremental sigmoid belief networks. In *Proceedings of the Forty-fifth Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 632–639. Association for Computational Linguistics, June 2007a. [12, 13, 23, 92, 93, 94, 98, 108, 110]

Ivan Titov and James Henderson. Incremental bayesian networks for structure prediction. In *Proceedings of the Twenty-fourth International Conference on Machine Learning (ICML)*, pages 887–894, New York, NY, USA, June 2007b. ACM Press. [12, 13, 23, 92, 93, 94, 98, 108, 110]

Ioannis Tsochantaridis, Thomas Hofmann, Thorsten Joachims, and Yasemin Altun. Support vector machine learning for interdependent and structured output spaces. In Carla E. Brodley, editor, *Proceedings of the Twenty-first International Conference on Machine Learning (ICML)*. ACM, July 2004. [12, 93, 99, 102, 104, 105]

Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, and Yasemin Altun. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6:1453–1484, 2005. [12, 93, 99, 102, 103, 104, 105]

Joseph Turian and I. Dan Melamed. Constituent parsing by classification. In *Proceedings of the Ninth International Workshop on Parsing Technologies (IWPT)*, 2005. [39, 53, 54]

Joseph Turian and I. Dan Melamed. Advances in discriminative parsing. In *Proceedings of the Forty-fourth Annual Meeting of the Association for Computational Linguistics (ACL)*, 2006a. [54, 59, 60, 90]

Joseph Turian and I. Dan Melamed. Computational challenges in parsing by classification. In *HLT-NAACL workshop on Computationally Hard Problems and Joint Inference in Speech and Language Processing*, 2006b. [57, 78]

Joseph Turian, Benjamin Wellington, and I. Dan Melamed. Scalable discriminative learning for natural language parsing and translation. In *Proceedings of the Twentieth Annual Conference on Neural Information Processing Systems (NIPS 19)*, Vancouver, BC, 2007. [iv, 54, 107]

Benjamin Wellington, Joseph Turian, Chris Pike, and I. Dan Melamed. Scalable purely-discriminative training for word and tree transducers. In *Proceedings of the Seventh Conference of the Association for Machine Translation in the Americas (AMTA)*, August 2006. [iv, 54, 107]

Dekai Wu. Stochastic inversion transduction grammars and bilingual parsing of parallel corpora. *Computational Linguistics*, 23(3):377–404, 1997. [95]

Hiroyasu Yamada and Yuji Matsumoto. Statistical dependency analysis with support vector machines. In *Proceedings of the Eighth International Workshop on Parsing Technologies (IWPT)*, 2003. [53, 54]

Rong Zhou and Eric A. Hansen. Beam-stack search: Integrating backtracking with beam search. In Susanne Biundo, Karen L. Myers, and Kanna Rajan, editors, *Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling (ICAPS)*, pages 90–98. AAAI, June 2005. [23, 30]