Abstract of “Any Domain Parsing: Automatic Domain
Adaptation for Natural Language Parsing” by David McClosky, Ph.D., Brown University, May, 2010.

Current efforts in syntactic parsing are largely data-driven. These methods require labeled examples of syntactic structures to learn statistical patterns governing these structures. Labeled data typically requires expert annotators which makes it both time consuming and costly to produce. Furthermore, once training data has been created for one textual domain, portability to similar domains is limited. This domain-dependence has inspired a large body of work since syntactic parsing aims to capture syntactic patterns across an entire language rather than just a specific domain.

The simplest approach to this task is to assume that the target domain is essentially the same as the source domain. No additional knowledge about the target domain is included. A more realistic approach assumes that only raw text from the target domain is available. This assumption lends itself well to semi-supervised learning methods since these utilize both labeled and unlabeled examples.

This dissertation focuses on a family of semi-supervised methods called self-training. Self-training creates semi-supervised learners from existing supervised learners with minimal effort. We first show results on self-training for constituency parsing within a single domain. While self-training has failed here in the past, we present a simple modification which allows it to succeed, producing state-of-the-art results for English constituency parsing. Next, we show how self-training is beneficial when parsing across domains and helps further when raw text is available from the target domain. One of the remaining issues is that one must choose a training corpus appropriate for the target domain or performance may be severely impaired. Humans can do this in some situations, but this strategy becomes less practical as we approach larger data sets. We present a technique, Any Domain Parsing, which automatically detects useful source domains and mixes them together to produce a customized parsing model. The resulting models perform almost as well as the best seen parsing models (oracle) for each target domain. As a result, we have a fully automatic syntactic constituency parser which can produce high-quality parses for all types of text, regardless of domain.

Abstract of “Any Domain Parsing: Automatic Domain
Adaptation for Natural Language Parsing” by David McClosky, Ph.D., Brown University, May, 2010.

Current efforts in syntactic parsing are largely data-driven. These methods require labeled examples of syntactic structures to learn statistical patterns governing these structures. Labeled data typically requires expert annotators which makes it both time consuming and costly to produce. Furthermore, once training data has been created for one textual domain, portability to similar domains is limited. This domain-dependence has inspired a large body of work since syntactic parsing aims to capture syntactic patterns across an entire language rather than just a specific domain.

The simplest approach to this task is to assume that the target domain is essentially the same as the source domain. No additional knowledge about the target domain is included. A more realistic approach assumes that only raw text from the target domain is available. This assumption lends itself well to semi-supervised learning methods since these utilize both labeled and unlabeled examples.

This dissertation focuses on a family of semi-supervised methods called self-training. Self-training creates semi-supervised learners from existing supervised learners with minimal effort. We first show results on self-training for constituency parsing within a single domain. While self-training has failed here in the past, we present a simple modification which allows it to succeed, producing state-of-the-art results for English constituency parsing. Next, we show how self-training is beneficial when parsing across domains and helps further when raw text is available from the target domain. One of the remaining issues is that one must choose a training corpus appropriate for the target domain or performance may be severely impaired. Humans can do this in some situations, but this strategy becomes less practical as we approach larger data sets. We present a technique, Any Domain Parsing, which automatically detects useful source domains and mixes them together to produce a customized parsing model. The resulting models perform almost as well as the best seen parsing models (oracle) for each target domain. As a result, we have a fully automatic syntactic constituency parser which can produce high-quality parses for all types of text, regardless of domain.

Any Domain Parsing: Automatic Domain Adaptation for Natural Language Parsing

by
David McClosky
B. S., University of Rochester, 2004
Sc. M., Brown University, 2006

A dissertation submitted in partial fulfillment of the
requirements for the Degree of Doctor of Philosophy in
the Department of Computer Science at Brown University.

Providence, Rhode Island
May, 2010

This dissertation by David McClosky is accepted in its present form by
the Department of Computer Science as satisfying the dissertation requirement
for the degree of Doctor of Philosophy.

Date _____ _____

Eugene Charniak, Director

Recommended to the Graduate Council

Date _____ _____

Mark Johnson, Reader
Cognitive and Linguistic Sciences

Date _____ _____

Dan Klein, Reader
University of California at Berkeley

Approved by the Graduate Council

Date _____ _____

Sheila Bond
Dean of the Graduate School

Acknowledgements

I am deeply thankful for the help and support I've received from others as a graduate student. This dissertation couldn't have been done without them (besides, even if it could, it wouldn't have been much fun).

My advisors Eugene Charniak and Mark Johnson were always helpful and insightful. Eugene and Mark's approaches complement each other nicely and I have been fortunate to be able to learn from both of them.

I am also indebted to the many faculty who gave me feedback on my work at conferences and elsewhere. Thanks to Jason Eisner, Tom Griffiths, Dan Klein, Alon Lavie, Joakim Nivre, Brian Roark, and Noah Smith for interesting discussions. Many of these lead to subtle or not-so-subtle changes to my research direction vector.

I will miss the BLLIP research meetings, especially those that opened with (generally groan-inspiring) jokes. My lab mates, Matt Lease, Jenine Turner, William "Sir" Headden (III), Micha Elsner, Sharon Goldwater, Stu Black, Engin Ural, Bevan Jones, David Ellis, Lenora Huang, and Stella Frank ensured that our meetings were always exciting, entertaining, and (nearly always) productive.

Additionally, I would like to thank friends from many different areas of my life for keeping things interesting. Thanks to Allison Moore, Alptekin Küpçü, Antske Fokkens, Benjamin Van Durme, Bronwyn Lovell, Carleton Coffrin, Celine Piser, C. Chris Erway, Collin Cherry, Dan Grollman, Dan Venning, Drew Perttula, Eric Rachlin, Frank Wood, Greg Cooper, Guillaume Marceau, Ian Morrissey, Justin Palmer, Kelsi Perttula, Liz Marai-Renieris, Maggie Benthall, Manos Marai-Renieris, Matt Wronka, Melissa Chase, Mike "Spock" Kass, Mira Belinkiy, Naomi Feldman, Nathan Backman, Reut Tsarfaty, Roi Reichart, Ryan Tarpine, Shane Bergsma, Sharon Goldwater, Stefan Roth, Stella Frank, Steve Bethard, Stu Black, Suman Karumuri, Tom Kollar, Tomer Moscovich, Tony Evans, Tori Sweetser, Warren Schudy, and Yanif Ahmad.

Finally, this would not possible without the love and support from my family. Thanks to my parents Nan and Dan and sister Karine. This dissertation is dedicated to my grandparents, Herb and Mitzi McClosky and Aaron and Shulamis Toder.

Contents

List of Tables

List of Figures

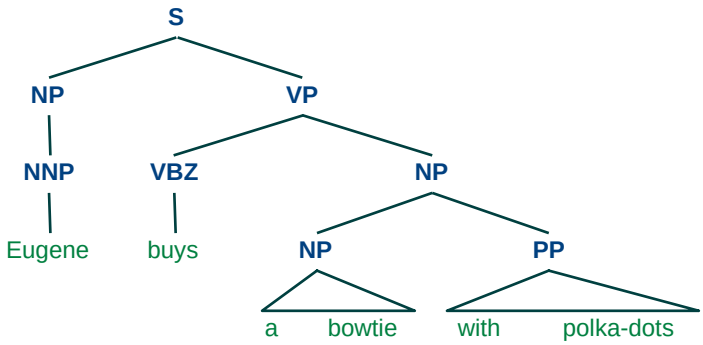
Chapter 1

Introduction

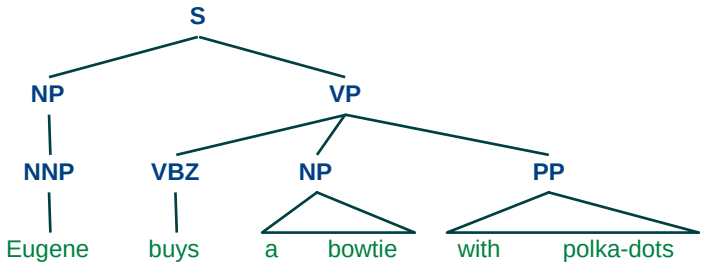
The syntax of natural language characterizes the possible structures and orderings of words within a sentence. It is generally accepted by linguists that the syntactic structure of a sentence is key to determining its meaning. Many theories of semantics state that meaning is compositional and that the compositional units are based on syntactic constructions. This dissertation centers around natural language parsing, which explores algorithms for finding the syntactic structure of sentences. Our parsing mechanism is statistical in nature meaning that it does not make categorical decisions about grammaticality — all structures for a sentence are possible, but some are better than others. Our training data consists of sentences labeled with their correct syntactic structure and we create our parsing models by estimating various statistical distributions over the syntactic structures within these corpora. These statistics can then be applied to new sentences which we assume follow the same distributions.

Current parsers are quite robust provided that the sentences we are parsing actually do follow these distributions. However, the accuracy of most statistical parsers degrades on sentences that have significantly different statistical patterns than the training data. Our goal is to produce parsers which can analyze sentences as well as humans, so this lack of generality is an issue and one of the primary concerns of this dissertation. For example, the text of Stephen Colbert's book "I am America (And So Can You!)" follows different statistical patterns than the text in a transcript of his television show, "The Colbert Report" since the former is written text and the latter is fluid speech. For example, the transcript of the television may be less fluent and contain more informal constructions than the text in the book. While humans do not have difficulty parsing either of these texts, current parsing models are not general enough to span both well. In the parsing literature, this difficulty is usually attributed to a difference in *domain*. By domain, we mean the style, genre, and medium of a document. Thus, Colbert's speech on his show, his writing in his book, and the news stories that inspire his show all come from different domains. Under current approaches, one would need to train a different parsing model for each domain (by and large requiring hand-annotated data) and select the appropriate parsing model for each document. This dissertation explores methods of performing automatic domain adaptation using either hand-annotated or raw text, should annotations be unavailable.

Before going further, we start with a concrete example of the types of structure we hope to learn. Figure 1.1 shows two of the possible parse trees for the sentence "Eugene buys a bowtie with polka-dots." The



(a) Parse tree where the bowtie has polka-dots.



(b) Parse tree where polka-dots are used to make the bowtie purchase.

Figure 1.1: Two parses showing the importance of correct prepositional phrases (PP) attachment.

leaves of the trees (terminals) are the words of the sentences, their parents are their parts-of-speech (preterminals), and the remaining internal nodes represent larger phrases and clauses. Since syntactic constituents are represented as nodes in the trees in our representation, we use the words "node" and "constituent" interchangeably. In both examples, the upper node represents the entire sentence (S) and expands to a noun phrase (NP) and a verb phrase (VP). The noun and verb phrases are in turn broken down into subunits. The triangles under the NPs and PPs indicate that the subtree has been elided for simpler exposition. The parse trees differ in their handling of the prepositional phrase "with polka-dots" (PP) — the upper parse tree attaches it under a noun phrase while the lower parse tree places it under the verb phrase. As a result, the two trees have different semantics. In the upper tree, Eugene is buying a bowtie which is patterned with polka-dots but in the lower tree, polka-dots are (somehow) being used to purchase a bowtie. Outside of computational linguistics articles like this one, few humans would notice the latter reading since the former reading is much more salient. However, we note that the lower tree is not universally incorrect as it represents the standard reading of the sentence "Eugene buys a bowtie with cash." Also note that these two trees are only two many possible parses that a statistical parser would find for this sentence.

Parse trees like those in Figure 1.1 are valuable building blocks for other linguistic applications (Lease et al., 2006). Examples of these include language modeling (Roark, 2001; Charniak, 2001), speech recognition (Chelba and Jelinek, 1998), machine translation (Charniak et al., 2003), dialogue systems, semantic role labeling (Pradhan et al., 2007), information extraction, sentiment analysis (Greene and Resnik, 2009), question answering, summarization (Turner and Charniak, 2005), coreference (Luo and Zitouni, 2005; Charniak and Elsner, 2009), and document coherence (Barzilay and Lapata, 2008; Elsner et al., 2007; Elsner and Charniak,

2008). Outside of computational linguistics, parsing has found applications in biology (Liu et al., 2007; Shi et al., 2007; Medlock, 2008; Miyao et al., 2008; Airola et al., 2008; Kilicoglu and Bergler, 2008) and forensics (van Halteren, 2004; Luyckx and Daelemans, 2008).

1.1 Road map

A brief overview of this dissertation follows. In the remainder of this section, we provide a brief history of statistical natural language parsing (Section 1.2) and previous work on domain dependence (Section 1.3). Section 2 gives an overview of our experimental setup including details of the parser, data sets, and evaluation metrics. We introduce and test self-training, a semi-supervised technique for improving parser performance[1] both in a single domain as well as across domains, in Section 3. Section 4 shows empirical analyses of self-training and explores several hypotheses of how it works. Our proposed work, a method of automatic domain adaptation, is covered in Section 5. Finally, we conclude with a summary and future work in Section 6.

1.2 Statistical Natural Language Parsing

There have been a wide variety of different statistical approaches to parsing natural language. As stated before, on the whole, these methods capture statistical patterns from labeled corpora for the purpose of applying them to new text. However, the specific types of patterns examined, how they're collected, and how they're applied vary greatly. Syntactic structures can be represented under many different formalisms, (context-free grammars, tree adjoining grammars (Schabes, 1992), combinatory categorical grammars (Steedman, 2000), and dependency graphs (Mel'čuk, 1988; McDonald et al., 2005; Nivre and Nilsson, 2005) to name a few) each allowing its own avenues of approaching the problem of parsing. Of course, the parsing formalism used is only one of many dimensions in the space of parsing algorithms. Other dimensions include the type of the model (generative, discriminative, hybrid, non-probabilistic) and the scope of parsing decisions (local or global). We cannot provide complete coverage here and so will only discuss some of the key papers related to our approach.

The first statistical natural language parsers (Black et al., 1993; Jelinek et al., 1994; Magerman, 1995) factored parsing decisions into three categories: part-of-speech tagging, connecting constituents, and labeling constituents. Each type of classification was made by decision trees where nodes in the tree ask questions about the words and previous tagging and parsing decisions. Collins (1997) and Charniak (1997) used generative probabilistic approaches for parsing where probabilistic context free grammars (PCFGs) were extended to include bilexical dependencies and parsing history. Much of the improvements came from annotating nodes with their lexical heads and careful smoothing to handle the sparsity introduced by additional conditioning events. Ratnaparkhi (1999) provided the first discriminative parsing model where a maximum entropy classifier chooses between several local parsing actions conditioned on parsing history and the current state. In addition to being one of the first works on parser adaptation, the paper also points out that the parsing charts contain substantially better possible parses than those found in the Viterbi parse. This has inspired work on

[1] Note that throughout this document, performance will refer to the accuracy of a parser, not its runtime speed.

n-best reranking (Collins, 2000; Charniak and Johnson, 2005) and ultimately forest-based reranking (Huang, 2008). Collins (2000) provided the first n-best reranker trained using the averaged perceptron algorithm (see also Collins and Koo (2005)). Bod (2003) presented an "all-subtrees" approach where the probability of a tree is proportional to the product of the probabilities of its subtrees in the training corpus (where there are several methods of determining each subtree's probability). Neural networks were used to estimate generative and discriminative parsing models in Henderson (2004). The reranking parser by Charniak and Johnson (2005) introduced a new search strategy (the coarse-to-fine parser) and a reranker estimated with a maximum-entropy model. We will discuss these two components in greater depth in Sections 2.1 and 2.2. Petrov et al. (2006) created a refined unlexicalized PCFG grammar by repeatedly splitting and merging nodes in order to improve corpus likelihood. Recently, there have been several attempts to train global discriminative models[2] of parsing (Petrov and Klein, 2008; Finkel et al., 2008; Carreras et al., 2008). Huang (2008) presents forest-based reranking which extends the ideas from n-best reranking to pick constituents from a parse chart instead complete parses in an n-best list.

1.3 Domain Dependence

While parsers have seen substantial improvements in accuracy on in-domain text (i.e. text from the same domain as the training set), their performance on sentences outside of their training domain has not necessarily followed the same trends. This has inspired work on the task of parser adaptation where the goal is to transfer knowledge about one domain to another. Work in parser adaptation is premised on the assumption that one wants a single parser that can handle a wide variety of domains. While this is the goal of the majority of parsing researchers, it is not quite universal. Sekine (1997) observes that for parsing a specific domain, data from that domain is most beneficial, followed by data from the same class, data from a different class, and data from a different domain. He also notes that different domains have very different structures by looking at frequent grammar productions. For these reasons he takes the position that we should, instead, simply create treebanks for a large number of domains. While this is a coherent position, it is far from the majority view.

One benchmark for parser adaptation has been the accuracy of newswire trained statistical parsers on literature. There have been a large number of studies on this task where parsers are trained on the WSJ corpus (Wall Street Journal news) and evaluated on the BROWN corpus (literature) (Ratnaparkhi, 1999; Gildea, 2001; Bacchiani et al., 2006; McClosky et al., 2006b). More details on these corpora can be found in Section 2.3. All of these works look at what happens to modern WSJ-trained statistical parsers (those by Ratnaparkhi, Gildea, Roark, and Charniak and Johnson respectively) as training data varies in size or relevance. We concentrate particularly on the work of (Gildea, 2001; Bacchiani et al., 2006), though these trends are echoed in McClosky et al. (2006b) which will be covered in greater depth in Sections 3.2 and 3.3.

In Table 1.1, we can see the effects of domain dependence. The precise meaning of f-score, our primary evaluation metric, will be covered in Section 2.4 — for now, simply note that higher f-scores mean more accurate parses. When WSJ-trained parsers are evaluated on the BROWN test set instead of the WSJ test set, performance drops by approximately 6% . If we train on the BROWN corpus instead (third row of the table),

[2] As opposed to local discriminative models where parsing decisions are made locally, e.g. (Ratnaparkhi, 1999; Henderson, 2004).

		f-score	
Training	Testing	Gildea (2001)	Bacchiani et al. (2006)
WSJ	WSJ	86.4	87.0
WSJ	BROWN	80.6	81.1
BROWN	BROWN	84.0	84.7
WSJ+BROWN	BROWN	84.3	85.6

Table 1.1: Effects of domain dependence when evaluating on WSJ and BROWN using different combinations of WSJ and BROWN for training. Gildea (2001) evaluates on sentences of length ≤ 40, Bacchiani et al. (2006) on all sentences.

we recover about half of this performance drop.[3] In the final row of the table, we see that there is a moderate improvement if we combine WSJ and BROWN training sets. One might be tempted to conclude that the best course of action is to combine all corpora into one training set. Indeed, as we will see in Section 5.4.1, this technique can obtain quite good performance in general. However, we will see that this is not always beneficial — while WSJ is not too distant from BROWN, other corpora, especially those constructed by parsers rather than humans, are different enough to cause damage.

There are many different approaches to parser adaptation. Steedman et al. (2003b) apply co-training to parser adaptation and find that co-training can work across domains. There is a considerable amount of biomedical text available now. These documents use language that is considerably different from typical training sets that special attention has been given to this domain and the desire to automatically extract key information and more accurately search these documents has inspired various works (Lease and Charniak, 2005; Clegg and Shepherd, 2005; Clegg and Shepherd, 2007). Clegg and Shepherd (2005) provide an extensive side-by-side performance analysis of several modern statistical parsers when faced with such data. They find that techniques which combine different parsers such as voting schemes and parse selection can improve accuracy on biomedical data. Lease and Charniak (2005) use the Charniak parser for biomedical data and find that the use of out-of-domain trees and in-domain vocabulary information can considerably improve performance. Gildea (2001) and Bacchiani et al. (2006) look at how much of an improvement one gets over a pure BROWN system by adding WSJ data (as seen in the last two lines of Table 1.1). Both systems use a "model merging" approach as described by Bacchiani et al. (2006). The different corpora are, in effect, concatenated together. However, Bacchiani et al. (2006) achieve a larger gain by weighting the in-domain BROWN data more heavily than the out-of-domain WSJ data. The above works focus on adapting constituency parsers. There has recently been a lot of attention given to adapting dependency parsers in the CoNLL 2007 Shared Task (Nivre et al., 2007).

[3] The BROWN training set is smaller than the WSJ training set. This may, in part, explain why only half of the performance is recovered.

Chapter 2

Background and Experimental setup

In this section, we describe the components of our experiments. This includes the Charniak and Johnson parser[1] (Charniak and Johnson, 2005), several labeled and unlabeled data sets (corpora), and our evaluation measures for determining the accuracy of candidate parse trees.

Our parsing model consists of two phases. First, we use a probabilistic generative parser to produce a list of the n most probable parses (which we will refer to as an n-best list). Next, a discriminative reranker reorders the parses within the n-best list. These components constitute two views of the data, though the reranker's view is heavily tied to the first stage parser. The reranker can only select parses from within the n-best list and, moreover, uses the probability of each parse tree according to the parser as a feature to perform the reranking. Nevertheless, the reranker's value comes from its ability to make use of more powerful features which would be difficult to express in a generative framework.

For some experiments, we evaluate only the first stage parser's performance to in order to isolate it from the reranker. In other cases, we evaluate the reranking parser as a whole. We distinguish these scenarios by using the term *parser* or *first stage parser* when we use only the generative parser and *reranking parser* for when we use both stages.

2.1 Generative parser

The first stage of our parser is the lexicalized probabilistic context-free parser described in (Charniak, 2000; Charniak and Johnson, 2005). The parser's grammar is a smoothed third-order Markov grammar, enhanced with lexical heads, their parts of speech, and parent and grandparent information. The parser uses five probability distributions, the head's part of speech tag, the head itself, the child constituent which includes the head, and children to the left and right of the child constituent. As all distributions are conditioned with five or more features, they are all heavily backed off using Chen back-off (the *average count* method from Chen and Goodman (1996)) to alleviate data sparsity. The backoff parameters are determined from held out data. Additionally, the statistics are lightly pruned to remove those that are statistically less reliable.

[1] Available for download at `http://bllip.cs.brown.edu/`

The parsing model assigns a probability to a parse π by a top-down process of considering each constituent c in π and, for each c, first guessing the preterminal (part of speech tag) of c, $t(c)$, then the lexical head of c, $h(c)$, and then the expansion of c into further constituents $e(c)$. Thus the probability of parse π is given by the equation

$$\begin{aligned} P(\pi) &= \prod_{c \in \pi} P(t(c) \mid l(c), \mathcal{H}(c)) \\ &\quad \cdot P(h(c) \mid t(c), l(c), \mathcal{H}(c)) \\ &\quad \cdot P(e(c) \mid l(c), t(c), h(c), \mathcal{H}(c)) \end{aligned}$$

where $l(c)$ is the label of c (e.g., whether it is a noun phrase (NP), verb phrase (VP), etc.) and $\mathcal{H}(c)$ is the relevant history of c — information outside c that the probability model deems important in determining the probability in question. $\mathcal{H}(c)$ may contain (among other possible features) the parent's part of of speech, the grandparent's head, and/or the part-of-speech of the previous sibling node.

For each expansion, $e(c)$, we distinguish one of the children as the "middle" child $M(c)$. $M(c)$ is the constituent from which the head lexical item h is obtained according to deterministic rules (called head finding rules) that pick the head of a constituent from among the heads of its children.[2] To the left of M is a sequence of one or more labels $L_i(c)$ including the special termination symbol $\triangle$ and similarly for the labels to the right, $R_i(c)$. Thus, an expansion $e(c)$ looks like: (all symbols in the following are functions of c which we have omitted for simplicity)

$$l \rightarrow \triangle L_m ... L_1 M R_1 ... R_n \triangle.$$

The expansion is generated by guessing first M, then L_1 through L_{m+1} (= $\triangle$) in order, and similarly for R_1 through R_{n+1}. In practice, we condition only on the previous three constituents generated rather all constituents between the current constituent and $M(c)$, forming a Markov grammar for expansions.

As in Charniak and Johnson (2005), the parser can produce an n-best list rather than a single parse. However, the n-best parsing algorithm described in that paper has been replaced by the much more efficient algorithm described in (Jiménez and Marzal, 2000; Huang and Chiang, 2005). For our experiments, we use the 50 most probable parses unless stated otherwise. This parameter value was chosen from evaluations on development data (Charniak and Johnson, 2005). Increasing the size of the n-best list improves performance but with diminishing returns.

2.2 Maximum Entropy Reranker

The second stage of our parser is a Maximum Entropy reranker as described in (Charniak and Johnson, 2005). The reranker takes the n-best parses for each sentence produced by the first stage generative parser and selects

[2] This may sound like a non-generative process since in order to expand constituent c, we need to know the middle child of c which relies on the children of c. However, the head finding rules are only used to mark the heads in the training data. The model is responsible for generating the middle children at test time. In a sense, the head finding rules are encoded as part of the grammar.

the highest scoring parse according to the its model. It does this using the reranking methodology described in Collins (2000), using a Maximum Entropy model with Gaussian regularization as described in Johnson et al. (1999). The reranker classifies each parse with respect to a large number features (typically about 1.3 million — most of which only occur on few parses). Features are defined by abstract feature schemas which produce specific feature instantiations when given the training data as input. The features we use consist of those described in Charniak and Johnson (2005), together with an additional feature schema for EDGE features. EDGE features consist of the parts-of-speech, possibly together with the words, that surround (i.e., precede or follow) the left and right edges of each constituent with the goal of capturing some distituent information. Other examples of reranker feature schemas include NGRAMTREE, (depth-limited subtrees of the original tree) RULE, (context-free rules with varying amounts of context) COPAR, (how many coordinated structures are parallel) and, most importantly, NLOGP (the log-probability of the tree from the first stage parser). As one can see, some of these features could be implemented in the first stage parser, but others (e.g. COPAR and EDGE features) would be nearly impossible to capture without introducing serious data sparsity.

Given a data set with training and development divisions, we create a reranker in the following way. First, we parse the training portion of the data set with 20-fold cross-validation. This gives us n-best lists for sentences in each fold of the data set as parsed by a model trained on the other 19 folds (one of these 19 folds is used as development data). Within an n-best list, each parse is either a *winner* (if its f-score matches the highest f-score within the n-best list — note that there can be multiple winners) or a *loser*. Our next step is to select features that distinguish winners from losers at least five times or more in the training set. This pruning step helps remove unpredictive and overly specific features. The n-best lists along with their gold parses are fed to a numerical optimizer to estimate feature weights. The regularization weights are tuned by evaluating on the development portion of the data set.

2.3 Corpora

In this work, there are two large classes of corpora: labeled and unlabeled. Each labeled corpus consists of a set of sentences which have been tagged with part of speech tags and bracketed into a constituency structure like those in Figure 1.1. Human annotators were trained and labeled the trees according to a set of standard guidelines (Bies et al., 1995; Bies et al., 2005). Unlabeled corpora are raw text and do not include part of speech tags, constituent brackets, or even sentence boundaries.[3] The names, descriptions, and basic statistics of the labeled and unlabeled corpora are included in Tables 2.1 and 2.2, respectively. "Tokens/type" is the average number of word tokens we've seen of each word type. A high number in this column indicates the corpus includes a large amount of vocabulary repetition, though this is also a function of corpus length — shorter corpora have a lower maximum tokens/type value. Note that these statistics are taken over each complete treebank and in many cases there are conventions for dividing the treebanks into training, development, and test sections. The rest of this section describes each corpus in further detail.

Our primary source of labeled data is the Penn Treebank (Marcus et al., 1993). The Penn Treebank

[3] For all unlabeled corpora, sentence boundaries were induced via a simple discriminative classifier trained from a portion of the labeled corpora.

Name	Description	Total sentences	Avg. sentence length	Tokens/type
WSJ	Newspaper	43,594	25.5	26.6
BROWN	Multiple genres	24,243	20.0	18.0
SWBD	Phone conversations	104,482	9.2	62.3
BNC	Multiple genres	1,000	28.3	3.9
ETT	Translated broadcast news	4,834	25.6	14.8
GENIA	Biomedical articles	10,848	27.5	19.9

Table 2.1: Summary of treebanked corpora and basic statistics.

Name	Description	Total sentences	Avg. sentence length	Tokens/type
NANC	Newspaper	23,075,637	23.2	407.5
GUTENBERG	Literature	687,782	26.2	83.1
BIOBOOKS	Biology textbooks	79,540	22.5	32.7
MEDLINE	Biomedical articles	278,192	27.2	41.5

Table 2.2: Summary of unlabeled corpora and basic statistics.

includes the Wall Street Journal (WSJ), BROWN, and SWBD corpora. WSJ collects approximately 40,000 sentences of newspaper stories from the newspaper of the same name in 1989. The corpus is divided into 25 sections, numbered 0 through 24. WSJ has become the de facto standard for statistical parser evaluation in English. Traditionally, sections 2–21 are used for training parsers, section 24 is used for held-out development (though some authors use 0 or 22), and section 23 is used for final evaluation.

The BROWN corpus (Francis and Kučera, 1979) consists of many different genres of text, intended to approximate a "balanced" corpus. While the complete BROWN corpus consists of domains in both the fiction and nonfiction categories, the sections that have been labeled with parse trees are primarily those containing fiction. Examples of these sections include science fiction, humor, romance, mystery, adventure, and "popular lore." We use the same divisions as Bacchiani et al. (2006), who base their divisions on Gildea (2001). Each division of the corpus consists of sentences from all available genres. The training division consists of approximately 80% of the data, while held-out development and testing divisions each make up 10% of the data. The treebanked sections contain approximately 25,000 sentences (458,000 words).

Switchboard (SWBD) is a collection of "spontaneous conversations" recorded from telephone calls. Participants in each telephone call were asked to converse about one of 52 possible topics. We ignore the actual audio portion of this corpus and use only the transcripts of these conversations along with their syntactic trees. These trees include disfluency information to indicate speech repairs and related acts. Parsing speech is a task unto itself and not the focus of this work. Thus, for most of our experiments, we assume that our corpora have had their speech repairs excised (e.g. as in Johnson et al. (2004)). Note, however, that many of the techniques in this dissertation could be incorporated into systems that jointly parse and perform speech repairs.

Like the BROWN corpus, the British National Corpus (BNC) aims to approximate a balanced corpus with a large number of genres using British English instead of American English. The full BNC contains over 100 million words but lacks syntactic annotations. A small subset of the sentences have been annotated by (Foster

and van Genabith, 2008; Foster and Dickinson, 2009).[4] The sentences were chosen randomly, so each one is potentially from a different domain.

The English Translation Treebank, ETT (Bies, 2007), is the translation of Arabic broadcast news from 2005. The English translations and the syntactic annotations were created by humans as opposed to any automatic mechanisms. In terms of domains, the corpus can be thought of as containing elements of both WSJ and SWBD. Many entities in the corpus are left as transliterated Arabic resulting in a unique and unusual vocabulary.

The GENIA treebank (Tateisi et al., 2005) is a corpus of abstracts from the Medline database selected from a search with the keywords "human," "blood cells," and "transcription factors." Medline[5] is a large database of abstracts from a wide variety of biomedical literature. Thus, the GENIA treebank data are all from a small domain within biology. Since a new version of this treebank was produced during our earlier experiments, we regrettably use two different versions of this treebank. Previous work and some of our earlier experiments use GENIA beta 2 while our more recent experiments use the larger version of GENIA which is a superset of the earlier treebank. Our division of the larger GENIA treebank is available online.[6]

We now turn to our unlabeled corpora. The North American News Text corpus, NANC (Graff, 1995), is roughly the unlabeled equivalent of WSJ and consists of approximately 24 million sentences from several news agencies. Our experiments use at most the first three million sentences from NANC. NANC contains no syntactic information. We perform some basic cleanups on NANC to ease parsing (some orthographic normalization). NANC contains news articles from various news sources including the Wall Street Journal, the New York Times, Los Angeles Times, and others. In our experiments, we only use articles from the LA Times. Note that WSJ and NANC do not overlap: The WSJ stories are from 1989 whereas NANC covers the period from 1994–1998.

Our GUTENBERG corpus is 214 randomly selected books from Project Gutenberg.[7] Project Gutenberg transcribes books which have entered into the public domain and releases them in machine-readable formats. Our selected books cover a broad range of subjects, including books such as "An Icelandic Primer," "Celtic Literature," "The Poetical Works of Henry Kirke White," and "Miss Parloa's New Cook Book." Unfortunately, the project does not track the year that each book was written, but copyright law requires that none of these books are especially modern (unless they were published without a restrictive license).

We created a corpus of seven online biology textbooks (BIOBOOKS). One textbook is a general biology textbook while others focus on more specific topics (bacteriology, biochemistry, or immunology). Various processing has been applied to extract the text from its HTML source while excluding figures and tables. The corpus contains almost 80,000 sentences.

MEDLINE is an unlabeled corpus of biomedical article abstracts we have collected from 50 different biomedical journals in the same online database as GENIA.[8] It contains approximately 270,000 sentences —

[4] `http://nclt.computing.dcu.ie/~jfoster/resources/`, downloaded January 8th, 2009.

[5] `http://www.ncbi.nlm.nih.gov/PubMed/`

[6] `http://bllip.cs.brown.edu/download/genia1.0-division-rel1.tar.gz`

[7] `http://www.gutenberg.org/`

[8] We use the EFetch interface: `http://www.ncbi.nlm.nih.gov/corehtml/query/static/efetchlit_help.html`

a random selection of about 31,000 abstracts from Medline. Since these articles were chosen randomly, they span a large number of biomedical subdomains, not just those domains present in GENIA.

For experiments in Chapter 5, we preprocessed these corpora to remove many of the differences in annotation. Examples of these changes include standardizing the use of tags like NML, NX, NAC, converting HVS and BES to VBZ (since they only appear in SWBD), removing EDITED nodes in SWBD, BROWN, and ETT, and removing rare tags from BROWN such as AUX and NEG. Nevertheless, it is inevitable that annotation differences remain as the corpora were each created under slightly different annotation guidelines by different annotators. One of the larger outstanding issues is the structure within noun phrases which some corpora annotate (e.g. GENIA and ETT) while others do not. When parsing across these corpora, the parser produces too much or too little internal structure in noun phrases, thus lowering our accuracy (thus, our parsers may be performing even better than we report). Additionally, while we aimed to normalize tag sets, we did not address how individual words are tagged across different corpora. To remove further differences of annotation or formalism, there are several works on the subject. Smith and Eisner (2009) addresses the issue of structural mismatches by using a quasi-synchronous grammar. The quasi-synchronous grammar allows them to automatically learn the rules to transform treebanks from one annotation style to another. Since a given set of sentences tend to be annotated under only one annotation scheme, they use automatic parses as an approximation of the other annotation scheme. Boyd et al. (2008) describes how to detect many structural errors from inconsistent annotations and Dickinson (2009) shows how dependency errors can be automatically corrected. To address mismatches in tagging, Dickinson and Jochim (2008) discusses a method for determining reliable tagging patterns. Niu et al. (2009) show how to convert a treebank in one formalism (dependency structures) to another (constituency) without heuristic rules. This could be useful for obtaining additional corpora to use as test sets.

2.4 Evaluation

Our evaluation follows the PARSEVAL standards (Black et al., 1991) for constituency evaluation. To evaluate a candidate parse against its human-annotated counterpart, (called the *gold parse*) we consider the overlap between candidate and gold constituents. Each constituent is represented as a labeled span (e.g. words 7 to 12 bracketed by an NP). We ignore the root node since it is trivial and preterminal nodes since they are considered outside the syntactic structure. Given these two lists of constituents, we can calculate the labeled *precision* and *recall*. Precision tells us how many of the nodes in the candidate tree are present in the gold parse whereas recall tells us how many nodes in the gold parse were present in the candidate tree. Precision and recall can be formally defined in terms of the number of true positives (TP), false positives (FP), and false negatives (FN).

$$\begin{aligned} \text{labeled precision (LP)} &= \frac{\text{TP}}{\text{TP} + \text{FP}} \\ \text{labeled recall (LR)} &= \frac{\text{TP}}{\text{TP} + \text{FN}} \end{aligned}$$

In many cases, we seek a single number summarizing performance. f-score (sometimes spelled F_1 or

F-measure) is the weighted harmonic mean of precision and recall (van Rijsbergen, 1979). The most general form is parameterized by β which determines the relative importance of precision and recall. By setting $\beta = 1$, we weight them equally which is the standard practice for parser evaluation. One of the reasons that the harmonic mean is used instead of the arithmetic mean is that the harmonic mean of two numbers is 0 if either of these numbers is 0. Thus, a good f-score requires both good precision and good recall scores (it is trivial to achieve a perfect precision score — simply don't return any constituents).

$$F_\beta = \frac{(1+\beta^2)\cdot(\mathrm{LP}\cdot\mathrm{LR})}{(\beta^2\cdot\mathrm{LP}+\mathrm{LR})}$$

$$f\text{-score} = F_1 = \frac{2\cdot\mathrm{LP}\cdot\mathrm{LR}}{\mathrm{LP}+\mathrm{LR}}$$

Two other statistics that we can compute given candidate and gold parse trees are *exact match* and *crossing brackets*. Exact match is the percentage of candidate parses that are identical to their gold parses (i.e. those with f-score = 1). Crossing brackets is the average number of constituent spans that cross between the candidate and gold parses. Given two constituent spans (x, y) and (a, b) where $x \leq y$, $a \leq b$, we say that they cross if $a < y$ or $x < b$. We use `evalb`[9] and Sparseval[10] (Roark et al., 2006) to obtain these measures. While f-score is useful for measuring the quality of a single candidate against the gold standard, we typically have multiple candidate parses for a given sentence due to our usage of an n-best parser. One measure that is commonly employed is the notion of the *oracle* f-score. Given a list of n candidate parses of a sentence and its corresponding gold parse, the oracle f-score is defined as the f-score of the highest scoring parse in the n-best list. This measure is useful as an upperbound on reranker performance.

To determine whether the difference in f-score between two sets of candidate parses is statistically significant, we use a randomized permutation test based on the randomization of the paired sample t-test described by Cohen (1995). We use Dan Bikel's implementation[11] as well as an in-house version by Mark Johnson (this is not to imply that either implementation is incorrect). For this test, the null hypothesis is that the two sets of candidate parses (call them A and B) were produced by the same model and that each sentence is equally likely to have either score. In each iteration of the test, we create two sets of scores which we'll call X and Y. The ith sentence in X, X_i, is given with equal probability either the score A_i or B_i and Y_i is assigned the other. Thus, we end up with two sets of scrambled scores. Next, we recalculate the score function in question (typically f-score in our case) for X and Y and take the difference. Over k iterations, we count how many times $|\mu(X) - \mu(Y)| \geq |\mu(A) - \mu(B)|$ where μ is the score function. If this occurs j times, our p-value for this test is $(j + 1)/(k + 1)$.

[9] `http://nlp.cs.nyu.edu/evalb/`

[10] `http://cslu.cse.ogi.edu/people/roark/papers_cv.html`

[11] `http://www.cis.upenn.edu/~dbikel/software.html`

Chapter 3

Self-training

In this section, we introduce *self-training*, a simple semi-supervised learning technique which can be used to improve parser performance. We first present a general form of self-training and review other semi-supervised learning techniques. Next, we show how self-training can be used to achieve state-of-the-art performance for parsing (Section 3.1). Finally, we demonstrate that self-training can also be applied successfully to the tasks of parser portability and parser adaptation (Sections 3.2 and 3.3 respectively).

To perform self-training, one needs a collection of labeled and unlabeled data, a labeling function which labels an unlabeled datum according to a specific model, and a training function which given labeled data creates a new model. The process is outlined in Algorithm 1. First, we create a base model from available labeled data which we use to label the unlabeled data. The resulting automatically labeled data is then treated as truth and combined with the actual labeled data to train a new model. Optionally, one may weight the labeled data more highly than the automatically labeled data in this combination. If there is significantly more unlabeled data than labeled data, this may be necessary to ensure that the labeled data is not completely washed out. Another variation is to only select a portion of the automatically labeled data ($\sigma(\cdot)$ in the pseudocode performs this operation). Ideally, one could select only the more reliable elements of the data without biasing the resulting distribution too heavily. In practice, this is often difficult and $\sigma(\cdot)$ ends up being the identity function. Self-training can be iterated over different sets of unlabeled data if desired: the self-trained model in one iteration becomes the base model in the next (Algorithm 1 thus shows the steps for a single iteration).

Semi-supervised learning has attracted much attention in recent years and has inspired a wide variety of approaches. A survey of these techniques can be found in Zhu (2007) and the ACL 2008 Semi-supervised Learning tutorial.[1] The Semi-supervised Learning for Natural Language Processing Workshop (Wang et al., 2009) was formed in response to this interest. Most semi-supervised learning approaches can be categorized as either bootstrapping, (which includes self-training and usually involves minimizing a proxy for error on the unlabeled data) graph regularization, (where the problem is expressed as a graph and unlabeled data provides a mechanism of smoothing the graph) or structural (where auxiliary problems which are predictive for the

[1] `http://ssl-acl08.wikidot.com/`

Input: $labeled\ data$, unlabeled data, weighting parameter α
Output: **self-trained model**

$\mathbf{base\ model} \leftarrow \mathsf{train}(labeled\ data)$
$autolabeled\ data \leftarrow \mathsf{label}(\mathbf{base\ model}, \text{unlabeled data})$
$selected\ autolabeled\ data \leftarrow \sigma(autolabeled\ data)$
$combined\ data \leftarrow \alpha \times labeled\ data + selected\ autolabeled\ data$
$\mathbf{self\text{-}trained\ model} \leftarrow \mathsf{train}(combined\ data)$
return **self-trained model**

Algorithm 1: Pseudocode for one iteration of self-training

original task are used to label the unlabeled data). We focus primarily on bootstrapping techniques here, but note that other approaches are often used in the field as well (Bennett and Demiriz, 1998; Nigam et al., 2000; Ng and Cardie, 2003; Zhu et al., 2003; Mihalcea, 2004; Ando and Zhang, 2005b; Ando and Zhang, 2005a; Blitzer et al., 2006; Fraser and Marcu, 2006; Deoskar, 2008; Koo et al., 2008; Wang et al., 2008).

3.1 Self-training for Parsing

Self-training has been attempted several times for parsing, usually without success. To our knowledge, the first reported use of self-training for parsing is by Charniak (1997). He used his parser trained on WSJ to parse 30 million words of unparsed news text from a different corpus. He then trained a self-trained model from the combination of the newly parsed text with WSJ training data. However, the self-trained model did not improve on the original model. Our work differs in that we use a different first stage parser and consider combining it with a reranker.

A close relative of self-training is *co-training* which is due to Blum and Mitchell (1998). Unlike self-training, co-training requires multiple learners, each with a different "view" of the data. Each view should provide a complementary interpretation and in the strongest case, these would be conditionally independent. When one learner is confident of its predictions about a data point, we add that data point with its predicted label to the training set of the other learners. A variation suggested by Dasgupta et al. (2001) is to add data points to the training set when multiple learners agree on the label. If this is the case, we can be more confident that the data was labeled correctly than if only one learner had labeled it since each learner has reached the same analysis via a different path.

Sarkar (2001) investigated using co-training for parsing under the LTAG formalism. The author shows that using about 10,000 labeled sentences and a large number of unlabeled sentences, co-training can be employed to raise performance from 70.6% f-score to 79.8% f-score on the test section of WSJ.

Self-training and co-training were subsequently investigated for parsing as part of the 2002 CLSP Summer Workshop at Johns Hopkins University (Steedman et al., 2003a). The study suggests that this type of co-training is most effective when small amounts (500-10,000 sentences) of labeled training data is available. Another one of their conclusions was that co-training can be used to improve parser portability. They experimented with several different parameter settings. In all cases, they performed multiple iterations of self-training and the number of sentences parsed per iteration was relatively small (30 sentences). The largest

Input: WSJ, NANC, weighting parameter α
Output: **self-trained parser model**

parser model $\leftarrow \mathsf{train}_1(\text{WSJ})$
$parsed$ NANC $\leftarrow \mathsf{parse}_1(\textbf{parser model}, \text{NANC})$
$combined\ data \leftarrow \alpha \times \text{WSJ} + parsed$ NANC
self-trained parser model $\leftarrow \mathsf{train}_1(combined\ data)$
return self-trained parser model

Algorithm 2: Pseudocode for self-training with first stage parser

Input: WSJ, NANC, weighting parameter α
Output: **self-trained parser model**

parser model $\leftarrow \mathsf{train}_1(\text{WSJ})$
reranker model $\leftarrow \mathsf{train}_2(\text{WSJ})$
$parsed$ NANC $\leftarrow \mathsf{rrp}(\textbf{parser model}, \textbf{reranker model}, \text{NANC})$
$combined\ data \leftarrow \alpha \times \text{WSJ} + parsed$ NANC
self-trained parser model $\leftarrow \mathsf{train}_1(combined\ data)$
return self-trained parser model

Algorithm 3: Pseudocode for self-training with reranking parser

amount of labeled training data (*seed size*) they used was 10,000 sentences from WSJ, though many experiments used only 500 or 1,000 sentences. They found that under these settings, self-training did not yield a significant gain.

In a closely related study, self-training and co-training were evaluated for part of speech tagging in Clark et al. (2003). Their conclusions are quite similar — co-training helps only when there are limited amounts of training data. Self-training either has a small positive effect, no effect, or a large negative effect depending on the specific tagger and seed size.

The unsupervised adaptation experiment by Bacchiani et al. (2006), initially presented in Roark and Bacchiani (2003), is the only previous successful instance of self-training for parsing that we have found. The authors use a parser trained on BROWN to parse WSJ along with additional unlabeled sentences from other years of the WSJ. The new parses are mixed into the BROWN training data as in Algorithm 2. This technique improves performance on the WSJ test set from 75.7% to 80.6%. Our experiments tend to focus on the opposite direction — using WSJ to parse BROWN.

3.1.1 Experiments

Our first self-training experiment uses the training portion of WSJ as our labeled data and variable amounts of NANC as the unlabeled data. We replace the general functions in Algorithm 1 with their more specific counterparts. Let $\mathsf{train}_n(\ell)$ be a function which trains the nth stage of the parser from labeled sentences ℓ and returns the new model. Let $\mathsf{parse}_n(u,m)$ be a function which parses unlabeled sentences u using model m and returns its parses of the sentences. As before, n indicates which stage to use, so the function for the reranking parser looks like:

$$\mathsf{rrp}(m_1, m_2, u) = \mathsf{parse}_2(\mathsf{parse}_1(u, m_1), m_2)$$

where m_1 is the generative parser's model and m_2 is the reranker's model. To replicate the results of Charniak (1997), we first use only the first stage parser. Thus, we use train_1 for train and parse_1 for as the label function (Algorithm 2). The "$+$" and "$\times$" operators warrant some discussion. The WSJ training data (sections 2-21) is combined with the NANC data in the following way: The count of each parsing event is the (optionally weighted) sum of the counts of that event in Wall Street Journal and NANC. Bacchiani et al. (2006) show that count merging is more effective than creating multiple models and calculating weights for each model (model interpolation). Intuitively, this corresponds to concatenating our training sets, possibly with multiple copies of each to account for weighting.[2] Note that the selection step has been removed. We refer to this scenario as "parser-best self-training" since we use the best parse according to the first stage parser.

Algorithm 3 shows the pseudocode for self-training with a reranker ("reranker-best self-training"). Here, our experiments depart from previous work. The primary difference is that the reranking parser is used to parse NANC instead of the first stage parser. Note that while both stages are trained from WSJ only the first stage is retrained from the combination of WSJ and NANC data. We attempted to retrain the reranker using the self-trained sentences, but found no significant improvement.

We evaluated the first stage models created by parser-best and reranker-best self-training by parsing held out WSJ data (section 22). Table 3.1 shows the difference in performance when using parser-best versus reranker-best models. Adding parser-best sentences of NANC reproduces previous self-training efforts and confirms that this strategy is not beneficial. However, we see a large improvement from adding reranker-best sentences. For our remaining experiments, we only use reranker-best self-training.

One may expect to see a monotonic improvement from this technique, but this is not quite the case, as seen when we add 1,000,000 sentences. This may be due to some sections of NANC being less similar to WSJ or containing more noise (NANC is quite noisy, including portions which are indistinguishable from line noise). Another possibility is that these sections contain harder sentences which we cannot parse as accurately and thus are not as useful for self-training.

We also attempt to discover the optimal number of sentences to add from NANC. Much of the improvement comes from the addition of the initial 50,000 self-trained trees. Recall that the experiments in (Steedman et al., 2003a) use a comparatively small amount of unlabeled data.[3] As we add more data, it appears that the maximum benefit to parsing accuracy by strictly adding reranker-best sentences is about 0.7% and that *f*-scores asymptotes around 91.0%. We return to this when we consider the relative weightings of WSJ and NANC data.

So far, we have only evaluated the first stage parser with the self-trained models. We now turn to the performance of the reranking parser. One hypothesis we considered is that the reranked NANC data had incorporated some of the features from the reranker. If this were the case, we would not see an improvement

[2] This implementation has the unfortunate requirement that all weights must be integers, of course. Combining corpora according to arbitrary distributions takes a bit of engineering and employed in Chapter 5.

[3] Most experiments perform 100–120 rounds of self- or co-training, adding 30 unlabeled sentences per round.

Sentences of NANC added	Parser-best	Reranker-best
(baseline) 0	90.3	90.3
50,000	90.1	90.7
250,000	90.1	90.7
500,000	90.0	90.9
750,000	89.9	91.0
1,000,000	90.0	90.8
1,500,000	90.0	90.8
2,000,000	—	91.0

Table 3.1: f-scores after adding either parser-best or reranker-best sentences from NANC to WSJ training data. While the reranker was used to produce the reranker-best sentences, we performed this evaluation using only the first stage parser (parse_1) to parse all sentences from section 22. We did not train a model where we added 2,000,000 parser-best sentences.

	WSJ section		
Sentences of NANC added	1	22	24
(baseline) 0	91.8	92.1	90.5
50,000	91.8	92.4	90.8
250,000	91.8	92.3	91.0
500,000	92.0	92.4	90.9
750,000	92.0	92.4	91.1
1,000,000	92.1	92.2	91.3
1,500,000	92.1	92.1	91.2
1,750,000	92.1	92.0	91.3
2,000,000	92.2	92.0	91.3

Table 3.2: f-scores from evaluating the reranking parser on three held-out sections after adding reranker-best sentences from NANC to WSJ training. These evaluations were performed on all sentences.

when evaluating a reranking parser on the same models. In Table 3.2, we see that our improvements from using self-trained parses and from using the reranker are orthogonal.

Up to this point, we have only considered giving our true training data a relative weight of one. A relative weight of n is equivalent to using n copies of a corpus, i.e. an event that occurred x times in the corpus would occur $x \times n$ times in the weighted corpus. Thus, larger corpora dominate smaller corpora of the same relative weight in terms of event counts. Increasing the weight of the WSJ data should improve, or at least not hurt, parsing performance. Indeed, this is the case for both the parser (Figure 3.1a) and reranking parser (Figure 3.1b). We considered assigning WSJ a relative weight of 1 through 5 while varying the number of sentences included from NANC.[4] Putting more weight on the WSJ trees ensures that the counts of our events are closer to our more accurate data source while still incorporating new statistics from NANC. While it appears that the performance still levels off after adding about one million sentences from NANC, the curves corresponding to higher WSJ weights achieve a higher asymptote.

Looking at the performance of these weighting schemes across sections 1, 22, and 24, we decided that the best combination of training data is to give WSJ a relative weight of 5 and use the first 1,750,000 reranker-best

[4] We arbitrarily chose a relative weight of 5 as our stopping point. However, we expect that there are diminishing returns for higher weights.

Model	Parser alone	Reranking parser
Charniak and Johnson (2005)	—	91.0
Current baseline (WSJ)	89.7	91.3
WSJ + NANC	91.0	92.1

Table 3.3: f-scores on all sentences in WSJ section 23. "WSJ + NANC" represents the system trained on WSJ training (with a relative weight of 5) and 1,750,000 sentences from the reranker-best list of NANC.

sentences from NANC.

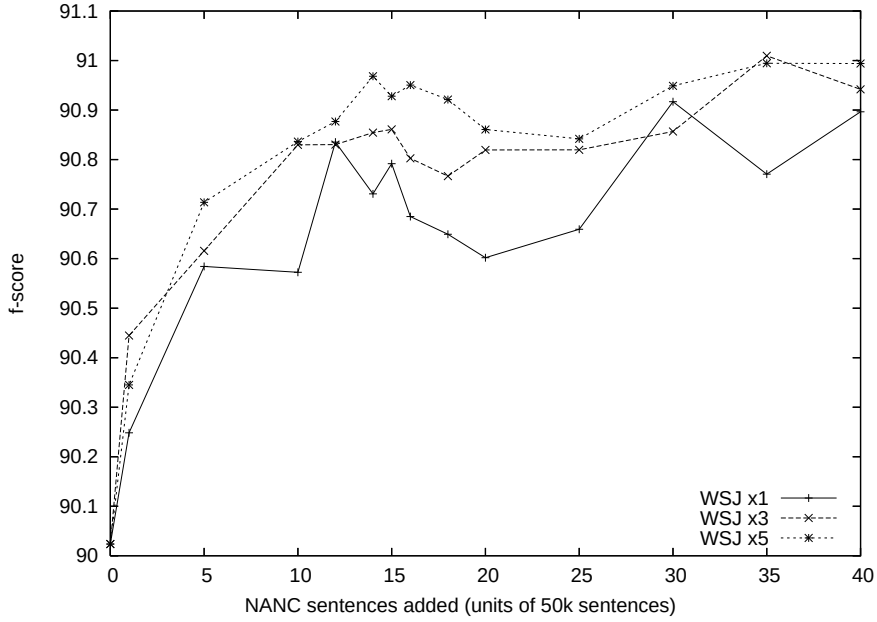
Finally, we evaluate our new model on the test section of Wall Street Journal in Table 3.3. We note that the baseline system (i.e. the parser and reranker trained purely on WSJ) has improved by 0.3% over Charniak and Johnson (2005). The improvement from self-training is significant in both macro and micro tests ($p < 10^{-5}$). We have shown that self-training can provide a substantial benefit when the training and testing data are drawn from the same domain. This study has raised the question of whether the parsing models are too finely tuned for parsing WSJ at the expense of portability to other genres. Such worries have merit. The next section should alleviate these concerns. In it, we show that self-training and reranking are also effective means of improving performance across domains in addition to within them.

3.2 Parser Portability

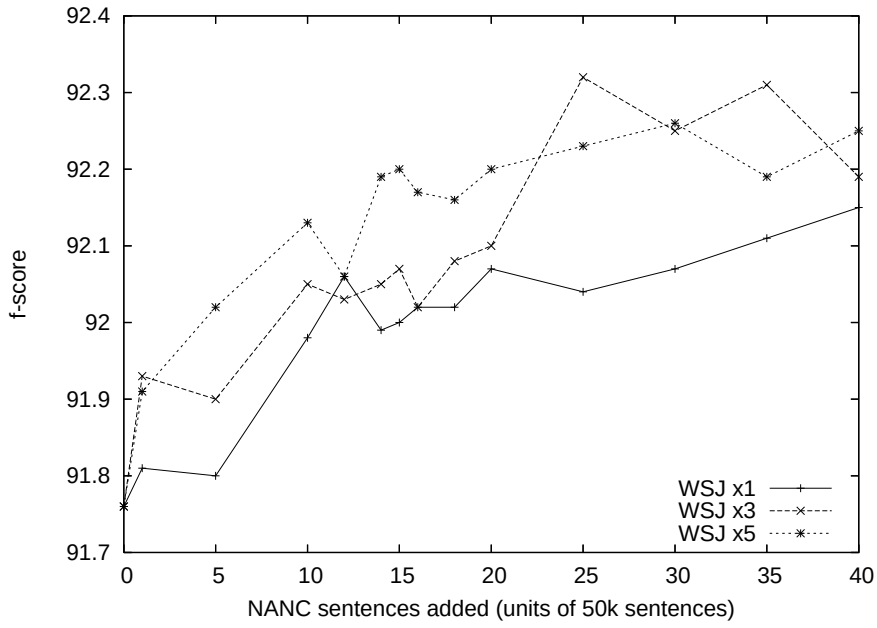
Parser portability studies examine how well parsers trained on one domain (*source domain*) perform on a different one (*target domain*). Unfortunately, there is little consensus in the field exactly what the tasks of parser portability and parser adaptation entail. We define *parser portability* as the task where we are given no labeled in-domain data for the target domain. When we do have access to some amount of labeled in-domain data, we call the task *parser adaptation*.

Naturally, there is always a penalty for changing domains if the source and target domains do not sufficiently overlap. Thus, parser portability informs us about the generality of the parser as well as being an approximate measure of distance between the source and target domains. As mentioned in Section 1.3, these studies have often been done by training parsers on WSJ and evaluating them on BROWN (Ratnaparkhi, 1999; Gildea, 2001; Bacchiani et al., 2006). For ease of comparison, we use the same setup. We also use the GENIA and SWBD corpora as alternative more distant target domains (Section 3.2.2).

Our first experiment examines the performance of the self-trained parsers. While the parsers are created entirely from labeled WSJ and unlabeled NANC data, they perform extremely well on BROWN development (Table 3.4). The trends are the same as in the previous section: Adding NANC data improves parsing performance on the BROWN development section considerably, improving the f-score from 83.9% to 86.4%. As more NANC data is added, the f-score appears to approach an asymptote. The NANC data appears to help reduce data sparsity and fill in some of the gaps in the WSJ model. Additionally, the reranker provides further benefit and adds an absolute 1-2% to the f-score. The improvements appear to be orthogonal, as our best performance is reached when we use the reranker and add 2,500,000 self-trained sentences from NANC.



(a) First stage parser alone



(b) Reranking parser

Figure 3.1: Effect of giving more relative weight to WSJ training data on parser and reranking parser f-score. Higher WSJ weights generally improve parsing accuracy. Evaluations were done from all sentences from section 1.

	f-score	
Model	Parser	Reranking Parser
Baseline BROWN	86.4	87.4
Baseline WSJ	83.9	85.8
WSJ + 50,000 sentences NANC	84.8	86.6
WSJ + 250,000 sentences NANC	85.7	87.2
WSJ + 500,000 sentences NANC	86.0	87.3
WSJ + 750,000 sentences NANC	86.1	87.5
WSJ + 1,000,000 sentences NANC	86.2	87.3
WSJ + 1,250,000 sentences NANC	86.3	87.5
WSJ + 1,500,000 sentences NANC	86.2	87.6
WSJ + 1,750,000 sentences NANC	86.0	87.5
WSJ + 2,000,000 sentences NANC	86.1	87.7
WSJ + 2,500,000 sentences NANC	86.4	87.7
WSJ + self-trained BROWN	85.6	86.1

Table 3.4: Effects of adding NANC sentences to WSJ training data on f-score. The parser and reranking parsers were evaluated on BROWN development data. The reranker model was trained on WSJ.

The results are even more surprising when we compare against a parser[5] trained on the labeled training section of the BROWN corpus, with parameters tuned against its held-out section (top line in Table 3.4). Despite having no access to in-domain example trees, the WSJ based parser is able to match the f-score of the BROWN based parser.

Recall that increasing the relative weight of WSJ sentences versus NANC sentences was effective when testing on WSJ in the previous section. However, when testing on the BROWN development section, this reweighting did not appear to have a significant effect. We believe this is because the true distribution was closer to WSJ in the previous section so it made sense to emphasize it. Here, the BROWN development data does not follow the same distribution as WSJ.

The model trained on WSJ+2,500,000 sentences of NANC is the best model for parsing BROWN of the ones we have considered. We also note that this "best" parser is different from the "best" parser for parsing WSJ, which was trained on WSJ with a relative weight of 5 and 1,750,000 sentences from NANC. For parsing BROWN, the difference between these two parsers is not large, though. We have shown that self-training is a valuable technique for improving parser portability. Our next section discusses the portability of the reranker briefly.

Bacchiani et al. (2006) applies self-training to parser adaptation to utilize unlabeled in-domain data. The authors find that it helps quite a bit when adapting from BROWN to WSJ. They use a parser trained from the BROWN train set to parse WSJ and add the parsed WSJ sentences to their training set. We perform a similar experiment, using our WSJ-trained reranking parser to parse the BROWN training division (ignoring its parse annotations) and testing on BROWN development. Parser f-score was boosted from 83.9% to 85.6% when we added the parsed BROWN sentences to our training set. Adding in 1,000,000 sentences from NANC as well, we saw a further increase to 86.3%. Adding self-trained sentences from BROWN improves performance over

[5] In this case, only the parser is trained on BROWN. The reranker is mismatched, but still helps in all cases. In Section 3.2.1, we compare against a fully BROWN-trained reranking parser as well.

Parser model	Parser alone	WSJ-reranker	BROWN-reranker
WSJ	82.9	85.2	85.2
WSJ + NANC	87.1	87.8	87.9
BROWN	86.7	88.2	88.4

Table 3.5: Performance of various combinations of parser and reranker models when evaluated on BROWN test set. The WSJ+NANC parser with the WSJ reranker comes close to the BROWN-trained reranking parser. The BROWN reranker provides only a small improvement over its WSJ counterpart for all three parser models, which is not statistically significant.

the pure WSJ baseline, but is unsurprisingly not as good as the pure BROWN model (which uses gold labels).

3.2.1 Reranker Portability

We have shown that the WSJ-trained reranker is actually quite portable to the BROWN fiction domain in Table 3.4. The baseline WSJ parser achieves 85.8% with the reranker and 83.9% without. When NANC is added, performance improves from 86.4% to 87.7% when we use the reranker. This may be surprising given the large number of features — over a million in the case of the WSJ reranker — which have been tuned to adjust for errors made in the n-best lists by the first stage parser. It would seem the corrections learned by the reranker are not as domain-specific as we might expect. The reranker's regularization process during training may encourage it to use only the more general features.

To compare against a model fully trained on BROWN data, we created a BROWN reranker. The resulting reranker model had approximately 700,000 features — about half as many as the WSJ trained reranker. This may be due to the smaller size of the BROWN training set or because the feature schemas for the reranker were developed on WSJ data. We evaluated three models (fully WSJ-trained, the self-trained WSJ, and fully BROWN-trained) on the testing section of BROWN, with and without the reranker. As seen in Table 3.5, the BROWN reranker does not provide a significant improvement over the WSJ reranker for parsing BROWN data. If no labeled BROWN data is available, the WSJ+NANC model with the WSJ reranker is our best bet for parsing BROWN and achieves an f-score of 87.8%. The fully BROWN-trained reranking parser is only slightly better at 88.4%.

3.2.2 Porting to more distant domains

As further evidence that self-training improves parser portability, we present the results of applying the WSJ models to two domains which have much less in common with WSJ than BROWN. We briefly present results on SWBD (transcribed telephone conversations) and then provide a more in-depth study for GENIA (biomedical article abstracts). Table 3.6 shows our evaluation of the original WSJ and self-trained WSJ models on the SWBD development section. We see that while the parser's performance is low, self-training and reranking continue to provide orthogonal benefits. The improvements represent a 12% error reduction and use no in-domain data. Naturally, in-domain labeled examples and speech-specific handling (e.g. disfluency modeling) would dramatically improve accuracy as well. Additionally, there are likely better choices of unlabeled corpora than NANC if our goal is to parse SWBD. We now turn to our study of GENIA where we explore using

Parser model	Parser	Reranking parser
WSJ	74.0	75.9
WSJ + 1,750,000 NANC sentences	75.6	77.0

Table 3.6: Parser and reranking parser f-score performance on the SWBD development corpus. The NANC data and reranker improvements are orthogonal, but less dramatic than they are on the BROWN corpus. This is likely because SWBD is considerably less similar to WSJ than the BROWN corpus.

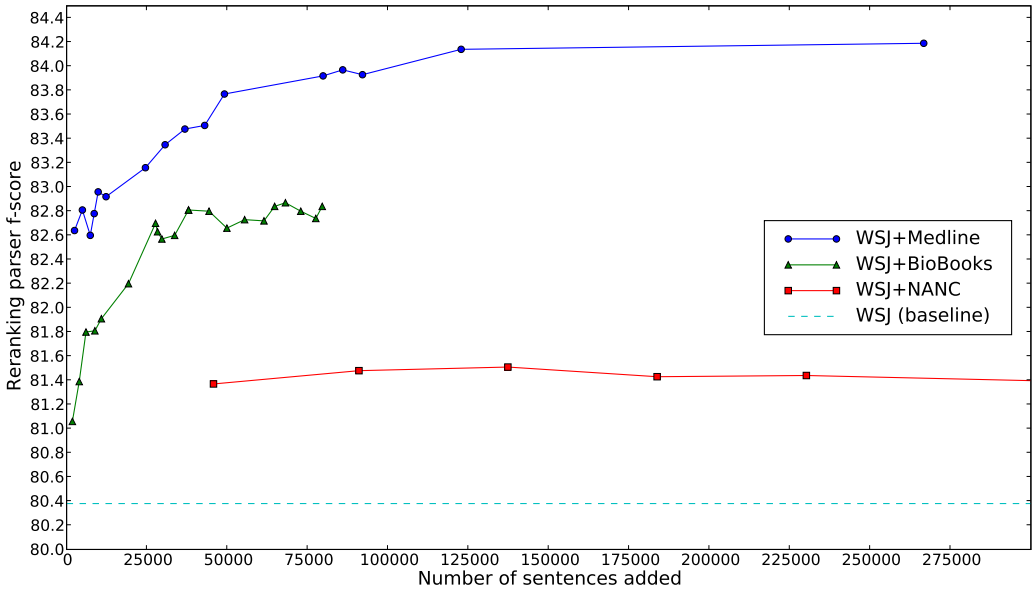


Figure 3.2: Reranking parser f-score on development data for four different self-training scenarios as a function of number of self-training sentences.

different unlabeled corpora for self-training.

We did several experiments on GENIA development data using different unlabeled corpora for self-training. As before, we use NANC but for this experiment, we also try two different in-domain corpora: BIOBOOKS and MEDLINE (descriptions of these corpora can be found in Section 2.3). These results are summarized in Figure 3.2. We show the f-score for four different self-training scenarios using the reranking parser as a function of number of self-training sentences. As before, the reranker was trained solely on WSJ data. The "WSJ (baseline)" line is the raw reranking parser with no self-training. At 80.4%, it is clearly the worst of the lot. On the other hand, it is already comparable to the best previous result (80.2%) for biomedical data from the parser by Lease and Charniak (2005), as reported by Clegg and Shepherd (2007).[6] The parser by Lease and Charniak (2005) is the first stage Charniak parser with access to an external in-domain tagger.

[6]We say comparable since the 80.4% is on the development while the 80.2% is on test (GENIA beta 2). However, these two data sets are similar in difficulty.

	Parser			Reranking parser		
System	Precision	Recall	f-score	Precision	Recall	f-score
Lease and Charniak (2005)	—	—	**80.2**	—	—	—
WSJ	79.3	76.6	77.9	82.4	78.7	80.5
WSJ + 266,000 MEDLINE sentences	85.0	81.9	83.4	86.3	82.4	**84.3**

Table 3.7: Comparison of the MEDLINE self-trained parser against previous best on GENIA beta 2.

We use the parser's internal tagger which has been trained on WSJ but we achieve similar performance due to the introduction of the 50-best reranker. If we self-train on NANC, our performance goes up to 81.4%, regardless of how much parsed NANC is incorporated.

Our best results come from self-training on MEDLINE instead of NANC. As seen in Figure 3.2, even a thousand sentences of MEDLINE is enough to drive our results up to a new level and accuracy continues to improve until about 150,000 sentences at which point it is relatively flat. However, as adding about 270,000 sentences is fractionally better than 150,000 sentences, we opted for the higher number of self-training sentences for our results on the test set.

The WSJ+BIOBOOKS line comes from interesting idea that failed to work. We mention it in the hope that others might be able to succeed where we have failed. We reasoned that biology textbooks would be a particularly good "bridging corpus." After all, they are written to introduce someone ignorant of a field to the ideas and terminology within it. Thus, one might expect that the English of a Biology textbook would be intermediate between the more typical English of a news article and the specialized English native to the biomedical domain.

To test this, we created a biology textbook corpus (BIOBOOKS). We observe in Figure 3.2 that for all quantities of self-training data one does better with MEDLINE than BIOBOOKS. For example, at 37,000 sentences the BIOBOOKS corpus is only able to achieve an f-score of 82.8% while the MEDLINE corpus is at 83.4% with the same amount of additional sentences. Furthermore, BIOBOOKS levels off in performance while MEDLINE has significant improvement left in it. Thus, while the hypothesis seems reasonable, we were unable to make it work.

Our evaluation on the GENIA test set is shown in Table 3.7. We weighted the original WSJ equally with self-trained MEDLINE data. We did not perform any tuning to find out if there is some better weighting. Given that reweighting was not helpful when evaluating on BROWN (Section 3.2), it is unlikely to help here since GENIA is quite different from WSJ. Clegg and Shepherd (2007) do not provide separate precision and recall numbers for the Lease and Charniak (2005) system. However, we can see that the MEDLINE self-trained reranking parser achieves an f-score of 84.3%, which is an absolute reduction in error of 4.1%. This corresponds to an error rate reduction of 20% over the Lease and Charniak (2005) baseline. Also note that, as before, our improvements come from a combination of self-training and using the out-of-domain trained reranker.

If labeled data is available, our accuracy is even higher. Table 3.8 gives our results on the more modern and significantly larger version of the GENIA treebank.[7] The "WSJ + 266,000 MEDLINE sentences" model from

[7] Experiments up to this point have been on the "GENIA beta 2" corpus.

Parser model	Reranker model	f-score
WSJ	—	74.9
WSJ	WSJ	76.8
WSJ + MEDLINE (parsed by WSJ)	WSJ	80.7
GENIA	—	83.6
GENIA	WSJ	84.5
GENIA	GENIA	85.7
WSJ + MEDLINE (parsed by GENIA)	GENIA	87.6
GENIA + MEDLINE (parsed by GENIA)	GENIA	87.6

Table 3.8: Evaluations on the larger GENIA data set (10,848 sentences across all divisions).

Table 3.7 corresponds to the third line in this table ("WSJ + GENIA (parsed by WSJ)"). As before, we receive a significant boost from both the reranker and self-training. The table shows that as labeled GENIA data is used in more steps (base parser training, reranker training, and as a corpus for self-training) performance improves substantially. Despite the success of our self-trained WSJ model from Table 3.7 which obtains an f-score of 80.7 on this dataset, once labeled data is available performance shoots up to 83.6 even without a reranker. Using an out-of-domain WSJ reranker gives us about an additional 1% in f-score and switching to an in-domain GENIA reranker provides another 1% improvement. Self-training using that GENIA reranking parser increases the f-score to 87.6%. Perhaps surprisingly, this level of accuracy can be achieved even if the parser is trained using WSJ and the self-trained MEDLINE corpus (as parsed by GENIA). Using out-of-domain WSJ data here does not hurt performance presumably since it is overwhelmed by the larger amount of self-trained MEDLINE data and because the GENIA reranker is able to correct enough of its mistakes.

3.3 Parser Adaptation

We now turn to the scenario where we have some labeled in-domain data. The most obvious way to incorporate labeled in-domain data is to combine it with the labeled out-of-domain data. We have already seen the results Gildea (2001) and Bacchiani et al. (2006) achieve in Table 1.1.

We explore various combinations of BROWN, WSJ, and NANC corpora. Because we are mainly interested in exploring techniques with self-trained models rather than optimizing performance, we only consider weighting each corpus with a relative weight of one for this experiment. The models generated are tuned on section 24 from WSJ. The results are summarized in Table 3.9.

While both WSJ and BROWN models benefit from a small amount of NANC data, adding more than 250,000 NANC sentences to the BROWN or combined models causes their performance to drop. For example, the WSJ + BROWN + 250,000 NANC model achieves an f-score of 88.1% with the reranking parser, but only 87.7% if an additional 250,000 sentences from NANC are added. Accuracy continues to fall as more NANC data is included. This is not surprising since adding "too much" NANC overwhelms the more accurate BROWN or WSJ counts. By weighting the counts from each corpus appropriately, this problem can be avoided.

Another way to incorporate labeled data is to tune the parser back-off parameters on it. Bacchiani et al. (2006) report that tuning on held-out BROWN data gives a large improvement over tuning on WSJ data.

	f-score	
Parser model	Parser	Reranking parser
WSJ alone	83.9	85.8
WSJ + 2,500,000 NANC	86.4	87.7
BROWN alone	86.3	87.4
BROWN + 50,000 NANC	86.8	88.0
BROWN + 250,000 NANC	86.8	88.1
BROWN + 500,000 NANC	86.7	87.8
BROWN + 1,000,000 NANC	86.6	87.8
WSJ + BROWN	86.5	88.1
WSJ + BROWN + 50,000 NANC	86.8	88.1
WSJ + BROWN + 250,000 NANC	86.8	88.1
WSJ + BROWN + 500,000 NANC	86.6	87.7
WSJ + BROWN + 1,000,000 NANC	86.6	87.6

Table 3.9: f-scores from various combinations of WSJ, NANC, and BROWN corpora on BROWN development. The reranking parser used the WSJ-trained reranker model. The BROWN parsing model is naturally better than the WSJ model for this task, but combining the two training corpora results in a better model (as in Gildea (2001)). Adding small amounts of NANC further improves the results.

The improvement is mostly (but not entirely) in precision. We performed a similar experiment using WSJ as training data, using either WSJ or BROWN data for parameter tuning to create parsing and reranker models (Figure 3.3). We do not see the same improvement as Bacchiani et al. (2006) on the non-self-trained parser ($x = 0$ NANC sentences) but this is likely due to differences in the parsers . However, we do see a similar improvement for parsing accuracy once the self-trained NANC data has been added. The reranking parser generally sees an improvement, but it does not appear to be significant. From these two experiments, it seems better to use labeled in-domain data for training rather than setting parameters.

3.4 Self-Training Extensions

There have been two follow-up studies on self-training which give us additional data points of self-training's capabilities. Reichart and Rappoport (2007) showed that one can self-train with only a generative parser if the seed size is small. The conditions are similar to those in Steedman et al. (2003a), but only one iteration of self-training is performed (i.e. all unlabeled data is labeled at once).[8] The authors show that self-training is beneficial for in-domain parsing and parser adaptation. In their case, they are able to demonstrate a reduction in the number of labeled sentences required to achieve a specific f-score.

Foster et al. (2007) use self-training to improve performance on BNC. Rather than using NANC as their unlabeled corpus, they use one million raw sentences from the complete BNC. They are able to improve performance on the 1,000 sentence BNC test set from 83.9% to 85.6% after adding the automatic BNC parses. Similarly, they are able to improve performance on the WSJ test set from 91.3% to 91.7%. This is smaller than the 0.8% improvement that we get from adding 1.7 million NANC parses[9] and reinforces our point that

[8] Performing multiple iterations presumably fails because the parsing models become increasingly biased.

[9] In both cases, performance has leveled off on the development set, so it is safe to assume that we would not see a similar improvement from BNC if an additional 700,000 automatically parsed BNC sentences were added.

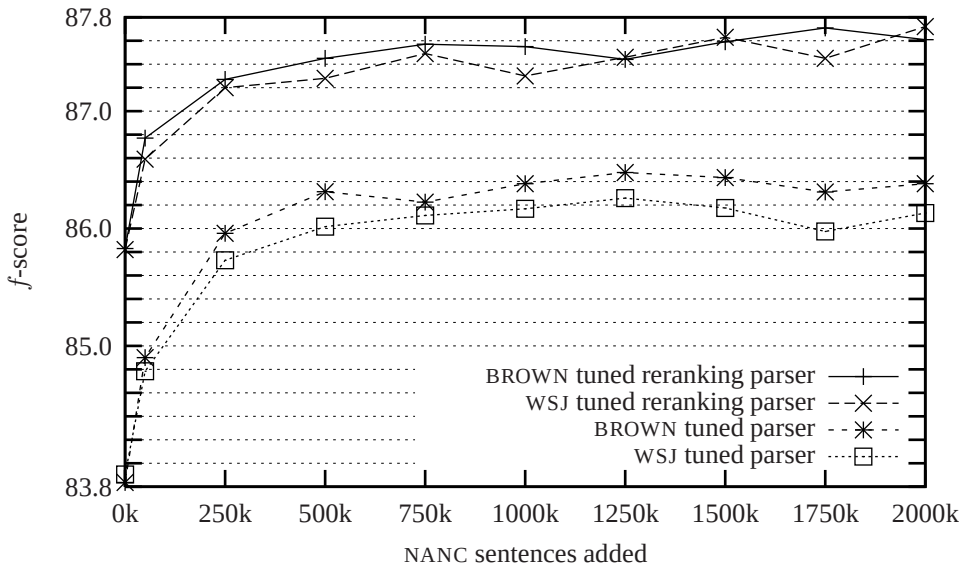


Figure 3.3: Precision and recall *f*-scores when testing on BROWN development as a function of the number of NANC sentences added under four test conditions. "BROWN tuned" indicates that BROWN training data was used to tune the parameters (since the normal held-out section was being used for testing). For "WSJ tuned," we tuned the parameters from section 24 of WSJ. Tuning on BROWN helps the parser, but not for the reranking parser.

picking a self-training corpus that matches the test set well is important. We return to this issue in Chapter 5.

Chapter 4

Analysis

While the success of self-training has demonstrated its merit, it remains unclear why self-training helps in some cases but not others. Our goal is to better understand when and why self-training is beneficial. We perform a variety of tests, covering the global behavior of the parser (Section 4.1), sentence-level changes (Section 4.2), and parser agreement between the BROWN and self-trained WSJ models (Section 4.3). Next, we explore four hypotheses for why self-training helps in Section 4.4. At a high level, the hypotheses are (1) self-training helps after a phase transition, (2) self-training reduces search errors, (3) specific classes of reranker features are needed for self-training, and (4) self-training improves because we see new combinations of words. We summarize our analysis in Section 4.5.

4.1 Global changes

It is important to keep in mind that while the reranker seems to be key to our performance improvement, the reranker per se never sees the extra automatically parsed sentences. It only sees the 50-best lists produced by the first stage parser. Thus, the nature of the changes to these lists are important.

We have already noted that the first stage parser's one-best f-score has significantly improved on WSJ when self-trained sentences from the reranking parser are added to training data (see Table 3.1). In Table 4.1, we see that the 50-best oracle score also improves from 95.5% (for the original first stage parser) to 96.4% (for our final model). We do not show it in the table, but if we self-train using first stage parser's one-best, there is no change in oracle score. The oracle scores on the BROWN development section are shown in Table 4.2. While the WSJ parser initially has relatively low oracle f-scores, adding sentences from NANC produces a parser with comparable oracle scores as the parser trained from BROWN training. The BROWN and self-trained WSJ models have essentially the same potential for good parses of the BROWN corpus. Thus, the self-trained models have better oracle scores than the original WSJ model for both the WSJ and BROWN domains. Overall, the oracle scores on BROWN are 2-3% lower than those on WSJ, probably due to the increased variability of the BROWN corpus.

The first stage parser also becomes more "decisive" after self-training. The average (geometric mean) of

Model	1-best	2-best	10-best	25-best	50-best
Baseline (WSJ)	89.0	91.0	94.1	95.3	95.9
WSJ + 250,000 NANC	89.8	91.4	94.6	95.5	96.1
WSJ×5 + 1,750,000 NANC	90.4	91.9	94.8	95.8	96.4

Table 4.1: Oracle f-scores of top n parses produced by the baseline WSJ parser, a small self-trained parser, and the "best" parser on WSJ section 24.

Model	1-best	2-best	10-best	25-best	50-best
WSJ	82.6	84.8	88.9	90.7	91.9
WSJ + 2,500,000 NANC	86.4	88.5	92.1	93.5	94.3
BROWN	86.3	88.4	92.0	93.3	94.2

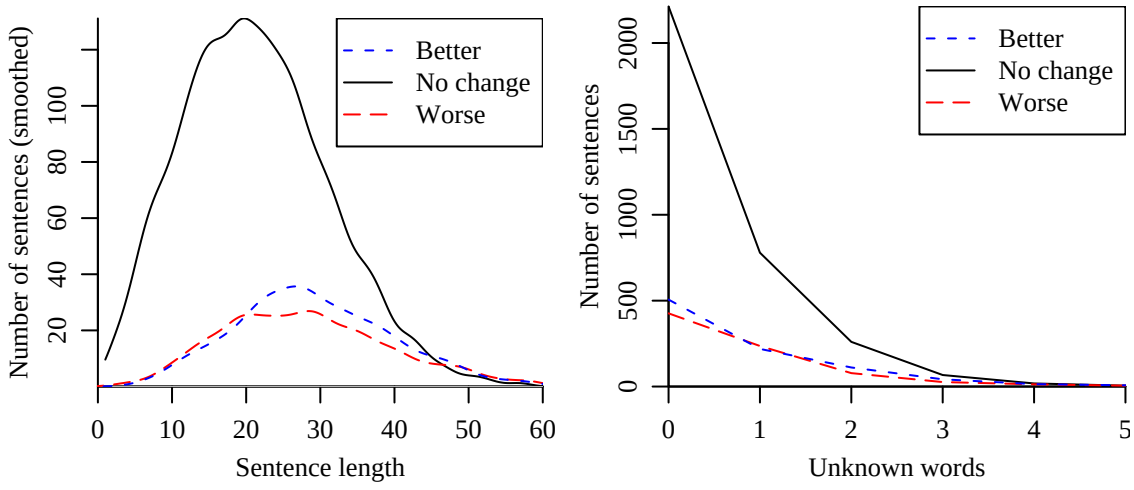
Table 4.2: Oracle f-scores of top n parses produced by the baseline WSJ parser, a combined WSJ and NANC parser, and the baseline BROWN parser on the BROWN development section.

$\log_2$(Pr(1-best parse) / Pr(50th-best parse)) (i.e. the ratios between the probabilities in log space) increases from 11.959 for the baseline parser to 14.104 for the final parser. In other words, the probability of the top parse increases relative to the 50th-best parse. We have seen earlier that this additional confidence is deserved, as the first stage one-best is much better. Additionally, with more data available, the self-trained parser backs off to smoothing less often which also has the effect of increasing the probabilities of parses.

4.2 Sentence-level analysis

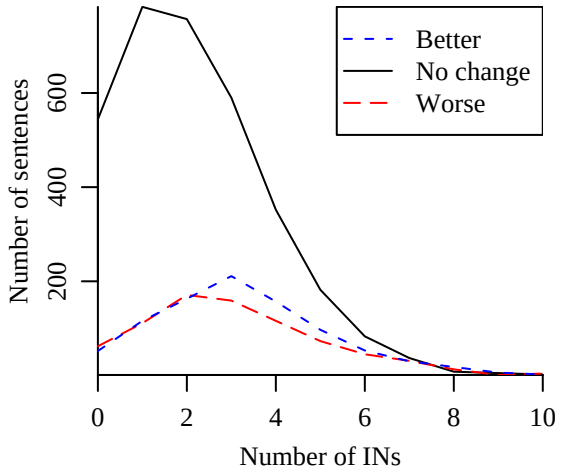
Until this point we have looked at bulk properties of the n-best lists fed to the reranker. We now turn to studying the performance of individual sentences. In particular, we analyzed the original and self-trained parsers' behavior on 5,039 sentences from sections 1, 22 and 24 of the Penn treebank. Specifically, we classified each sentence into one of three classes: those where the self-trained parser's f-score increased relative to the baseline parser's f-score, those where the f-score remained the same, and those where the self-trained parser's f-score decreased relative to the baseline parser's f-score. We charted the distribution of sentences into these classes with respect to four factors: sentence length, the number of unknown words (i.e., words not appearing in sections 2–21 of the Penn treebank) in the sentence, the number of coordinating conjunctions (CC) in the sentence, and the number of prepositions (IN) in the sentence. The distributions of classes (better, worse, no change) with respect to each of these factors individually are graphed in Figures 4.1a through 4.1d.

Figure 4.1a shows how the self-training affects f-score as a function of sentence length. The top line shows that the f-score of most sentences remain unchanged. The middle line is the number of sentences that improved their f-score, and the bottom are those which got worse. So, for example, for sentences of length 30, about 80 were unchanged, 25 improved, and 22 worsened. It seems clear that there is no improvement for either very short or very long sentences. (For long sentences the graph is hard to read. We show a regression analysis later in this section that confirms this statement.) While we did not predict this effect, in retrospect it seems reasonable. The parser was already doing very well on short sentences. The very long ones are

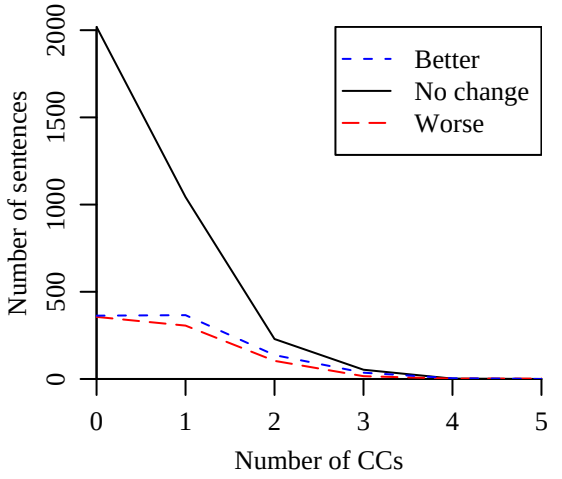


(a) Sentence length

(b) Number of unknown words



(c) Number of prepositions



(d) Number of conjunctions

Figure 4.1: Effect of self-trained model on performance for four different variables.

Feature	Estimate	Pr(> 0)
(Intercept)	-0.25328	0.3649
BinnedLength(10,20]	0.02901	0.9228
BinnedLength(20,30]	0.45556	0.1201
BinnedLength(30,40]	0.40206	0.1808
BinnedLength(40,50]	0.26585	0.4084
BinnedLength(50,200]	-0.06507	0.8671
CCs	0.12333	0.0541

Table 4.3: Predictors for the question: "does the self-trained parser improve the f-score of the parse with the highest probability?"

hopeless, and the middle ones are just right. We call this the Goldilocks effect.

As for the other three of these graphs, their stories are by no means clear. Figure 4.1b seems to indicate that the number of unknown words in the sentence does *not* predict that the reranker will help. Figure 4.1c might indicate that the self-trained parser improves prepositional-phrase attachment, but the graph looks suspiciously like that for sentence length, so the improvements might just be due to the Goldilocks effect. Finally, the improvement in Figure 4.1d is hard to judge.

To get a better handle on these effects, we performed predictor selection within a selection model using the same four factors. As Figure 4.1a makes clear, the relative performance of the self-trained and baseline parsers does not vary linearly with sentence length, so sentence lengths were binned (with each bin of length 10). Because the self-trained and baseline parsers produced equivalent output on 3,346 (66%) of the sentences, we restricted attention to the 1,693 sentences on which the self-trained and baseline parsers' f-scores differ. The results are shown in Table 4.3.

The regression analysis is trying to model the log odds as a sum of linearly weighted factors. That is:

$$\log\left(\frac{P(1|x)}{1-P(1|x)}\right) \quad = \quad \alpha_0 + \sum_{j=1}^{m} \alpha_j f_j(x)$$

In Table 4.3, the first column gives the name of the factor. The second displays the change in the log-odds resulting from this factor being present (in the case of CCs and INs, multiplied by the number of them) and the last column is the probability that this factor is really non-zero. "(Intercept)" refers to α_0 in the equation.

Note that there is no row for either PPs or unknown words. This is because we also asked the program to do a model search using the Akaike Information Criterion (AIC) over all single and pairwise factors. The model it chooses predicts that the self-trained parser is likely to produce a better parse than the baseline only for sentences of length 20–40 or sentences containing several CCs. It did not include the number of unknown words and the number of INs as factors because they did not receive a weight significantly different from zero, and the AIC model search dropped them as factors from the model.

In other words, the self-trained parser is more likely to be correct for sentences of length 20–40 and as the number of CCs in the sentence increases. The self-trained parser does *not* improve prepositional-phrase attachment or the handling of unknown words.

This result is mildly perplexing. It is fair to say that neither we, nor anyone we talked to, thought conjunction handling would be improved. Conjunctions are about the hardest things in parsing, and we have no

grip on exactly what it takes to help parse them. Conversely, everyone expected improvements on unknown words, as the self-training should drastically reduce the number of them. It is also the case that we thought PP attachment might be improved because of the increased coverage of preposition-noun and preposition-verb combinations that work such as Hindle and Rooth (1993) show to be so important.

Currently, our best conjecture is that unknowns are not improved because the words that are unknown in the WSJ are not significantly represented in the LA Times we used for self-training. CCs are difficult for parsers because each conjunct has only one secure boundary. This is particularly the case with longer conjunctions, those of VPs and Ss. One thing we know is that self-training always improves performance of the parsing model when used as a language model. We think CC improvement is connected with this fact and our earlier point that the probabilities of the 50-best parses are becoming more skewed. In essence the model is learning, in general, what VPs and Ss look like so it is becoming easier to pull them out of the stew surrounding the conjunct. Conversely, language modeling has comparatively less reason to help PP attachment. As long as the parser is doing it consistently, attaching the PP either way will work almost as well.

4.3 Portability Studies

We perform several types of analysis to measure the differences and similarities between the BROWN trained and self-trained WSJ reranking parsers. Despite their different training sources, these two parsers are extremely close in performance on BROWN. While the two parsers agree on a large number of parse brackets (Section 4.3.1), there are categorical differences between them (Section 4.3.2). In Section 4.3.3, we perform feature selection to better understand the circumstances in which each parser does better.

4.3.1 Parser Agreement

In this section, we compare the output of the WSJ+NANC-trained and BROWN-trained reranking parsers. We use *evalb* to calculate how similar the two sets of output are on a bracket level by treating one set of parses as candidate parses and the other as gold.[1] Table 4.4 shows the results. The two parsers agree on a good portion of the brackets and achieved an 88.0% f-score between them. Additionally, the two parsers agreed on the entire parse almost half the time and have approximately one crossing bracket on average. The part of speech tagging agreement is fairly high as well. Considering they were trained from different corpora, this seems like a high level of agreement. It would be nice to perform a finer-grained analysis of this to better determine the nature of the disagreements.

4.3.2 Statistical Analysis of f-score differences

We are interested in whether the differences between the various parsers produced while adapting the WSJ-trained parser to the BROWN corpus are statistically significant. To test this, we conducted randomized permutation tests for the significance of the difference in corpus f-score (recall Section 2.4). The null hypothesis

[1] We can choose these arbitrarily since all of these metrics are symmetric.

Bracketing agreement f-score	88.0%
Complete match	44.9%
Average crossing brackets	0.94
POS tagging agreement	94.9%

Table 4.4: Agreement between the WSJ+NANC parser with the WSJ reranker and the BROWN parser with the BROWN reranker. Complete match is how often the two reranking parsers returned the exact same parse. Though created by different sets of data, the two parsers come up with surprisingly similar results on the bracket level.

Parser/Reranker model	WSJ+NANC/WSJ	BROWN/WSJ	BROWN/BROWN
WSJ/WSJ	0.025 (0)	0.030 (0)	0.031 (0)
WSJ+NANC/WSJ		**0.004** (**0.1**)	0.006 (0.025)
BROWN/WSJ			**0.002** (**0.27**)

Table 4.5: The difference in f-score between different combinations of parsers and rerankers, and the significance of the difference in parentheses as estimated by a randomization test with 10^6 samples. "x/y" indicates that the first stage parser was trained on data set x and the second stage reranker was trained on data set y. Differences between combinations that are not significantly different are shown in bold font.

is that the two parsers being compared are in fact behaving identically, so permuting or swapping the parse trees produced by the parsers for the same test sentence should not affect the corpus f-scores. By estimating the proportion of permutations that result in an absolute difference in corpus f-scores at least as great as that observed in the actual output, we obtain a distribution-free estimate of significance that is robust against parser and evaluator failures. The results of this test are shown in Table 4.5. The "x/y" notation indicates that the first stage parser was trained on data set x and the second stage reranker was trained on data set y. The table shows that the BROWN reranker is not significantly different from the WSJ reranker (the difference between the BROWN and WSJ rerankers is significant with a p-value $= 0.27$ and thus not statistically significant). We can also see that the difference between the WSJ+NANC and BROWN parsing models is not statistically significant if the WSJ reranker is used for both (p-value $= 0.1$). The difference between the WSJ-trained reranking parser (WSJ+NANC/WSJ) and BROWN-trained reranking parser (BROWN/BROWN) is, however, statistically significant (p-value ≈ 0).

4.3.3 Feature selection for regression

In order to better understand the difference between the fully BROWN-trained and the self-trained WSJ reranking parsers on BROWN data, we constructed a logistic regression model of the difference between the two parsers' f-scores on the development data using the R statistical package.[2] Of the 2,078 sentences in the development data, 29 sentences were discarded because *evalb* failed to evaluate at least one of the parses.[3] A Wilcoxon signed rank test on the remaining 2,049 paired sentence level f-scores was significant at $p = 0.0003$. Of these 2,049 sentences, there were 983 parse pairs with the same sentence-level f-score. Of

[2] `http://www.r-project.org/`

[3] This occurs when an apostrophe is analyzed as a possessive marker in the gold tree and a punctuation symbol in the parse tree, or vice versa.

Feature	Estimate	z-value	$\Pr(>\|z\|)$	
(Intercept)	0.054	0.3	0.77	
# INs	-0.134	-4.4	8.4e-06	$***$
ID=Letters, bibliography, memories	0.584	2.5	0.011	$*$
ID=General fiction	0.697	2.9	0.003	$**$
ID=Mystery	0.552	2.3	0.021	$*$
ID=Science fiction	0.376	0.9	0.33	
ID=Adventure and Western fiction	0.642	2.7	0.0055	$**$
ID=Romance and Love story	0.624	2.7	0.0069	$**$
ID=Humor	0.040	0.1	0.90	

Table 4.6: The logistic model of BROWN/BROWN f-score $>$ WSJ+NANC/WSJ f-score identified by model selection. The feature "# INs" is the number prepositions in the sentence, while ID identifies the BROWN subcorpus that the sentence comes from. Stars indicate significance level.

Category	Description	# Sentences	BROWN	WSJ+NANC	Δ
F	Popular Lore	271	87.3	89.6	2.28
G	Letters, bibliography, memories	281	87.6	87.1	-0.45
K	General fiction	333	87.2	85.9	-1.29
L	Mystery	318	88.7	88.3	-0.45
M	Science fiction	76	87.7	88.8	1.17
N	Adventure and Western fiction	378	89.7	89.0	-0.64
P	Romance and Love story	338	88.0	86.6	-1.40
R	Humor	83	84.6	87.0	2.45

Table 4.7: Performance of BROWN-trained reranking parser vs. best WSJ+NANC reranking parser on various categories of the BROWN development division. Both rerankers use WSJ-trained models.

the 1,066 sentences for which the parsers produced parses with different f-scores, there were 580 sentences for which the BROWN/BROWN parser produced a parse with a higher sentence-level f-score and 486 sentences for which the WSJ+NANC/WSJ parser produced a parse with a higher f-score. We constructed a generalized linear model with a binomial link to predict when BROWN/BROWN f-score $>$ WSJ+NANC/WSJ f-score. For explanatory variables, we used sentence length, the number of prepositions (IN), the number of conjunctions (CC), and the BROWN subcorpus ID. The goal of the last class of features is to determine if there are certain genres within BROWN that are harder for WSJ to parse. Model selection (using the "step" procedure) discarded all but the IN and BROWN ID explanatory variables. The final estimated model is shown in Table 4.6. The WSJ+NANC/WSJ parser becomes more likely to have a higher f-score than the BROWN/BROWN parser as the number of prepositions in the sentence increases. The BROWN/BROWN parser is more likely to have a higher f-score on sections K (general fiction), N (adventure and western fiction), P (romance and love story), G (letters and memories) and L (mystery) sections of the BROWN corpus. The three sections of BROWN not in this list are F (popular lore), M (science fiction), and R (humor) which contain too few sentences to result in significant effects. The performance on individual categories of the BROWN corpus is shown in Table 4.7. This table confirms that the BROWN based model performs better on categories with sufficient sentences with the exception of category F ("popular lore").

4.4 Four Hypotheses

The question of why self-training helps in some cases (e.g. Section 3 and Reichart and Rappoport (2007; Foster et al. (2007)) but not others (Charniak, 1997; Steedman et al., 2003a) has inspired various theories. We investigate four of these to better understand the circumstances which govern self-training's success.

4.4.1 Phase Transition

.

The phase transition hypothesis is that once a parser has achieved a certain threshold of performance, it can label data sufficiently accurately. Once this happens, the labels will be "good enough" for self-training.

To test the phase transition hypothesis, we intentionally degrade the parser's performance to see if it can still self-train. We do this by using the same parser as our earlier self-training experiments in Section 3 but train on only a fraction of WSJ to see if self-training is still helpful. This is similar to some of the experiments by Reichart and Rappoport (2007) but with the use of a reranker and slightly larger seed sizes. The self-training protocol is the same as in (Charniak, 1997; McClosky et al., 2006a; Reichart and Rappoport, 2007): we parse the entire unlabeled corpus in one iteration. We start by making a random subset of the WSJ training sections (2–21), accepting each sentence with probability k. We create two subsets of WSJ ($k \in \{0.1, 0.5\}$). This approach to testing the phase transition hypothesis does not include other parsers or other methods of performance degradation. It is possible that we might see a phase transition in these cases (James Henderson, personal communication).

With the sampled training section and the standard development data, we train a parser and a reranker. In Table 4.8, we show the performance of the parser with and without the reranker. For reference, we show the performance when using the complete training division as well. Unsurprisingly, both metrics drop as we decrease the amount of training data. These scores represent our baselines for this experiment.

Using these parser models, we parse one million sentences from NANC, both with and without the reranker. We combine parsed sentences with the sampled subsets of WSJ training and train new parser models from them.[4]

Finally, we evaluate these self-trained models (Table 4.9). The numbers in parentheses indicate the change from the corresponding model made without self-training. As in Reichart and Rappoport (2007), we see large improvements when self-training on a small seed size (10%) without using the reranker. We still see significant improvements in most cases when the seed size is 50%, but the absolute values of these improvements are smaller. However, using the reranker to parse the self-training and/or evaluation sentences further improves results. From Section 3.1.1, we know that when 100% of the training data is used, self-training does not improve performance unless the reranking parser is used to parse NANC.

From this we conclude that there is no such threshold phase transition in this case. High performance is not a requirement to successfully use self-training for parsing, since there are lower performing parsers which can self-train and higher performing parsers which cannot. The higher performing Charniak and Johnson

[4] We do not weight the original WSJ data, though our expectation is that performance would improve if WSJ were given a higher relative weight.

% WSJ	# sentences	Parser f-score	Reranking parser f-score
10	3,995	85.8	87.0
25	9,903	87.9	89.3
50	19,975	89.0	90.4
100	39,832	89.9	91.5

Table 4.8: Parser and reranking parser performance on sentences ≤ 100 words in sections 1, 22, and 24 when trained on different amounts of training data. % WSJ is the probability of selecting a sentence from WSJ training (this is why the 10% column doesn't have exactly 10% of the sentences, etc.). Note that the full amount of development data is still used as held out data.

% WSJ	Parsed NANC with reranker?	Parser f-score	Reranking parser f-score
10%	No	87.7 (+1.9)	88.7 (+1.7)
10%	Yes	88.4 (+2.6)	89.0 (+2.0)
50%	No	89.3 (+0.3)	91.0 (+0.6)
50%	Yes	89.7 (+0.7)	91.0 (+0.6)

Table 4.9: Effect of self-training using only a portion of WSJ as labeled data. The parser model is trained from WSJ and one million parsed sentences from NANC. The first column indicates whether the million NANC sentences were parsed by the parser or reranking parser. The second and third columns differ in whether the reranker is used to parse the test sentences (WSJ sections 1, 22, and 24, sentences 100 words and shorter). Numbers in parentheses are the improvements over the corresponding non-self-trained parser.

(2005) parser without reranker achieves an f-score of 89.0 on section 24 when trained on all of WSJ. This parser does not benefit from self-training unless paired with a reranker. Contrast this with the same parser trained on only 10% of WSJ, where it gets an f-score of 85.8 (Table 4.9) or the small seed models of Reichart and Rappoport (2007). Both of these lower performing parsers can successfully self-train. Additionally, we now know that while a reranker is not required for self-training when the seed size is small, it still helps performance considerably (f-score improves from 87.7 to 89.0 in the 10% case and 89.3 to 91.0 in the 50% case).

4.4.2 Search Errors

Another possible explanation of self-training's improvements is that seeing newly labeled data results in fewer search errors (Daniel Marcu, personal communication). A search error would indicate that the parsing model could have produced better (more probable) parses if not for heuristics in the search procedure. The additional parse trees may help produce sharper distributions and reduce data sparsity, making the search process easier. To test this, first we present some statistics on the n-best lists ($n = 50$) from the baseline WSJ trained parser and self-trained model. We use each model to parse sentences from held-out data (sections 1, 22, and 24) and examine the n-best lists.

We compute statistics of the WSJ and self-trained n-best lists with the goal of understanding how much they intersect and whether there are search errors. On average, the n-best lists overlap by 66.0%. Put another way, this means that about a third of the parses from each model are unique, so the parsers do find a fair number of different parses in their search. The next question is where the differences in the n-best lists lie — if all the differences were near the bottom, this would be less meaningful. Let W and S represent the

Model	f-score
WSJ	91.5
WSJ with search help	91.7
Self-trained	92.0

Table 4.10: Test of whether "search help" from the self-trained model impacts the WSJ trained model. WSJ with search help is made by adding self-trained parses not proposed by the WSJ trained parser but to which the parser assigns a positive probability. The WSJ reranker is used in all cases to select the best parse for sections 1, 22, and 24.

n-best lists from the baseline WSJ and self-trained parsers, respectively. The $top_m(\ell)$ function returns the highest scoring parse in the n-best list ℓ according to the reranker and parser model m.[5] Almost 40% of the time, the top parse in the self-trained model is not in the WSJ model's n-best list, ($top_s(S) \notin W$) though the two models agree on the top parse roughly 42.4% of the time ($top_s(S) = top_w(W)$). Search errors can be formulated as $top_s(S) \notin W \wedge top_s(S) = top_w(W \cup S)$. This captures sentences where the parse that the reranker chose in the self-trained model is not present in the WSJ model's n-best list, but if the parse were added to the WSJ model's list, the parse's probability in the WSJ model and other reranker features would have caused it to be chosen. These search errors occur in only 2.5% of the n-best lists. At first glance, one might think that this could be enough to account for the differences, since the self-trained model is only several tenths better in f-score. However, we know from Section 4.2 that on average, parses do not change between the WSJ and self-trained models and when they do, they only improve slightly more than half the time. For this reason, we run a second test more focused on performance.

For our second test we help the WSJ trained model find the parses that the self-trained model found. For each sentence, we start with the n-best list ($n = 500$ here) from the WSJ trained parser, W. We then consider parses in the self-trained parser's n-best list, S, that are not present in W (in other words, we take the set $S - W$). For each of these parses, we determine its probability under the WSJ trained parsing model. If the probability is non-zero, we add the parse to the n-best list W, otherwise we ignore the parse. In other words, we find parses that the WSJ trained model could have produced but didn't due to search heuristics. In Table 4.10, we show the performance of the WSJ trained model, the model with "search help" as described above, and the self-trained model on WSJ sections 1, 22, and 24. The WSJ reranker is used to pick the best parse from each n-best list. WSJ with search help performs slightly better than WSJ alone but does not reach the level of the self-trained model. From these experiments, we conclude that reduced search errors can only explain a small amount of self-training's improvements.

4.4.3 Non-generative reranker features

We examine the role of specific reranker features by training rerankers using only subsets of the features. Our goal is to determine whether some classes of reranker features benefit self-training more than others. We hypothesize that features which are not easily captured by the generative first stage parser are the most beneficial for self-training. If we treat the parser and reranking parser as different (but clearly dependent)

[5] Recall that the parser's probability is a reranker feature so the parsing model influences the ranking.

		f-score	
Feature set	# features	Original parser	Self-trained parser
GEN	448,349	89.8	90.4
NON-GEN	885,492	90.5	91.1
EDGE	601,578	90.2	91.0
NON-EDGE	732,263	90.3	91.1
ALL	1,333,519	90.5	91.3

Table 4.11: Sizes and f-scores of reranker feature subsets. Reranking parser f-scores are on all sentences in section 24. Recall that the original parser without a reranker gets an f-score of 89.0% on this section.

views, this is a bit like co-training. If the reranker uses features which are captured by the first stage, the views may be too similar for there to be an improvement.

We consider two classes of features (GEN and EDGE) and their complements (NON-GEN and NON-EDGE).[6] GEN consists of features that are roughly captured by the first stage generative parser: rule rewrites, head-child dependencies, etc. EDGE features describe items across constituent boundaries. This includes the words and parts of speech of the tokens on the edges between constituents and the labels of these constituents. This represents a specific class of features not captured by the first stage. These subsets and their sizes are shown in Table 4.11. For comparison, we also include the results of experiments using the full feature set, as in Section 3.1.1, labeled ALL. The GEN features are roughly one third the size of the full feature set.

We evaluate the effect of these new reranker models on self-training (Table 4.11). For each feature set, we do the following: We parse one million NANC sentences with the reranking parser. Combining the parses with WSJ training data, we train a new first stage model. Using this new first stage model and the reranker subset, we evaluate on section 24 of WSJ. GEN's performance is weaker while the other three subsets achieve almost the same score as the full feature set. This confirms our hypothesis that when the reranker helps in self-training it is due to features which are not captured by the generative first stage model.

4.4.4 Unknown Words

Given the large size of the parsed self-training corpus, it contains an immense number of parsing events which never occur in the training corpus. The most obvious of these events is words — the vocabulary grows from 39,548 to 265,926 words as we transition from the WSJ trained model to the self-trained model. Slightly less obvious is bigrams. There are roughly 330,000 bigrams in WSJ training data and approximately 4.8 million new bigrams in the self-training corpus.

One hypothesis (Mitch Marcus, personal communication) is that the parser is able to learn a lot of new bilexical head-to-head dependencies (biheads) from self-training. The reasoning is as follows: Assume the self-training corpus is parsed in a mostly correct manner. If there are not too many new pairs of words in a sentence, there is a decent chance that we can tag these words correctly and bracket them in a reasonable fashion from context. Thus, using these parses as part of the training data improves parsing because should we see these pairs of words together in the future, we will be more likely to connect them together properly.

[6] A small number of features overlap hence these sizes do not add up.

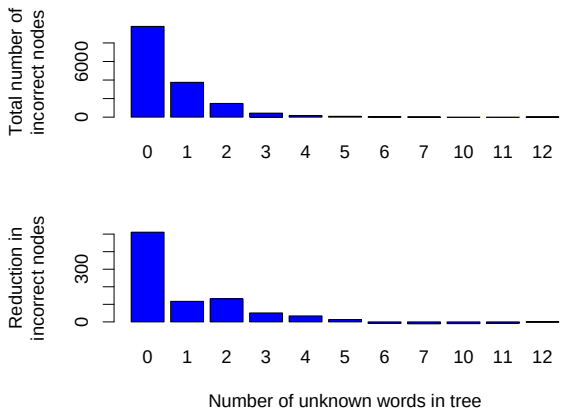
We test this hypothesis in two ways. First, we perform an extension of the feature selection similar to that in Section 4.2. This is done via a generalized linear regression model intended to determine which features of parse trees can predict when the self-training model will perform better. We consider many of the same features (e.g. bucketed sentence length, number of conjunctions, and number of unknown words) but also consider two new features: unknown bigrams and unknown biheads. Unknown items (words, bigrams, biheads) are calculated by counting the number of items which have never been seen in WSJ training but have been seen in the parsed NANC data. Given these features, we take the f-scores for each sentence when parsed by the WSJ and self-trained models and look at the differences. Our goal is to find out which features, if any, can predict these f-score differences. Specifically, we ask the question of whether seeing more unknown items indicates whether we are more likely to see improvements when self-training.

The effect of unknown items on self-training's relative performance is summarized in Figure 4.2. For each item, we show the total number of incorrect parse nodes in sentences that contain the item. We also show the change in the number of correct parse nodes in these sentences between the WSJ and self-trained models. A positive change means that performance improved under self-training. In other words, looking at Figure 4.2a, the greatest performance improvement occurs, perhaps surprisingly, when we have seen no unknown words. As we see more unknown words, the improvement from self-training decreases. This is a pretty clear indication that unknown words are not a good predictor of when self-training improves performance.

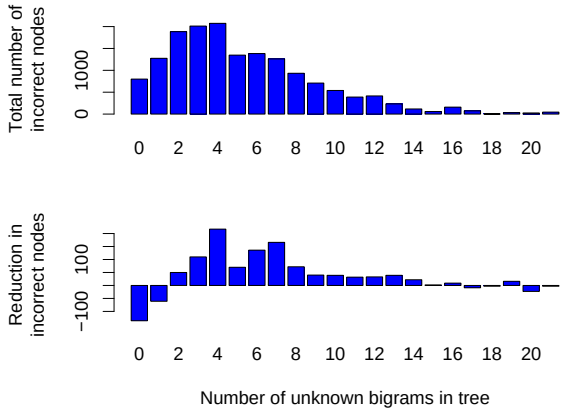
A possible objection that one might raise is that using unknown biheads as a regression feature biases our results if they are counted from gold trees instead of parsed trees. Seeing a bihead in training causes the otherwise sparse biheads distribution to be extremely peaked around that bihead. If we see the same pair of words in testing, we are likely to connect them in the same fashion. Thus, if we count unknown biheads from gold trees, this feature may explain away other improvements: When gold trees contain a bihead found in our self-training data, we almost always see an improvement. However, given the similar trends in Figures 4.2b and 4.2c, we propose that unknown bigrams can be thought of as a rough approximation of unknown biheads.

The regression analysis reveals that unknown bigrams and unknown biheads are good predictors of f-score improvements. The significant predictors from Section 4.2 such as the number of conjunctions or sentence length continue to be helpful whereas unknown words are a weak predictor at best. These results are apparent in Figure 4.2: as stated before, seeing more unknown words does not correlate with improvements. However, seeing more unknown bigrams and biheads does predict these changes fairly well. When we have seen zero or one new bigrams or biheads, self-training negatively impacts performance. After seeing two or more, we see positive effects until about six to ten after which improvements taper off.

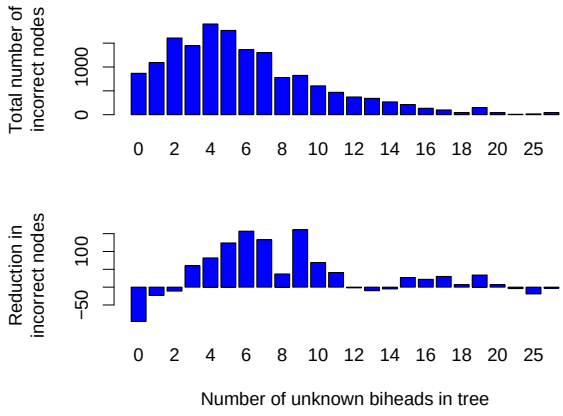
To see the effect of biheads on performance more directly, we also experiment by interpolating between the WSJ and self-trained models on a distribution level. To do this, we take specific distributions (see Section 2.1) from the self-trained model and have them override the corresponding distributions in a compatible WSJ trained model. From this we hope to show which distributions self-training boosts. According to the biheads hypothesis, the biheads distribution (which captures information about head-to-head dependencies) should account for most of the improvement.



(a) Effect of unknown words on performance



(b) Effect of unknown bigrams on performance



(c) Effect of unknown biheads on performance

Figure 4.2: Change in the number of incorrect parse tree nodes between WSJ and self-trained models as a function of number of unknown items. Seeing any number of unknown words results in fewer errors on average whereas seeing zero or one unknown bigrams or biheads is likely to hurt performance.

f-score		Model
89.8	$*$	WSJ (baseline)
89.8	$*$	WSJ + NANC M
89.9	$*$	WSJ + NANC T
89.9	$*$	WSJ + NANC L
90.0	$*$	WSJ + NANC R
90.0		WSJ + NANC MT
90.1		WSJ + NANC H
90.2		WSJ + NANC LR
90.3		WSJ + NANC LRT
90.4		WSJ + NANC LMRT
90.4		WSJ + NANC LMR
90.5		WSJ + NANC LRH
90.7	$\circledast$	WSJ + NANC LMRH
90.8	$\circledast$	WSJ + NANC (fully self-trained)

Table 4.12: Performance of the first stage parser under various combinations of distributions from the WSJ and WSJ+NANC (self-trained) models on sections 1, 22, and 24. Distributions are L (left expansion), R (right expansion), H (head word), M (head phrasal category), and T (head POS tag). $*$ and $\circledast$ indicate the model is not significantly different from baseline and self-trained model, respectively.

The results of swapping these distributions is shown in Table 4.12. For each new model, $*$ indicates that the model's performance is not significantly different than the baseline WSJ model and $\circledast$ that it is not significantly different than the self-trained model. H (biheads) is the strongest single feature and the only one to be significantly better than the baseline. Nevertheless, it is only 0.3% higher, accounting for 30% of the full self-training improvement. In general, the performance improvements from distributions are additive ($+/-$ 0.1%). Self-training improves all distributions, so biheads are not the full picture. Nevertheless, they remain the strongest single feature.

4.5 Summary

The experiments in this chapter have clarified many details about the nature of self-training for parsing. We have shown that the phase transition hypothesis does not explain when self-training is effective. Reduced search errors are responsible for some, but not all, of the improvements in self-training. We have confirmed that non-generative reranker features are more beneficial than generative reranker features since they make the reranking parser more different from the base parser. Finally, we have found that while unknown bigrams and biheads are a significant source of improvement, they are not the sole source of it. Since unknown words do not correlate well with self-training improvements, we believe it is the unknown bigrams and biheads which aid the parser in self-training. Our belief is that new combinations of words we have already seen guides the parser in the right direction. Additionally, these new combinations result in more peaked distributions which decreases the number of search errors.

However, while these experiments and others get us closer to understanding self-training, we still lack a complete explanation. Naturally, the hypotheses tested are by no means exhaustive. Additionally, we

have only considered generative constituency parsers here and a good direction for future research would be to see if self-training generalizes to a broader class of parsers. We suspect that using a generative parser/discriminative reranker paradigm should allow self-training to extend to other parsing formalisms and related tasks.

Finally, we believe that there are actually two different types of self-training happening, depending on the amount of labelled data available. Recall that in Reichart and Rappoport (2007) where only a small amount of labeled data was used, self-training was possible without the use of a reranker (see also our experiments in Section 4.4.1). Reichart and Rappoport (2007) showed that the number of unknown words in a sentence was a strong predictor of when self-training benefits. When a large amount of labeled data is available, unknown words are no longer correlated with these gains, but unknown bigrams and biheads are. Our theory is that when using a small amount of training data, unknown words are useful since we have not seen very many words yet and the increased lexical coverage is key. As the amount of training data increases, we see fewer new words but the number of new bigrams and biheads remains high. We postulate that this difference may help explain the shift from unknown words to unknown bigrams and biheads. We hope to further investigate the role of these unknown items by seeing how our analyses change under different amounts of labeled data relative to unknown item rates.

Chapter 5

Automatic Domain Adaptation

Until this point, our experiments have been designed given the identity of the domain being parsed. We have assumed we (i.e. human agents) can easily determine which domains would be useful as training data or for self-training. In many cases, this is not an unreasonable assumption. For example, the bioinformatics community's desire for highly accurate parses has produced the GENIA corpus as well as numerous other resources. However, relatively few domains have received this type of treatment and there are still many applications where this assumption does not hold. For example, consider the task of parsing text on an arbitrary web page. Each web page is potentially a different domain with a different optimal mixture of training corpora. Our goal is to explore methods of automatically creating parsing models tailored to specific target text, ideally without a significant drop in accuracy.

To study this, we create an extension to the standard semi-supervised parser adaptation task allowing for multiple source domains rather than just one. For example, suppose that we need to parse the BROWN corpus and we're given newswire text, biomedical abstracts, and automatically parsed Gutenberg books as possible training data. It is clear that the newswire text is helpful, (as seen in Section 3.2) the Gutenberg books are likely to help despite their automatic nature, and biomedical abstracts are probably not be as useful and may even hurt performance. However, the optimal mixture (i.e. weightings) of these training corpora is far from clear. In earlier experiments, we were able to tune our weightings based on development data (albeit in a time consuming manner). For this scenario, we are not allowed any in-domain development data.[1] We refer to this task as *multiple-source domain adaptation*.

Multiple-source domain adaptation also has applications to two issues that come up when applying self-training. We have shown earlier in Chapter 3 that self-training is a valuable tool for improving parser portability and parser adaptation. In our previous earlier experiments, the texts and base parsers used for self-training were selected by hand. Currently, no mechanism exists for automatically determining this. The choice of target domain can have a significant effect on parsing performance (as seen in Figure 3.2) thus selecting the correct training corpora is an important task. Our formulation of the problem allows us to answer all of these questions using the same framework.

[1] We note, though, that like other tasks, performance on this task should improve in the presence of in-domain data when available.

Our proposed solution, *Any Domain Parsing*, is based on the assumption that the target domain is a mixture of the source domains. That is, using a combination (linear, in this case) of the statistics from each source domain, we can create a new parsing model specifically for the target domain. Each source domain is treated atomically, though in theory they could be split to increase granularity.[2] We divide the task into two steps. The first step, *domain detection*, is to weight the importance of each source domain with regards to the target domain text. For the example mentioned in the second paragraph where the target domain is the BROWN corpus, this might mean giving a weight of 0.6 to the newswire text, 0.3 to the automatically parsed Gutenberg books, and 0.1 to the biomedical journal article abstracts. Given these weights, we linearly combine each source domain and create a new parsing model in the second step (*model combination*). Domain detection is the focus of our exploration.[3] Broadly put, we learn a model of how domain differences influence parsing accuracy. This is done by taking several computational measures of domain differences (*domain divergence measures*) between the target text and each source domain. We use these measures as features in a regression model. The regression model predicts the accuracy of the model produced by the source domain mixture on the target domain. To parse the target text, one simply uses the best predicted scoring source domain mixture. We show that our method is able to predict these accuracies quite well and that the source domain mixtures it suggests are among the best we have seen for parsing the target text in question.

Before delving into the details of our model, we note that the two step approach described above is not the only way do multiple-source domain adaptation, of course. One could imagine other approaches where existing resources are augmented or selected rather than used atomically (note, though, that some of these approaches would simply correspond to a more sophisticated model combination function). Additionally, while we use regression in our first step, there are additional ways of formulating the regression problem as well as classification-based approaches one could use instead. As an example of another way to phrase the regression problem, imagine that each domain is a point in some space. The axes of these spaces are the relative weights of each source domain and moving around in the space corresponds to choosing different mixtures of source domains. One could use multi-dimensional regression to learn where a new corpus should map into this domain mixture space and then perform model combination as before. However, this has the downside that it results in a small number of training data points (one data point per target domain) which would create severe data sparsity. Our formulation of the regression allows for an essentially unbounded number of possible training data points.

Note that in this chapter, our experiments only use the first-stage parser. This is because our approach is specific to models with easily blendable models. However, while reranker models cannot be linearly interpolated like generative parsing models, the scores in their outputs can still be blended. Our two stage detect-and-combine framework is thus still applicable. A simpler approach would be to use a single reranker model for all domains. While this may seem unsatisfying, from earlier experiments (Sections 3.2.1 and 3.2.2) we can see that using the WSJ-trained reranker improves performance for many domains. It is likely that a reranker trained from all available domains would perform even better across multiple domains. As with the first-stage parser, we could simultaneously train reranker models for all available domains while explicitly

[2] Splitting could be done using the divisions given in each corpora, e.g. article boundaries in WSJ. Alternatively, the splits could be created automatically via syntactically-driven clustering.

[3] However, we will discuss some possible variations of the model combination step.

learning which features were domain-specific (Daumé III, 2007; Finkel and Manning, 2009).

In Section 5.1, we detail recent work on similar tasks. Domain detection, the first step of our system, is covered in Section 5.2. Section 5.3 describes the second step where the weights from domain detection are used as input to a model mixing procedure to produce a new parsing model. We describe an evaluation strategy in Section 5.4. Since multiple-source domain adaptation is a new parsing task, we have created some baselines and upper bounds to give a sense of current approaches to the task (Section 5.4.1). The results of our experiments are detailed in Section 5.5 where we show that our system outperforms all non-oracle baselines. In our discussion (Section 5.6), we describe how to apply our model to the questions raised in this section.

5.1 Related work

The closest work to ours is Plank and Sima'an (2008) where unlabeled text is used to group text within WSJ into subdomains. The authors create a model for each subdomain which weights trees from its subdomain more highly than others. The weights are based on probabilities from an n-gram language model. Given the domain specific models, they consider two parser combination strategies. The first approach is to pick a single model to parse the target domain. The second technique parses sentences using models from all subdomains and creates a single tree from their outputs (along similar lines to Sagae and Lavie (2006)). Unfortunately, these methods do not result in a statistically significant improvement.

Multiple source domain adaptation has been done for other tasks, e.g. classification in (Blitzer et al., 2007; Daumé III, 2007; Dredze and Crammer, 2008) and is related to multitask learning. Daumé III (2007) shows that an extremely simple method delivers solid performance on a number of domain adaptation classification tasks. This is achieved by making a copy of each feature for each source domain plus the "general" pseudodomain (for capturing domain independent features). This allows the classifier to directly model which features are domain-specific and share the statistics of the rest. Finkel and Manning (2009) demonstrate the hierarchical Bayesian extension of this where domain-specific models draw from a general base distribution. This is applied to classification (named entity recognition) as well as dependency parsing. Dredze and Crammer (2008) approach this problem by combining multiple confidence-weighted linear classifiers. All of these works have the nice property that they extend naturally to any number of source domains. However, it is not obvious how work on classifiers can be applied to our parsing model (though it would fit nicely with a parsing model based on classifiers such as Ratnaparkhi (1999)). Additionally, these works describe how to train models in many different domains but sidestep the problem of domain detection. Thus, our work could be combined with theirs.

Our domain detection step draws on work in parser accuracy prediction (Ravi et al., 2008; Kawahara and Uchimoto, 2008). These works aim to predict the parser performance on a given target sentence. Ravi et al. (2008) frame this as a regression problem. Kawahara and Uchimoto (2008) treat it as a binary classification task and predict whether a specific parse is at a certain level of accuracy or higher. Some examples of features used in these systems include sentence length, estimates of lexical and orthographic difficulty (rarer/unknown words and more sentence-internal punctuation marks like commas tend to make parsing less accurate), and

	Test						
Train	BNC	GENIA	BROWN	SWBD	ETT	WSJ	Average
GENIA	66.3	**83.6**	64.6	51.6	69.0	66.6	67.0
BROWN	81.0	71.5	**86.3**	79.0	80.9	80.6	79.9
SWBD	70.8	62.9	75.5	**89.0**	75.9	69.1	73.9
ETT	72.7	65.3	75.4	75.2	81.9	73.2	73.9
WSJ	**82.5**	74.9	83.8	78.5	**83.4**	**89.0**	**82.0**
Average	74.7	71.6	77.1	74.7	78.2	75.7	75.3

Table 5.1: Cross-domain parser performance (first stage parser only). Averages are macro-averages. Unsurprisingly, most domains perform the best when parsing themselves (the lone exception is ETT which is better parsed by WSJ possibly due to decreased sparsity). On average, WSJ is the most accurate. GENIA and SWBD have the highest variation.

structural information such as the counts of labels in the parse tree. While accurately predicting the accuracy of a sentence is not our primary concern, we are interested in the relative performance of parsing under different source model combinations and we incorporate several of their features. Ravi et al. (2008) show that their system can be used to return a ranking over different parsing models which we extend to the multiple domain setting. They also demonstrate that training their model on WSJ allows them to accurately predict parsing accuracy on the BROWN domain. In contrast, our models are trained with multiple domains in mind giving them a better sense of which factors influence cross-domain performance.

5.2 Domain detection

The goal of this subtask is to predict the relative proportions of our source domains which should be used to parse a given set of target text. Our input consists of a vector of labeled source domains $\mathbf{C}$ (each ideally its own domain, though this is not required) and unlabeled target text t. Our goal is to produce domain divergence functions which assign weights to the labeled corpora, $\mathsf{detect} : \mathbf{C} \times t \rightarrow \mathbf{w}$ where $\mathbf{w}$ is a weight vector of positive real numbers with the same cardinality as $\mathbf{C}$. Higher weights in $\mathbf{w}$ should indicate that the corresponding source domain is more similar to the target text.

Imagine a very simple approach to this problem involving a basic notion of how domains differ (e.g. cosine similarity between vectors of common word frequencies). We'll refer to these notions as *domain divergence measures*. The approach is to weight each source domain in proportion to its "closeness" to the target text where closeness is determined by the domain divergence measure.

While this method is a good first approximation to a solution, it is likely to run into difficulties. Our problems can be summarized by the phrase "not all corpora are created equally" — that is, some corpora are larger, more accurate, and/or more general. Smaller corpora result in less accurate predictions for domain divergence measures due to data sparsity. Regarding accuracy, while we hope that all our labeled corpora are of comparable accuracy and their annotations standardized, our system should also be able to make effective use of self-trained corpora. Our system needs a mechanism to ensure that our self-training corpora obtain a reasonable weight relative to the hand-labeled corpora. Finally, regarding generality, we can see in Table 5.2

that WSJ performs quite well over a range of different corpora whereas GENIA works well only within its own domain. There is also the issue that this strategy ties us to a single domain divergence measure when better performance may be achieved by a combination of domain divergence measures.

To handle these issues, our approach follows a machine learning-inspired route to automatically learn per-corpora biases and which features are useful for predicting cross-domain performance with a regression model. Our detect function uses the predictions of the regression model to determine its results. Each input to the regression model describes a mixture of source domains (i.e. a distribution over them) and the domain divergences between those source domains and the target text in question. Regression outputs are the f-score of the parsing model created from the source domain mixture on the target text. By using multiple domain divergence measures, we allow them to complement each other, possibly providing more reliable estimates of domain divergence. Our system is similar to Ravi et al. (2008) inasmuch as they both use regression to predict f-scores and some of our features are similar.

Our inputs to the regressor can be any function of the source domain mixture (s) and the surface form of the target text (t):

$$\mathsf{predict}(s, t) = y$$

where y is the predicted f-score. Assuming we can build such a model with reasonable accuracy, the question remains: How would this be useful for the problem of creating the detect function? If we have a set of source domain models, $\mathcal{S}$, we can use our regression function to select the best model from this set:

$$\mathsf{detect}(\mathbf{C}, t) = \arg\max_{s \in \mathcal{S}} \mathsf{predict}(s, t)$$

where all source domains in each source domain mixture s are contained in $\mathbf{C}$. However, we can also attempt to optimize our regression function predict if it is convex and otherwise we can find local maxima. Now our $\arg\max$ function can (in theory) select any source domain mixture:

$$\mathsf{detect}(\mathbf{C}, t) = \arg\max_{s} \mathsf{predict}(s, t)$$

In terms of practical performance, these techniques do not differ significantly if one has a sufficiently large set of source domain models.

In the following subsections, we provide further details of the regression model. Our regression model itself is a generalized linear model (GLM), as outlined in Section 5.2.1. Section 5.2.2 describes the domain divergence functions and other functions of the source domain.

5.2.1 Regression model

One of the simplest forms of regression is the linear regression model. Linear regression learns functions where the output is linear combination of the inputs $\mathbf{x}$:

$$y = \mathbf{a}^T \mathbf{x} + b$$

When fitting the model to a set of inputs and outputs, the variables $\mathbf{a}$ and b are adjusted. Linear regression assumes a linear relationship between the inputs and output and that all inputs are independent. In practice,

linear regression works best among all the regression models that we explored so the linear relationship appears to be at least a reasonable assumption overall. It is possible that a more sophisticated regression model which mapped each feature onto its own scale would perform better.

For completeness, we describe the other regression models that we explored. The generalized linear model allows us to break down the first assumption. In this case, we use a link function, g, which allows us to map the inputs nonlinearly:

$$y = g^{-1}(\mathbf{a}\mathbf{x} + b)$$

Note that if the link function is the identity, generalized linear regression reduces to linear regression. The link function in a generalized linear model is actually an estimation of the distribution of errors between our prediction and the mean of the output:

$$\mathrm{E}[y] = \mu = g(\mathbf{a}\mathbf{x} + b)$$

In our specific case, this would be beneficial if the components of $\mathbf{x}$ (i.e. our domain divergence measures along with the other features) were not on a linear scale and applying the link function resulted in a linear between inputs and outputs. However, it would still require that they all be on the same scale. Naturally, if one has prior knowledge that a component of $\mathbf{x}$ is on a specific nonlinear scale, one can preprocess that component to make it more amenable to linear regression.

Prediction and estimation are fairly straightforward. The learned function can be used immediately to predict new outputs for a given input. These models are typically estimated using least squares methods to obtain the maximum likelihood estimate. More details on the actual data points for this estimation is forthcoming. The next section describes what each data point looks like and Section 5.5.2 where explain the origin of the points.

In our experiments, we explored several different families of error distributions including Gaussian (which uses the identity link function), gamma, inverse gamma, and Poisson. As stated earlier, the Gaussian error distribution outperformed the others. In pilot studies, we also explored other forms of regression such as Locally Weighted Projection Regression (LWPR) (Vijayakumar et al., 2005). LWPR essentially works by clustering the data points and finding local linear approximations of each cluster. However, LWPR did not perform as well as linear regression in our pilot studies. Our regression training dataset is somewhat small which may have made it tricky for LWPR to find decent clusters. Additionally, LWPR is stochastic and for this task we prefer to have a more stable prediction function.

5.2.2 Domain divergence measures and other features

We describe the possible features which are designed to help the regression model determine if a particular source domain mixture is well suited for a specific target domain. Some of these features directly connect the two (domain divergence measures) whereas others serve as general information about the source domain mixture. The latter allow the regression model to capture nonlinear patterns about good source domain mixtures (e.g. how many/which source domains should be used, how uniform the distribution should be, etc.).

As stated earlier, domain divergence measures are designed to approximate how different the target domain is from a specific source domain. Only the surface form of the target text and automatic analyses are available (e.g. we can tag or parse the target text but cannot use gold tags or trees). Our features make use of word frequency, vocabularies, sentence lengths, and simple n-gram language models.

While this section describes all the features that we explored, we note that our feature selection step selected only three of them (COSINETOP50, UNKWORDSREV, and ENTROPY from uniform). The rest of the details are here for completeness.

Word frequencies and vocabulary are an important indicator of domain. We can use a spatial representation to summarize the vocabulary of a corpus as a single vector. Vector space approaches map each source domain to a point in a metric space allowing distances between domains to be measured using standard methods (Euclidean, Manhattan, cosine similarity etc.). Naturally, there are a large number of ways that the contents of a corpus can be mapped into vector space. A common method is to represent each corpus as a vector of frequencies of the k most frequent words (Schütze, 1995). To find the k most frequent words across corpora, we take the count of all words in each corpus divided by the number of tokens in the corpus and sum these counts across all corpora. This ensures that our list of the most frequent words are not dominated by the words in our larger corpora. The vectors are normalized to ensure that they all have the same magnitude. Typically, one applies dimensionality reduction (e.g. singular value decomposition) to these points to focus on the most salient differences, though in our case we do not have enough points to warrant dimensionality reduction. Under this setup, this method assigns high similarity to domains with a large amount of overlap in the high-frequency vocabulary items. We create these vectors and use cosine similarity (i.e. the angle between the two vectors) to determine the divergence between two domains:

$$\mathsf{similarity}(\vec{A}, \vec{B}) = \frac{\vec{A} \cdot \vec{B}}{\|\vec{A}\| \, \|\vec{B}\|}$$

We refer to these features as COSINETOPK where K $\in (5, 50, 500, 5000)$ indicates how many of the most frequent words we include in our vector. We show the top 50 words across our corpora in Figure ref-fig:Top50Words. An example of the raw values from the COSINETOP5000 domain divergence measure can be found in Figure 5.2a. Note how WSJ is fairly similar to almost all domains whereas SWBD is similar only to itself. Additionally, we can use cosine similarities over vectors of punctuation (COSINEPUNC). Domains differ in their usage of punctuation (e.g. we would expect WSJ to use quotation marks more than other corpora) so these statistics may provide a fast way of distinguishing gross differences in domain.

Another way of comparing the vocabularies across domains is to determine how many words would be unknown if one built a vocabulary from a different domain. This can be done on the word type or word token level. We opt for the word token level since unknown words pose problems for parsing each time they occur. The domain divergence measure UNKWORDS computes the percentage of word tokens in the target domain that are unknown given the source domain's vocabulary. UNKWORDSREV is the same idea with the source and target domains swapped (i.e. percentage of words in the source domain that are never seen in the target domain). The raw values of the latter feature can be seen in Figure 5.2b. The broad trend of this chart is that GENIA and MEDLINE are similar to each other but very dissimilar from everything else. Other domains tend to have about a 5–15% unknown word rate with themselves and 20–25% unknown word rate

```
, the .  of and to in a that is i for it ( was with ) 's by on as '' ""
you he we this from at not but be have are or were they an his uh had
which cells n't has said do one there all
```

Figure 5.1: Fifty most frequent words across all corpora sorted by decreasing frequency. Unsurprisingly, nearly all of these words are closed-class words. "cells" appears due to its high frequency in the MEDLINE corpus.

with non-medical domains.

We also consider language models as domain divergence measures. Given a language model and some input text, a language model estimates the probability of producing the input text. Thus, given a collection of domains we can make a language model for each domain and find the probability of generating the target domain text from each source domain. The most common form of language models use n-grams (c.f. (Chen and Goodman, 1996; Goodman, 2001; Brants and Franz, 2006)) which assume that generating each word in a sentence is only conditioned on the previous $n-1$ words. More sophisticated approaches (Chelba and Jelinek, 1998; Roark, 2001; Charniak, 2001; Xu et al., 2002) take syntax into account and generate all possible trees which have the sentence as the leaves. The probability of each sentence is the sum of the probabilities of each of its possible trees. We could potentially use a syntax-based language model which would allow us to make full use of the syntactic trees in our training data. Using a language model for domain detection may be able to make better use of context than the other approaches. For experiments, we use simple Kneser-Ney smoothed trigram model with an open vocabulary from SRILM.[4] We create three domain divergence measures corresponding to the three scores from the language model (perplexity, perplexity ignoring sentence boundaries, and log probability).

A quick study of sentence lengths revealed that these may vary substantially across domains (see Tables 2.1 and 2.2). We allow our model to capture this information by introducing two domain divergence measures, AVERAGELENGTH and AVERAGELENGTHDIRECTED. Both features examine the difference between the average sentence lengths. The former feature returns the absolute value of these differences.

Note that since domain divergence measures merely measure the similarity between each source domain and target text, the raw values of measures must remain constant across all source domains mixtures parsing the same target text. In other words, the raw domain divergence measures fail to distinguish any source domains from each other. Naturally, this is undesirable. Thus, our computed domain divergence measures must connected to their corresponding source domain mixture weight in some way. The best method which we have found is to divide the mixture weight of the source domain by the divergence. When the source is not used, the adjusted divergence is zero regardless of the raw divergence (which is reasonable). Given the choice between two source domains, we obtain a higher adjusted divergence score from the source domain with the smaller divergence measure, thus encouraging our system to use the more similar (i.e. less divergent) source domains. Thus, we apply this procedure to all domain divergence measures. For the remainder of this chapter, domain divergence measures will refer to their computed values unless explicitly stated.

In addition to the domain divergence measures, we include several features which are purely a function

[4] SRILM is available from `http://www.speech.sri.com/projects/srilm/`

Source domain	BNC	GENIA	BROWN	SWBD	ETT	WSJ
GENIA	0.894	0.998	0.860	0.676	0.887	0.881
MEDLINE	0.911	0.977	0.875	0.697	0.895	0.897
BROWN	0.976	0.862	0.999	0.828	0.917	0.960
GUTENBERG	0.982	0.868	0.977	0.839	0.929	0.957
SWBD	0.779	0.663	0.825	0.992	0.695	0.789
ETT	0.971	0.896	0.937	0.766	0.992	0.959
WSJ	0.968	0.880	0.963	0.803	0.941	0.997
NANC	0.983	0.888	0.979	0.801	0.950	0.987

(a) Divergences from COSINETOP5000. Higher values are more similar.

Source domain	BNC	GENIA	BROWN	SWBD	ETT	WSJ
GENIA	33.3	10.8	40.5	45.8	43.1	38.9
MEDLINE	32.5	21.5	36.5	45.4	42.0	35.5
BROWN	14.3	38.5	10.7	21.5	22.7	18.3
GUTENBERG	16.0	36.9	14.3	23.7	23.2	20.0
SWBD	9.0	30.6	6.1	4.6	11.1	11.4
ETT	18.1	35.3	17.4	22.1	10.3	16.6
WSJ	23.1	41.1	22.5	30.1	25.4	14.2
NANC	20.4	39.8	19.3	27.1	24.5	18.3

(b) Divergences from UNKWORDSREV. Lower values are more similar.

Figure 5.2: Raw values from two domain divergence measures. We use the training division for the source domains and the development division for the target text — this is why the charts are not symmetric even for symmetric measures like cosine similarity. This is also why, for example, the WSJ source domain doesn't have cosine similarity 1 with the WSJ target text. Cells have been colored from white (more similar) to black (less similar). For self-trained corpora (GUTENBERG, MEDLINE, and NANC) we do not list the base parser used to parse it since neither of these divergence measures use information from syntactic trees and thus gave the same scores regardless of base parser.

of the source domain mixture. From pilot studies, we learned that models with a large number of source domains tended to perform well. We created several features with this in mind to allow the regression model to capture the relevant properties. One feature, "# source domains used," lets the regressor learn a weight for adding or removing an arbitrary source domain. In case this is too coarse we create a NONZERO feature for each source domain which is 1 when the source domain is given positive weight. To capture the uniformity of the distribution of source domains, we introduce the ENTROPY which measures the entropy of the distribution over source domains. Finally, to provide the regression model with a control for how much self-trained data is used, we create a feature which measures the percent of the source domain mixture which consists of self-trained corpora.

5.3 Model combination

Given parsing models for each source domain, $\mathbf{m}_1, \ldots, \mathbf{m}_N$, and a mixing distribution, $\lambda_1, \ldots, \lambda_N$, over the source domains as input, this step combines them into a new parsing model. As mentioned before, $\mathsf{mix}(\mathbf{m}, \lambda)$ creates a new model by linearly interpolating models $\mathbf{m}$ using weights λ. In this model, the probability of parsing event e is

$$P_{\text{mix}}(e) \propto \lambda_1 C_1(e) + \lambda_2 C_2(e) + \cdots + \lambda_N C_N(e)$$

where $C_k(e)$ is the count of event e in parsing model $\mathbf{m}_k$. This approach performs *mixing by counts* but we could also do *mixing by models*: (Bacchiani et al., 2006)

$$P_{\text{mix}}(e) = \lambda_1 P_1(e) + \lambda_2 P_2(e) + \cdots + \lambda_N P_N(e)$$

where $P_k(e)$ is the probability of event e in parsing model $\mathbf{m}_k$. While these both have similar forms, they make different predictions. As in Bacchiani et al. (2006), we expect mixing by counts to perform better than mixing by models.

One detail to consider is that models may be mixed completely or on a per-component basis (in this case, component refers to one of the five distributions that make up the parsing model described in Section 2.1). For example, we could allow for a different mixing distribution for the left and right expansion components (L and R) than for the biheads component (H). This is potentially useful since the left and right expansion components are more likely to be shared across models than the biheads component because the latter encodes more lexical information. To do mixing at this level, we would need to train separate regression functions for each component which may be too computationally expensive. Alternatively, we could select only a subset of distributions to mix. In future work, we plan to investigate parser portability on a per-component level. This could result in better model combination strategies.

5.4 Evaluation

Multiple-source domain adaptation is a new task for parsing and thus some thought must be given to evaluation methodology. We describe two evaluation scenarios which differ in how foreign the target text is from

Train		Test	
Source	Target	Source	Target
$\mathbf{C} \setminus \{t\}$	$\mathbf{C} \setminus \{t\}$	$\mathbf{C} \setminus \{t\}$	$\{t\}$

(a) Out-of-domain evaluation

Train		Test	
Source	Target	Source	Target
$\mathbf{C}$	$\mathbf{C} \setminus \{t\}$	$\mathbf{C}$	$\{t\}$

(b) In-domain evaluation

Table 5.2: List of domains allowed in single round of evaluation. In each round, the evaluation corpus is t. $\mathbf{C}$ is the set of all target domains. For example, when training a domain detection system in the in-domain scenario, one may build models using all domains and evaluate them on any domain except t. One detail not shown is that any derived corpora are removed as well (i.e. if $t =$ WSJ, we must remove NANC as well since NANC is created from a WSJ base parser).

our source domains. Schemas for these evaluation scenarios are shown in Table 5.2. Note that training and testing here refer to training and testing of our regression model, **not** the parsing models which are trained in the conventional fashion.

In the first scenario, *out-of-domain evaluation*, one target domain is completely removed from consideration and only used to evaluate proposed models at test time. The regressor is trained on training points that use any of the remaining corpora, $\mathbf{C} \setminus \{t\}$, as sources or targets. For example, if $t =$ WSJ, we can train the regressor on all data points which don't use WSJ (or any self-trained corpora derived from WSJ) as a source or target domain. At test time, we are given the text of WSJ's test set. From this, our system creates a parsing model using the remaining available corpora for parsing the raw WSJ text.

This evaluation scenario is intended to evaluate how well our system can adapt to an entirely new domain with only raw text from the new domain (for example, parsing biomedical text when none is available in our list of source domains). Ideally, we would have a large number of web pages or other documents from other domains which we could use solely for evaluation. Unfortunately, at this time, only a handful of domains have been annotated with constituency structures under the same annotation guidelines. Instead, we hold out each hand-annotated domain, t, (including any automatically parsed corpora derived from that source domain) as a test set in a round-robin fashion.[5] For each round of the round robin we obtain an f-score and we report the mean and variance of the f-scores for each model.

The second scenario, *in-domain evaluation*, allows the target domain, t, to be used as a source domain in training but not as a target domain. This is intended to evaluate the situation where the target domain is not actually that different from our source domains. The in-domain evaluation can approximate how our system would perform when, for example, we have WSJ as a source domain and the target text is news from a source other than WSJ. Thus, our model still has to learn that WSJ and the North American News Text corpus (NANC) are good for parsing news text like WSJ without seeing any direct evaluations of the sort (WSJ and NANC can be used in models which are evaluated on all *other* corpora, though).

[5] Thus, the schemas in Table 5.2 are schemas for each round.

5.4.1 Baselines

Given that this is a new task for parsing, we needed to create baselines which demonstrate the current approaches to multiple-source domain adaptation. One approach is to take all available corpora and mix them together uniformly.[6] The UNIFORM baseline does exactly this using the available hand-built training corpora. SELF-TRAINED UNIFORM uses self-trained corpora as well in its mixtures. In the out-of-domain scenario, these exclude the held out domain. When used in the in-domain setting, the held out domain is included. These baselines are similar to the ALL and WEIGHTED baselines in Daumé III (2007).

Another simple baseline is to use the same parsing model regardless of target domain. This is essentially how large heterogeneous document collections are generally handled currently. We use the WSJ corpus since it is the best single corpus for parsing all six domains (see Table and Section 3.2). We refer to this baseline as FIXED SET: WSJ. In the out-of-domain scenario, we fall back to SELF-TRAINED UNIFORM when the target domain is WSJ while the in-domain scenario uses the WSJ model throughout.

There are several interesting oracle baselines as well which serve to measure the limits of our approach. These baselines examine the resulting *f*-scores of models and pick the best model according to some criteria. The first oracle baseline is SINGLE CORPUS which parses each corpus with the training corpus that maximizes performance on the test corpus. In almost all cases, this baseline selects each corpus to parse itself when possible.[7] This baseline roughly corresponds to a human picking the appropriate source domain in each case (though it could easily outperform the human given some of the surprises we have seen).

Our second oracle baseline, BEST SEEN, chooses the best parsing model from all those explored for each test set. Recall that while training the regression model in Section 5.2.1, we needed to explore many possible source domain mixtures which approximate the complete space of mixed parsing models. To the extent that we can fully explore the space of mixed parsing models, this baseline represents an actual upper bound for model mixing approaches. Since fully exploring the space of possible weightings is intractable, it is not a true upper bound. Nevertheless, we believe that we have obtained sufficient samples (we elaborate on this when we describe our sampling strategy in Section 5.5.2). While it is theoretically possible to beat this baseline, (indeed, this is the mark of a good domain detection system) it is far from easy. We provide SINGLE CORPUS and BEST SEEN for both in-domain and out-of-domain scenarios. The out-of-domain scenario restricts the set of possible models to those not including the target domain.

Finally, we searched for the BEST OVERALL MODEL. This is the model with the highest average *f*-score across all six target domains. This baseline can be thought of as an oracle version of FIXED SET: WSJ and demonstrates the limit of using a single parsing model regardless of target domain. Naturally, the very nature of this baseline places it only in the in-domain evaluation scenario. Since it was able to select the model according to *f*-scores on our six target domains, its performance on domains outside that set is not guaranteed.

To provide a better sense of the space of mixed parsing models, we also provide the WORST SEEN

[6] Accounting for corpus size so that the larger corpora don't overwhelm the smaller ones.

[7] For corpora that are too small to have both training and testing divisions, (BNC in our case) this baseline has to choose a different corpus. Additionally, WSJ actually performs better at parsing ETT than the training portion of ETT — this is most likely due to the small size of ETT.

baseline which picks the worst model available for a specific target corpus.[8]

5.5 Experiments

We discuss the specifics of our experiments in this section. We start with our rationale for the selection of source and target domains (Section 5.5.1). Next, we describe our strategy for randomly sampling parsing models and empirical evidence that we have enough samples despite the sampling space's high dimensionality in Section 5.5.2. In Section 5.5.3, we describe a greedy strategy for picking which features (domain divergence measures and other features) to include in our regression model. The results of our experiments with baseline comparisons are described in Section 5.5.5. Finally, we conclude this chapter with some discussion (Section 5.6).

5.5.1 Corpora

We had a variety of goals for selecting the source and target domains which ultimately resulted in nine source domains and six target domains. The breakdown of how corpora are used is shown in Table 5.3. The primary goal was to cover as many source domains as possible. Ideally, we would include a large number of self-trained texts and each hand-labeled corpus would have at least one self-trained text derived from it. Initially, we were concerned that the space of parsing models would be too large. We opted towards fewer corpora to make experiments more reproducible by other researchers. While there is essentially an infinite amount of raw text that we could use for self-training, we selected only four self-trained texts to use. We include GUTENBERG as a self-trained corpus as parsed by WSJ. Parsing GUTENBERG with WSJ rather than the presumably more closely matched BROWN surprisingly resulted in better performance during pilot studies. We also include two versions of the self-trained MEDLINE corpus — one parsed by GENIA, the other parsed by WSJ — to see if our system can learn preferences between the two. Lastly, we include the NANC corpus as parsed by WSJ. We omitted the BIOBOOKS corpus given its relatively small size and lower performance on GENIA. When training parsing models from self-trained corpora, we need to trees to use for tuning. Since these need to be gold trees, we use the development portion of base parser's corpus. In other words, the parsing model for MEDLINE (by WSJ) uses WSJ's development section for tuning. Finally, note that there is a mismatch in the number of source and target domains since BNC is too small to be used as a source domain. Self-trained corpora shouldn't be used as target domains as their trees are not necessarily correct.

As mentioned in Section 2.3, the corpora in this chapter have been preprocessed to standardize many of the differences in annotation. Thus, results on them are slightly different than in previous chapters. Nevertheless, we do not expect these changes to significantly impact overall performance.

5.5.2 Sampling parsing models

We wish to sample parsing models which have varied performance across all corpora to use as training data for our regression model. We present here a simple strategy which empirically achieves our goal. First, we

[8] This turns out to be GENIA for all corpora other than GENIA and SWBD when the target domain is GENIA.

Corpus	Source domain	Target domain
BNC		•
BROWN	•	•
ETT	•	•
GENIA	•	•
MEDLINE	•	
SWBD	•	•
WSJ	•	•
NANC	•	
GUTENBERG	•	
MEDLINE	•	

Table 5.3: List of source and target domains. Indented rows indicate self-trained corpora parsed using the non-indented row as a base parser.

sample the number of source domains to use. We draw values from an exponential distribution until we find one between two and the maximum number of source domains (nine, in our case). Using an exponential distribution encourages this number to be closer to two while still allowing for the occasional nine. This means that we try many different subsets of source domains. The λ parameter for the exponential distribution was adjusted by hand to place most probability mass on smaller numbers of source domains and we found that $\lambda = 0.4$ produces a reasonable curve.

Once we know the number of source domains, we sample their names uniformly at random without replacement from the list of all source domains. Finally, we sample the weights for the source domains uniformly from a simplex. The dimension of the simplex is the same as the number of source domains so we end up with a probability distribution over the sampled source domains.

In total, we created 1,040 sampled source domain mixtures and their corresponding parsing models. Each of these parsing models is evaluated on each of the six target domains giving us 6,240 data points total. To ensure that the simple cases are covered, we made 40 configurations which include each single source domains and several simple combinations of source domains. The above strategy was used to create 500 samples. For the remaining 500 samples, we made a small modification. Since our evaluation scheme excludes one target domain and all corpora derived from it, if we select the corpora to use uniformly at random, there is a good chance that the model could be excluded for a large number of target domains. To work around this, we rotated through the set of target domains, holding out each target domain and any derived corpora each time. This guarantees that each model is usable for at least one target domain.

As stated before, this is a large space and we were initially concerned about covering it. To alleviate these concerns, we show a graph of the cumulative oracle score for each corpus (Figure 5.3). Each data point is the average of the f-scores of the best parsing models seen for the six target domains (they need not be the same model). In other words, for each of the six target domains, the oracle is allowed to pick the best of the first k models when plotting the kth point. The shape of Figure 5.3 implies that we have covered the space well. After the first 200 samples, the cumulative oracle f-score does not increase much meaning that we find models which perform well for each domain quickly. This curve shows the performance for the *in-domain* cumulative oracle f-score — that is, where models trained on the target domain are included. The curve for

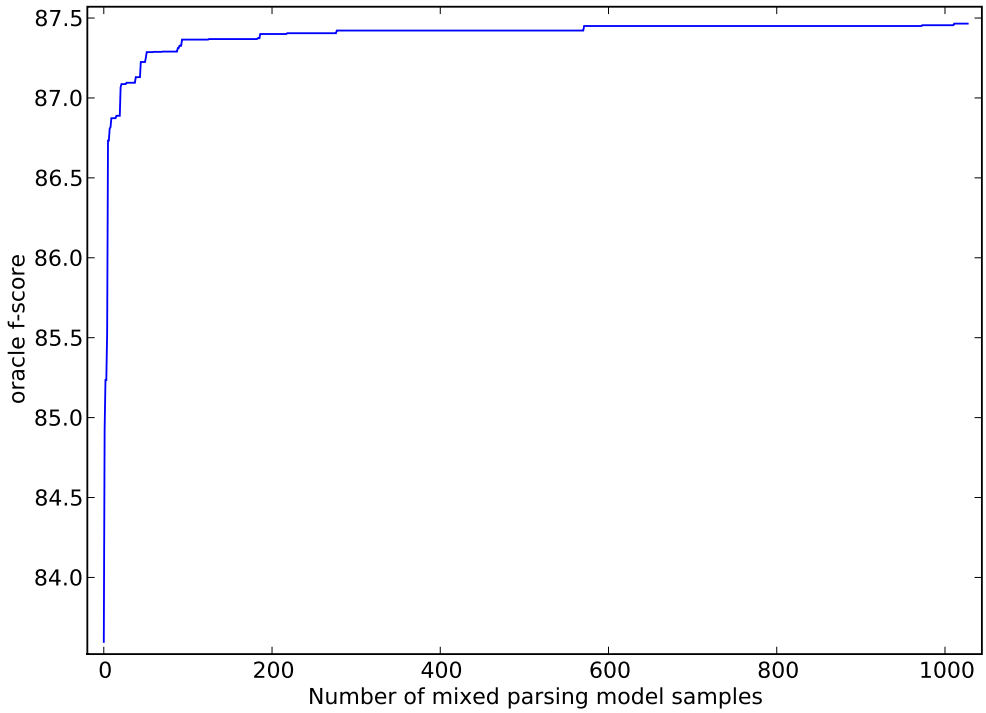


Figure 5.3: Cumulative oracle f-score (averaged over all target domains) as more models are randomly sampled. Most of the improvement comes the first 200 samples indicating that our samples are sufficient to cover the space of good source domain mixtures.

the more conservative *out-of-domain* cumulative oracle f-score has a similar shape but is nearly completely flat after the 400 samples instead of 200. In both cases, we sample more than enough points to reach a performance plateau.

5.5.3 Model and feature selection

In order to explore many different regression models and features for said models without hill climbing on our test data, we created a tuning scenario. Since the out-of-domain evaluation scenario holds out one target domain, this gives us six test evaluation rounds. For each of these six rounds, we hold out one of the remaining five target domains for tuning. This gives us 30 tuning evaluation rounds and we tune our parameters by optimizing our aggregate performance over all of them. A model that performs well in this situation has proven that it has useful features which transfer to unknown target domains.

The next step is to determine the loss function to optimize. Our primary guide is *oracle f-score loss* which is determined as follows. We take all test data points (i.e. points that evaluated on target domain) and predict their f-scores. In particular for this measure, we are interested in the point with the highest predicted f-score. We take its actual f-score and call that the *candidate f-score*. When tuning, we know the true f-scores of all test points. The difference between the highest f-score (the oracle f-score for this dataset) and the candidate f-score is the oracle f-score loss. Ties need to be handled correctly to avoid degenerate models. If there is a tie for highest predicted f-score, the candidate f-score is the one with the *lowest* actual f-score. This approach is conservative but ensures that regression models which give everything the same predicted f-score do not receive zero oracle f-score loss.

Since oracle f-score loss is only concerned with a single data point, we use two other loss functions to ensure a good holistic fit. The first is the common *mean squared error* where we sum the squared differences between the predicted and true f-scores. To encourage fits which do better on points with higher true f-scores, we also introduce *modified mean squared error*:

$$\sum_{\{\text{predicted,true}\}} |\text{true} - \text{predicted}|^{1+\text{true}}$$

Modified mean squared error interpolates between L_1 and L_2 loss as data points increase in their true f-score. Thus, errors on points with higher true f-scores are more heavily penalized.

Armed with a tuning regime and loss functions to guide us, we can now use them to select regression models and features for those models. We created a parallelized best-first feature searcher which performs best-first search. We provided it with several seed parameter settings (one for each domain divergence measure). These settings are evaluated in parallel to determine their oracle f-score loss. At each stage, the setting with the lowest loss is expanded by toggling all possible settings (e.g. if the setting included the COSINETOP5000 divergence measure, we create a copy of the setting without that domain divergence measure). These new settings are evaluated and the cycle repeats. If we exhaust all expansions of the setting with the lowest loss, we backtrack to the next best scoring setting.

For demonstration purposes, Table 5.4 provides an example regression input data point. It includes two domain divergence measures (COSINETOP5000 and UNKWORDSREV), the source domain distribution, whether each source domain is non-zero and the three other features of the source domain distribution.

Description	Value	Description	Value
% WSJ	—	COSINETOP5000: WSJ	0.000
% BROWN	6.9%	COSINETOP5000: BROWN	0.083
% GENIA	27.4%	COSINETOP5000: GENIA	0.405
% SWBD	—	COSINETOP5000: SWBD	0.000
% ETT	—	COSINETOP5000: ETT	0.000
% NANC	55.7%	COSINETOP5000: NANC	0.696
% GUTENBERG	10.0%	COSINETOP5000: GUTENBERG	0.119
% MEDLINE (by WSJ)	—	COSINETOP5000: MEDLINE (by WSJ)	0.000
% MEDLINE (by GENIA)	—	COSINETOP5000: MEDLINE (by GENIA)	0.000
used WSJ?	FALSE	UNKWORDSREV: WSJ	0.000
used BROWN?	TRUE	UNKWORDSREV: BROWN	0.321
used GENIA?	TRUE	UNKWORDSREV: GENIA	0.599
used SWBD?	FALSE	UNKWORDSREV: SWBD	0.000
used ETT?	FALSE	UNKWORDSREV: ETT	0.000
used NANC?	TRUE	UNKWORDSREV: NANC	2.057
used GUTENBERG?	TRUE	UNKWORDSREV: GUTENBERG	0.421
used MEDLINE (by WSJ)?	FALSE	UNKWORDSREV: MEDLINE (by WSJ)	0.000
used MEDLINE (by GENIA)?	FALSE	UNKWORDSREV: MEDLINE (by GENIA)	0.000
# source domains used	4		
ENTROPY	1.591		
% self-trained corpora	65.7%		

Table 5.4: An example regression input data point with SWBD as the target text. Percentages and booleans have been color-coded. Features in the left half of the table are functions solely of the source domain mixture whereas the right half has features which are functions of the target text as well. Only two domain divergence measures are listed (COSINETOP5000 and UNKWORDSREV) but in practice many others are available.

The best setting we found uses only three features: ENTROPY with the COSINETOP50 and UNKWORDSREV domain divergence measures. We evaluated over 6,000 settings for the GLM model, though this setting was found very early on (within the first 200 settings) so we have some degree of confidence that this is one of the best settings. The setting gets an average 0.37 oracle f-score loss on the 30 tuning datasets. The average unmodified and modified mean squared errors are 0.48 and 0.96 respectively. These settings make a reasonable schema for a regression model — it uses two relatively orthogonal domain divergence measures (see Figure 5.2.2) and ENTROPY feature allows it to prefer more uniform distributions and encourages it to use more source domains. The ENTROPY feature is especially valuable when considering the high performance of the Self-trained Uniform baseline (see Table 5.5).

5.5.4 Maximizing the regression function

Once we have trained our regression function, we can use it to select the model with the highest predicted f-score on the target domain. However, as mentioned in Section 5.2, we can also attempt to search for an even better model by maximizing the regression function. Recall that the regression function takes information about the source domain distribution and the target text as input. Here, we hold the target text constant and create a proxy function which takes weights (i.e. unnormalized probabilities) for the source domains. This proxy function computes the correct input to the regression function by normalizing its input and calculating any features of the source domain distribution (e.g. # source domains used, entropy, etc.). The proxy function

returns the predicted f-score on the target text. We optimize the proxy function with the L-BFGS-B constrained numerical optimizer package (Byrd et al., 1995; Zhu et al., 1997). We prefer a numerical optimizer with constraints since it allows us to constrain all parameters to be non-negative.

Unfortunately, due to features like ENTROPY and "# source domains used," the proxy function is highly non-convex. We are likely to get trapped in local maxima when optimizing it. To alleviate this, we perform ten numerical optimizations, each initialized from one of the points with the ten best predicted f-scores. We then use whichever optimized setting results in the highest predicted f-score. In practice, the predicted f-scores from these ten optimizations do not differ too greatly from each other. They tend to be about 0.5%-1% higher than the original predicted f-score. When actually evaluating these new settings in the tuning scenario, they show a small improvement over the previous non-optimized settings. The improvement, however, is probably not statistically significant. Nevertheless, we maximize the regression function for our final results.

The most exciting aspect of this experiment is that in several cases, the settings discovered by the optimizer are better than any we have seen from sampling. This indicates to us that the model is guiding us well towards better source domain mixtures. It is difficult to determine if there are global maxima in the proxy function which would result in significantly improved performance. Our hypothesis is that we would need a more sophisticated regression model rather than better numerical optimization here to close the gap between our system and the best seen settings. This takes into account the assumption that the best seen settings are in fact close to the best settings for this type of model combination, as supported by Section 5.5.2.

5.5.5 Results

We present an overview of our final results for out-of-domain and in-domain evaluation in Table 5.5. The results include the f-score averaged over the six target domains and the standard deviation. More detailed results on individual target domains can be seen in Figures 5.4 and 5.5. As stated earlier, these experiments use only the first-stage parser and thus have lower performance than some of our previous experiments.

Our system, Any Domain Parsing, is the best non-oracle system for both tasks. For out-of-domain evaluation, our system is only 0.3% worse than the best seen models for each target domain. For the in-domain scenario, we are within 0.6% of the BEST SEEN models. Additionally, our model is 0.7% better than the BEST OVERALL MODEL. Recall that the BEST OVERALL MODEL is the single model with the best performance across all six target domains. By beating this baseline, we show that there is value in customizing parsing models to the target domain.

Our baselines reveal some interesting properties of our task and corpora. In both situations, the FIXED SET: WSJ baseline performs fairly poorly. Not surprisingly, assuming all of our target domains are close enough to WSJ works badly for our set of target domains and it does particularly bad on SWBD and GENIA. On average, the UNIFORM baseline does slightly better for out-of-domain and over 3% better for in-domain. UNIFORM actually does fairly well for out-of-domain except on GENIA. In general, using more source domains is better which partially explains the success of UNIFORM. This seems to be the case since even if a source domain is terribly mismatched with the target domain, it may still be able to fill in some holes left by the other source domains. Of course, if it overpowers more relevant domains, performance may suffer. The SELF-TRAINED UNIFORM baseline uses even more source domains which are also the largest ones. In both

Oracle	Baseline or model	Average f-score
•	Worst seen	62.0 ± 6.1
•	Single corpus	81.0 ± 2.9
	Fixed set: WSJ	81.2 ± 3.1
	Uniform	81.4 ± 3.6
	Self-trained Uniform	83.4 ± 2.5
	Any Domain Parsing	84.0 ± 2.5
•	Best seen	84.3 ± 2.6

(a) Out-of-domain evaluation

Oracle	Baseline or model	Average f-score
	Fixed set: WSJ	82.3 ± 4.4
	Uniform	85.4 ± 2.4
•	Single corpus	85.6 ± 2.9
	Self-trained Uniform	86.1 ± 2.0
•	Best overall model	86.2 ± 1.9
	Any Domain Parsing	86.9 ± 2.4
•	Best seen	87.5 ± 2.1

(b) In-domain evaluation

Table 5.5: Baselines and final results for each multiple-source domain adaptation evaluation scenarios. Results are f-scores, averaged over all six target domains with their standard deviation. Our model, Any Domain Parsing, is the best non-oracle based system.

evaluations, this dramatically improves performance and is the second-best non-oracle system. This baseline provides more evidence as to the power of self-training for improving parser adaptation. If we excluded all self-trained corpora, our performance on this task would be substantially worse. The BEST SINGLE CORPUS is poor in the out-of-domain scenario primarily because the actual best single corpus is excluded by the task specification in most cases. When we move to in-domain, this baseline improves but is still worse than SELF-TRAINED UNIFORM on average. It beats SELF-TRAINED UNIFORM primarily on WSJ, SWBD, and GENIA indicating that these three domains are best when not diluted by others. Perhaps surprisingly, BEST SINGLE CORPUS and FIXED SET: WSJ perform similarly in the out-of-domain setting. This is because WSJ is the best single corpus to use in most cases. FIXED SET: WSJ ends up doing slightly better because it is forced to fall back to uniform when evaluating on WSJ. By definition, the WORST SEEN baseline does terribly, almost 20% worse then BEST SINGLE CORPUS. We omit this baseline from Figures 5.4 and 5.5 to avoid skewing the scale.

5.5.6 Analysis

For the in-domain evaluation, our biggest loss comes from our evaluation on GENIA. At the same time, our results on WSJ are near the best seen models for both scenarios — for the out-of-domain evaluation, it actually performs better than the BEST SEEN baseline (although not by a significant margin). In Table 5.6, we show the weights on the linear regression model for the GENIA and WSJ evaluation rounds to give examples of bad and good regression models. We note that the magnitudes of weights are not directly comparable across groups of features (i.e. weights for COSINETOP50 are on a different scale than UNKWORDSREV). However, it is meaningful to compare weights of features within groups and across evaluation scenarios.

This discrepancy in performance warrants some exploration. In the out-of-domain scenario, GENIA and MEDLINE (by GENIA) are excluded, but the regressor learns that it should favor MEDLINE (by WSJ) and gives it the highest weights among COSINETOP50 and UNKWORDSREV. Note that self-trained corpora tend to have higher weights. Presumably this is because the self-trained corpora are larger and thus applicable to a large number of target domains. In the in-domain evaluation, both versions of MEDLINE are available. The regressor assigns a higher weight to the MEDLINE which uses WSJ as its base parser — this is most likely

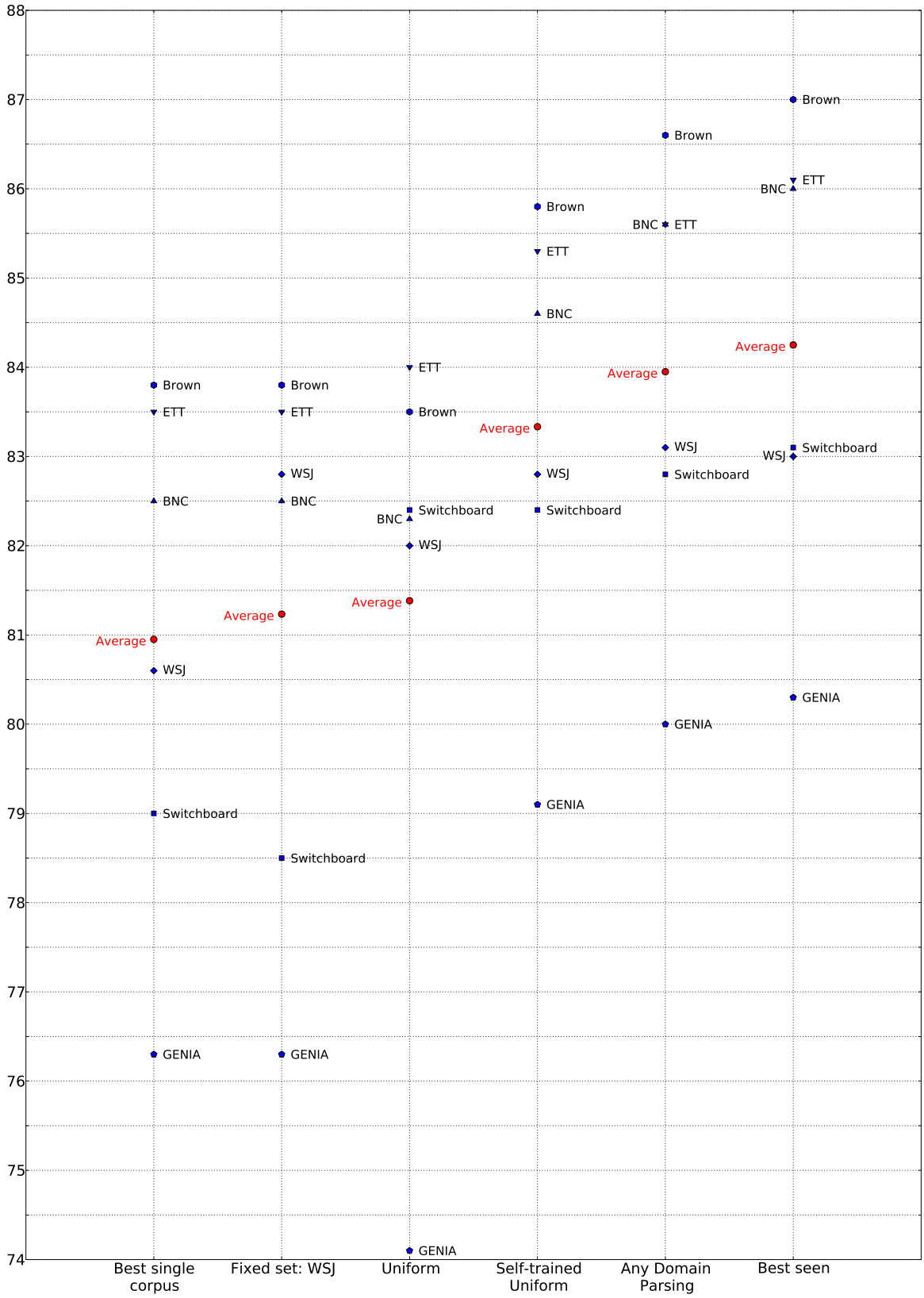


Figure 5.4: Out-of-domain evaluation

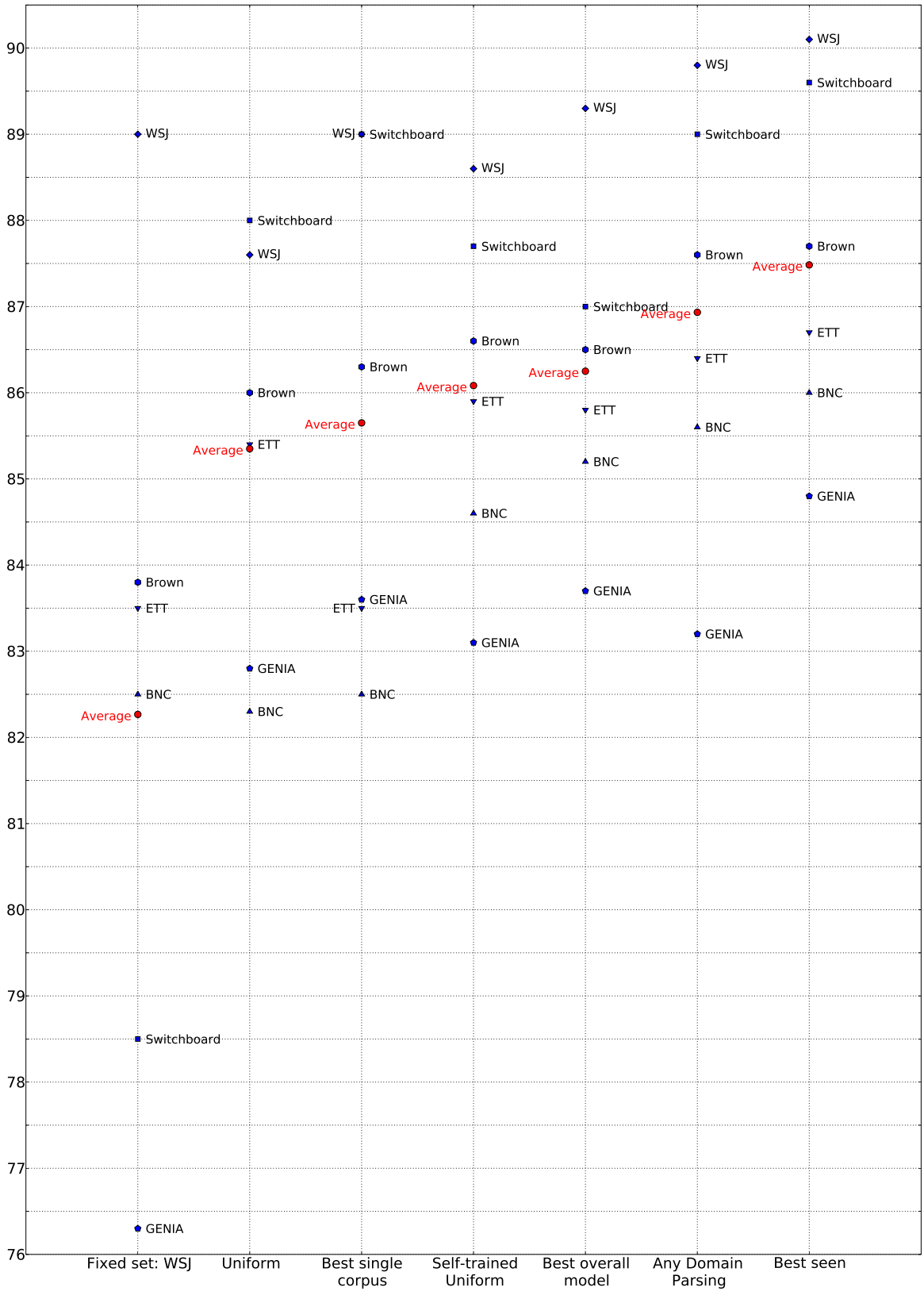


Figure 5.5: In-domain evaluation

the main factor in why it produces a suboptimal parsing model in this evaluation. The question remains, why would it assign a higher weight to MEDLINE (by WSJ)? The answer seems to be because self-trained corpora incorporate many of the statistics of their base parser. Thus, MEDLINE (by WSJ) acts in many ways like a larger version of WSJ. Since we aren't allowed to evaluate on GENIA when testing on it, the regressor only knows how well the models do on other target domains. By and large, the other target domains are more like WSJ than GENIA (this is certainly the case from an unknown word point of view — recall Figure 5.2b). Thus, the regressor believes it should trust MEDLINE (by WSJ) more.

We had hoped that our system would be able to handle the case where the same raw corpus has been parsed by multiple base parsers. Given the above, it is clear that this is a limitation of our model. One problem is that from a domain divergence standpoint, the two MEDLINE corpora are nearly identical since these features only look at surface words. Adding tree-based domain divergence measures (e.g. syntactic language models) may improve the situation, though there is no guarantee. The easiest solution is to only include each raw corpus once and to use the best model as the base parser. In the next section, we discuss how to select the best base parser for raw corpus and we are confident that it would select GENIA over WSJ to parse MEDLINE. If the experiment was repeated without MEDLINE (by WSJ), we suspect that the resulting model for GENIA would improve significantly.

One may be concerned that the *sign* of the weights for the COSINETOP50 and UNKWORDSREV features is the same even though they are on reversed scales.[9] Indeed, this was an unexpected finding. To investigate, we created two regression models to isolate the features. The first used only UNKWORDSREV and ENTROPY as features while the second used only COSINETOP50 and ENTROPY as features. When UNKWORDSREV and COSINETOP50 operate on their own, their feature weights do obtain different signs (UNKWORDSREV is positive while COSINETOP50 is negative). Thus, the reason that they both have positive weights in the original model is the result of feature interactions. The model without COSINETOP50 performs nearly as well as the one with it, so it seems that UNKWORDSREV is doing most of the work while COSINETOP50 is acting as a small correction factor.

5.6 Discussion

We have shown that for both evaluation conditions, our system is excellent at predicting the effects of domain divergence on parsing accuracy. Now we return to the questions that we raised at the start of this chapter. Any Domain Parsing suggests a combination of source domains to use when we wish to parse a new text. However, while we know that self-trained corpora can dramatically improve performance, we also hoped to create a tool for determining the best corpus to self-train on. We posit that our model can be used here as well: Simply choose the raw text with the highest predicted f-score. Treat each raw text as a candidate target text and determine the best mixture of source domains to parse it along with its predicted f-score. Since our system translates domain divergences into lowered f-score performance, it should assign a higher f-score to a more similar domain. This approach also answers our other main questions regarding self-training — namely,

[9] COSINETOP50 has a raw divergence score of 1.0 when the two domains are identical according to the measure and 0.0 when they are completely orthogonal. UNKWORDSREV has the opposite behavior and ranges from 0.0 (high similarity) to 1.0 (low similarity).

GENIA		WSJ		
Out	In	Out	In	Feature
1.548	1.715	2.464	1.946	CosineTop50: BROWN
1.192	1.226	1.878	1.521	CosineTop50: ETT
—	1.341	2.047	1.684	CosineTop50: GENIA
—	1.833	2.987	2.556	CosineTop50: MEDLINE (by GENIA)
2.659	2.851	—	2.690	CosineTop50: MEDLINE (by WSJ)
0.747	0.814	1.094	0.956	CosineTop50: SWBD
2.254	2.482	—	2.767	CosineTop50: WSJ
2.492	2.724	—	2.874	CosineTop50: NANC
1.982	2.151	—	2.317	CosineTop50: GUTENBERG
1.568	1.729	2.514	1.913	UnkWordsRev: BROWN
1.015	1.119	1.791	1.422	UnkWordsRev: ETT
—	0.930	1.783	1.558	UnkWordsRev: GENIA
—	1.699	3.361	2.512	UnkWordsRev: MEDLINE (by GENIA)
2.635	2.530	—	2.640	UnkWordsRev: MEDLINE (by WSJ)
0.716	0.808	1.101	0.941	UnkWordsRev: SWBD
2.291	2.437	—	2.707	UnkWordsRev: WSJ
2.472	2.683	—	2.868	UnkWordsRev: NANC
1.931	2.183	—	2.354	UnkWordsRev: GUTENBERG
2.199	2.643	4.601	2.463	Entropy
77.537	77.527	76.826	75.123	Intercept

Table 5.6: Regression weights learned for the GENIA and WSJ evaluations round for out-of-domain and in-domain scenarios. '—'s indicate that this domain was excluded since it was the target domain in an out-of-domain evaluation.

which parsing model should we use to parse our raw text and how should self-trained data be combined with hand-annotated corpora?

There are a number of practical concerns should this system actually be employed to perform large-scale parsing of heterogeneous domains. Making a new model for each article is likely to be prohibitively expensive. However, as we have seen, making a small number of randomly sampled models is likely to be sufficient. This step would be necessarily to generate training data to train an initial regression model. Subsequent models could be mixed on demand, using existing models when they're sufficiently close (closeness can be measured by KL divergence or the difference in predicted f-scores on the text in question among other methods). To determine which raw texts to use, one might cluster the articles by topic or syntactically. Alternatively, using the domain divergence measures, it may be possible to frame the problem as a multi-cut graph problem where edges represent divergences and the goal is to minimize divergences within each cluster.

One interesting question is how much variation our model sees in corpora with multiple domains such as BROWN or BNC. As in Ravi et al. (2008), we would need a mechanism of dividing up corpora into smaller units which could be articles, groups of articles, or simply contiguous blocks of sentences. One segmented, we can could ask our model to calculate the best source domain mixture for each segment. If these source domain mixtures vary significantly, it would be interesting if improved corpora can be obtained for these corpora. As stated earlier, Plank and Sima'an (2008) attempted to automatically uncover these subdomains for WSJ without much success but our approach is sufficiently different that it is worth looking into.

Chapter 6

Conclusion

In this dissertation, we have described a semi-supervised method for statistical natural language parsing called self-training. Self-training for parsing works by treating the parse trees of raw sentences from a supervised parser as correct. One would not expect this to work well in general and, indeed, this is not the case (Charniak, 1997; Steedman et al., 2003a). We have shown that when the generative parser is used in conjunction with a discriminative reranker, self-training produces a parser with state-of-the-art accuracy. Additionally, self-training has proved to be extremely valuable for improving the parser portability and parser adaptation tasks. Unlabeled data can be effectively leveraged to help cover domains which lack sufficient labeled training data.

We have addressed the issue of how to best parse completely new target texts given multiple source domains with our Any Domain Parsing model. To our knowledge, the problem of multiple source parser adaptation has not been tackled before. Without an automatic mechanism for performing this task, a human would have to select the best model for the text to be parsed. While this is feasible in some cases, there are many cases (e.g. parsing the web where there are a large number of domains which may change frequently) where it is less obvious what the best mixture of source domains might be. Our model learns which properties of domain divergence influence parsing accuracy. It uses this information to suggest combinations of source domains which perform extremely well in practice. Our high accuracies on both evaluation scenarios is in part due to the use of self-trained corpora and our crossdomain performance would certainly suffer without them.

Looking beyond this work, there are some broader issues which suggest future avenues of investigation. Syntactic parsing is but one of many tasks in natural language processing and machine learning. It is possible that other tasks which lend themselves to a two-stage generative/discriminative framework may benefit from self-training as well. Additionally, despite our work on analyzing self-training, we do not yet know the effect of self-training on other parsers with different formalisms. It would be interesting to investigate how well self-training and Any Domain Parsing work for CCG, LTAG, and so on. Finally, the machine learning community has proposed many new semi-supervised learning techniques, generally involving discriminative models. The challenge in this case is finding ways of integrating these techniques into parsing frameworks.

However, the parsing accuracy for English is quite good, at least for the level of annotation used in this dissertation. The main challenge for parsing lies in other languages. Other languages may have complex

morphological systems, (e.g. Turkish, Czech, and Hungarian) difficult segmentation problems, (e.g. Chinese and Japanese), or a large amount of dialectal variation (e.g. Arabic). In theory, the techniques described in this thesis should be applicable to other languages and we leave this to future studies.

Bibliography

Antti Airola, Sampo Pyysalo, Jari Bjorne, Tapio Pahikkala, Filip Ginter, and Tapio Salakoski. 2008. All-paths graph kernel for protein-protein interaction extraction with evaluation of cross-corpus learning. *BMC Bioinformatics*, 9(Suppl 11):S2.

Rie Kubota Ando and Tong Zhang. 2005a. A framework for learning predictive structures from multiple tasks and unlabeled data. *Journal of Machine Learning Research*, 6:1817–1853.

Rie Kubota Ando and Tong Zhang. 2005b. A high-performance semi-supervised learning method for text chunking. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics (ACL 2005)*, pages 1–9, Morristown, NJ, USA. Association for Computational Linguistics.

Michiel Bacchiani, Michael Riley, Brian Roark, and Richard Sproat. 2006. MAP adaptation of stochastic grammars. *Computer Speech and Language*, 20(1):41–68.

Regina Barzilay and Mirella Lapata. 2008. Modeling local coherence: an entity-based approach. *Computational Linguistics*, 34(1):1–34.

Kristin P. Bennett and Ayhan Demiriz. 1998. Semi-supervised support vector machines. In Michael J. Kearns, Sara A. Solla, and David A. Cohn, editors, *NIPS*, pages 368–374. The MIT Press.

Ann Bies, Mark Ferguson, Karen Katz, and Robert MacIntyre, 1995. *Bracketting Guideliness for Treebank II style Penn Treebank Project*. LDC.

Ann Bies, Justin Mott, and Colin Warner, 2005. *Addendum to the Switchboard Treebank Guidelines*. LDC.

Ann Bies. 2007. *GALE Phase 3 Release 1 - English Translation Treebank*. Linguistic Data Consortium. LDC2007E105.

Ezra Black, Steven P. Abney, D. Flickenger, Claudia Gdaniec, Ralph Grishman, P. Harrison, Donald Hindle, Robert Ingria, Frederick Jelinek, Judith L. Klavans, Mark Liberman, Mitchell P. Marcus, Salim Roukos, Beatrice Santorini, and Tomek Strzalkowski. 1991. Procedure for quantitatively comparing the syntactic coverage of English grammars. In *Proceedings of Workshop on Speech and Natural Language*, pages 306–311. Morgan Kaufmann.

Ezra Black, Fred Jelinek, John Lafrerty, David M. Magerman, Robert Mercer, and Salim Roukos. 1993. Towards history-based grammars: Using richer models for probabilistic parsing. In *Proceedings of the 31st Annual Meeting of the Association for Computational Linguistics*, pages 31–37, Columbus, Ohio, USA, June. Association for Computational Linguistics.

John Blitzer, Ryan McDonald, and Fernando Pereira. 2006. Domain adaptation with structural correspondence learning. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 120–128, Sydney, Australia, July. Association for Computational Linguistics.

John Blitzer, Mark Dredze, and Fernando Pereira. 2007. Biographies, bollywood, boom-boxes and blenders: Domain adaptation for sentiment classification. In *Association for Computational Linguistics*, Prague, Czech Republic.

Avrim Blum and Tom Mitchell. 1998. Combining labeled and unlabeled data with co-training. In *Proceedings of the 11th Annual Conference on Computational Learning Theory (COLT-98)*.

Rens Bod. 2003. An efficient implementation of a new DOP model. In *10th Conference of the European Chapter of the Association for Computational Linguistics*, Budapest, Hungary.

Adriane Boyd, Markus Dickinson, and Detmar Meurers. 2008. On detecting errors in dependency treebanks. *Research on Language and Computation*, 6(2):113–137.

Thorsten Brants and Alex Franz. 2006. *Web 1T 5-gram Version 1*. Linguistic Data Consortium. LDC2006T13.

R.H. Byrd, P. Lu, J. Nocedal, and C. Zhu. 1995. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, 16(5):1190–1208.

Xavier Carreras, Michael Collins, and Terry Koo. 2008. TAG, dynamic programming, and the perceptron for efficient, feature-rich parsing. In *CoNLL 2008: Proceedings of the Twelfth Conference on Computational Natural Language Learning*, pages 9–16, Manchester, England, August. Coling 2008 Organizing Committee.

Eugene Charniak and Micha Elsner. 2009. Em works for pronoun anaphora resolution. In *Proceedings of the Conference of the European Chapter of the Association for Computational Linguistics (EACL-09)*, Athens, Greece.

Eugene Charniak and Mark Johnson. 2005. Coarse-to-fine *n*-best parsing and MaxEnt discriminative reranking. In *Proceedings of the 2005 Meeting of the Assoc. for Computational Linguistics (ACL)*, pages 173–180.

Eugene Charniak, Kevin Knight, and Kenji Yamada. 2003. Syntax-based language models for statistical machine translation. In *MT Summit*. International Association for Machine Translation.

Eugene Charniak. 1997. Statistical parsing with a context-free grammar and word statistics. In *Proceedings of AAAI*, pages 598–603.

Eugene Charniak. 2000. A maximum-entropy-inspired parser. In *Proceedings of the North American Chapter of the ACL (NAACL)*, pages 132–139.

Eugene Charniak. 2001. Immediate-head parsing for language models. In *Proceedings of the Assoc. for Computational Linguistics (ACL)*, pages 116–123.

Ciprian Chelba and Frederick Jelinek. 1998. Exploiting syntactic structure for language modeling. In Christian Boitet and Pete Whitelock, editors, *Proceedings of the Thirty-Sixth Annual Meeting of the Association for Computational Linguistics and Seventeenth International Conference on Computational Linguistics*, pages 225–231, San Francisco, California. Morgan Kaufmann Publishers.

Stanley F. Chen and Joshua Goodman. 1996. An empirical study of smoothing techniques for language modeling. In *Proceedings of the Assoc. for Comp. Linguistics (ACL)*, pages 310–318.

Stephen Clark, James Curran, and Miles Osborne. 2003. Bootstrapping POS-taggers using unlabelled data. In *Proceedings of CoNLL-2003*.

Andrew B. Clegg and Adrian Shepherd. 2005. Evaluating and integrating treebank parsers on a biomedical corpus. In *Proceedings of the ACL Workshop on Software*.

A.B. Clegg and A.J. Shepherd. 2007. Benchmarking natural-language parsers for biological applications using dependency graphs. *BMC Bioinformatics*, 8(1):24.

Paul R. Cohen. 1995. *Empirical Methods for Artificial Intelligence*. The MIT Press, Cambridge, Massachusetts.

Michael Collins and Terry Koo. 2005. Discriminative Reranking for Natural Language Parsing. *Computational Linguistics*, 31(1):25–69.

Michael Collins. 1997. Three generative, lexicalised models for statistical parsing. In *Proceedings of the Assoc. for Computational Linguistics*, pages 16–23.

Michael Collins. 2000. Discriminative reranking for natural language parsing. In *Machine Learning: Proceedings of the Seventeenth International Conference (ICML 2000)*, pages 175–182, Stanford, California.

Sanjoy Dasgupta, M.L. Littman, and D. McAllester. 2001. PAC generalization bounds for co-training. In *Advances in Neural Information Processing Systems (NIPS), 2001*.

Hal Daumé III. 2007. Frustratingly easy domain adaptation. In *Conference of the Association for Computational Linguistics (ACL)*, Prague, Czech Republic.

Tejaswini Deoskar. 2008. Re-estimation of lexical parameters for treebank PCFGs. In *Proceedings of the 22nd International Conference on Computational Linguistics (Coling 2008)*, pages 193–200, Manchester, UK, August. Coling 2008 Organizing Committee.

Markus Dickinson and Charles Jochim. 2008. A simple method for tagset comparison. In *Proceedings of the 6th Language Resources and Evaluation Conference (LREC 2008)*, Marrakech, Morocco.

Markus Dickinson. 2009. Correcting dependency annotation errors. In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics (EACL-09)*, Athens, Greece.

Mark Dredze and Koby Crammer. 2008. Online methods for multi-domain learning and adaptation. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 689–697, Honolulu, Hawaii, October. Association for Computational Linguistics.

Micha Elsner and Eugene Charniak. 2008. Coreference-inspired coherence modeling. In *Proceedings of ACL-08: HLT, Short Papers*, pages 41–44, Columbus, Ohio, June. Association for Computational Linguistics.

Micha Elsner, Joseph Austerweil, and Eugene Charniak. 2007. A unified local and global model for discourse coherence. In *Proceedings of HLT-NAACL '07*, Rochester, New York, April. Association for Computational Linguistics.

Jenny Rose Finkel and Christopher D. Manning. 2009. Hierarchical bayesian domain adaptation. In *Proceedings of HLT-NAACL 2009*, pages 602–610, Boulder, Colorado, June.

Jenny Rose Finkel, Alex Kleeman, and Christopher D. Manning. 2008. Efficient, feature-based, conditional random field parsing. In *Proceedings of ACL-08: HLT*, pages 959–967, Columbus, Ohio, June. Association for Computational Linguistics.

Jennifer Foster and Markus Dickinson. 2009. Similarity rules! exploring methods for ad-hoc rule detection. In *Proceedings of the Fifth Workshop on Treebanks and Linguistic Theories (TLT-2009)*, Groningen, The Netherlands.

Jennifer Foster and Josef van Genabith. 2008. Parser evaluation and the bnc: Evaluating 4 constituency parsers with 3 metrics. In European Language Resources Association (ELRA), editor, *Proceedings of the Sixth International Language Resources and Evaluation (LREC'08)*, Marrakech, Morocco, May.

Jennifer Foster, Joachim Wagner, Djamé Seddah, and Josef van Genabith. 2007. Adapting WSJ-trained parsers to the British National Corpus using in-domain self-training. In *Proceedings of the Tenth International Conference on Parsing Technologies*, pages 33–35, Prague, Czech Republic, June. Association for Computational Linguistics.

W. Nelson Francis and Henry Kučera. 1979. *Manual of Information to accompany a Standard Corpus of Present-day Edited American English*, for use with Digital Computers. Brown University, Providence, Rhode Island.

Alexander Fraser and Daniel Marcu. 2006. Semi-supervised training for statistical word alignment. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 769–776, Sydney, Australia, July. Association for Computational Linguistics.

Daniel Gildea. 2001. Corpus variation and parser performance. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 167–202.

Joshua T. Goodman. 2001. A bit of progress in language modeling extended version. Technical Report 2001-72, Microsoft Research.

David Graff. 1995. *North American News Text Corpus*. Linguistic Data Consortium. LDC95T21.

Stephan Greene and Philip Resnik. 2009. More than words: Syntactic packaging and implicit sentiment. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 503–511, Boulder, Colorado, June. Association for Computational Linguistics.

James Henderson. 2004. Discriminative training of a neural network statistical parser. In *Proc. 42nd Meeting of Association for Computational Linguistics (ACL 2004), Barcelona, Spain*.

Donald Hindle and Mats Rooth. 1993. Structural ambiguity and lexical relations. *Computational Linguistics*, 19(1):103–120.

Liang Huang and David Chiang. 2005. Better k-best parsing. In *Proceedings of the Ninth International Workshop on Parsing Technology*, pages 53–64, Vancouver, British Columbia, October. Association for Computational Linguistics.

Liang Huang. 2008. Forest reranking: Discriminative parsing with non-local features. In *Proceedings of ACL-08: HLT*, pages 586–594, Columbus, Ohio, June. Association for Computational Linguistics.

Frederick Jelinek, John D. Lafferty, David M. Magerman, Robert L. Mercer, Adwait Ratnaparkhi, and Salim Roukos. 1994. Decision tree parsing using a hidden derivation model. In *HLT*. Morgan Kaufmann.

Victor M. Jiménez and Andres Marzal. 2000. Computation of the n best parse trees for weighted and stochastic context-free grammars. In *Joint Intl. Workshops on Advances in Pattern Recognition (IAPR)*.

Mark Johnson, Stuart Geman, Stephen Canon, Zhiyi Chi, and Stefan Riezler. 1999. Estimators for stochastic "unification-based" grammars. In *The Proceedings of the 37th Annual Conference of the Association for Computational Linguistics*, pages 535–541, San Francisco. Morgan Kaufmann.

Mark Johnson, Eugene Charniak, and Matthew Lease. 2004. An improved model for recognizing disfluencies in conversational speech. In *Proc. of the Rich Text 2004 Fall Workshop (RT-04F)*.

Daisuke Kawahara and Kiyotaka Uchimoto. 2008. Learning reliability of parses for domain adaptation of dependency parsing. In *Third International Joint Conference on Natural Language Processing (IJCNLP '08)*.

H. Kilicoglu and S. Bergler. 2008. Recognizing speculative language in biomedical research articles: a linguistically motivated perspective. *BMC Bioinformatics*, 9(11):S10.

Terry Koo, Xavier Carreras, and Michael Collins. 2008. Simple semi-supervised dependency parsing. In *Proceedings of ACL-08: HLT*, pages 595–603, Columbus, Ohio, June. Association for Computational Linguistics.

Matthew Lease and Eugene Charniak. 2005. Parsing biomedical literature. In *Second International Joint Conference on Natural Language Processing (IJCNLP'05)*.

Matthew Lease, Eugene Charniak, Mark Johnson, and David McClosky. 2006. A look at parsing and its applications. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI-06)*, 16–20 July.

Yudong Liu, Zhongmin Shi, and Anoop Sarkar. 2007. Exploiting rich syntactic information for relationship extraction from biomedical articles. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Companion Volume, Short Papers*, pages 97–100, Rochester, New York, April. Association for Computational Linguistics.

Xiaoqiang Luo and Imed Zitouni. 2005. Multi-lingual coreference resolution with syntactic features. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 660–667, Vancouver, British Columbia, Canada, October. Association for Computational Linguistics.

Kim Luyckx and Walter Daelemans. 2008. Authorship attribution and verification with many authors and limited data. In *Proceedings of the 22nd International Conference on Computational Linguistics (Coling 2008)*, pages 513–520, Manchester, UK, August. Coling 2008 Organizing Committee.

David M. Magerman. 1995. Statistical decision-tree models for parsing. In *The Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*, pages 276–283, San Francisco. The Association for Computational Linguistics, Morgan Kaufman.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Comp. Linguistics*, 19(2):313–330.

David McClosky, Eugene Charniak, and Mark Johnson. 2006a. Effective self-training for parsing. In *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*, pages 152–159.

David McClosky, Eugene Charniak, and Mark Johnson. 2006b. Reranking and self-training for parser adaptation. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics (ACL'06)*, pages 337–344, Sydney, Australia, July. Association for Computational Linguistics.

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajič. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 523–530, Vancouver, British Columbia, Canada, October. Association for Computational Linguistics.

B. Medlock. 2008. Exploring hedge identification in biomedical literature. *Journal of Biomedical Informatics*, 41(4):636–654.

Igor A. Mel'čuk. 1988. *Dependency Syntax: Theory and Practice*. State University of New York Press, Albany.

Rada Mihalcea. 2004. Co-training and self-training for word sense disambiguation. In Hwee Tou Ng and Ellen Riloff, editors, *HLT-NAACL 2004 Workshop: Eighth Conference on Computational Natural Language Learning (CoNLL-2004)*, pages 33–40, Boston, Massachusetts, USA, May 6 - May 7. Association for Computational Linguistics.

Yusuke Miyao, Rune Sætre, Kenji Sagae, Takuya Matsuzaki, and Jun'ichi Tsujii. 2008. Task-oriented evaluation of syntactic parsers and their representations. In *Proceedings of ACL-08: HLT*, pages 46–54, Columbus, Ohio, June. Association for Computational Linguistics.

Vincent Ng and Claire Cardie. 2003. Weakly supervised natural language learning without redundant views. In *HLT-NAACL*.

Kamal Nigam, Andrew Kachites Mccallum, Sebastian Thrun, and Tom Mitchell. 2000. Text Classification from Labeled and Unlabeled Documents using EM. *Machine Learning*, 39(2):103–134.

Zheng-Yu Niu, Haifeng Wang, and Hua Wu. 2009. Exploiting heterogeneous treebanks for parsing. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 46–54, Suntec, Singapore, August. Association for Computational Linguistics.

Joakim Nivre and Jens Nilsson. 2005. Pseudo-projective dependency parsing. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 99–106, Ann Arbor, Michigan, June. Association for Computational Linguistics.

Joakim Nivre, Johan Hall, Sandra Kübler, Ryan McDonald, Jens Nilsson, Sebastian Riedel, and Deniz Yuret. 2007. The CoNLL 2007 shared task on dependency parsing. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, pages 915–932, Prague, Czech Republic, June. Association for Computational Linguistics.

Slav Petrov and Dan Klein. 2008. Discriminative log-linear grammars with latent variables. In J.C. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 1153–1160. MIT Press, Cambridge, MA.

Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. 2006. Learning accurate, compact, and interpretable tree annotation. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 433–440, Sydney, Australia, July. Association for Computational Linguistics.

Barbara Plank and Khalil Sima'an. 2008. Subdomain sensitive statistical parsing using raw corpora. In *Proceedings of the Sixth International Language Resources and Evaluation (LREC'08)*, Marrakech, Morocco, May.

Sameer Pradhan, Wayne Ward, and James Martin. 2007. Towards robust semantic role labeling. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference*, pages 556–563, Rochester, New York, April. Association for Computational Linguistics.

Adwait Ratnaparkhi. 1999. Learning to parse natural language with maximum entropy models. *Machine Learning*, 34(1-3):151–175.

Sujith Ravi, Kevin Knight, and Radu Soricut. 2008. Automatic prediction of parser accuracy. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 887–896, Honolulu, Hawaii, October. Association for Computational Linguistics.

Roi Reichart and Ari Rappoport. 2007. Self-training for enhancement and domain adaptation of statistical parsers trained on small datasets. *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 616–623.

Brian Roark and Michiel Bacchiani. 2003. Supervised and unsupervised PCFG adaptation to novel domains. In Marti Hearst and Mari Ostendorf, editors, *HLT-NAACL 2003: Main Proceedings*, pages 205–212, Edmonton, Alberta, Canada, May 27 – June 1. Association for Computational Linguistics.

Brian Roark, Mary Harper, Eugene Charniak, Bonnie Dorr, Mark Johnson, Jeremy Kahn, Yang Liu, Mari Ostendorf, John Hale, Anna Krasnyanskaya, Matthew Lease, Izhak Shafran, Matthew Snover, Robin Stewart, and Lisa Yung. 2006. Sparseval: Evaluation metrics for parsing speech. In *Proceedings of LREC*.

Brian Roark. 2001. Probabilistic top-down parsing and language modeling. *Computational Linguistics*, 27(2):249–276.

Kenji Sagae and Alon Lavie. 2006. Parser combination by reparsing. In *Proceedings of the Human Language Technology Conference of the NAACL, Companion Volume: Short Papers*, pages 129–132, New York City, USA, June. Association for Computational Linguistics.

Anoop Sarkar. 2001. Applying cotraining methods to statistical parsing. In *Proceedings of the 2001 NAACL Conference*.

Yves Schabes. 1992. Stochastic lexicalized tree-adjoining grammars. In *The Proceedings of the fifteenth International Conference on Computational Linguistics, COLING-92*, volume 2, pages 426–432, Nantes, France.

Hinrich Schütze. 1995. Distributional part-of-speech tagging. In *Proceedings of the 7th conference of the EACL*, pages 141–148.

Satoshi Sekine. 1997. The domain dependence of parsing. In *Proc. Applied Natural Language Processing (ANLP)*, pages 96–102.

Zhongmin Shi, Anoop Sarkar, and Fred Popowich. 2007. Simultaneous identification of biomedical named-entity and functional relation using statistical parsing techniques. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Companion Volume, Short Papers*, pages 161–164, Rochester, New York, April. Association for Computational Linguistics.

David A. Smith and Jason Eisner. 2009. Parser adaptation and projection with quasi-synchronous grammar features. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 822–831, Singapore, August. Association for Computational Linguistics.

Mark Steedman, Steven Baker, Jeremiah Crim, Stephen Clark, Julia Hockenmaier, Rebecca Hwa, Miles Osborne, Paul Ruhlen, and Anoop Sarkar. 2003a. CLSP WS-02 Final Report: Semi-Supervised Training for Statistical Parsing. Technical report, Johns Hopkins University.

Mark Steedman, Miles Osborne, Anoop Sarkar, Stephen Clark, Rebecca Hwa, Julia Hockenmaier, Paul Ruhlen, Steven Baker, and Jeremiah Crim. 2003b. Bootstrapping statistical parsers from small datasets. In *Proc. of European ACL (EACL)*, pages 331–338.

Mark J. Steedman. 2000. *The syntactic process*. MIT Press, Cambridge, Massachusetts.

Yuka Tateisi, Akane Yakushiji, Tomoko Ohta, and Jun'ichi Tsujii. 2005. Syntax Annotation for the GENIA corpus. *Proceedings of IJCNLP 2005, Companion volume*, pages 222–227.

Jenine Turner and Eugene Charniak. 2005. Supervised and unsupervised learning for sentence compression. In *Proc. Assoc. for Computational Linguistics (ACL)*, pages 290–297.

Hans van Halteren. 2004. Linguistic profiling for authorship recognition and verification. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume*, pages 199–206, Barcelona, Spain, July.

C. J. van Rijsbergen. 1979. *Information Retrieval, 2nd edition*. Department of Computer Science, University of Glasgow.

S. Vijayakumar, A. D'souza, and S. Schaal. 2005. Incremental online learning in high dimensions. *Neural Computation*, 17(12):2602–2634.

Qin Iris Wang, Dale Schuurmans, and Dekang Lin. 2008. Semi-supervised convex training for dependency parsing. In *Proceedings of ACL-08: HLT*, pages 532–540, Columbus, Ohio, June. Association for Computational Linguistics.

Qin Iris Wang, Kevin Duh, and Dekang Lin, editors. 2009. *Proceedings of the NAACL HLT 2009 Workshop on Semi-supervised Learning for Natural Language Processing*. Association for Computational Linguistics, Boulder, Colorado, June.

Peng Xu, Ciprian Chelba, and Frederick Jelinek. 2002. A study on richer syntactic dependencies for structured language modeling. In *Proc. Assoc. for Computational Linguistics (ACL)*, pages 191–198.

Ciyou Zhu, Richard H. Byrd, Peihuang Lu, and Jorge Nocedal. 1997. Algorithm 778: L-BFGS-B: Fortran subroutines for Large-Scale bound constrained optimization. *ACM Transactions on Mathematical Software*, 23(4):550–560, December.

Xiaojin Zhu, Zoubin Ghahramani, and John D. Lafferty. 2003. Semi-supervised learning using gaussian fields and harmonic functions. In Tom Fawcett and Nina Mishra, editors, *Machine Learning, Proceedings of the Twentieth International Conference (ICML)*, pages 912–919, Washington, DC, USA. AAAI Press.

Xiaojin Zhu. 2007. Semi-supervised learning literature survey. Technical Report 1530, Computer Science, University of Wisconsin-Madison.