

TOWARDS EFFECTIVE DOMAIN ADAPTATION OF DEPENDENCY
PARSING

Atreyee Mukherjee

Submitted to the faculty of the University Graduate School
in partial fulfillment of the requirements
for the degree
Doctor of Philosophy
in the Department of School of Informatics, Computing and Engineering,
Indiana University

Accepted by the Graduate Faculty, Indiana University, in partial fulfillment of the
requirements for the degree of Doctor of Philosophy.

Doctoral Committee

Sandra Kübler, PhD

David Crandall, PhD

David Leake, PhD

Donald Williamson, PhD

Date of Defense: 07/04/2019

Copyright ©
Atreyee Mukherjee

A great dedication goes here.

ACKNOWLEDGEMENTS

Atreyee Mukherjee

TOWARDS EFFECTIVE DOMAIN ADAPTATION OF DEPENDENCY PARSING

abstract?

Sandra Kübler, PhD

David Crandall, PhD

David Leake, PhD

Donald Williamson, PhD

TABLE OF CONTENTS

Acknowledgements	v
Abstract	vi
List of Tables	xii
List of Figures	xiv
Chapter 1: Introduction	1
Chapter 2: Methodology	3
2.1 Topic Modeling	3
2.2 POS Tagging	5
2.3 Dependency Parsing	6
2.3.1 Transition based approaches	8
2.3.1.1 Projective	9
2.3.1.1.1 Pseudo projective approaches	13
2.3.1.2 Non-projective	14
2.3.2 Graph based approaches	15
2.3.2.0.1 Non-projective approaches	19
2.3.2.0.2 Higher order models	20
2.3.3 Combining transition-based and graph-based models	21

2.3.4	Unsupervised and semi-supervised approaches	21
2.3.5	Neural Network based approaches	22
2.4	Transformation Based Error Driven Learning	23
2.5	Evaluation	25
2.5.1	POS tagging	26
2.5.2	Dependency Parsing	26
Chapter 3: Domain Adaptation		28
Chapter 4: Corpora		29
4.1	Wall Street Journal section of Penn Treebank (WSJ)	29
4.2	GENIA Corpus	30
4.3	CReST Corpus	30
4.4	WSJ-GENIA mixed corpus (WSJ+GENIA)	31
4.5	Summary	31
Chapter 5: Domain Experts via Topic Modeling		32
5.1	Introduction	32
5.2	Research Questions	33
5.3	Experimental Setup	36
5.3.1	Overall Architecture	36
5.3.2	Clustering Decision	38
5.3.3	Baselines	39
5.3.4	Evaluation	40

5.4	Experimental Results	40
5.4.0.1	Using Gold POS Tags	43
5.4.0.2	Using the POS Tagger	44
5.4.1	Analysis of results	45
5.4.1.1	POS Tagging	46
5.4.1.2	Dependency Parsing	48
5.5	Summary	49
Chapter 6: Automated Sentence Assignment to the Domain Experts		51
6.1	Introduction	51
6.2	Architecture	52
6.2.1	Topic words from LDA	53
6.2.2	k -nearest neighbors	53
6.2.3	Language Model Perplexity	56
6.3	Similarity Estimation Techniques	58
6.3.1	Topic Words from LDA	59
6.3.2	k -Nearest Neighbors based similarity metric	60
6.3.3	Perplexity-Based Similarity	60
6.4	Experimental Setup	61
6.4.1	Baselines	61
6.4.2	Similarity Estimation	61
6.4.3	Evaluation	61
6.5	Experimental Results	62

6.5.1	Genre Assignment	62
6.5.2	POS Tagging	63
6.5.3	Dependency Parsing	63
6.6	Summary	65
Chapter 7: Transformation-based Error-driven Learning in Dependency Parsing		67
7.1	Introduction	67
7.2	Related Work	68
7.3	Issues in Domain Adaptation	70
7.3.1	Error Types	70
7.3.2	Error Sources	71
7.4	Transformation-Based Error-Driven Learning for Domain Adaptation . . .	73
7.4.1	TBL: The Brill Tagger	73
7.4.2	Domain Adaptation Using TBL	74
7.5	Experimental Setting	77
7.5.1	Data Sets	77
7.5.1.0.1	Target Domain	78
7.5.2	TBL Components	78
7.5.2.0.1	Initial State System	78
7.5.2.0.2	Baseline	78
7.5.3	Evaluation	79
7.6	Experimental Results	79
7.7	Discussion	82
Chapter 8: Conclusion		84

Appendix A: Placeholder	86
Appendix B: Placeholder	87

LIST OF TABLES

2.1	2 topics distribution	4
2.2	10 topics distribution	4
4.1	Target domain training and test set for TBL	31
4.2	Overview of the mixed dataset (WSJ+GENIA)	31
5.1	2 topics distribution	39
5.2	10 topics distribution	39
5.3	Distribution of sentences from the WSJ+GENIA data set given 2 and 10 topics (showing the percentage of GENIA sentences per topic).	40
5.4	Examples of words in topics for the 2-topic experiments on the WSJ+Genia corpus.	41
5.5	Comparing the topic model experts to the baselines on the WSJ+GENIA data set.	41
5.6	Results of the dependency parsing experiments using gold POS tags.	43
5.7	Results of the dependency parsing experiments using TnT POS tags.	44
5.8	Comparison of LAS for the sentences with the lowest LAS in the fulltext setting.	44
5.9	The 5 most frequent dependency label confusions of the full baseline parser.	45
5.10	Unknown word rates and accuracies for known and unknown words in the WSJ+GENIA experiment using 2 topics.	46

5.11	The 6 most frequent POS tags assigned to unknown words (2 topics).	47
5.12	The 8 most frequent confusion sets (2 topics).	48
6.1	Classification parameters for k -nearest neighbors	60
6.2	Accuracy of genre assignment for different similarity metrics.	62
6.3	Results for the POS tagging experiments.	63
6.4	Attachment scores for the dependency parsing experiments.	64
6.5	Results for POS tagging and dependency parsing when we separate incor- rectly assigned sentences from correct ones.	65
7.1	Analysis of sentences from CReST containing incorrect dependency pre- dictions.	71
7.2	Differences in the dependency label sets across treebanks.	72
7.3	Sample sentences from Wall Street Journal (WSJ) & CReST corpus	78
7.4	Target domain training and test set for TBL	78
7.5	Results of applying TBL rules on test set.	79
7.6	Accuracy in predicting CReST-specific labels.	80
7.7	Top learned rules for the setting using 2 templates.	81
7.8	Top learned rules for the setting using 4 templates.	81
7.9	The 10 most frequent label confusions across the different settings.	82

LIST OF FIGURES

1.1	Dependency parsing tree ?	2
2.1	Graphical model for LDA [Blei et al., 2003]	5
2.2	Projective and Non-projective dependency trees ?	7
2.3	Three probabilistic models in Eisner’s work	16
2.4	Neural Network architecture	22
2.5	Transformation Based Error Driven Parsing	23
5.1	Overview of the architecture of the POS tagging and parsing experts.	37
6.1	Overview of the architecture of the POS tagging and parsing experts.	52
7.1	Reducing dependency label errors via transformation-based error-driven learning	76

CHAPTER 1

INTRODUCTION

TBD!!

Syntactic parsing refers to the process of extracting and analyzing the syntactic structure or representation of a natural language text. This is very useful in a variety of different domains such as machine translation, information retrieval, question answering ?. More formally, we can define it as a structural prediction problem. Hence, there is an input which are sentences; an output, which are syntactic representation of a sentence; and a model that maps the input to the output. The input sentence is usually split into tokens. The output is represented as trees. Dependency parsing is a type of syntactic parsing analyses the dependency structure of a sentence based on a dependency grammar. As illustrated in figure 1.1, the words in a sentence are linked with relations (shown using labeled arrows), also known as dependency relation (dependencies) where one word is the dependent ¹ which depends on the head ². In order to simplify the problem to provide a syntactic head to the word which does not have a head, an artificial root is introduced ?. Dependency parsing gained prominence because of the flexibility it provides for free word order languages such as Czech.

Most commonly used dependency parsing approaches are transition-based and graph-based. The transition based approach is a greedy locally optimized classifier that learns how to get from one parse state to the next. In contrast, for graph-based dependency parsing, the problem lies in finding maximum spanning trees (MST). The MST parser is state of the art in graph based parsing. The core idea of the parser to compute a score of the dependency tree ³ which is a dot product of weight and a high dimensional feature vector. The resulting

¹can also be referred to as child

²can also be referred to as parent

³Score of the tree is the sum of the score of each edge of the tree

dependency parse is the highest scoring tree. I use MATE Bohnet [2010a]⁴ which is a reimplementation of MST parser[?] for my experiments. The parser is highly efficient in terms of memory utilization and CPU requirements while maintaining the accuracy of the resultant parses.

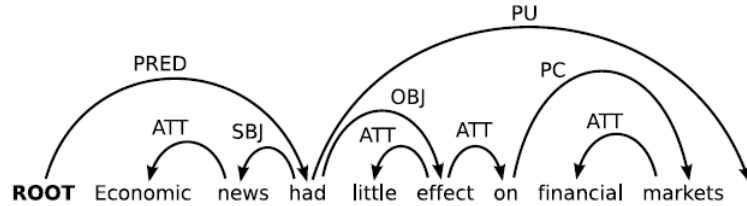


Figure 1.1: Dependency parsing tree[?]

Part of speech (POS) tagging refers to finding part of speech tags (e.g., nouns, verbs, adjectives) in a sentence. Often POS tagging is a precursor step in parsing, which is why, I look at POS tagging in my experiments, as well. These tasks generally work well when applied on the dataset from the same domain. However, the performance suffers when source and target are from different domains. The work in the area is largely driven by unavailability of data from the target domain. It is nearly impossible to hand annotate the huge amount of data generated by various sources. Manual annotation is even harder when the syntactic structure of the data differs widely. E.g., the syntactic structure of a newspaper corpus is very different from that of social media or academic journals from biomedical domain or literary works. This is an interesting problem since a deep understanding of syntactic structure of a sentence helps in variety of NLP tasks such as question answering, sentiment analysis, opinion mining, etc. Dependency parsing is particularly useful because the dependency relations serve as an important feature to detect negation or multi word expressions, for instance.

⁴code.google.com/p/mate-tools

CHAPTER 2

METHODOLOGY

POS tagging; Dependency Parsing ; TM; TBL

2.1 Topic Modeling

Probabilistic topic modeling is a class of algorithms which detects the thematic structure in a large volume of documents. Topic modeling is unsupervised, i.e., it does not require annotated documents Blei [2012] but rather discovers similarity between documents. Latent Dirichlet Allocation (LDA) is one of the topic modeling algorithms. It is a generative probabilistic model that approximates the underlying hidden topical structure of a collection of texts based on the distribution of words in the documents Blei et al. [2003]. I explain LDA in more detail below.

LDA Intuitively, Latent Dirichlet Allocation (LDA) is a method for discovering the hidden topics in a sentence. E.g., consider the following WSJ sentence. If we choose to model this in terms of LDA, we get the output shown in table 5.1 and 5.2.

“Though growers can’t always keep the worm from the apple , they can protect themselves against the price vagaries of any one variety by diversifying – into the recently imported Gala , a sweet New Zealand native ; the Esopus Spitzenburg , reportedly Thomas Jefferson’s favorite apple ; disease-resistant kinds like the Liberty.”

I.e., when we choose the number of topics as 2 & 10, LDA creates a probabilistic distribution of topics in the sentence. This process of discovering topics in a document can be represented by a generative model. Figure 2.1 shows the plate diagram for graphical model for LDA. Plate diagrams are standard for representing repeating entities in a graphical model for Bayesian inference. Each document(W), i.e., a sentence in this case,

Table 2.1: 2 topics distribution

0	1
95.90	4.10

Table 2.2: 10 topics distribution

0	1	2	3	4	5	6	7	8	9
0.11	0.10	0.08	0.10	0.09	30.58	15.28	49.55	3.93	0.16

is a mixture of topics. In other words, a sentence consists of a collection of words, i.e., $W = w_1, w_2, \dots, w_N$. A corpus can be represented as a collection of M sentences or documents, i.e., $C = W_1, W_2, \dots, W_M$. For each sentence in a corpus, the generative steps for LDA can be given as below:

- The number of words in a sentence i.e., N is chosen from a Poisson distribution.

$$N \sim \text{Poisson}(\lambda)$$

- The mixture of topic for a sentence is chosen according to the dirichlet distribution over a fixed set of topics.

$$\theta \sim \text{Dirichlet}(\alpha)$$

- Each word (w_i) in sentence (W) can be generated as follows:
 - Pick a topic according to the multinomial distribution sampled above.

$$Z_n \sim \text{Multinomial}(\theta)$$

- Generate the word from the topic according to the multinomial distribution of topics. I.e., we choose a word from $p(w_n|Z_n, \beta)$, where β represents word probabilities.

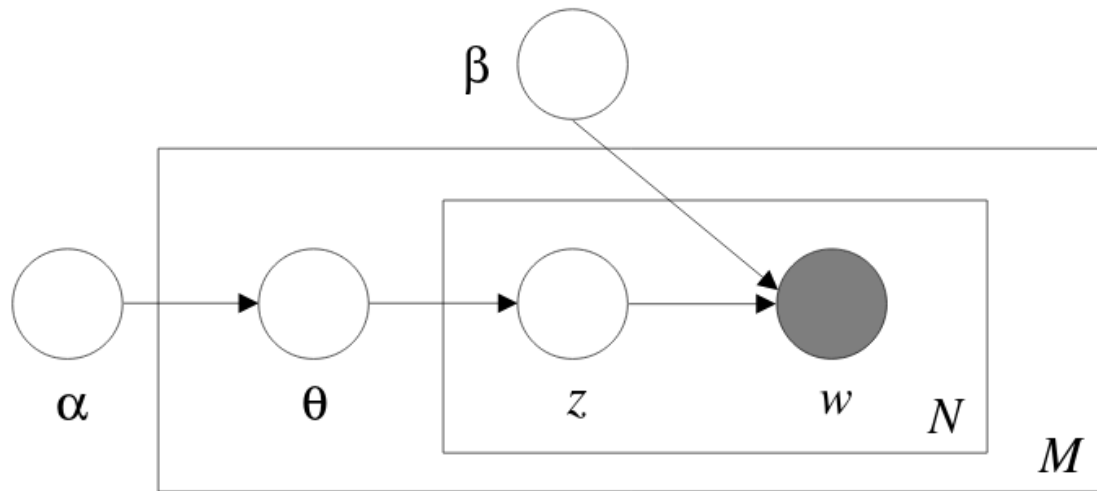


Figure 2.1: Graphical model for LDA [Blei et al., 2003]

The inference/parameter estimation step follows the generative step, which requires calculating the posterior distribution of the hidden variables given a sentence. to be finished

LDA toolkit There are several open sourced toolkits available for LDA. We use the topic modeling toolkit MALLET McCallum [2002]. The topic modeler in MALLET implements Latent Dirichlet Allocation (LDA), clustering documents into a predefined number of topics. As a result, it provides different types of information such as:

- Topic keys: The highest ranked words per topic with their probabilities;
- Document topics: The topic distribution for each document (i.e., the probability that a document belongs to a given topic); and
- Topic state: This correlates all words and topics.

2.2 POS Tagging

POS Tagging Toolkit For part of speech tagging, I use the TnT (Trigrams'n'Tags) tagger Brants [2000]. TnT is based on a second order Markov Model and has an elaborate model

for guessing the POS tags for unknown words. I use TnT mainly because of its speed and because it allows the manual inspection of the trained models (emission and transition frequencies).

2.3 Dependency Parsing

The process of automatically analyzing dependency structures to an input sentence is referred to as dependency parsing. A dependency tree needs to satisfy the following properties ?:

- A dependency tree needs to have an artificial root which specifies that there must not exist an arc which comes into the root from another node(word).
- A dependency tree must satisfy the spanning property for all the words in a sentence.
- A dependency tree must be connected i.e., if we ignore the directed edges then every word in a sentence must be connected.
- Every word in the dependency tree must have one head only.
- A dependency tree must be acyclic, i.e., should not contain cycle.
- Number of arcs in the dependency tree must be one less than the number of words, i.e.,

$$|E| = |W| - 1$$

for a dependency tree $G = (W, E)$ where E is number of arcs and W is number of words.

There are two different kinds of dependency structures based on which, dependency trees can be categorized into two types. Figure 2.2 shows an example illustrating the difference between projective (right) and non-projective (left) trees.

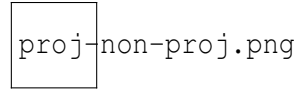


Figure 2.2: Projective and Non-projective dependency trees ?

- Projective dependency trees - For these dependency trees, the dependencies do not cross each other. Usually, the sentences in English are projective.
- Non-projective dependency trees - For these type, the dependency branches can cross each other. This is more prevalent in the languages with free word order like German.

There are two major approaches for this problems - data-driven and grammar-based. The grammar-based approaches make use of a formal grammar and the problem in this case is defined as - whether the input sentence belongs to the language defined by this formal grammar. In contrast to grammar-based approaches, data-driven approaches utilize machine learning approaches on annotated data from a corpus to parse a given input sentence. Since data-driven approaches rely on machine learning approaches, we will only discuss these methods in the rest of the report.

- Transition-based - In a transition-based approach are based on the notion of transition system which is a finite state automaton. An FSA consists of states, transitions, mapping states and input symbols and transitions from initial to final states. The state transition happens on an input to a state. In case of transition based parsing, each of the states represent steps for deriving a dependency tree. These approaches make use of a stack-based technique for parsing. Transition based systems are mostly stack-based. Initially, the stack contains the artificial root and the buffer is empty. There are three kinds of transitions - shift, left and right. This is a bottom-up approach, i.e., for an arc to be constructed between two nodes, it is imperative that the dependent node has already a dependency tree constructed. This approach is also known as arc-standard. There is another approach called arc-eager where the terminal configuration is guided by the state of the buffer. If it is empty the arc-eager system

terminates. In addition to the transitions in the arc-standard system, there is an additional transition called reduce. This is a top-down parsing approach as the arcs are added as and when found ?.

- Graph-based - Graph-based approaches rely on the algorithms for directed graphs like finding the maximum spanning trees. There is a scoring function which evaluates how correctly a particular tree analyzes a sentence ?.

Various machine learning techniques have been implemented in different stages of parsing. In general, machine learning algorithms are used to compute scores in case of graph-based algorithms while for transition-based systems, it is used as an oracle. Supervised learning methods have proven useful while most recently, semi-supervised and unsupervised methods have shown promise. The goal is to reduce the time taken in parsing while not compromising the quality of the parse. The parsing problems by using these approaches are discussed in further details in the subsequent sections. We have further sub-categorized problems by whether they work for projective or non-projective sentence structures. Usually, the problem with projective sentences are a subclass of that of the non-projective sentences. The pseudo-projective approaches are also listed under the non-projective approaches. Neural network approaches are categorized in a separate section for ease of readability.

2.3.1 Transition based approaches

Broadly, there are two main approaches to transition based parsing.

- Greedy classifier based - The transition-based approach has a set of transitions between the states (or configuration). However, to categorize this as a parsing problem, we need to score these transitions and estimate the highest scoring transition by using a model. A classifier can be used as this model which maps configurations to the most optimal transition. The optimal transition can be determined in a greedy way,

where the final optimal transition is determined by selecting the optimal transitions locally. A representation of the scoring is as below:

$$Score(C, T) = \sum_{i=1}^N f_i(C, T) \cdot w_i$$

where C = configuration, T = transition Many machine learning techniques are used to learn the weights such as SVMs, perceptrons, etc.(more recently, neural networks)

- Beam search and structure learning - The beam search and structure learning has a similar strategy, however, in this case, instead of a greedy search, beam search is applied. Thus instead of 1 optimal transition, n number of possible optimal solutions are considered at each step where n is the beam size. The second part is similar to greedy classifier method, except that it has been seen (as we will see later in this chapter) that structured learning approaches tend to have better results.

We discuss the existing methods for transition based learning from the point of view of projectivity. Since our objective is to highlight the machine learning approaches, we will look at it from the perspective of the machine learning techniques used.

2.3.1.1 Projective

Probabilistic approaches to parsing initially showed promising results in dependency parsing as compared to rule-based techniques. Previous work in syntactic parsing (constituency parsing, in most cases) did not address the feature selection or in other words, how selecting good features could be a key in improving the results in dependency structure analysis. So, Kudo and Matsumoto? applied Support Vector Machines (SVMs) for analysis of dependency structure in Japanese. They assume that the dependency structure holds the following constraints.

- Each item depends on exactly one item appearing to the right.

- The dependencies are projective.

Since, the problem of statistical dependency analysis deals with finding the best dependency structure(D_{best}) given a sequence of chunks of a sentence (B) (chunks are specific to Japanese dependency structure and are defined as relation between phrasal units), it can be formulated as follows:

$$D_{best} = \arg \max_D P(D|B)$$

Considering the dependent probabilities are independent of each other, $P(D|B)$ can be written as:

$$P(D|B) = \prod_{i=1}^{m-1} P(Dep(i) = j|f_{ij})$$

where, $f_{ij} = f_1, \dots, f_n \in R^n$ (2.1)

$P(Dep(i) = j|f_{ij})$ denotes the probability that the chunk i depends on the chunk j given an n -dimensional linguistically-motivated feature set. To find D_{best} , the authors have used the backward beam search technique for statistical dependency analysis of Japanese sentence by ? which processes a sentence backwards.

The feature set consists of static features such as, head words and their parts-of-speech, particles and inflection forms of the words appearing at the end of chunks, distance between two chunks, existence of punctuation marks. In order to handle long sentences, dynamic features are included which resolves syntactic ambiguity at the time of parsing. These are mainly “form of functional words or inflection that modifies the right chunk”. The authors report a better accuracy compared to the previous work on the same corpus.

A similar discriminative model for dependency analysis on Wall Street Journal section (02-21 for training and 23 for test) of the Penn Treebank Marcus et al. [1994a] using SVMs by Yamada and Matsumoto?. They apply a deterministic bottom-up parsing algorithm which comprises of three actions - *Shift*, *Left* and *Right*. The parsing algorithm undergoes a two-step procedure to parse an input sentence considering words from left to right. The

first procedure involves assessing the appropriate action (*Shift* or *Left* or *Right*) from the contextual information provided by the surrounding words. Then, the parser builds the dependency tree by executing these actions determined in the previous step. During the training phase, each sentence is parsed using this algorithm. The contextual features with the right parsing action serves as an example for the SVM. Thus the estimation phase is a multiclass classification problem comprising of three binary classifiers for each action in a pairwise fashion:

- Left vs. Right
- Left vs. Shift
- Right vs. Shift

The decision on the action that needs to be implemented at any stage is determined by the cumulative votes from each of these SVMs. The features are as follows:

- word
- Part of Speech (POS) tags
- child word modifying the parent node on the right hand side
- child word modifying the parent node on the left hand side
- POS tag of the child node modifying the parent node on the right hand side
- POS tag of the child node modifying the parent node on the left hand side

Each feature is defined as a triplet consisting of the position from target node, feature type and the feature value. The accuracy of this parser is not at par with the contemporary phrase structure parsers owing to the fact that the phrase structures are not utilized in this parser. Malt parser (discussed in the non-projective section) addressed the problems faced by this parser viz., requirement to iterate for long sentences.

Cheng *et al.* [10] applied a similar approach using SVMs to determine if a dependency relation exists for any two pair of words based on Chinese Treebank. The parsing method is based on Nivre and Scholz's [11] bottom-up deterministic algorithm which parses a sentence in linear time. The basic difference is in the machine learning algorithm used for

these two papers. While Nivre and Scholz used memory based learners (5-nearest neighbors (IB 1) from TiMBL ?). This algorithm is also stack-based, where the analyzer states are represented as a triple $\langle S, I, A \rangle$ where S contains the words which are being considered, I contains the words which are to be processed and A is a list of dependency relation. It is also important to point out the differences between Yamada and Matsumoto’s approach to that of Nivre’s since both are based on English text and use a deterministic parsing algorithm. The primary difference would be the choice of classifiers, SVMs vs. MBL. Yamada and Matsumoto’s parser requires multiple passes but Nivre’s take one pass which makes Yamada’s algorithm’s worst case time complexity to be quadratic as compared Nivre’s algorithm’s linear time. The data-driven dependency parsing was further used for parsing Swedish ?, Bulgarian ? and Turkish ? treebanks with improvement over baseline.

At any stage, there are four possible operations for a given configuration - right, left, reduce & shift. Right & left operations add dependencies based on whether the top word in I depends on the top word in S or in the other direction respectively. For reduce, if the top element of S has no dependents, it is removed. For shift, the analyzer checks whether there is any possible dependency relation between the top elements of S and I . If it does not find one and the conditions for reduce are not met as well, then the top element of I is pushed into S .

Since the two methods (Cheng vs. Nivre) use different machine learners, the feature set is different. Nivre and Scholz considered the top word of the stack, the next input token, left and right dependent of the top element in the stack, left dependent of the next input token, the possible tokens from next configurations which are connected through a dependency arc. For the tokens in consideration, the lemma and part of speech tags are considered, while for the “lookahead” tokens, they have only considered the part of speech tags. For Cheng’s work, each node comprises of the word, POS tags and information of its children. They also take context features, such the preceding and succeeding nodes and its children into consideration. These are treated as local features. The global features are used for the

long distance dependencies. If the local features arrive at a decision which is shift/reduce, the analyzer uses SVMs to determine the correct operation using global features. They have also proposed a two-step process for finding the root words using SVMs. They have reported an increase in accuracy by employing the global features and the root word finder.

This method was further extended to account for multiword units (MWUs) such as multiword names, function words by Nivre and Nilsson ?. Particularly, to examine whether multiword units help in improving the parser results and if so, at which point should this be implemented. They experimented on the lexicalized and non-lexicalized versions of the parser based on memory-based learning. They show a significant improvement over the baselines for the syntactic structures in addition to the MWUs.

In order to analyze whether there is a “better” machine learning algorithm for parsing, Hall *et al.* ? presented a comparison between SVMs and memory based learning classifiers in deterministic dependency parsing for English, Chinese and Swedish using a variety of features. They show that the accuracy achieved by these classifier based models are almost at par with more complex parsing models. For their experiments, SVMs outperform the memory based learners but with a tradeoff in training times.

2.3.1.1.1 Pseudo projective approaches English sentences are generally projective, i.e., the edges of the dependency trees do not cross each other in a sentence. However, in the languages with a flexible word order like German, Czech and Dutch, the sentences tend to be non-projective with crossing dependencies. To solve, this problem, Nivre and Nilsson ? devised a “pseudo projective” method for parsing and reported an improvement over the state-of-the-art non-projective parsing results for Prague Dependency Treebank ?? by using a combination of data-driven deterministic dependency parsing (memory-based) with a graph transformation technique called lifting. Typically, a non projective dependency graph can be converted to a projective one by replacing the non-projective arc with a projective one. The authors have implemented a transformation involving a minimal

number of lifts. Thus, at the first step, the dependency trees are projectivized by applying the minimal lift transformation technique. In the second step, they have implemented an inverse transformation based on the breadth-first search algorithm using three encoding schemes. As a part of the experiments, memory based dependency parsers are used and the projectivized trees are used for training. The output is transformed using the inverse transformation and compared to the gold standard test set.

2.3.1.2 *Non-projective*

While the parsing techniques for English have proven to be efficient enough, parsing non-projective tree structures can prove to be challenging. A lot of techniques, as we will see later for the generative models as well, work for non-projective sentences with some modification on the projective counterparts.

This problem has been addressed by Nivre *et al.* [1], who introduced MaltParser, which they describe as “data-driven parser-generator for dependency parsing”. The parser requires a treebank and not a grammar unlike contemporary parser-generators. The parsing comprises of - building dependency graphs by deterministic parsing algorithms as shown by Yamada and Matsumoto and Nivre, estimating the next action of the parser by building a history-based feature model and finally mapping the history to parser action using discriminative machine learning algorithms (as demonstrated in Yamada and Matsumoto’s and Nivre’s work) such as memory based learning, SVMs. MaltParser specifies a fixed set of data structures and typically any parsing algorithm which follows this architecture would work in the framework. It supports Nivre’s [1] algorithm for projective dependency parsing and Covington’s [2] algorithms which uses three approaches - Brute-force search, Exhaustive left-to-right search and enforcing uniqueness. The feature models consists of word, lemma, part of speech tags, dependency type described in terms of the provided data structures. The evaluations provided by Nivre *et al.* for MaltParser is for Swedish, English, Czech, Danish and Bulgarian based on memory based learning.

It is clear that, parsing accuracy and time is an issue for non-projective parsing, which caused largely due to the way non-projective dependencies are handled. Also, it has been argued that, even for the free word order languages, the dependency structures tend to be either projective or “very nearly projective”. This leads to the discussion of finding the appropriate “degrees of non-projectivity” ?, which can be stated as finding a balance such that a small amount of non-projectivity might improve the parsing accuracy and time over strict projectivity and it is also more efficient than considering non-projectivity in abundance. Nivre ? investigated this appropriate degree of non-projectivity by using Covington’s parsing algorithm with history based SVM classifier to predict the next parser action. The results indicate that the languages exhibiting extensive non-projectiveness, the parser accuracy can be boosted if the non-projective dependencies are derived. However, on the other hand, parsing times can be improved by curbing the non-projectivity with a small decrease in parsing accuracy.

Although SVMs work well in case of parsing, it can be expensive in terms of training and memory requirements. A better workaround for this can be, to use multi layer perceptrons as the classifier Attardi et al. [2009] as perceptrons are faster and require less memory.

2.3.2 Graph based approaches

The most basic model for graph based parsing is the arc-factored model. An arc is an edge in the dependency tree. This is also commonly referred to as the first order model since the score is computed based on the scores of each arc. This is an exact inference. However, research has indicated that better parsing accuracy can be achieved when we considered bigger subgraphs i.e., set of 2 or 3 arcs in dependency tree. These models are referred to as second and third order models respectively but it increases the parsing complexity i.e., for learning and parsing. These are approximate inferences. In the subsequent sections, we have divided graph-based models in terms of projectivity and non-projectivity.

Eisner’s Algorithm Quite a lot of work has been done using generative models to improve the results of dependency parsing. To address the “lexical blindspot” of context-free grammars (CFGs), Eisner ? proposed three different probabilistic approaches to describe the basic structure of a sentence and apply it to a dependency framework for improving the resulting parses. This is a constituent CKY based algorithm for dependency parsing which uses words as node labels. This has been used for parsing projective dependencies The probabilistic models are as follows:

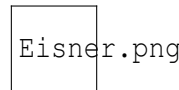


Figure 2.3: Three probabilistic models in Eisner’s work

- Bigram lexical affinities [expressions 1-3 in Fig. 2.3]

The first step is to generate tags by a Markov process given the previous two tags. Next, a word is chosen, given each tag. To account for the dependencies, there is a third step in this process which considers pairs of words and makes a random decision whether to link the two based on whether the probability of the words being linked together is “lexically sensitive”. I.e., both the tag and word are considered for any pair of words. This model also accounts for the inter-dependencies between the children of a word.

- Selectional preferences [expressions 4 in Fig. 2.3]

A sequence consisting of word and its tags are generated by a Markov process and then for each word, a parent is described. Hence the “selectional preference”¹ for each word is being considered in this case. Then, each of the words are independently “sense tagged” based on its selectional preference.

- Recursive generation [expressions 5 in Fig. 2.3]

¹referred to as disjunct, in the paper

This is a generative model as opposed to the two previously described comprehensive models. Every time a word is added, two separate Markov sequence of tag/word pairs are generated to serve as its left and right children. This process continues recursively for each child which is generated.

A probabilistic algorithm which is similar to CKY is proposed as the parsing algorithm. CKY considers substrings of increasing length for parsing and thus each such substring is represented as lexical trees. But, in this case, instead of considering substring as trees, the parser considers these as spans. These approaches when tested on the WSJ corpus indicate that the recursive generation model performs better as compared to the other two. The parsing algorithm has a $O(n^3)$ complexity.

Training these models is particularly easy as it constitutes mainly of estimating probabilities by counting the events related to parsing in the training set. However, this advantage is sometime nullified due to the poor decisions in the independence assumptions. This is where the discriminative models tend to perform better. However, these especially Maximum Entropy Markov Models based parsers ? tend to run into the label bias problem (?) because it does not take the parsing decisions based on an observation which might be seen later in the sequence. Parsing the training corpus repeatedly can solve this problem to some extent. However, it is really expensive - $O(n^5)$ as opposed to $O(n^3)$ for generative models. Hence, to strike a balance and utilize reduced parsing complexity of generative models like Eisner's, McDonald *et al.* ? described a method of training dependency parsers for English and Czech using margin-sensitive online training algorithms ?. The authors mention that this method can be translated for non-projective cases (where it may appear in Czech, for instance) as well. Since the reported results are for projective cases only, we have listed it so.

Chu-Liu-Edmonds Algorithm Chu-Liu-Edmonds algorithm?? often serves as a basis for non-projective parsing approach which determines the maximum spanning tree (MST)

of a graph in a greedy and recursive way. We will informally explain the intuition behind the algorithm as follows:

- Motivation: All the nodes in a graph need to be in arborescence. I.e., there should be exactly one directed path between two nodes in the graph.
- An incoming edge is selected for a node in a greedy way.
- The highest scoring incoming edge is stored for every non-root node.
- If a cycle is formed, a decision has to be made to remove one of the nodes to resolve the cycle.
- There are two stages - contract and expand.
- Following steps are followed for contract phase:
 - ◇ At first, the nodes in a cycle are contracted to form a new node/vertex.
 - ◇ All the incoming edges to the nodes in the cycle are now redirected to the new node.
 - ◇ Similarly, the outgoing edges have the new vertex as its starting vertex.
 - ◇ The incoming and outgoing edge weights for this new “contracted” vertex are recalculated.
 - ◇ This process is repeated recursively until every non-root vertex contains exactly one incoming node and there are no cycles.
- The MST formed on this contracted graph can be shown to be equivalent to MST for the original graph. ?
- Thus the algorithm is recursively implemented on the new graph to find the MST.

2.3.2.0.1 Non-projective approaches While the graph transformation techniques yielded good results with a cubic complexity, the parsing complexity was reduced by McDonald *et al.* [10] who proposed a dependency framework for both projective and non-projective dependency parsing by formalizing the problem as searching for maximum spanning tree (MST). They have used a similar approach as Hirakawa's [11] spanning tree search for dependency parsing. However, the worst case performance is exponential in this case as branch and bound algorithm is used. They used Eisner's spanning trees approach for projective and Chu-Liu-Edmonds algorithm [12] for non-projective dependency trees. At first, an edge-based factorization is applied to the projective languages with Eisner's parsing algorithm and Chu-Liu-Edmonds maximum spanning tree algorithm for non-projective languages. The edge factorization for an edge is done by computing a dot-product between a high dimensional feature representation of the edge in consideration and a weight vector. The score of a dependency tree is then a sum of the scores of all edges in the tree. The weight is updated by using the online learning algorithm implemented in McDonald *et al.* [10]. Their results show significant improvement in accuracy even with a small amount of non-projective sentences. They also reported improved parsing times - $O(n^2)$ as compared to $O(n^3)$ - in favor of the Chu-Liu-Edmonds algorithm as opposed to Eisner's. Their parser performs well even when compared to state-of-the-art [13] lexicalized phrase structure parsers such as Collins *et al.* [14] and Zeman [15], whose parsing complexity is $O(n^5)$. This framework was further account for higher-order feature representation and acyclic dependencies, i.e., multiple heads for each word [16] by using approximate inferencing of the online learning algorithms. The defined approximation is to start with a reasonably good baseline structure and then continue making transformations until the structure converges.

In order to devise more effective ways of solving multilingual parsing problems, McDonald *et al.* [10] proposed a two-stage multilingual parser and evaluated it on the 13 language treebanks provided in different CoNLL shared tasks [17]. The first stage in this model is to create an unlabeled parse for an input sentence. They extend the existing models by Mc-

Donald & Pereira ?, adding morphological features (where available) for each token. The morphological features (includes features for parent and its dependent and also various conjunctions of features from each of sets. The second stage is label classification which takes the output parse from stage 1 and assigns each edge with a label with the highest score. A first order Markov factorization has been used. Each factor is defined as the score of labeling adjacent edges. As defined in the earlier work by McDonald *et al.*, the score function is defined as the dot product between the high dimensional feature representation and a weight vector. The most likely sequence of labels is then ascertained by applying Viterbi's algorithm.

2.3.2.0.2 Higher order models As explained before, first order model is one in which the dependency tree is split into head and modifier dependencies. Second order models look into the adjacent dependencies in addition to these primary dependencies. McDonald & Pereira ? experimented with second order graph models. Carreras ? extended this to experiment with other types of second-order relations. In particular, they look into PP-attachment which requires looking into grand parent relations. The training is done with averaged perceptron on Eisner's(1996) algorithm. Although it reported one of the best reported labeled attachment scores for some languages, this model suffers in terms of time and memory utilization. Bohnet Bohnet [2010a] used this parser as a basis for their work on parallelizing the feature extraction and parsing algorithm by using passive-aggressive perceptron algorithm ? as Hash Kernel. The concepts from this model can be extended to the transition based parsers as well. Bohnet reported a 3.5 times increase in speed over the baseline MST parser using a single core CPU and it also requires a lot less memory than the contemporary parsers by using Hash Kernel. The speed increases further by using parallel algorithms and it can be further reduced at the cost of accuracy.

2.3.3 Combining transition-based and graph-based models

With the diverse amount of work on dependency parsers there has been work based on combining components from generative models with that of the discriminative approaches.

Graph-based and transition-based parsers have reported good accuracies for different criteria. Thus, a combination of both promises better results than individual. This premise was explored by Zhang & Clark [10] for projective dependency parsing. They considered Malt and MST for transition and graph-based respectively. Their basis was to use of beam search framework. They have used perceptron for training and beam search for decoding. They tested the combined parser on English and Chinese with comparable results with that of the best parsers for both models.

CoNLL-X Shared Task on Multilingual Dependency Parsing [11] constituted multilingual parsing using a single dependency parser which can learn from treebank data. Nivre *et al.* [12] used MaltParser to solve this problem for Swedish and Turkish. For mapping parser actions to history, they used SVMs and used graph transformations described by Nivre and Nilsson [13] to restore the non-projective structures.

2.3.4 Unsupervised and semi-supervised approaches

Some unsupervised and semi-supervised techniques have also been suggested to improve parser accuracy and addressing the concerns about portability of the system to other parsing frameworks. Blunsom & Cohn [14] reported higher attachment scores by focusing on dependency grammar induction using tree substitution grammar because of its ability to learn large chunks of dependency tree. They devised a hierarchical non-parametric prior due to its bias towards simple productions. Spitkovsky *et al.* [15], on the other hand, argued that Viterbi actually is better suited for the problem of grammar induction. They tested their approach on Brown corpus. However, there is no direct comparison with Blunsom's approach.

2.3.5 Neural Network based approaches

Although transition-based dependency parsers work reasonably well, but these parser tend to suffer due to poor estimation of feature weights, the incompleteness of manual feature templates and the time complexity for extraction of these features. The problem with sparse indicator features can be mitigated by using dense word embeddings as indicated in some of the recent works, such as for POS tagging ? . Chen and Manning ? use a neural network based classifier for making the parsing decisions in a transition based dependency parser ? . The architecture of the neural network based parser is given shown in figure 2.4. It contains exactly one hidden layer with a cubic activation function ($g(x) = x^3$).

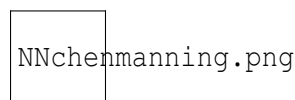


Figure 2.4: Neural Network architecture

A greedy decoding is performed for parsing. At every step, word, its POS and label embeddings are extracted from the current configuration and like all the transition based parsers, the transition with the highest score is selected. This work is shown for the projective case only. For the selected sentences in PTB and CTB, this parser works better than MaltParser and MST in terms of parsing time and accuracy. ? followed a similar approach but instead of one hidden layer, they added two which slightly improved the parser accuracy. Their work differs from Chen and Manning's by the use of semi-supervised structured learning which is implemented for the training. To learn the final layer of the model, they make use of structured perceptron. They also introduce unlabeled data by using word embeddings. This work was further extended for multilingual cases ?. The difference is in the use of set or bag of features which are embedded into the same embedding space. Chen and Manning's neural network architecture for parsing was further modified by ? using beam search for the decoding step and contrastive learning to maximize the sentence-level log likelihood. They reported 1.8% increase in accuracy over Chen and Manning's greedy

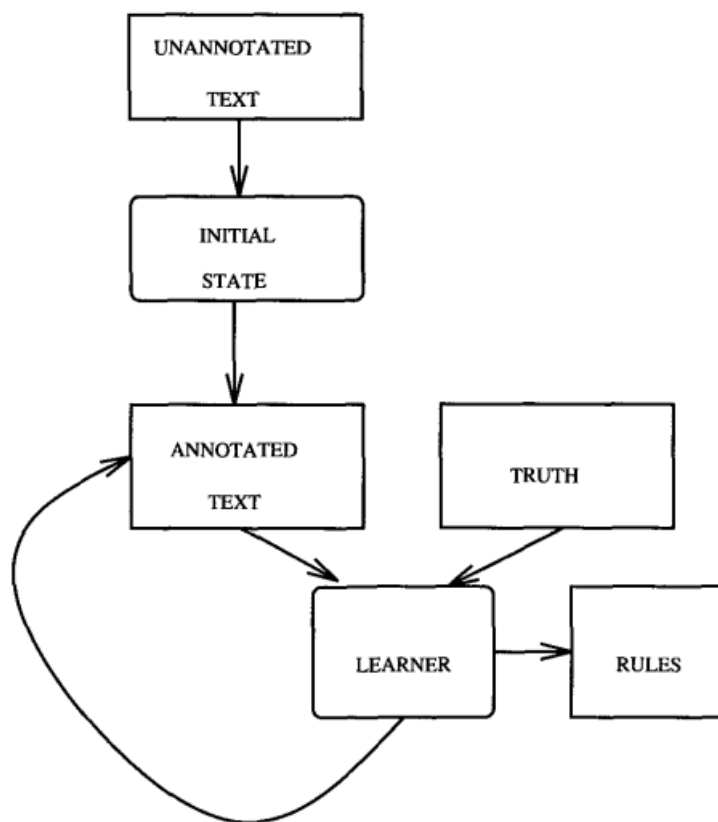


Figure 2.5: Transformation Based Error Driven Parsing
[Brill, 1992]

neural parser.

A graph-based neural network parser was proposed by Pei *et al.* [1] which made use of a $\tanh - cubic$ activation function instead of Chen and Manning cubic activation function. The parser outperformed the baseline graph-based parsers. However, it is not clear if this parser outperforms the other neural network based parsers. Neural network parsers for multilingual settings are implemented by using convolutional neural networks by Zhang *et al.* [2] by using the basic architecture of Pei *et al.*.

2.4 Transformation Based Error Driven Learning

The underlying idea of transformation based error driven learning (TBL) is shown in figure 2.5. Transformation based error driven learning (TBL) was originally developed for

POS tagging Brill [1992, 1995]. The idea is straightforward, yet effective. We have a set of unannotated text. We run it through an “initial” state system, which is dependent on the task. E.g., in Brill’s case, it was a stochastic n -gram tagger. This produces an initial annotation, albeit not a good/accurate one. However, the errors in the initial annotation can be continually improved by comparing it to the ground truth or the gold standard. Each such annotation passes through multiple iterations of error correction, until there is no more improvement in terms of reduction of errors. In repeating this process, the TBL process accumulates a set of rules, which is the final output. This is the training phase, which essentially learns from the errors and creates a set of rules. Now, that we have learned these rules, we can apply it to another corpus.

As described in the previous paragraph, the method works with two versions of the training data: the gold standard and (an initially unannotated) working copy that simulates the learning process and records the errors. The unannotated version of the training data is tagged by a simple part of speech tagger², to create the initial state of the working copy of the text. The goal is to bridge the difference between the gold standard text and the working copy by learning rules that change the incorrect POS tags to the correct ones.

Learning is based on rule templates, which consist of two parts: rewrite rules and triggering environment. The templates need to be determined by the researcher, based on the problem. In general, a rule template is of the form:

Rule Template $(A, B) : X \rightarrow Y$

I.e., if we observe conditions A and B (triggering environment), we change the output variable (POS tags, for Brill and dependency labels, for us) from X to Y (rewrite rule). Note that X can remain unspecified, then the rule is applied independent of the current variable (label or POS tag).

The following is an example of a rule template for POS tagging: Change the POS tag of the current word from X to Y if the previous word is tagged as Z , where X , Y , and Z are

²This could be any simple part of speech tagger, or a heuristic that labels every word with the most frequent POS tag

variables that need to be instantiated during learning.

The learner creates rule hypotheses out of those templates by identifying incorrectly tagged words in the working copy and instantiating the template variables. For example, one possible rule hypothesis could be the following: Change the POS tag of the word “impact” from verb to noun if the previous word is tagged as a determiner.

In one pass through the system, all possible rule hypotheses are created and ranked based on an objective function which determines for each hypothesis how many corrections occur. The rule hypothesis with the highest score is applied to the working copy and added to the final list of transformations, stored in order, during the training process. Then the process repeats, and new rule hypotheses are generated from the templates based on the remaining errors in the working copy. The process finishes when there is no more improvement in terms of reduction of errors. _____

more?

Brill [1993] also used this approach to parse text by learning a “transformational” grammar. The algorithm repeatedly compares the bracketed structure of the syntactic tree to the gold standard structure and learns the required transformations in the process . _____

Expand

Brill and Resnik [1994] also use this technique for prepositional phrase attachment disambiguation. _____

Expand

Transformation based error driven learning has also been used for information retrieval by Woodley and Geva [2005]. _____

Expand??

2.5 Evaluation

Example? _____

Look
at hulk

In this section I discuss the different evaluation techniques that will be used in this thesis. I report on the standard metrics for POS tagging and dependency parsing. I discuss these metrics in detail below.

2.5.1 POS tagging

- Overall Accuracy: For POS tagging, I evaluate the results based on the accuracy of identifying POS tags correctly in the test set as shown in 2.2. I.e., I measure how many tags have been correctly identified in the test set.

$$Accuracy = \frac{\text{Number of correctly identified tokens}}{\text{total number of tokens}} \quad (2.2)$$

- Accuracy for Known & Unknown tokens: For POS tagging, unknown or out of vocabulary words (OOV) pose as a challenging problem. `tnt-diff` provides an utility to measure accuracy based on number of correct predictions for known vs. unknown words. I discuss the method employed by TnT to determine the POS tags for unknown words in section ***TBD***. I.e., we can determine the accuracy on known and OOV words separately. Since performance on OOV words is an essential determining factor, I use this metric to further analyze the results from domain experts in greater detail. This is a bigger challenge in a domain adaptation situation since there are domain specific words for each domain, which tend to get misclassified in a heterogeneous dataset.

2.5.2 Dependency Parsing

- Labeled Attachment Scores (LAS): For evaluating the results of dependency parsing experiments in this chapter, I report LAS³. As 2.3 shows, LAS estimates the number of words with correctly predicted head and label.

$$LAS = \frac{\text{number of words with correct head and label}}{\text{total words}} \quad (2.3)$$

- Unlabeled Attachment Scores (UAS): UAS, as the name suggests, evaluates based

³I report micro-averaged LAS. Micro-averaged LAS is reported on words as opposed to macro-averaged LAS, which considers sentences as the grain.

on correctly predicted head.

$$UAS = \frac{\textit{number of words with correct head}}{\textit{total words}} \quad (2.4)$$

- Exact Match (EM): This metric determines how many sentences are parsed accurately by the parser.

CHAPTER 3

DOMAIN ADAPTATION

domain adaptation lit review

Domain adaptation is a well studied problem in machine learning and natural language processing. Usually, there is a plenty of labeled data available for one domain (also known as the source domain) but nearly not enough or in some cases, none available from a different domain (also known as target). The challenge in that case, is to apply well performing systems from source domain and adapt it the target domain. In most problems, this results in a drop in performance, sometimes severe. The same holds true for POS tagging and dependency parsing. These systems perform well when trained and tested on datasets that are predominantly in the same text domain. However, there is a considerable decrease in accuracy if the domains under consideration, are markedly different. It is a well-studied problem for POS tagging and dependency parsing (more so for dependency parsing) but the improvement proposed by these systems have been negligible at best. This is mainly due to unavailability of annotated data from target domain. Daume III [2007] notes that this problem is “frustratingly easy” when some annotated data is available from the target domain and “frustratingly hard” if no such target data is available [Dredze et al., 2007a].

CHAPTER 4

CORPORA

In this chapter, I elaborate on the corpora I used for the experiments conducted on domain adaptation for POS tagging and dependency parsing. I use three main corpora as a representative of their domains. I discuss this in detail in the next sections. In addition to classical domain adaptation, the goal is also to achieve a more generic form of domain adaptation, i.e., to evaluate if my system can detect domains in a heterogeneous or mixed dataset. Thus, I create an artificial corpus from the corpora, which contains equal representation of two domains. Although all the corpora used in this thesis are annotated in the PTB style, each has a distinctive syntactic style, which ...

4.1 Wall Street Journal section of Penn Treebank (WSJ)

¹ This is the Wall Street Journal section [Marcus et al., 1994a] of the Penn Treebank [San-
torini, 1990]. This corpus contains annotated newspaper articles from WSJ from 1987 -
1989. The corpus is annotated for POS tags and parsed in PTB bracketed style. The depen-
dency trees are then derived using the Penn Converter Tool [Johansson and Nugues, 2007]
and the inconsistencies are addressed manually. I use this corpus as a representative of the
newspaper domains. There are various different kinds of news articles from financial news,
theater critique, weather, sports, politics, etc. Thus, in addition to contributing as a main
domain, we can also exploit the corpus for micro-genres or more fine-grained domains.
The sentences in WSJ are reasonably longer with an average length of 24 words. I use the
standard split for parsing, which is, 02-21 for training, 22 for test and 23 for validation.
Some of the example sentences from WSJ are as given below.

¹<https://catalog.ldc.upenn.edu/LDC2000T43>

dont
know
how to
finish
this
sen-
tence
check!!

EXAMPLE

4.2 GENIA Corpus

The GENIA Corpus [Tateisi and Tsujii, 2004] comprises of biomedical abstracts from Medline, and it is annotated on different linguistic levels, including POS tags, syntax, coreference, and events, among others. I use GENIA 1.0 trees Ohta et al. [2002] created in the Penn Treebank format². The treebank is converted to dependencies using pennconverter Johansson and Nugues [2007]. The tagset used in GENIA is based on the Penn Treebank tagset, but it uses the tags for proper names and symbols only in very restricted contexts. This treebank serves as a representative of biomedical domain for my experiments. Clearly, this is very distinct from WSJ in terms of the nature of the content. Some of the sentences from the corpus are shown below.

EXAMPLE

4.3 CReST Corpus

The CReST corpus [Eberhard et al., 2010] constitutes of natural language dialogues between two individuals performing a “cooperative, remote, search task” (CReST). This is a multimodal corpus which is annotated for speech signals and their corresponding transcriptions. The transcribed text is annotated in PTB format for constituent trees. These constituent trees are converted to dependency trees using Penn Converter [Johansson and Nugues, 2007]. Since this is transcribed from spoken dialogues, the average length of sentence is 7. In terms of linguistic factors, CReST has a subset of WSJ’s dependency labels with 10 new labels added. A few examples of the CReST corpus are given below:

EXAMPLE

I use CReST as a representative target domain in my experiments. The CReST corpus consists of 23 dialogues that were manually annotated for dependencies. I randomly select 19 dialogues as my training data for the TBL algorithm and the rest as test. Since the system needs to learn rule hypotheses from incorrect predictions of the source domain parser, I can safely ignore sentences that were parsed completely correctly³. For the test data, I use all

²<http://nlp.stanford.edu/mcclosky/biomedical.html>

³Details about the split are discussed in Chapter TBL *TBD*

	# Dialogues	# Sentences	
		with errors	without errors
Training Set	19	4384	459
Test Set	4	831	85

Table 4.1: Target domain training and test set for TBL

	Training Set	Test Set
WSJ	17 181	850
GENIA	17 181	850
Total	34 362	1 700

Table 4.2: Overview of the mixed dataset (WSJ+GENIA)

the sentences from the designated dialogues (with correct and incorrect dependency label predictions). Table 7.4 shows the division of sentences for training and test.

4.4 WSJ-GENIA mixed corpus (WSJ+GENIA)

Since my objective is to identify domains from a heterogeneous dataset and then adapt POS taggers and dependency parsers to the corresponding domain, I need a dataset which contains sentences from different domains. In order to test my hypothesis, I create an artificial corpus by incorporating data from WSJ as well as the GENIA corpus in equal measure.

I randomly select 17 181 sentences from section 02-21 of WSJ corpus as training data and similarly select 850 sentences from section 22 for test. Then, I add equal amount of sentences from GENIA to create a heterogeneous balanced corpus.

4.5 Summary

TBD

CHAPTER 5

DOMAIN EXPERTS VIA TOPIC MODELING

5.1 Introduction

In this chapter, I address a problem which is analogous to the classic domain adaptation problem since the aim is to improve (morpho-)syntactic analysis for different domains, but more generic in nature. In order to simulate a more realistic scenario, I assume that the dataset on which the taggers and the parsers is trained on, does not come from a single domain but may contain a mix of different domains. The same is true for the sentence that is parsed (or tagged) using the model trained on this dataset. I.e., the test sentence could potentially come from any of the domains. Thus, in my case, the problem is two fold - identify domains automatically in the training dataset and then suitably adapt the method to parse sentences more accurately. There is no manual work involved.

I describe my method on improving POS tagging and dependency parsing for such heterogeneous datasets from a variety of different genres by creating experts for automatically detected topics. In this case, the datasets consist of newspaper reports on the one hand and biomedical extracts on the other. I assume that the domains have an equal participation in the dataset. I use Latent Dirichlet Allocation (LDA) to determine the topic of a sentence. LDA displays the latent topic structure in a document. In this case, a document to be clustered consists of a single sentence. I then assign each sentence to the most likely topic, for both training and test sentences. I train an expert for each topic and then use this expert to POS tag and parse the test sentences belonging to this topic. I assume that the topics detected by the topic modeler do not only pertain to lexical differences, which can be beneficial for the POS tagger and the parser, but also to syntactic phenomena. Thus, one topic may focus on "incomplete" sentences, such as headlines in a newspaper.



The rest of the chapter is structured in the following way: *TBD after completion*

5.2 Research Questions

The goal is to create POS tagging and parsing experts for heterogeneous datasets which consist of sentences from different genres. For example, the dataset might be a mixture of newspaper articles, blogs, financial reports, research papers and even specialized texts such as biomedical research papers and law texts. I create experts such that each expert would learn specific information about its own genre. I determine these experts by performing topic modeling on sentences and then train an expert on the sentences of the topic. I group sentences based on their most probable topic. To test the hypothesis that topic modeling can serve to group sentences into topics, I create a mixed dataset from the financial domain (using the Penn Treebank Marcus et al. [1994b]) and from the biomedical domain (using the GENIA Corpus Tateisi and Tsujii [2004]) such that the new handcrafted corpus consists of sentences from both domains in equal measure. Consequently, there is a clear difference in the genres in the corpus, and I have gold standard topic information.



In this chapter, I investigate the possibility of creating genre experts, given a heterogeneous dataset. Thus, I perform topic modeling on training and test data simultaneously: I assign a test sentence to the topic with the highest probability. This means that I currently simplify the problem of assigning new sentences to topics. In the next chapter, I assign new sentences to topics based their similarity to sentences in the topics created during training, following the work by [Plank and van Noord, 2011].

To summarize, I present the results on the following research questions in this chapter.

Question 1: Can topic modeling successfully identify genres in a dataset?

In this question, I determine whether the data splits obtained from the topic modeler are meaningful for creating domain experts. I.e., I investigate whether an unsupervised topic modeler can detect topics in a heterogeneous corpus (table 4.2). I use an artificially created

heterogeneous corpus containing sentences from the Wall Street Journal (WSJ) section of the Penn Treebank [Marcus et al., 1994b] and from the GENIA Corpus [Tateisi and Tsujii, 2004] and take their original corpus as the gold standard topic. I assume that a good split into the known topics, financial news and biomedical abstracts, will also improve POS tagging and parsing accuracy. If I assume two topics, we should be able to see a clear distinction between WSJ and GENIA sentences. I.e., for each topic, we should have a clear correspondence of its sentences to either WSJ or GENIA. I thus calculate the percentage of sentences in a given topic that belong to GENIA and expect that one topic should have a high percentage and the other one a low percentage. I also experiment with a larger number of topics, to see if I can profit from a finer grained topic definition. However, this advantage will be offset by a smaller training set since we split into more sets.

Question 2: Does POS Tagging Benefit from Using Topics?

In this question, I examine whether the performance of POS tagging improves if we create experts based on the topics detected by the topic modeler. In order to investigate this question, I generate a two-topic corpus by combining data from the Wall Street Journal (WSJ) section of the Penn Treebank Marcus et al. [1994b] and from the GENIA corpus Tateisi and Tsujii [2004] as shown in table 4.2. The WSJ covers financial news while GENIA uses Medline abstracts as its textual basis. As a consequence, I have sentences from two different genres, but also slight variations in the POS tagsets. The tagset used in GENIA is based on the Penn Treebank tagset, but it uses the tags for proper names and symbols only in very restricted contexts. This setup allows me to test whether the topic modeler is able to distinguish the two genres, and whether POS tagging experts can profit from this separation. I use the topics created for the previous sections and train a POS tagging expert on the training part of each topic. I then use the expert to tag the test sentences from this topic. In this setting, we can see if the experts can effectively handle the data sparseness caused by dividing the training set into multiple experts. I



experiment with one setting in which I use topic modeling as hard clustering, i.e., I assign each sentence to the topic for which the topic modeler gave the highest probability. I could also potentially experiment with a different clustering technique - soft clustering, in which I add each sentence to all topics, weighted by its probability distribution. I discuss these clustering techniques in more detail in the next section

Question 3: Does Dependency Parsing Benefit from the Topics?

Here, I investigate the effects of using topic modeling experts for dependency parsing. I first use gold POS tags in order to abstract away from POS tagging quality. In a second step, I investigate the interaction between POS tagging and parsing experts. I.e., I am interested in whether dependency parsing can profit from using the POS tags that were determined by the POS tagging experts. This helps in determining whether integrating POS information given by the POS experts can improve dependency parsing or whether there is no interaction between the two levels. Similar to POS tagging experts, I experiment with two different clustering techniques - hard & soft clustering to address the data sparsity issue.



Question 4: What do the Experts Learn?

In this question, I take a closer look at the results from the previous question to learn where the improvements by the experts (POS tagging & parsing) are come from. I investigate on certain known issues for both POS tagging and dependency parsing in domain adaptation situation and in general, to ascertain the source of these improvements.



For POS tagging, out of vocabulary words are a major concern. Hence, I analyze whether all the improvements based on lower rates of out-of-vocabulary words. For example, suppose we have two experimental settings, both using the same size of the training set, but in one setting, the majority of the training set is from GENIA while in the second setting, the training set is a mix of GENIA and WSJ. It is more likely that the former will

contain a wider range of biomedical vocabulary than the latter. However, it is also possible that the experts will learn different regularities, for example with regard to how the proper name tags are used in the two corpora. Thus, I look at the ratio of unknown words in the different experiments and at the error rates of known and unknown words. I additionally look at the confusion matrices.

I also analyze the parsing results in more detail to gauge the effect of using topic modeling experts. Here, I'm primarily interested in whether there are specific types of sentences or dependencies that are grouped by the topic models, so that the parsing experts focus on a specific subset of syntactic properties. Genre differences can be captured by adapting to certain syntactic phenomena. Hence, I look more closely if there is a particular type of sentence that benefit from using experts over the general case. Mislabeling of dependency is another problem, which can impact overall performance of the parser. I look at confusion matrices for dependency labels to further investigate if there is an improvement from using the experts.

5.3 Experimental Setup

5.3.1 Overall Architecture



Figure 5.1 shows the overall architecture of the system. I use sentences as documents. Based on the document topic information, I then group the sentences into genre topics. Note that, I utilize the artificial heterogeneous WSJ+GENIA corpus, described in section 4.4. I collect all sentences from the training and test set, cluster them via the MALLET topic modeler, and determine for which expert(s) the sentence is relevant to. There are several ways of determining the best expert based on the probability distribution of topics in a sentence. Then, we separate the sentences for each expert into training and test sentences, based on the previously determined data splits (see above).



The experts can be determined based on hard or soft clustering decisions: For hard clustering, the sentences are assigned to hard topics, based on the topic that has the highest

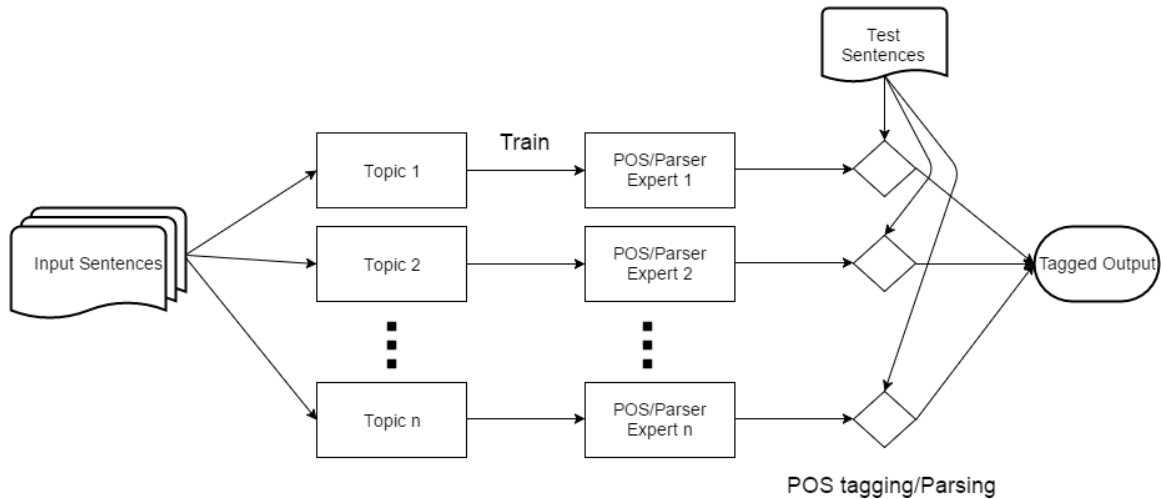


Figure 5.1: Overview of the architecture of the POS tagging and parsing experts.

probability in that sentence. I.e., if for sentence s_x , MALLET lists the topic t_1 as the topic with the highest probability, then s_x is added to the data set of topic t_1 . In other words, the data set of topic t_1 consists of all sentences for which MALLET showed topic t_1 as the most likely topic. This means that the data set sizes vary between topics. This is a simplification since a sentence can represent different topics to different degrees. Thus, I investigate whether I can utilize the soft clustering information directly and add every sentence to every POS tagging expert, weighted based on the degree to which it represents the topic of this expert. This not only allows me to model topics in more detail, it can also help combating data sparsity since every sentence contributes to every domain expert.

Hence, for soft clustering experiments, I utilize the entire topic distribution of a sentence by weighting sentences in the training data based on their topic distribution. I simulate weighting training sentences by adding multiple copies to the training files of the experts. Thus, for 2-topic experiments, a sentence with 80% probability for topic 1 will be included 8 times in the expert for topic 1 and 2 times in the expert for topic 2, rounding up small percentages so that every sentence will be added to every expert at least once. Thus, I use a more fine grained topic model while mitigating data sparseness, but we risk adding non-typical / irrelevant sentences to experts.

5.3.2 Clustering Decision

Not sure if this should be here or in a separate chapter or a research question



In the research questions, I outlined a setting of using a finer grained split with a larger number of topics. However, evidently, larger number of topics could result in severe data sparseness. This is assuming that I treat topics as hard clusters, i.e., every sentence belongs to the topic with the highest probability. So, creation of experts causes a sparsity in the training data because I am dividing the training set into a number of experts. For example, if I create 10 experts, I am basically dividing the training set to a fraction of 10. This essentially means that, now my training experts can only train on $\frac{1}{10}^{th}$ of the full training set. This is also a simplification since a sentence can represent different topics to different degrees. This is especially true in case of identifying micro-genres or larger number of topics, in general. Consider the following example:

In order to represent the distinct domains (WSJ, GENIA) and also the sub-genres of a particular domain, I create 2 as well as 10 topics experts. Typically, if I consider the topic prior as 2 & 10, the distribution looks as given in Table 5.1 & 5.2¹, for the following WSJ sentence:

“Though growers can’t always keep the worm from the apple , they can protect themselves against the price vagaries of any one variety by diversifying – into the recently imported Gala , a sweet New Zealand native ; the Esopus Spitzenburg , reportedly Thomas Jefferson’s favorite apple ; disease-resistant kinds like the Liberty.”

In Table 5.1 and 5.2, we can see the probability of the highest (7) and the second highest (5) topic are fairly close. Thus, if I take topic 7 and discard the other topics, I lose a fair share of information given by LDA. In other words, dividing a corpus into experts creates a significant sparsity in training data if I consider the highest probability topic and discard the rest. This problem can be mitigated if we effectively utilize the whole topic probability distribution by giving a sentence to more than one topic. Thus, I also investigate whether we

¹an approximation

Table 5.1: 2 topics distribution

0	1
95.90	4.10

Table 5.2: 10 topics distribution

0	1	2	3	4	5	6	7	8	9
0.11	0.10	0.08	0.10	0.09	30.58	15.28	49.55	3.93	0.16

can utilize the soft clustering information directly and add every sentence to every domain expert, weighted based on the degree to which it represents the topic of this expert. This not only allows us to model topics in more detail, it can also help combating data sparsity since every sentence contributes to every expert. The risk is that I diffuse the expert knowledge too much by adding all sentences even if they are weighted.



5.3.3 Baselines

I define two baselines to compare my results with. As the first baseline, I take the complete training set when no topic modeling is performed. Note that this is a very competitive baseline since the topic modeling experts have access to considerably smaller amounts of training data. In order to avoid differences in accuracy resulting from different training set sizes, I create a second baseline by splitting the sentences randomly into the same number of groups as the number of topics, while maintaining the equal distribution of WSJ and GENIA sentences where applicable. I.e., I assume the same number of random “topics”, all of the same size. Thus, in the 2-topic setting with the the genres, I create two separate training sets, each containing half of the WSJ training set and half of the GENIA one. In this setting, I test all experts on the whole test set and average over the results.

T.	2 topics		10 topics	
	% in train	%in test	% in train	% in test
1	0.71	0.71	0.48	0.52
2	97.99	98.6	98.58	98.35
3			1.16	0.73
4			94.87	97.14
5			0.17	0
6			0.28	0.29
7			99.47	99.12
8			98.93	100
9			98.92	99.33
10			94.85	95.35

Table 5.3: Distribution of sentences from the WSJ+GENIA data set given 2 and 10 topics (showing the percentage of GENIA sentences per topic).

5.3.4 Evaluation

I use the script `tnt-diff` that is part of TnT to evaluate the POS tagging results and the CoNLL shared task evaluation script² for evaluating the parsing results. I report the overall accuracy for POS tagging and attachment scores (labeled & unlabeled) for dependency parsing.

5.4 Experimental Results

In this section, I discuss the results of the experiments based on the research questions delineated in section .

Question 1: Can topic modeling successfully identify genres in a dataset?

Based on EACL split

Following question 1, I investigate whether LDA can separate the sentences into meaningful topics. Table 5.3 shows the distribution of sentences in the training and test set into different topics when I assume 2 or 10 topics. These results indicate that the topic modeler


²<http://ilk.uvt.nl/conll/software/eval.pl>

1	mr million ui year company market stock billion share corp years shares trading president time quarter sales government business
2	cells cell expression il nf activation human binding gene transcription protein kappa ab cd ti factor alpha activity induced


Table 5.4: Examples of words in topics for the 2-topic experiments on the WSJ+Genia corpus.

Setting	Accuracy	
	2 topics	10 topics
Full training set	96.64	
Random split	96.48	95.49
Topic model	96.84	96.34
Soft Clustering	96.73	96.84

Table 5.5: Comparing the topic model experts to the baselines on the WSJ+GENIA data set.

separates topics very efficiently. For the 2-topic experiments, a clear split is evident as the majority of the GENIA sentences are clustered in topic 2; the misclassified sentences constitute less than 1%. For the 10-topic experiments, we notice that topics 2, 4, 7, 8, 9, and 10 contain mainly GENIA sentences while the remaining topics cover mainly WSJ sentences. In both settings, the error rate is between 0.2% and 5%, i.e., I obtain a distinct split between GENIA and WSJ, which should give us a good starting point for the following experiments. Table 5.4 shows example words from the 2-topic experiment, which show a clear separation of topics into biomedical and financial terms. 

Question 2: Does POS Tagging Benefit from Using Topics?

In this section, I present the results on the experiments for question 2. I investigate whether the POS tagger can benefit from using topic modeling, i.e., whether POS tagging results can be improved by training experts for genres provided by topic modeling. I compare the topic modeling approach to two baselines for the 2-topic and 10-topic setting. I also perform a soft clustering experiment, in which each sentence is added to every topic, weighted by its probability. 

The results in Table 5.5 show that if I assume a 2-topic setting, the experts perform

better than both baselines, i.e., the model trained on the full training set and the model with randomly chosen “topics”. The 2-topic expert model reaches an accuracy of 96.84%, which is slightly higher than the full training set accuracy of 96.64%. We know that the 2-topic setting provides a clear separation between WSJ and GENIA (Table 5.3). Thus, this setting outperforms the full training set using a smaller amount of training data. There is also an increase of 0.36 percent points over the accuracy of the 2 random split setting.

For the 10-topic setting, the topic expert model outperforms the random split of the same size by 0.85 percent points, which is a higher difference than for the 2-topic setting. This shows that the finer grained splits model important information. However, the topic expert model does not reach the accuracy of the baseline using the full training set. This can be attributed to the reduced size of the training set for the experts.

Since training set size is a detrimental factor for the larger number of topics, I also conducted an experiment where I use soft clustering so that every sentence is represented in every topic, but to a different degree. The last row in table 5.5 reports the results of this experiment. We notice that the 2-topic experts cannot benefit from the soft clustering. Since the separation between WSJ and GENIA is very clearly defined for the 2-topic experiments, the advantage of having a larger training set is outweighed by too many irrelevant examples from the other topic. However, the 10-topic model profits from the soft clustering, which indicates that soft clustering can alleviate the data sparseness problem of the POS tagging experts for larger numbers of topics.

Question 3: Does Dependency Parsing Benefit from the Topics?

In this section, I discuss my findings for question 3. I present my results from two perspectives. One, where I assume I have gold POS tags and another, where I use POS tags from TnT as an input to the parser. This will help me in determining the effect of POS tags on parsing choices.

Setting	LAS		UAS	
	2 topics	10 topics	2 topics	10 topics
Full training set	88.67		91.71	
Random split	87.84	84.91	90.86	88.64
Topic model	90.51	88.38	92.14	90.3
Soft clustering	89.86	89.91	91.99	91.84



Table 5.6: Results of the dependency parsing experiments using gold POS tags.

5.4.0.1 Using Gold POS Tags

I now look into the parsing experiments using gold standard POS tags. The choice of gold POS tags allows me to focus on the contribution of the topic modeling experts on parsing results.

The results of the experiments are shown in Table 5.6, for 2-topic and 10-topic settings and in comparison to the two baselines, for the hard and soft clustering experiments. The hard clustering results indicate that the 2-topic expert model reaches an improvement over the baseline using the full training set for both the labeled attachment score (LAS) and the unlabeled attachment score (UAS). There is an increase of around 2% over the baseline for LAS, and an increase of 0.43% for UAS. However, for the 10-topic setting, both the LAS and the UAS are slightly lower than the baseline. For LAS, the difference is 0.29 percent points while for UAS, the difference is 1.41 percent points. This shows that the gain in LAS and UAS is offset by the reduced training set, parallel to the results for POS tagging. Both the 2-topic and the 10-topic experts outperform the random split baseline (which uses similar training set sizes), with a gain of more than 3 percent points.

The soft clustering results show the same trends as in the POS tagging experiments: For the 2-topic setting, soft clustering outperforms the full baseline by 1.19 percent points. But it does not exceed the hard clustering results. In the 10-topic setting, soft clustering outperforms the full baseline as well as the hard clustering setting. This is because sentences with a 50% probability of belonging to topic 1 and a 40% probability for topic 3 need to be considered to belong to both topics. This result also shows that this method effectively

Setting	LAS		UAS	
	2 topics	10 topics	2 topics	10 topics
1. Full set POS + full set parsing	86.70		90.26	
2. Random split POS + random split parsing	85.77	81.33	89.11	85.73
3. Full set POS + topic model parsing	88.30	86.13	90.43	88.47
4. Topic model POS + Topic model parsing	88.35	85.68	90.55	88.15

Table 5.7: Results of the dependency parsing experiments using TnT POS tags.

Sentence	Fulltext LAS	2-topic LAS
Phyllis Kyle, Stephenson Newport News , Va .	0	25.00
But volume rose only to 162 million shares from 143 million Friday .	46.15	61.54
Fidelity , for example , prepared ads several months ago in case of a market plunge .	47.06	82.35
CALL IT un-advertising .	50.00	75.00
(See related story : " And Bills to Make Wishes Come True " – WSJ Oct. 17 , 1989 .	52.38	61.90

Table 5.8: Comparison of LAS for the sentences with the lowest LAS in the fulltext setting.

handles the training data sparsity in the 10-topic setting.

5.4.0.2 Using the POS Tagger

In section 5.4.0.1, I use the gold standard POS tags in the POS tags. In this section, I explore the results of using POS tags from the POS tagger TnT as the input for the parser. This gives rise to four major scenarios:

1. The full training set is used for POS tagging and for parsing (full baseline).
2. Random splits are used for parsing and POS tagging. I.e., the POS tagger and parser are trained on random splits (random baseline).
3. Topic models are used for training the parser, but TnT is trained on the whole training set.
4. Topic models are used for training the parser and the POS tagger.



Gold Dep.	Pred. Dep.	Fulltext	Topic 1	Topic 2
ADV	NMOD	121	37	86
PMOD	NMOD	101	21	67
NMOD	ADV	100	34	57
AMOD	NMOD	91	26	83
CONJ	NMOD	86	13	56

Table 5.9: The 5 most frequent dependency label confusions of the full baseline parser.

I use the random split case as the lower baseline for these experiments and the full training set as the more competitive baseline. Table 5.7 shows the results.

Table 5.7 shows that in the 2-topic setting, using topic modeling experts on the POS level as well as on the parsing level reaches the highest results with an improvement of around 2% in LAS in comparison to the full baseline parser, from 86.70% to 88.35%. The gain in UAS is considerably smaller: The topic modeling expert reaches 90.55% as opposed to 90.26% for the full baseline. In contrast, the topic modeling setting for the 10-topic setting outperforms the random baseline but does not reach the full baseline, thus mirroring the trends we have seen before.

When I compare the experiments where I use the full POS tagging baseline along with topic model parsing experts (row 3 in table 5.7) to the full topic model (row 4), I observe that the latter model reaches only very minimal gains by using the topic modeling POS tagger when I use 2 topics, and there is a negative trend when I use 10 topics. I.e. the overall quality of the POS tagger is more important than its specialization. Thus, even if the topic model POS tagger outperforms its full baseline, the learned adaptations only have a minimal effect on parsing accuracy.

5.4.1 Analysis of results

This could be like this or it could be 2 separate research question - one for POS tagging and one for parsing

It is important to delve deeper into the results to understand where improvement stems from. For POS tagging, the experts outperformed the random split baseline by a greater

Topic	Random split			Topic model		
	% Unknown	Known Acc.	Unknown Acc.	% Unknown	Known Acc.	Unknown Acc.
1	4.79	97.06	82.84	4.29	96.29	85.31
2	4.86	97.25	83.38	3.85	98.35	85.12
avg.	4.83	97.16	83.11	4.07	97.33	85.22

Table 5.10: Unknown word rates and accuracies for known and unknown words in the WSJ+GENIA experiment using 2 topics.

margin. Hence I take a closer look at the differences. I analyze the results for gold POS tags experiments to better understand the improvement for dependency parsing, since it yields most accurate predictions.

5.4.1.1 POS Tagging

In this section, I investigate the differences between the models learned based on a random split as opposed to the models learned based on the topic models. I concentrate on the 2 topic models since this the closest approximation of the mixed domain problem that I am addressing in this chapter.

First, I take a closer look at the distribution of unknown words, and the POS taggers' accuracy on known and unknown words. Unknown words are defined as those words from the test set that do not occur in the training set. This means that the POS tagger needs to guess the word's possible tags without having access to its ambiguity class. The results for this investigation are listed in Table 5.10. These results show that the percentage of unknown words is higher by 0.76 percent points in the random split setting. This means that the two topic models acquire more specialized lexicons that allow the taggers to cover more words. A look at the accuracies shows that, as expected, the accuracy for known words is higher in the topic model setting. However, the results also show that the accuracy on unknown words is significantly higher in this setting, 85.22% for the topic model experts vs. 83.11% for the random splits. This means that the POS tagging models learned from the topic model data split has acquired better models of unknown words based on the word distribution from the training corpora.

Random split				Topic model			
split 1		split 2		GENIA-majority		WSJ-majority	
NN	335	NN	300	NN	387	CD	227
JJ	219	JJ	187	JJ	217	NNP	226
CD	151	CD	162	CD	70	NN	132
NNP	132	NNP	162	NNS	51	JJ	104
NNS	67	NNS	69	NNP	28	NNS	57
VBN	31	VBG	30	FW	13	VBN	32

Table 5.11: The 6 most frequent POS tags assigned to unknown words (2 topics).

Then, I investigate which POS labels are assigned to unknown words in the two settings. The 6 most frequent POS tags per setting and topic are shown in table 5.11. A comparison shows that for the random split, both subsets have a very similar distribution: Unknown words are assigned one of the following labels: noun (NN), adjective (JJ), cardinal number (CD), proper name (NNP), plural noun (NNS), past participle (VBN) or present participle (VBG). The distributions for the topic models show a visibly different picture: In the WSJ-majority topic (topic 1), see table 5.3), cardinal numbers are the most frequent class for unknown words, followed closely by names. These two labels are three times and ten times more frequent than in topic 1. In contrast, GENIA-majority topic (topic 2) is closer to the distribution of the models based on random sampling, but it has a higher number of foreign words (FW), which is an indication that some biomedical terms are not recognized as such and are then marked as foreign words. Examples of such cases are the words “aeruginosa” and “Leishmania”. Overall, these results corroborate our hypothesis that the topic models learn individual characteristics of unknown words.

Finally, I consider the types of errors that the POS taggers make by looking at confusion sets. The 8 most frequent confusion sets under both conditions are shown in table 5.12. A closer look at the confusion sets of the two experiments shows that the categories in the random split setting are consistent with standard errors that POS taggers make: These POS taggers mostly confuse nouns (NN) with adjectives (JJ) and with names (NNP), past tense verbs (VBD) with participles (VBN), prepositions (IN) with adverbs (RB). One notable

Random split			Topic model		
Gold	TnT	No.	Gold	TnT	No.
NN	JJ	141	NN	JJ	122
JJ	NN	111	JJ	NN	104
NNP	NN	93	VBD	VBN	82
VBD	VBN	88	NNP	NNPS	70
NN	NNP	66	RB	IN	64
IN	RB	65	IN	RB	61
RB	IN	62	NN	NNP	53
NNP	NNPS	53	VBG	NN	50

Table 5.12: The 8 most frequent confusion sets (2 topics).

difference in the topic modeling setting is that the number of confusions between nouns (NN) and names (NNP) (in both directions) is almost reduced by half in comparison to the random split setting: 88 vs. 159 cases (note that the condition NN NNP is not among the 8 most frequent cases for the topic model as shown in table 5.12, it is the 12th most frequent confusion set). Names are generally difficult because they constitute an open set, and thus not all of them will be found in the training set. For example, names that were misclassified as nouns in the random split data set included “BART”, “Jefferies”, and “Tulsa”. Thus, a reduction of these errors means that the topic model experts are learning characteristics that allow them to handle domain specific names better, even though the respective learned model files of the topic model setting contain considerably fewer lexical entries.

5.4.1.2 *Dependency Parsing*

Since the goal is to determine the performance of experts, I take a closer look at the results presented for the parsing experiments using gold POS tags in section 5.4.0.1. The results show that the 2-topic parsing experts outperform the general parser trained on the full training set by almost 2 percent points. I looked at the 5 sentences that had the lowest LAS when I used the general parser. These sentences are shown in table 5.8, along with their LAS for both settings. The table clearly shows that the topic expert parsers reach a much higher LAS across all these sentences, and the highest increase reaches 35 percent points.

We also see that there are two headlines among these sentences. They are different in their syntactic patterns from other sentences and thus difficult to parse. For this reason, I decided to have a closer at all “incomplete” sentences, i.e., sentences that do not have verbs, as an approximation of headlines. I found that of the 1 310 sentences in the training set, 437 were grouped into topic 1, the other 873 sentences in topic 2. In the test set, I had 65 such sentences, 15 in topic 1 and 50 in topic 2. For the sentences in topic 1, I calculate an LAS of 76.54, for the ones in topic 2 an LAS of 89.91. These results show that the parser expert for topic 2 has adapted substantially better to the syntax of such untypical sentences than the parser expert for topic 1.

I also looked at the dependency labels that were mislabeled most often by the more general, full baseline parser. The 5 most frequent combinations are shown in table 5.9, with their frequencies in the test sentences of the two topics. These numbers show that the topic 1 expert is much better adapted to these confusion sets, resulting in lower error rates than the topic 2. This shows very dramatically that the two topics learn different patterns.

5.5 Summary

In this chapter, I have presented a flexible and fully automated methodology for POS tagging and parsing for different genres. These experts can be extracted from a heterogeneous text source, without the need of having to separate the genres manually. Additionally, I obtain individual experts, which can be used separately. The results show considerable improvement in POS and parsing results on heterogeneous domains by using unsupervised topic modeling to separate the data into different topics. I can then train POS tagging and parsing experts on the individual topics, which show an increased accuracy in comparison to their counterparts trained on the whole, heterogeneous training set. In theory, I can repeat the experiments for any number of topics but at the cost of reducing training data . This data sparsity resulting from having to split the training set into different topics can be mitigated by assigning every sentence to every topic but weighting their importance to a topic

by the probabilities of the topic modeler. I also showed that while the POS tagger and the dependency parser individually profit from the split into topic experts, the combination of topic expert POS tagger and parser does not improve over using a POS tagger trained on the whole data set.

A deeper analysis of the results show interesting observations. For POS tagging, the analysis shows that a significant improvement is achieved, particularly, for proper names. The topic model experts are almost three times more likely to tag a name correctly than the random split models. The parsing results show that the experts are indeed successful in adapting to certain syntactic aspects than the full training baseline. Further applications for this kind of technology can be found in adapting POS taggers & parsers to characteristics of different speech or cognitive impediments but also to the characteristics of non-native speakers.

In this chapter, I have simplified the problem of assigning sentences to the experts. I.e., I retrain the topic modeler for new test sentences. However, since topic modeling is non-parametric, this could potentially change the topic composition. A better approach is to estimate similarity between a test sentence and the domain experts and then assign the sentence to be tagged/parsed by that training expert. This potentially alleviates the problem posed by retraining the topic experts. I discuss the methods in detail in the next chapter.

CHAPTER 6

AUTOMATED SENTENCE ASSIGNMENT TO THE DOMAIN EXPERTS

6.1 Introduction

In the previous chapter, we observed that, POS tagging and dependency parsing results can be considerably poor for out-of-domain/heterogeneous datasets. This means that a tagger/parser trained on a homogeneous dataset, e.g., Penn Treebank works reliably well on a similar dataset but fails to achieve similar results for heterogeneous datasets. I approached this problem by creating genre/domain experts using topic modeling: I use Latent Dirichlet Allocation (LDA) [Blei et al., 2003, Blei, 2012] for an unsupervised clustering of sentences into topics. The assumption is that these topics correspond to genres. We observed that the topic modeler models the split into genres in a very similar way to the original split, with error rates around 2%. I then train one expert per topic. I.e., I train the expert on all the training sentences that were assigned to the corresponding topic. During testing, I assign test sentences to topics, which means that they are POS tagged and parsed by the corresponding expert. I tested the approach on an artificial, heterogeneous corpus, consisting of a balanced mix of sentences from the WSJ portion of the Penn Treebank (financial news) [Marcus et al., 1994b] and from the GENIA corpus (biomedical abstracts) [Tateisi and Tsujii, 2004]. For POS tagging, there is a moderate increase in performance over a competitive baseline of training on the full training set, and a considerable increase for dependency parsing.

However, in the previous chapter, assigning test sentences to the relevant training topic experts is handled in the simplest possible way: I perform topic modeling on the combination of training and test data due to the non-parametric nature of the algorithm. This means that each time a new test sentence is encountered, the topic modeler is rerun to consistently

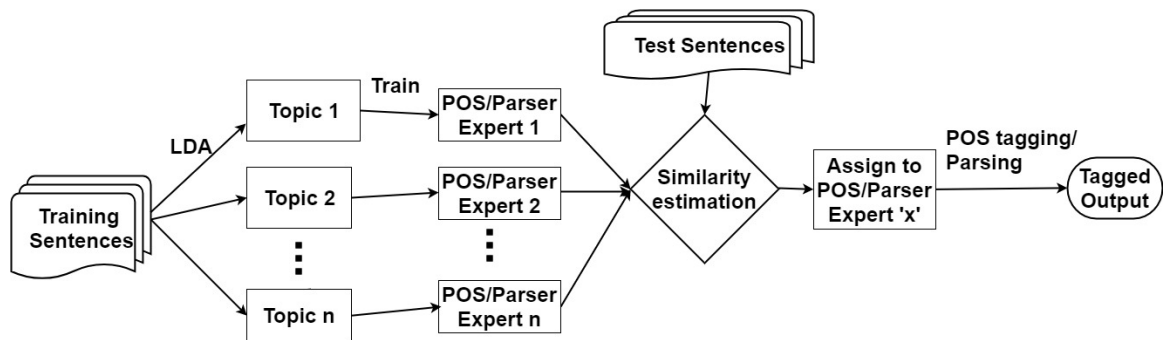


Figure 6.1: Overview of the architecture of the POS tagging and parsing experts.

determine the appropriate genres across training and test sentences. In order to avoid re-training, I propose to use similarity estimation techniques to determine which genre the test sentence belongs to. In this setup, I only create the experts once, and then assign new sentences to genres in an asynchronous fashion. For a new test sentence, I evaluate the similarity of the test sentence to the sentences of a genre and then assign it to the expert with the highest similarity score. I investigate a range of different techniques, based on 1) words closely associated with a topic by the LDA, 2) k -nearest neighbors, or 3) perplexity models to estimate the similarity.

The remainder of this chapter is structured as follows: **TBD**

6.2 Architecture

Do I need to cite knn, dice coeff, ig/gain ratio, perplexity?

The training part of the process remains same as shown in figure 5.1 - LDA [Blei, 2012, Blei et al., 2003] is used to generate genres and train genre experts. In this chapter, I focus on the 2-topics case, which is the closest approximation for the mixed domain adaptation problem. Since in this case, we observed the best performance using hard clustering, I will use this as the base setting for similarity estimation experiments. I use 2 topics as prior, parallel to the 2 domains, and assign each training sentence via hard clustering to the genre for which LDA showed the highest probability. I then test the different similarity metrics: I

compute the similarity of a test sentence to the training sentences of the individual experts and then assign the sentence to the expert for tagging/parsing for which it has the highest similarity. I use techniques such as topic words from LDA, k -nearest neighbors & language model perplexity for estimating similarity. Detailed description of these techniques are given below. I substantiate the use of these metrics as similarity estimation techniques in the next section.

6.2.1 Topic words from LDA

LDA does not only cluster sentences into genres, it also determines which words are highly correlated with each genre. Thus, we can utilize these words along with their probabilities for a specific genre. I sum over all genre words that we find in the sentence, weighted by their probability, and then assign the sentence to the topic that has the highest score. In the next section, I describe the process of using topic words as a similarity metric in detail.

6.2.2 k -nearest neighbors

: There exist a wide range of metrics to calculate similarity between two feature vectors. However, since I compare a sentence to a set of genre sentences, I decided to use memory-based classification using the k -nearest neighbors to classify each test sentence into the relevant class (or genre, in our case). k -nearest neighbor is a supervised, non-parametric lazy learning algorithm. This has the advantage over a pure similarity metric that we have a principled way of handling the comparison to a set of sentences. Additionally, I do not consider the whole search space, as I would if I used a centroid.

The typical steps for the algorithm is given as follows:

- The training data is stored in memory as feature vectors and associated class labels.
- The value of k can be determined on a validation set based on the training errors for different k .

- For each datapoint in test, we do the following:
 - We calculate the distance between the datapoint and each training data vector. Usually there are several metrics such as Euclidean distance, Cosine Similarity, etc.
 - We sort these distances and consider the top k values.
 - We choose the most frequent class from these and return it to be the predicted class for the test datapoint.

TiMBL [Daelemans et al., 1998] enables us to implement variety of parameters to find the most optimum results. Given below is the detailed description of the parameters which I used with k -nearest neighbors.

Dice Coefficient Dice coefficient computes the similarity between two samples. In terms of set theory, this can be shown as:

$$DC(X, Y) = \frac{2|X \cap Y|}{|X| + |Y|} \quad (6.1)$$

For language, dice coefficient can be used to calculate the number of common occurrences of character bigrams in the strings. Thus, given two strings x and y , 6.1 can be rewritten as:

$$DC(x, y) = \frac{2n_{x \cap y}}{n_x + n_y} \quad (6.2)$$

where n_x is the number of character bigrams for string x and n_y is the number of character bigrams for string y . Since we need to measure dissimilarity/distance rather than similarity, our metric is computed as below:

$$d(x, y) = 1 - \frac{2n_{x \cap y}}{n_x + n_y} \quad (6.3)$$

Gain Ratio In the previous paragraph, I discussed Dice Coefficient as the distance metric used in k -NN. However, it is a naive assumption that, all features are equally predictive of class labels. In other words, there are some features which are more informative than others. If we assume all features weigh same, we disregard this concept. Hence it is important to weigh features based on their predictive power.

One of the widely used weighting scheme is Information Gain (IG). It measures how informative a feature is, to determine the class label. It relates to entropy which measures the amount of “randomness” or uncertainty in a probability distribution. Entropy is represented as:

$$H(Y) = - \sum_{y \in Y} p(y) \log p(y) \quad (6.4)$$

where Y is a random variable. In this case, Y can be considered as set of class labels such that $y \in Y$. Now, when we have the information on the value of the feature, we can compute a conditional entropy as below.

$$H(Y|x) = \sum_{x \in X_n} p(x) H(Y|x) \quad (6.5)$$

where X_n is the value of the n -th feature.

Therefore, we can define information gain as difference between the overall entropy and the entropy conditional on the value of the feature.

$$IG(Y, x) = H(Y) - H(Y|x) \quad (6.6)$$

While it is one of the commonly used metric, information gain tends to bias towards features with greater number of values i.e., the multi-valued features. Thus, a feature which is a good indicator of class label but contains lower number of values could potentially not given enough importance in information gain. To counter this effect, gain ratio is introduced, which normalizes information gain using a value referred to as split information

(SI). This measures entropy on the value of the features. Thus SI is given as:

$$SI(n) = - \sum_{x \in X_n} p(x) \log p(x) \quad (6.7)$$

Gain ratio can be represented as following:

$$Gain\ ratio = \frac{IG(Y,x)}{SI(n)} \quad (6.8)$$

6.2.3 Language Model Perplexity

As the third set of methods, I turn to language modeling and use perplexity as a measure to determine the similarity. I explain the concept in details below.

Probabilistic Language Models Language models assign probabilities to the sequence of words in a sentence [Jurafsky and Martin, 2014]. It is essential to compute the most likely word that follows a sequence of words. N-gram is the simplest language model, which is a sequence of n words. We define a unigram, bigram and a trigram are sequences of one, two and three words respectively. If we consider a sentence to be a sequence of consecutive words, $S = w_1, w_2, \dots, w_n$, the primary goal of a language model is to estimate the following:

$$P(S) = P(w_1, w_2, \dots, w_n) \quad (6.9)$$

According to the chain rule of probability, we can calculate this as:

$$\begin{aligned} P(w_1, w_2, \dots, w_n) &= P(w_1)P(w_2|w_1)P(w_3|w_1, w_2) \dots P(w_n|w_1, \dots, w_{n-1}) \\ &= \prod_i P(w_i|w_1^{i-1}) \end{aligned} \quad (6.10)$$

So, in order to find n-th word, we can simply estimate, $P(w_n|w_{n-1}, w_{n-2}, \dots, w_1)$. How-

ever, estimating conditional probabilities for sequences of any length is intractable and it is also nearly impossible to see all the probable sequences in the training data. Thus, it does not generalize well. Hence, we can make the Markov assumption, i.e., the future state depends on the current state only. In terms of language, this translates as:

$$P(w_1, w_2, \dots, w_n) \approx \prod_i P(w_i | w_{i-k} \dots w_{i-1}) \quad (6.11)$$

For a bigram model, we can make the following approximation:

$$P(w_i | w_1, w_2, \dots, w_{i-1}) \approx P(w_i | w_{i-1}) \quad (6.12)$$

We use maximum likely estimates (MLE) to calculate these probabilities. Hence,

$$P(w_i | w_{i-1}) = \frac{\text{Count}(w_{i-1}w_i)}{\text{Count}(w_{i-1})} \quad (6.13)$$

This means that, we calculate bigram probability of a word w_n , given the previous word w_{n-1} by counting the number of times these words appear in conjunction in our corpus, normalized by the number of times w_{n-1} appears in the corpus. Once we know these probabilities, it is trivial to estimate the most likely word that follows a sequence of words.

Evaluation of Language models - Perplexity There is an extrinsic evaluation method for language models, i.e., we use language model and then measure the end to end performance of a system. E.g., if we want to use language models for a machine translation system, we can judge the efficiency of using language model by simply evaluating the MT system. However, I discuss one of the widely used intrinsic evaluation method, perplexity. Perplexity is a function of probability of a sentence - it computes the inverse probability for an unseen test set, normalized by the number of words [Jurafsky and Martin, 2014]. Hence,

lower the perplexity, higher the probability, which implies that the model generalizes well.

$$\begin{aligned}
Perplexity(S) &= P(w_1, w_2, w_3, \dots, w_n)^{-\frac{1}{n}} \\
&= \sqrt[n]{\frac{1}{P(w_1, w_2, w_3, \dots, w_n)}} \\
&= \sqrt[n]{\prod_{i=1}^n \frac{1}{P(w_i | w_1, \dots, w_{i-1})}}
\end{aligned} \tag{6.14}$$

Thus, for bigrams, we can compute perplexity as:

$$Perplexity(S) = \sqrt[n]{\prod_{i=1}^n \frac{1}{P(w_i | w_{i-1})}} \tag{6.15}$$

Smoothing *more?*

Unknown or out of vocabulary (OOV) words are a major problem language model performance. Often a test set has words that never appeared in the training set. Thus the performance might suffer if we assign zero probability to these cases. The idea of smoothing is to “reallocate” probability mass from seen n -grams to unseen ones. There are several smoothing methods. However, since I use additive smoothing, I discuss the methods as below:

- Laplace smoothing: Adds one to the counts of n -grams before estimating the probabilities. This is the simplest smoothing technique.
- Add- k smoothing: Instead of adding one, we add a value k to the counts. The value of k can be determined on a validation set.

6.3 Similarity Estimation Techniques

There are different ways of determining the similarity of a sentence to the sentences in a genre. I investigate methods based on the topic words from LDA, k -nearest neighbor

approaches, and perplexity. Given below is the use of these techniques for similarity estimation for this work.

6.3.1 Topic Words from LDA

In this section, I explore whether we can define a metric from topic modeling that can be used for similarity estimation. LDA provides a list of the words most closely associated with a topic and assigns a weight to each word. So, the words that are highly correlated with each topic can be considered as good indicators for a sentence belonging to the genre represented by the topic. Additionally, I utilize the weights for each word assigned by LDA to determine a word's contribution to the similarity. For this experiment, I select the top 50/100/200 words from each topic. I.e., I assume that these words can be considered to be the most representative words in their respective domain. Then, for each sentence, I check how many of those words occur in the current sentence and add up their weights. So for each test sentence, I compute the weights for each domain experts as follows:

$$W_{t_j} = \sum_{i=1}^N w_{t_j}^i \quad (6.16)$$

W_{t_j} is the overall weight for a test sentence for the j^{th} topic; $w_{t_j}^i$ is the topic weight for the i^{th} top word for the training topic expert; $N = 50/100/200$.

I then assign the sentence to the topic with the higher value. Since I only look at a small number of words, I have to consider the cases where a sentence does not contain any of the topic words. I resolve these cases by extending beyond the top words and considering all the words in the training set for each genre. If there is a tie in values, we assign the sentence randomly to one of the experts.

Feature Vector Size	Number of Nearest Neighbor	Distance Metric	Feature Weighting
50	7	Dice Coefficient	Gain Ratio
100	3	Dice Coefficient	Gain Ratio

Table 6.1: Classification parameters for k -nearest neighbors

6.3.2 k -Nearest Neighbors based similarity metric

In this method, I perform k -nearest neighbor classification to assign test sentences to topic experts.

I create a feature vector by using the top 50/100 words from each genre, as determined by LDA, and assigning the weights as values. The class label is the domain-membership, i.e., either WSJ or GENIA.

The classification parameters are determined by tuning these on a validation set. The settings are shown in table 6.1. It is based on the setting which had least training/validation errors. I explain these metrics in the next paragraphs.

6.3.3 Perplexity-Based Similarity

In the previous section, we learned that perplexity can be a quick way to evaluate language models. I use perplexity as a measure to determine the similarity of a sentence to the training sentences. I have two training experts¹, specializing in both domains in question. I calculate the perplexity of a test sentence to the experts. I then assign this test sentence to the expert which exhibits lower perplexity. I estimate unigram, bigram and trigram perplexity.

¹See previous chapter for detailed explanation

6.4 Experimental Setup

6.4.1 Baselines

In the previous chapter, I have used two baseline cases: The first baseline considers the entire training set and does not employ any topic modeling. Since the topic experts have access to only a fraction of the data, a second and more comparable baseline consists of randomly distributing training and test sentences into sets that correspond in size to the genres. I use these baselines and add a third, which is more relevant to our current setting. In this case, I use the experts trained on the genres but then randomly assign test sentences to the experts. This allows me to gauge how important a correct assignment to the corresponding expert is.

6.4.2 Similarity Estimation

For the k -nearest neighbor estimation, I use the Tilburg Memory-Based Learner (TiMBL) [Daelemans et al., 1998]. For the perplexity models, I derive n -grams of the training experts with additive smoothing using NLTK [Loper and Bird, 2002], and compute the perplexity of the training experts to a test sentence ².

6.4.3 Evaluation

The evaluation is two-fold - intrinsic & extrinsic. As a part of intrinsic evaluation, I compute the accuracy of assigning sentences to its appropriate genre. For extrinsic or end-to-end evaluation, I use the script `tnt-diff` that is part of TnT to evaluate the POS tagging results and the CoNLL shared task evaluation script³ for evaluating the parsing results.

²For Topic Modeling, POS tagging & Dependency Parsing, I use MALLET, TnT & MATE respectively, as done in the previous chapters.

³<http://ilk.uvt.nl/conll/software/eval.pl>

Similarity metric	Setting	Accuracy
joint LDA		98.94
topic words	50	97.59
	100	97.53
	200	97.35
perplexity	unigrams	99.76
	bigrams	84.71
	trigrams	81.53
k -NN	50	90.59
	100	91.18

Table 6.2: Accuracy of genre assignment for different similarity metrics.

6.5 Experimental Results

6.5.1 Genre Assignment

I first investigate how well the different similarity metrics can assign the test sentences to the correct genre. I.e., I calculate accuracy in terms of whether a GENIA sentence is assigned to the GENIA genre, and a WSJ sentence to the WSJ genre. Table 6.2 shows the results of classification accuracy of using different similarity estimation techniques. The reference here is the joint LDA for training and test data, with an accuracy of 98.94%.

Perplexity based on unigrams reaches the highest accuracy, reaching an accuracy of 99.76%, thus surpassing the joint LDA. Surprisingly, using bi- or trigrams instead decreases accuracy by 15-20 points absolute. I assume that this is due to data sparsity since the language model is trained on a relatively small dataset. The second highest accuracy is reached by the methods based on topic words. Here, the number of words considered does not seem to make a significant difference. The k -NN approach performs at around 91%. These results indicate that using single words without context (i.e., the context in bi- and trigrams) provides the most reliable information. The language model has an additional advantage, potentially because it can smooth over unseen words.

Setting	Similarity metric	Accuracy
full training		96.69
random split		96.41
topic experts + random test		91.36
joint LDA		96.95
topic words	50	96.80
	100	96.81
	200	96.81
perplexity	unigrams	96.92
	bigrams	95.64
	trigrams	95.22
k -NN	50	96.05
	100	96.09

Table 6.3: Results for the POS tagging experiments.

6.5.2 POS Tagging

Table 6.3 shows the accuracies of the POS tagging experiments for different similarity metrics. We can see that assigning the test sentences randomly to genres has a detrimental effect, and reaches an accuracy of 91.36%, which is more than 5 points absolute lower than the random split baseline. This difference shows how important it is that sentences are assigned to the correct genre.

Perplexity based on unigrams and the method using topic words reach comparable accuracies to the original topic expert results based on the joint LDA clustering. These results follow the same trends as the genre assignment accuracies, but the differences between the methods are smaller. The perplexity setting using unigrams does not only surpass the joint LDA scores but also all the baselines: by nearly 0.3 percent points for the full training set, by 0.5 percent points for the random split, and by 5 percent points for the random test assignment.

6.5.3 Dependency Parsing

Table 6.4 shows labeled and unlabeled attachment scores of the dependency parses. These results mirror the trends in the POS tagging experiments: Randomly assigning sentences to

Setting	Similarity metric	LAS	UAS
full training		88.67	91.71
random split		87.84	90.86
topic experts + random test		82.17	88.13
joint LDA	-	90.51	92.14
topic words	50	90.30	92.07
	100	90.30	92.07
	200	90.30	92.07
perplexity	unigrams	90.54	92.16
	bigrams	88.33	91.13
	trigrams	87.50	90.68
<i>k</i> -NN	50	89.45	91.82
	100	89.52	91.84

Table 6.4: Attachment scores for the dependency parsing experiments.

genres results in the lowest LAS score of 82.17%, and using perplexity based on unigrams reaches the highest LAS of 90.54%. This LAS is considerably higher than the full training baseline of 88.67%. Note that the differences between the accuracies based on different similarity metrics are considerably more pronounced than in the POS tagging experiments. This mirrors the trend that we have previously seen for the joint LDA assignment. It is also interesting to see that the results for all the topic word settings are the same. This is due to 5 sentences that did not contain any of the topic words and thus had to be randomly assigned to one genre.

I now have a closer look at the two best settings, i.e., the perplexity experiment using unigrams and 50 topic words: I separate the sentences that were assigned to the wrong genre from the correctly assigned ones and evaluate them separately. For the unigram setting, 4 sentences were assigned to the wrong genre, for the 50 topic words, 41 sentences. The results are shown in Table 6.5. These results show that the sentences that were assigned to the wrong genre receive POS and dependency analyses with significantly lower accuracies, the difference to the correct ones ranging between 5 points absolute for POS tagging and 10-14 points for parsing. This corroborates my findings that the correct assignment to a genre is of utmost importance, which also corroborates my conclusion that the genre

	Setting	Correct genre	Incorrect genre	Overall
POS tagging (acc.)	unigram	96.92	91.84	96.92
	topic words 50	98.23	88.59	96.80
parsing (LAS)	unigram	90.54	85.72	90.54
	topic words 50	90.55	76.19	90.30

Table 6.5: Results for POS tagging and dependency parsing when we separate incorrectly assigned sentences from correct ones.

experts model genre-specific information. If they did not, mis-assigning sentences would not have any impact.

6.6 Summary

Using topic modeling to create experts can be very beneficial, but this approach is only viable if we can assign new sentences asynchronously to genres without having to retrain the LDA to determine genres that include the new sentences. I have investigated similarity based methods for assigning the new sentences to genres. More specifically, I have investigated the following methods: 1) using topic words that LDA associates with a genre, 2) k -nearest neighbor models, and 3) perplexity in language models. A baseline that assigns test sentences randomly to genres performs poorly, thus showing that the correct assignment to genres is indispensable.

The results show that the perplexity model based on unigrams surpasses the accuracy of a joint LDA model that assigns the sentences synchronously. For POS tagging, the accuracy of the unigram perplexity model is very close to that of the joint LDA. For parsing, the unigram perplexity model outperforms the joint LDA model. Using the 50 topic words to assign sentences to their genre reaches accuracies close to the best performing model. This shows that word-based methods are more robust in comparison to bigram and trigram methods, which should be able to profit from more context but also face data sparsity issues.

Not sure if this is needed For the future, I plan to investigate models with a more dynamic mix of genres during the POS tagging and parsing process. I.e. rather than creating independent experts, I will investigate methods to create a POS tagger and parser that have

access to expert views during each decision about the next POS tag or parsing step. I will also investigate whether I can integrate gold POS tags or dependency information into the topic modeling process, so that the topic modeler has access not only to the specialized lexical information but also to the linguistic information that it is ultimately tasked to distinguish.

CHAPTER 7

TRANSFORMATION-BASED ERROR-DRIVEN LEARNING IN DEPENDENCY PARSING

7.1 Introduction

Dependency parsing results tend to be reliable as long as the parser is trained and tested on data from a single domain. However, the situation is considerably more challenging when the data set on which the parser is tested/used is different from the sentences on which it is trained. For example, a parser trained on newspaper corpus would not parse texts based on spontaneous dialogues accurately. Since it is impossible to manually annotate the amount of data needed to train a parser in any domain that we need to parse, we need automatic methods to adapt a parser to those new domains. This problem is generally addressed in domain adaptation. Domain adaptation attempts to use a large set of annotated data from a different domain plus specific strategies to adapt the annotations to the target domain, for which a small amount of annotated data may exist. The problem is compounded when there are differences in the annotation schemes between the source and target domain. Different treebanks, representing different domains, tend to use somewhat different annotations Dredze et al. [2007b]. These differences can be due to individual syntactic interpretations of different annotators, but in many cases, they are necessitated by phenomena in the target domain that do not exist in the source domain. Spontaneous dialogues, for example, have a large number of incomplete sentences with words that cannot be attached easily if their head is missing (e.g., the determiner in “I bought the -”).

Out-of-domain dependency parsing results tend to be low as compared to in-domain results. The problem can be looked at from different perspectives. Previous work in this area has extensively looked at the structural errors, where the dependency arcs are corrected.

For instance, the DeSR parser Attardi et al. [2007], Attardi and Ciaramita [2007], Attardi et al. [2009] implements tree revisions. Structural change of dependency trees prove to be an effective domain adaptation method. Yu et al. [2015] report a 1.6% improvement by using self-training to generate training examples for the target domain. To our knowledge, no methods have been reported addressing functional errors, i.e., errors in the dependency labels.

Our work focuses on a target domain of spontaneous dialogues, using the CReST Corpus [Eberhard et al., 2010], which uses labels that do not occur in the source domain corpus. We propose to use transformation based error driven learning (TBL) Brill [1995] to address the problem. The method has been proven effective in a variety of NLP problems including POS tagging and syntactic parsing. The idea is simple: We use a small annotated data set in the target domain and automatically annotate it using a source domain parser. Our goal to learn a set of rules from the errors that occur and a set of rule templates. Although we focus on correcting dependency labels in the current paper, our method can be extended to correct the errors in dependency arcs as well. We demonstrate that using TBL with a small target domain training set improves dependency parsing accuracy by about 10 % absolute, and it learns to use the target-specific labels.

The paper is structured as follows: Section 7.2 discusses related work, section 7.3 describes the issues in domain adaptation in more detail, and section 7.4 explains our approach of using TBL for domain adaptation. In section 7.5, we describe our experiment setup, and section 7.6 discusses the results. Section 7.7 concludes and delineates future work.

7.2 Related Work

There has been a significant amount of work done on domain adaptation in dependency parsing. The primary challenge of domain adaptation is the unavailability of annotated examples from the target domain. Previous work in this area focused on analyzing how this

affects parsing quality and on building systems to bypass the need for having a large amount of target domain data. Although distributional difference between the domains is a common problem, in general, Dredze et al. [2007b] conclude that domain adaptation is most challenging when there are dissimilarities in annotation schemes between the domains.

Domain adaptation for dependency parsing was one of the tasks in the CoNLL 2007 Shared Task. The shared task was focused on the scenario where there is no data available from the target domain. Out of the 10 systems participating in the domain adaptation task, the highest results (81.06% LAS; 83.42% UAS) are achieved by Sagae and Tsujii [2007]. To add target domain sentences to the training set, they emulate a single iteration of co-training by using MaxEnt and SVMs, selecting the sentences where both models agreed. The system by Attardi and Ciaramita [2007] (80.40% LAS; 83.08% UAS) produced similar results. They use a tree revision method Attardi and Ciaramita [2007] to correct structural mistakes. They formulate the problem as a supervised classification task using a multiclass perceptron, where the set of revision rules is the output space and features are based on syntactic and morphological properties of the dependency tree.

A popular approach for domain adaptation is selecting appropriate training data by using self-training or co-training for domain adaptation. Although testing on a single domain, McClosky et al. [2006a,b] show the effectiveness of using reranking using self training. They achieved an improvement of around 1% on unlabeled data. Kawahara and Uchimoto [2008] devised a method to select reliable parses from the output of a single dependency parser, MST parser McDonald et al. [2005], instead of using an ensemble of parsers for domain adaptation. They use a self training method and combine labeled data from target domain with unlabeled data from the source by “concatenation”. To estimate whether a parse is reliable, they applied binary classification using SVM on features such as sentence length, dependency length, unknown words, punctuation, average frequency of words in a sentence. They report a 1% increase in accuracy over the contemporary state of the art CoNLL shared task results on the shared task data. Yu et al. [2015] applied self-training for

domain adaptation using confidence scores to select appropriate parse trees. Their highest improvement in terms of LAS is 1.6% on the CoNLL data.

blitzer:mcdonald:ea:06 used structural correspondence learning (SCL) for POS tagging and parsing to find “frequently occurring” pivot features, i.e., features that occur frequently in unlabeled data and equally characterize source and target domains. They used the WSJ as the source and MEDLINE abstracts as the target domain. They established that SCL reaches better results in both POS tagging and parsing than supervised and semi-supervised learning even when there is no training data available on the target domain.

Transformation based error driven learning (TBL) was originally developed for POS tagging Brill [1992, 1995]. Brill [1993] also used this approach to parse text by learning a “transformational” grammar. The algorithm repeatedly compares the bracketed structure of the syntactic tree to the gold standard structure and learns the required transformations in the process. Brill and Resnik [1994] also use this technique for prepositional phrase attachment disambiguation. Transformation based error driven learning has also been used for information retrieval by Woodley and Geva [2005].

7.3 Issues in Domain Adaptation

Before we present our approach to domain adaptation, we need to better understand the different facets of the domain adaptation problem. This analysis will motivate our solution, which can handle all of those cases.

7.3.1 Error Types

The first phenomenon that we look at is the types of errors that can occur in dependency parsing:

Predicting the **head of a dependency arc** inaccurately, resulting in a structural error.

Predicting the **label of a dependency arc** wrong, resulting in a functional error.

	Cases	# instances
1	Incorrect dependency label (incl. correct & incorrect heads)	13 839
2	Incorrect dependency head (incl. correct & incorrect label)	10 294
3	Incorrect dependency label & correct dependency head	5 389
4	Incorrect dependency label & incorrect dependency Head	8 450

Table 7.1: Analysis of sentences from CReST containing incorrect dependency predictions.

Predicting both **label and the head of a dependency arc** wrong, resulting in structural and functional problems.

Note that structural errors affect the labeled and unlabeled attachment scores, functional errors only affect labeled attachment scores.

Previous work focused on correcting structural errors introduced by inaccurate prediction of the head of a dependency arc. To our knowledge, there are no approaches to address these functional errors.

Next, we need to determine how serious these problems are. We concentrate on a setting where the source domain is the WSJ part of the Penn Treebank Marcus et al. [1994a] and the target domain is the CReST Corpus Eberhard et al. [2010] (for more details on the corpora, see section 7.5). We now take a closer look at a parsing experiment where we use 17 181 WSJ sentences for training the MATE parser and 5 759 CReST sentences for testing. The different types of errors are shown in table 7.1. This analysis shows that functional errors, i.e., errors involving dependency labels, are more frequent than structural errors. Thus, by ignoring those errors in domain adaptation, we artificially limit the possible improvements.

7.3.2 Error Sources

A second type of difference concerns the source of the parser errors. Here we need to distinguish two cases:

Parser inherent errors, i.e., errors that the parser makes independent of the out-of-domain setting.

Parser errors caused by differences in distribution between the two domains.

Treebank	No. of dep. labels	Intersection with WSJ
WSJ	67	-
CReST	31	22
Brown	71	60

Table 7.2: Differences in the dependency label sets across treebanks.

Differences in annotation, i.e., the domain data on which we evaluate differ from the annotations in the source data.

In the following, we focus on the differences of type 2 and 3. More specifically, we work on errors in the dependency labels since these are easier to distinguish. Structural differences tend to be differences of degree, which are difficult to separate into distributional differences (type 2) and annotation differences (type 3). In order to establish whether annotation differences cause issues, We have analyzed a range of treebanks that are annotated using (variants of) the Penn Treebank label set.

Table 7.2 looks at differences in the dependency label sets in the following treebanks: the WSJ part of the Penn Treebank, the Brown Corpus Francis and Kucera [1979] (Sections cg, ck, cl, cm, cn, cp, cr), and the CReST Corpus Eberhard et al. [2010]. The CoNLL style dependency trees of the WSJ and the Brown Corpus are created using the pennconverter tool Johansson and Nugues [2007]; CReST was natively annotated in dependencies as well as constituents. It is obvious that there are considerable differences in the annotation schemes, especially between WSJ and CReST, which only have 22 labels in common. We can establish three different scenarios:

The source is a superset of the target data set.

The source is a subset of the target data set,

Both the source and target annotations have labels that do not occur in the other data set.

In the first case, the distribution of labels may be different between source and target, which can cause incorrect parsing decisions. The problem in the second case is more difficult since the parser is supposed to predict labels that it has not seen in training. The

third case is a combination of the first two and thus the most difficult to handle.

Returning to our treebanks, we assume that since WSJ has the largest number of manually annotated sentences, it will serve as the source domain. In this case, CReST mainly shows a subset of labels from WSJ, but also has 9 labels that do not occur in WSJ.

Our hypothesis is that we can address *all* of the problems described above by using transformation based error driven learning Brill [1995] (for a description of this method see section 7.4). The method has been proven effective in many tasks such as POS tagging, syntactic parsing, and machine translation.

For the work presented here, we will focus on cases where the parser has predicted an incorrect label. This is motivated by our findings in table 7.1, which show that label errors are more frequent than structural ones. We investigate the scenario using CreST, where the source has a superset of labels in the target annotations, but with some unique labels in the target annotations. We hypothesize that our method can lead to an improved performance of the parser on the target domain, including dependency labels that do not exist in the source treebank. We evaluate this by the Labeled Attachment Scores (LAS) and Label Accuracy (LA).

7.4 Transformation-Based Error-Driven Learning for Domain Adaptation

7.4.1 TBL: The Brill Tagger

We implement the idea of transformation based error driven learning, which was originally developed by Brill [1992] for POS tagging. The method works with two versions of the training data: the gold standard and (an initially unannotated) working copy that simulates the learning process and records the errors. The unannotated version of the training data is tagged by a simple part of speech tagger¹, to create the initial state of the working copy of the text. The goal is to bridge the difference between the gold standard text and the working

¹This could be any simple part of speech tagger, or a heuristic that labels every word with the most frequent POS tag

copy by learning rules that change the incorrect POS tags to the correct ones.

Learning is based on rule templates, which consist of two parts: rewrite rules and triggering environment. The templates need to be determined by the researcher, based on the problem. In general, a rule template is of the form:

Rule Template $(A, B) : X \rightarrow Y$

I.e., if we observe conditions A and B (triggering environment), we change the output variable (POS tags, for Brill and dependency labels, for us) from X to Y (rewrite rule). Note that X can remain unspecified, then the rule is applied independent of the current variable (label or POS tag).

The following is an example of a rule template for POS tagging: Change the POS tag of the current word from X to Y if the previous word is tagged as Z , where X , Y , and Z are variables that need to be instantiated during learning.

The learner creates rule hypotheses out of those templates by identifying incorrectly tagged words in the working copy and instantiating the template variables. For example, one possible rule hypothesis could be the following: Change the POS tag of the word “impact” from verb to noun if the previous word is tagged as a determiner.

In one pass through the system, all possible rule hypotheses are created and ranked based on an objective function which determines for each hypothesis how many corrections occur. The rule hypothesis with the highest score is applied to the working copy and added to the final list of transformations, stored in order, during the training process. Then the process repeats, and new rule hypotheses are generated from the templates based on the remaining errors in the working copy. The process finishes when there is no more improvement in terms of reduction of errors.

7.4.2 Domain Adaptation Using TBL

We use the underlying concept of transformation based error driven learning for reducing the dependency label errors resulting from parsing across domains. In particular, we ad-

dress the scenario where the target domain contains a subset of the labels in the source annotation, but also contains new labels.

To use TBL for domain adaptation in dependency parsing, we need to adapt the following parts: In our case, the initial state consists of the training set from the target corpus parsed using a dependency parser, which was trained on the treebank from the source domain. We need to choose rule templates describing the relevant information that can help decide in which case to change a dependency label. We use POS and lemma² information rather than word forms in order to balance generality and specificity in the transformation rules. We utilize information on head of a word and its dependents since these provide essential contextual information. We currently do not use dependency labels as variables in our rule templates, but using this type of information is a viable modification in the future.

We use the following rule templates to describe the contextual properties of situations when a dependency label needs to be changed. Note that it is straightforward to include more templates, and in doing so, we can also model cases of structural change. In the latter case, the replacement part would take the form of "replace the current head of the word by word X". This will need to be accompanied by a check to ensure that the resulting dependency graph is still valid.

For this paper, we examine the following four rule templates.

- Rule template 1 (RT1)- $[(P_S, P_P, P_D, L_S, L_P, L_D) \rightarrow D_1]$

If the part of speech tags of the word, its parent, and its dependent(s) are P_S , P_P and P_D , and the lemma of the word, its parent, and its dependent(s) are L_P , L_S , L_D respectively, the dependency label should be D_1 . We consider all the dependents of a word as a single variable. E.g., $[(NN, VBZ, [DT, JJ, CC], box, be, [a, pink, and])] \rightarrow PRD]$

- Rule template 2 (RT2)- $[(P_S, P_P, L_S, L_P) \rightarrow D_1]$

If the part of speech tags of the word and its parent are P_S and P_P , and the lemma of

²We derive lemma information using the TreeTagger Schmid [1995]

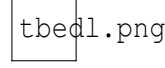


Figure 7.1: Reducing dependency label errors via transformation-based error-driven learning

the word and its parent are L_S and L_P respectively, the dependency label should be D_1 .

- Rule template 3 (RT3) - $[(P_S, P_P, P_D) \rightarrow D_1]$

If the part of speech tags of the word, its parent and its dependent(s) are P_S , P_P and P_D respectively, the dependency label should be D_1 .

- Rule template 4 (RT4) - $[(P_S, P_P) \rightarrow D_1]$

If the part of speech tags of the word and its parent are P_S and P_P respectively, the dependency label should be D_1 .

Figure 7.1 shows the TBL approach through four iterations of creating rule hypotheses from the templates, selecting and applying the best one.

We outline the process in algorithm 1. The learning iterates until there are no more rule hypotheses that would cause a further reduction in errors. During each iteration, based on the templates, the algorithm creates rule hypotheses from the errors found by comparing the working copy to the gold standard. It then chooses the best rule hypothesis that maximally reduces the errors in the working copy and applies this rule to the working copy. Since the templates that we use are independent of each other, we currently apply the top 10 rules per iteration. Choosing to apply multiple rules speeds up the overall process. To prevent over-generation, we favor the more specific hypothesis if there is a tie in the scores. We then store these rules in the final list of learned rules. After the application of the rules chosen in each iteration, we use the updated work copy in the next iteration.

After learning, a new text is processed by first having it parsed by the source domain

Algorithm 1 TBL for Domain Adaptation

```
1: procedure TBL_learn(training_data)
2:   work_copy = extract text from training_data; parse using WSJ model
3:   learned_rules  $\leftarrow$  []
4:   repeat Generate rule hypothesis from errors in corpus given rule templates
5:     rule_hypotheses = generate_hypotheses(training_data, work_copy) calculate
      improvement score for each rule: # errors fixed
6:     rule_scores = determine_scores(rule_hypotheses, work_copy) Rank rules
      given scores; select rule with highest score
7:     best_rule, best_score = argmax(rank_rules(rule_hypotheses, rule_scores))
      If best rule causes improvement, apply rule to work copy
8:     if best_score > 0 then
9:       work_copy = apply_rule(work_copy, best_rule)
10:    learned_rules += best_rule
11:  until best_score <= 0
12:  return learned_rules

14: procedure TBL_apply(learned_rules, test)
15:   test_annotated = parse test data using WSJ model Apply learned rules in order
      to test set
16:   for each rule in learned_rules do
17:     test_annotated = apply_rule(test_annotated, rule)
```

dependency parser and then applying the learned rules in order.

7.5 Experimental Setting

7.5.1 Data Sets

In our current work, we focus on two domains: financial news articles and dialogues in a collaborative task. We use the Wall Street Journal Marcus et al. [1994a] and the CReST corpus Eberhard et al. [2010] as representative corpora for these two domains.

These corpora are very dissimilar in nature. On average, the length of the sentences for WSJ is 24 and 7 for CReST. In terms of linguistic factors, CReST has a subset of WSJ's dependency labels with 10 new labels added. Example sentences from each corpora are shown in table 7.3. We use WSJ as the source domain and CReST as the target domain.

WSJ	In an Oct. 19 review of " The Misanthrope " at Chicago 's Goodman Theatre (" Revitalized Classics Take the Stage in Windy City , " Leisure & Arts) , the role of Celimene , played by Kim Cattrall , was mistakenly attributed to Christina Haag .
CReST	it's like as soon - like if I were to close the door it's right next to like the bottom of the floor like where the door closes

Table 7.3: Sample sentences from Wall Street Journal (WSJ) & CReST corpus

	# Dialogues	# Sentences	
		with errors	without errors
Training Set	19	4384	459
Test Set	4	831	85

Table 7.4: Target domain training and test set for TBL

7.5.1.0.1 Target Domain The CReST corpus consists of 23 dialogues that were manually annotated for dependencies. We randomly select 19 dialogues as our training data and the rest as test. since the system needs to learn rule hypotheses from incorrect predictions of the source domain parser, we can safely ignore sentences that were parser completely correctly. For the test data, we use all the sentences from the designated dialogues (with correct and incorrect dependency label predictions). Table 7.4 shows the division of sentences for training and test.

7.5.2 TBL Components

7.5.2.0.1 Initial State System We use the MATE parser Bohnet [2010b] as the initial state annotator. We use 17 181 sentences from WSJ to train the MATE parser. We use this model to parse the sentences from the CReST training set to obtain the initial state annotated text.

7.5.2.0.2 Baseline We benchmark our results against the performance of the MATE parser trained on WSJ sentences, i.e., the out-of-domain parser, without any modification. We also report results for an in-domain setting where the MATE parser is trained on the CReST training set.

	Setting	# Incorrect	LAS	LA	EM
1	Baseline (trained on WSJ)	2 112	59.02	63.73	8.41
2	In-domain baseline (trained on CReST)	572	87.39	90.18	67.58
3	Rule Templates (1+2)	600	69.91	89.70	48.03
4	Rule Templates (1+2+3+4)	451	70.10	92.25	48.80

Table 7.5: Results of applying TBL rules on test set.

7.5.3 Evaluation

For evaluation, we use the CoNLL-X script. We report the Labeled Attachment Scores (LAS) and Label Accuracy (LA) as the primary metrics for evaluating our system. LAS measures the percentage of words which have the correct head and dependency label. LA measures the number of correctly assigned labels. Reporting LA is important since this measure focuses solely on the accuracy of the labels while LAS also evaluates structural issues in the parse, which we do not address currently. We also report the exact syntactic match (EM), which measures the percentage of sentences that have correct overall predicted dependency trees.

7.6 Experimental Results

In this section, we discuss the outcome of our experiments for the TBL process. The results of applying the rules learned from TBL are given in table 7.5.

The first setting is our baseline. We evaluate the performance of our system against the results of parsing the sentences from the test set with the model trained on WSJ. The second baseline consists of an in-domain parser trained on the small CReST training set. Settings 3 and 4 evaluate the performance of the learned rules. The third setting derives rule hypotheses instantiated by 2 rule templates (rule templates 1 & 2). Since rule templates 1 & 2 are more specific (it accounts for lemma as well as part of speech tags), we add more general templates such as POS tags specifically (rule templates 3 & 4) to investigate if that leads to a significant difference in the results.

From table 7.5, we observe that applying TBL has a significant effect on the quality

Label	# in Gold	% Correct		
		baseline	2 templates	4 templates
INTJ	690	0	98.26	99.71
ROOT*	195	0	63.59	77.44

Table 7.6: Accuracy in predicting CReST-specific labels.

of the predicted dependency labels, with an improvement in LAS of almost 20% absolute over the out-of-domain parses. The number of completely correct sentences shows an improvement of 40% absolute over this baseline. We also see that extending the set of templates to include less-specific templates (templates 3 and 4) using POS tags information, has a minimal effect on LAS but increases label accuracy (LA) by 2.5% absolute.

When comparing the TBL results to the in-domain baseline, we notice that the baseline reaches an LAS considerably higher than the TBL results, but the TBL approach using all 4 templates reaches higher results with regard to LA (around 2% improvement). The different trends can be explained by the fact that we are currently restricting the TBL approach to functional changes, and we ignore structural errors. Since LAS evaluates both aspects, we see that the in-domain parser fares better in this respect. However, the LA, which only evaluates dependency labels, shows that our method is successful in improving the labels. Since the results of the experiment with 2 templates are similar to the in-domain treebank, we assume that we need to experiment with more specific templates in the future.

There are 10 dependency labels in CReST which do not occur in WSJ. Out of these, 2 labels (ROOT*, INTJ) occur in the test data. ROOT* is used for words whose head is missing because the sentence was interrupted; and INTJ is used for interjections. Since WSJ does not use these labels, these are predicted incorrectly in all the cases in the out-of-domain baseline.

We investigate the number of times these labels have been corrected by TBL. Table 7.6 shows the results. As expected, the labels do not show up in the baseline setting. For the settings using 2 or 4 templates, there is a significant improvement for the labels INTJ and ROOT*.

Rule Template	Lemma (self)	Pos (self)	Lemma (head)	POS (head)	Lemma (children)	POS (children)	Deprel	Improvement
RT2	okay	UH	(root word)	(root word)	-	-	INTJ	1292
RT2	um	UH	(root word)	(root word)	-	-	INTJ	386
RT2	be	VBZ	(root word)	(root word)	-	-	ROOT	384
RT2	alright	UH	(root word)	(root word)	-	-	INTJ	216
RT2	yeah	UH	(root word)	(root word)	-	-	INTJ	205
RT2	the	DT	(root word)	(root word)	-	-	ROOT*	131
RT3	and	CC	be	VBZ	(no children)	(no children)	COORD	117
RT2	go	VBI	(root word)	(root word)	-	-	ROOT	100
RT3	that	DDT	be	VBZ	(no children)	(no children)	SBJ	83
RT2	well	UH	(root word)	(root word)	-	-	INTJ	77

Table 7.7: Top learned rules for the setting using 2 templates.

Rule Template	Lemma (self)	POS (self)	Lemma (head)	POS (head)	Lemma (children)	POS (children)	Deprel	Improvement
RT4	-	UH	-	(root word)	-	-	INTJ	2886
RT5	-	UH	-	(root word)	-	(no children)	INTJ	2766
RT2	okay	UH	(root word)	(root word)	-	-	INTJ	1292
RT3	okay	UH	(root word)	(root word)	(no children)	(no children)	INTJ	1271
RT4	-	AP	-	(root word)	-	-	INTJ	453
RT5	-	AP	-	(root word)	-	(no children)	INTJ	438
RT2	um	UH	(root word)	(root word)	-	-	INTJ	386
RT2	be	VBZ	(root word)	(root word)	-	-	ROOT	384
RT3	um	UH	(root word)	(root word)	(no children)	(no children)	INTJ	383
RT5		XY		(root word)		(no children)	ROOT*	268

Table 7.8: Top learned rules for the setting using 4 templates.

Table 7.7 and 7.8 show the 10 learned rules with the highest scores, for the setting with 2 or 4 templates respectively. We can observe that many of the rules concern the labels INTJ and ROOT*. Since these labels do not appear in the source domain, applying rule hypotheses specific to these labels leads to the highest gains.

To have a closer look, we extracted the most frequent label confusions for each setting after parsing and TBL domain adaptation. The results are shown in table 7.9. For the baseline, the most frequently incorrect label is INTJ, followed by ROOT and ROOT*. By applying TBL, we have countered the most frequent label confusion as evident from the settings using 2 or 4 templates, where the numbers are lower across the board, but also the types of confusion are more typical of standard parsing issues (e.g., subject (SBJ) vs. predicate (PRD)).

Baseline			2 Templates			4 Templates		
Gold	Predicted	Counts	Gold	Predicted	Counts	Gold	Predicted	Counts
INTJ	ROOT	354	ROOT*	ROOT	39	ROOT*	ROOT	38
INTJ	DEP	210	SBJ	PRD	20	SBJ	PRD	25
ROOT	NMOD	136	ROOT*	NMOD	20	ROOT	ROOT*	21
ROOT*	NMOD	100	LOC	NMOD	19	DIR	LOC	19
INTJ	NMOD	85	ROOT	ROOT*	17	DIR	ADV	17
LOC	NMOD	55	DIR	ADV	15	LOC	NMOD	15
COORD	DEP	38	ROOT	NMOD	13	TMP	ADV	10
ROOT	COORD	37	DIR	NMOD	12	LOC	ADV	10
LOC	LOC-PRD	35	LOC	ADV	11	ROOT	NMOD	8
LOC	PRD	33	PMOD	NMOD	10	OBJ	SBJ	8

Table 7.9: The 10 most frequent label confusions across the different settings.

7.7 Discussion

In this paper, we introduce transformation-based error-driven learning (TBL) for domain adaptation in dependency parsing. Since there tend to be significant differences between annotation schemes of different corpora from different domains, we focus on methods that can address those differences systematically along with differences due to the domain itself. When a text is parsed with a model trained on one domain, it leads to a significant number of incorrect predictions, especially for dependency labels. We address this problem in our work by using TBL to learn rules from a small target domain training set and a small set of manually defined rule templates. In comparison to a pure out-of-domain parser, we observe a significant increase in the labeled attachment scores (~20%), labeled accuracy (~30%) and exact match (~40%) for WSJ as source and CReST as target corpus. The observed improvement is largely due to the labels that are used in CReST, but do not occur in WSJ since those are mislabeled consistently by the out-of-domain parser. However, when we apply TBL, these labels are corrected in most of the cases. When we compare our results to an in-domain parser, the results show that the TBL approach is very good at correcting the dependency labels, but we need to extend the approach and use more specific templates and cover structural errors as well to be able to compete with the in-domain parser with regard to LAS.

Our method is flexible in that any number of variables can be used in the templates, and it can be extended to cover structural changes, i.e., to correct dependency head annotations. In addition, the templates are valid across different target domains.

As a part of our future work, we plan to evaluate the effectiveness of this method for multiple domains. We will also introduce new variables in our rule templates including contextual information on dependency labels. In addition to correcting dependency labels, we will extend the method to correct predicted dependency heads by introducing new templates. This will need to be accompanied by a check to ensure that the resulting analysis is still a valid dependency graph.

CHAPTER 8
CONCLUSION

Appendices

APPENDIX A

PLACEHOLDER

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

APPENDIX B

PLACEHOLDER

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

BIBLIOGRAPHY

- Giuseppe Attardi and Massimiliano Ciaramita. Tree revision learning for dependency parsing. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference*, pages 388–395, 2007.
- Giuseppe Attardi, Felice Dell’Orletta, Maria Simi, Atanas Chanev, and Massimiliano Ciaramita. Multilingual dependency parsing and domain adaptation using DeSR. In *EMNLP-CoNLL*, pages 1112–1118, 2007.
- Giuseppe Attardi, Felice Dell’Orletta, Maria Simi, and Joseph Turian. Accurate dependency parsing with a stacked multilayer perceptron. *Proceedings of EVALITA*, 9:1–8, 2009.
- David M. Blei. Probabilistic topic models. *Communications of the ACM*, 55(4):77–84, 2012. ISSN 0001-0782. doi: 10.1145/2133806.2133826. URL <http://doi.acm.org/10.1145/2133806.2133826>.
- David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003. ISSN 1532-4435. URL <http://dl.acm.org/citation.cfm?id=944919.944937>.
- Bernd Bohnet. Very high accuracy and fast dependency parsing is not a contradiction. In *Proceedings of the 23rd international conference on computational linguistics*, pages 89–97, 2010a.
- Bernd Bohnet. Top accuracy and fast dependency parsing is not a contradiction. In *Proceedings of the 23rd International Conference on Computational Linguistics (COLING)*,

- pages 89–97, Beijing, China, 2010b. URL <http://www.aclweb.org/anthology/C10-1011>.
- Thorsten Brants. TnT—a statistical part-of-speech tagger. In *Proceedings of the 1st Conference of the North American Chapter of the Association for Computational Linguistics and the 6th Conference on Applied Natural Language Processing (ANLP/NAACL)*, pages 224–231, Seattle, WA, 2000.
- Eric Brill. A simple rule-based part of speech tagger. In *Proceedings of the DARPA Speech and Natural Language Workshop*, pages 112–116, Harriman, NY, 1992.
- Eric Brill. Automatic grammar induction and parsing free text: A transformation-based approach. In *Proceedings of the 31st Annual Meeting of the Association for Computational Linguistics*, pages 259–265, 1993.
- Eric Brill. Transformation-based error-driven learning and natural language processing: A case study in part of speech tagging. *Computational Linguistics*, 24(1):543–565, 1995.
- Eric Brill and Philip Resnik. A rule-based approach to prepositional phrase attachment disambiguation. In *Proceedings of the 15th conference on Computational linguistics-Volume 2*, pages 1198–1204, 1994.
- Walter Daelemans, Jakub Zavrel, Ko Van der Sloot, and Antal Van den Bosch. Timbl: Tilburg memory-based learner-version 1.0-reference guide. 1998.
- Hal Daume III. Frustratingly easy domain adaptation. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 256–263, Prague, Czech Republic, 2007.
- Mark Dredze, John Blitzer, Partha Pratim Talukdar, Kuzman Ganchev, João Graca, and Fernando Pereira. Frustratingly hard domain adaptation for dependency parsing. In

- Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, pages 1051–1055, Prague, Czech Republic, 2007a.
- Mark Dredze, John Blitzer, Partha Pratim Talukdar, Kuzman Ganchev, Joao Graca, and Fernando CN Pereira. Frustratingly hard domain adaptation for dependency parsing. In *EMNLP-CoNLL*, pages 1051–1055, 2007b.
- Kathleen Eberhard, Hannele Nicholson, Sandra Kübler, Susan Gunderson, and Matthias Scheutz. The Indiana "Cooperative Remote Search Task" (CReST) Corpus. In *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC)*, Valetta, Malta, 2010.
- W. Nelson Francis and Henry Kucera. Brown corpus manual. Brown University, 1979.
- Richard Johansson and Pierre Nugues. Extended constituent-to-dependency conversion for English. In *Proceedings of NODALIDA 2007*, pages 105–112, Tartu, Estonia, May 25-26 2007.
- Dan Jurafsky and James H Martin. *Speech and language processing*, volume 3. Pearson London, 2014.
- Daisuke Kawahara and Kiyotaka Uchimoto. Learning reliability of parses for domain adaptation of dependency parsing. In *Proceedings of the Third International Joint Conference on Natural Language Processing (IJCNLP)*, Hyderabad, India, 2008.
- Edward Loper and Steven Bird. Nltk: the natural language toolkit. *arXiv preprint cs/0205028*, 2002.
- Mitchell Marcus, Grace Kim, Mary Ann Marcinkiewicz, Robert MacIntyre, Ann Bies, Mark Ferguson, Karen Katz, and Britta Schasberger. The Penn Treebank: Annotating predicate argument structure. In *Proceedings of the Workshop on Human Language*

- Technology*, HLT '94, pages 114–119, Plainsboro, NJ, 1994a. ISBN 1-55860-357-3. doi: 10.3115/1075812.1075835. URL <http://dx.doi.org/10.3115/1075812.1075835>.
- Mitchell Marcus, Grace Kim, Mary Ann Marcinkiewicz, Robert MacIntyre, Ann Bies, Mark Ferguson, Karen Katz, and Britta Schasberger. The Penn Treebank: Annotating predicate argument structure. In *Proceedings of the ARPA Human Language Technology Workshop, HLT 94*, pages 114–119, Plainsboro, NJ, 1994b.
- Andrew Kachites McCallum. Mallet: A machine learning for language toolkit. <http://mallet.cs.umass.edu>, 2002.
- David McClosky, Eugene Charniak, and Mark Johnson. Effective self-training for parsing. In *Proceedings of the Main Conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics, HLT-NAACL '06*, pages 152–159, New York, New York, 2006a. doi: 10.3115/1220835.1220855. URL <https://doi.org/10.3115/1220835.1220855>.
- David McClosky, Eugene Charniak, and Mark Johnson. Reranking and self-training for parser adaptation. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics, ACL-44*, pages 337–344, Sydney, Australia, 2006b. doi: 10.3115/1220175.1220218. URL <https://doi.org/10.3115/1220175.1220218>.
- Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajič. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 523–530, 2005.
- Tomoko Ohta, Yuka Tateisi, and Jin-Dong Kim. The GENIA corpus: An annotated research abstract corpus in molecular biology domain. In *Proceedings of the Second International*

- Conference on Human Language Technology Research*, pages 82–86, San Francisco, CA, 2002. URL <http://dl.acm.org/citation.cfm?id=1289189.1289260>.
- Barbara Plank and Gertjan van Noord. Effective measures of domain similarity for parsing. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 1566–1576, Portland, OR, 2011.
- Kenji Sagae and Jun’ichi Tsujii. Dependency parsing and domain adaptation with LR models and parser ensembles. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, pages 1044–1050, Prague, Czech Republic, 2007.
- Beatrice Santorini. Part-of-speech tagging guidelines for the Penn Treebank Project. Department of Computer and Information Science, University of Pennsylvania, 3rd Revision, 2nd Printing, 1990. URL <ftp://ftp.cis.upenn.edu/pub/treebank/doc/tagguide.ps.gz>.
- Helmut Schmid. Treetagger – a language independent part-of-speech tagger. *Institut für Maschinelle Sprachverarbeitung, Universität Stuttgart*, 43:28, 1995.
- Yuka Tateisi and Jun’ichi Tsujii. Part-of-speech annotation of biology research abstracts. In *Proceedings of 4th International Conference on Language Resource and Evaluation (LREC)*, Lisbon, Portugal, 2004.
- Alan Woodley and Shlomo Geva. Applying transformation-based error-driven learning to structured natural language queries. In *International Conference on Cyberworlds*, pages 8–pp, 2005.
- Juntao Yu, Mohab Elkaref, and Bernd Bohnet. Domain adaptation for dependency parsing via self-training. In *Proceedings of the 14th International Conference on Parsing Technologies*, pages 1–10, 2015.