ENSEMBLES OF DIVERSE CLUSTERING-BASED DISCRIMINATIVE DEPENDENCY PARSERS

by

Marzieh Razavi
B.Sc., Sharif University of Technology, 2009

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
in the School
of
Computing Science
Faculty of Applied Sciences

SIMON FRASER UNIVERSITY
Summer 2012

APPROVAL

Name: Marzieh Razavi

Degree: Master of Science

Title of thesis: Ensembles of Diverse Clustering-Based Discriminative Dependency Parsers

Examining Committee: Dr. Richard Vaughan
Chair

Dr. Anoop Sarkar, Senior Supervisor
Associate Professor, School of Computing Science
Simon Fraser University

Dr. Fred Popowich, Supervisor,
Professor, School of Computing Science
Simon Fraser University

Dr. Giuseppe Carenini, External Examiner,
Associate Professor, Computer Science Department
University of British Columbia

Date Approved: _____

Abstract

Syntactic parsing and dependency parsing in particular are a core component of many Natural Language Processing (NLP) tasks and applications. Improvements in dependency parsing can help improve machine translation and information extraction applications among many others. In this thesis, we extend the framework of (Koo, Carreras, and Collins, 2008) for dependency parsing which uses a single clustering method for semi-supervised learning. We make use of multiple diverse clustering methods to build multiple discriminative dependency parsing models in the Maximum Spanning Tree (MST) parsing framework (McDonald, Crammer, and Pereira, 2005). All of these diverse clustering-based parsers are then combined together using a novel ensemble model, which performs exact inference on the shared hypothesis space of all the parser models. We show that diverse clustering-based parser models and the ensemble method together significantly improves unlabeled dependency accuracy from 90.82% to 92.46% on Section 23 of the Penn Treebank. We also show significant improvements in domain adaptation to the Switchboard and Brown corpora.

Keywords: Natural Language Processing, Discriminative Dependency Parsing, Clustering Algorithms, Ensemble Learning

To my family!

Language is a process of free creation; its laws and principles are fixed, but the manner in which the principles of generation are used is free and infinitely varied. Even the interpretation and use of words involves a process of free creation.

Noam Chomsky

Acknowledgments

First and foremost, I would like to express my sincere gratitude towards my supervisor Dr. Anoop Sarker, for his support, encouragement, and patience during my Master's career. I truly enjoyed learning from him, with his enthusiasm for sharing his vast knowledge of Natural Language Processing, and working with him, with his brilliant insights and gentle humor during my work. Without his great supervision this work would not have been possible.

I have been very fortunate to have Dr. Fred Popowich, Dr. Giuseppe Carenini and Dr. Richard Vaughan on my committee and their valuable comments and feedback helped me improve this thesis, of which I am appreciative.

I am also very much indebted to Dr. Reza Haffari, for his guidance and patience during the joint work with him. I learned a lot from him about the different aspects involved during the project.

I have shared numerous happy and sad moments with my friends in Vancouver, that can hardly be acknowledged in words. I would like to especially thank Parnian, Hengameh, Yasaman, Maryam, Parastoo and Zahra for their extreme support and kindness during the past two years. Many thanks to my lab-mates and friends particularly, Ravi, Majid, Baskaran, Ann, Max, David, Milan, Young-chan, Porus, Sara, Hoda, Farnaz, Mohsen, Arash and Hossein with whom I had many interesting discussions and unforgettable memories.

Finally, I would like to express my endless gratitude towards my family. I am extremely fortunate for having been blessed with my incredible parents who always supported me with their unconditional love. My Sisters Mariam and Razieh, with all their care and love and my brother Mohsen with his unconditional support, as well as my caring brothers-in-law Afshin and Mehrdad, my lovely sister-in-law Sanaz and my cheerful nephews Mohsen and Matin are the greatest treasures of my life.

Contents

Approval **ii**

Abstract **iii**

Dedication **iv**

Quotation **v**

Acknowledgments **vi**

Contents **vii**

List of Tables **x**

List of Figures **xi**

1 Introduction **1**

1.1 Motivation 1

1.2 Our Approach 2

1.3 Comparison to Related Work 3

1.4 Contributions of the Thesis 4

1.5 Outline of the Thesis 4

2 Background **6**

2.1 Dependency Trees 6

2.1.1 Relation Between Phrase Structure and Dependency Structure Representations 7

2.2 Discriminative Dependency Parsing . . . 8
2.3 Arc-factored Parsing Algorithms . . . 9
2.3.1 Chu-Liu-Edmonds Algorithm . . . 9
2.3.2 Eisner Algorithm . . . 10
2.4 Parameter Estimation . . . 12
2.4.1 Perceptron for Structured Output . . . 13
2.4.2 Margin Infused Relaxed Algorithm . . . 13
2.5 Graph-based vs Transition-based Dependency Parsing . . . 15

3 Multiple Word Representations **17**
3.1 Introduction . . . 17
3.2 Brown Clustering Algorithm . . . 18
3.3 HMM State Splitting . . . 19
3.4 Syntactic clustering . . . 22
3.5 Feature Design . . . 24
3.6 Experiments . . . 27
3.6.1 Empirical Results . . . 28
3.7 Summary of the Chapter . . . 30

4 Ensemble Model **31**
4.1 Introduction . . . 31
4.2 Ensemble Model . . . 33
4.3 Setting Ensemble Model Weights for Discriminative Dependency Parsing . . . 34
4.3.1 Uniform Setting . . . 34
4.3.2 Learning the Model Weights . . . 34
4.4 Experiments . . . 38
4.4.1 Experimental Setup . . . 38
4.4.2 Empirical Results in In-domain Experiments . . . 39
4.4.3 Empirical Results in Out-of-domain Experiments . . . 44
4.4.4 Error Analysis . . . 47
4.5 Summary of the Chapter . . . 49

5 Conclusion and Future Work **52**
5.1 Thesis Summary . . . 52

5.2 Future Work 53

Bibliography **54**

List of Tables

3.1 Total number of features for each model 27
3.2 Unlabeled-accuracy/unlabeled-complete-correct for the baseline and different clustering-based models 29

4.1 Different POS tag categorizations 36
4.2 Unlabeled-accuracy/unlabeled-complete-correct for in-domain experiments using uniform model weights 40
4.3 For different size K of initial coarse annotations, comparing the ensemble of five split-merge HMM models vs the individual models and the Baseline. . . . 41
4.4 The effect of learning the model weights based on partial scores from dev set on Section 23 of PTB. 44
4.5 The effect of using different modifier POS categories on the accuracy of the ensemble Brown on Section 22 and 23 of PTB using two different learning strategies. 44
4.6 The effect of using different combinations of modifier POS categories on Section 22 of PTB. 45
4.7 The Baseline (no cluster) vs individual cluster-based models as well as an Ensemble of 3 Browns and 5 HMMs for the SwitchBoard and Brown corpora. 46
4.8 The effect of using different modifier POS categories on the accuracy of the ensemble Brown and ensemble HMM on Brown corpus using two different learning strategies. 48
4.9 The effect of using different modifier POS categories on the accuracy of the ensemble Brown and ensemble HMM on SwitchBoard corpus using two different learning strategies. 48

List of Figures

2.1 Dependency and phrase structure tree for the example sentence *For Japan, the trend improves access to American markets* 7
2.2 Different steps of Chu-Liu-Edmonds algorithm (Chu and Liu, 1965; Edmonds, 1967) 10
2.3 The dynamic-programming structures and derivations of the Eisner algorithm (Eisner, 1996) 11
2.4 Extension of Eisner algorithm for second order sibling factorization (McDonald and Pereira, 2006) 12

3.1 The frequency, in the English training corpus, of head-modifier bigrams encountered in the English held-out development corpus (Koo, 2010) 18
3.2 Class-based bigram language model 19
3.3 An example of a Brown word-cluster hierarchy 20
3.4 Clusterings of different granularities 20
3.5 An example of hard clustering of Brown as opposed to soft clustering of HMM 22
3.6 Dependency tree with cluster identifiers obtained from the split non-terminals from the Berkeley parser output 23
3.7 Baseline feature templates and examples 25
3.8 Cluster feature templates and examples 26

4.1 Error rate of the head attachment for different types of modifier categories on Section 22 of PTB 35
4.2 Different feature templates for learning model weighs based on partial scores and examples 37

4.3 Different possible feature templates when learning model weights based on binary indicator features and examples 38
4.4 Comparing Unlabeled Attachment Score (UAS) of the ensemble model with the state-of-the-art graph-based second-order dependency parser and some of the relevant works on Section 23 of PTB. 41
4.5 The unlabeled accuracy for individual and ensemble models. The x and y axis show the accuracy on Section 00 and 23 of PTB respectively 42
4.6 The word *stuff* is clustered with diverse set of words in each of the HMM models 45
4.7 The word *got* is clustered with diverse set of words in each of the HMM models 46
4.8 Error Analysis of different Ensemble models on in-domain experiments 50
4.9 Error Analysis of different Ensemble models on out-of-domain experiments . 51

Chapter 1

Introduction

1.1 Motivation

Syntactic parsing is a core component of many Natural Language Processing (NLP) tasks and applications. Obtaining accurate parsers can be essential in applications such as machine translation (Huang, 2006), summarization (Soricut and Marcu, 2003), information extraction (Finkel and Manning, 2009) and question answering (Hermjakob, 2001). Recently, *dependency parsing*, which provides the abstraction of syntactic information in the form of *head* and *modifier* dependencies, has attracted a number of researchers in the field. Some of the reasons behind the increasing popularity of dependency structure representations can be listed as follows:

- Head-modifier dependencies provide convenient, adjustable and intuitive representations and the simplicity of dependency representations makes the learning and parsing strategies more efficient (Eisner, 2000; McDonald et al., 2005).
- The clear encoding of predicate-argument structure in dependency parsing makes it useful in many NLP applications such as machine translation (Ding and Palmer, 2005) and information extraction (Culotta, 2004).
- Considering that dependency grammar is more appropriate for languages with free or flexible word orders compared to phrase structure grammar, it is possible to provide a common framework for a diverse set of languages (Nivre and Hall, 2005; Ryan, 2006).
- The existence of dependency Treebanks for different languages such as Czech, Swedish,

Turkish and Arabic gives the opportunity to take advantage of machine learning techniques to develop accurate parsers for a number of languages (Nivre et al., 2007).

In discriminative models for dependency parsing, we can take advantage of flexible feature vector representations. Typically the feature representations capture lexical information and part of speech (POS) tagging of the words in the sentence (McDonald, Crammer, and Pereira, 2005). But we face the sparsity problem when dealing with lexical statistics. A simple solution is to provide a semi-supervised method to incorporate word clusters obtained from unlabeled data in discriminative dependency parsing. This idea was provided in (Koo, Carreras, and Collins, 2008) which involved clustering the unlabeled data and then assigning a cluster identifier to each word in the dependency Treebank. These identifiers were used to augment the feature representation of the edge-factored or second-order features, and this feature set was used to discriminatively train a dependency parser.

The use of clusters leads to the question of how to integrate different types of clusters (possibly from different clustering algorithms) and take advantage of multiple word representations in discriminative dependency parsing. This has been the motivation of the works in this thesis. We provide multiple word representations based on different types of *clustering methods* and use each of them to build discriminative dependency parsers. Each parser uses cluster-based features in order to improve accuracy. These diverse clustering-based parsers are then combined together using an *ensemble model* which is shown to result in a more *powerful* model that obtains *consistent* significant improvements in unlabeled dependency parsing.

1.2 Our Approach

In order to help alleviate the sparse data in dependency parsing, we use three different clustering methods. In the first method, the clusters obtained from the (Brown et al., 1992) clustering algorithm are used that can help deal with unknown words since a large amount of unlabeled data is used to extract clusters, e.g. one cluster might contain *plan, letter, request, memo, ...* while another could contain *people, customers, employees, students, ...* These clusters might inform the parser about unknown words in test data that are clustered with known words and how to correctly attach them in the dependency tree. In the second method, a split-merge method is used to train an Hidden Markov Model (HMM) in order to further refine the coarse clusters obtained from the (Brown et al., 1992) algorithm and

obtain context-specific clusters. The third clustering approach that is more "syntactic" comes from the use of state-splitting in Probabilistic Context Free Grammars (PCFGs). For instance, we could extract a syntactic cluster *loss, time, profit, earnings, performance, rating, etc*: all head words of noun phrases corresponding to cluster of direct objects of verbs like *improve*. We create syntactic clusters using the Berkeley parser (Petrov et al., 2006). Chapter 3 describes these clustering methods in more detail.

In order to integrate different models based on different clustering annotations, we provide an ensemble model that is a linear combination of the cluster-based models. Our ensemble model performs *exact inference* on the *shared* hypothesis space of all the parser models. This is in contrast to Bagging (Breiman, 1996) and Boosting (Schapire, 1999) typically used in ensemble learning, merging the outputs from multiple models, or stacking. The ensemble model has the privilege to combine the *expertise* of the cluster-based parsing models and provide a more powerful model. In addition, the *scalability* of the ensemble model enables incrementally adding a large number of models using different clustering algorithms to further improve the accuracy. We implement the ensemble model by extending the MSTParser framework (McDonald, Crammer, and Pereira, 2005). Chapter 4 describes the ensemble model in more detail.

1.3 Comparison to Related Work

Several ensemble models have been proposed for dependency parsing (Sagae and Lavie, 2006; Hall et al., 2007; Nivre and McDonald, 2008; Attardi and Dell'Orletta, 2009; Surdeanu and Manning, 2010). Essentially, most of these approaches combine *different* dependency parsing systems, i.e. transition-based and graph-based. Although graph-based models are globally trained and can use exact inference algorithms, their features are defined over a limited history of parsing decisions. Since transition-based parsing models have the opposite characteristics, the idea is to combine these two types of models to exploit their complementary strengths (Section 2.5 provides more details about the differences between transition-based and graph-based models.). Moreover, in these approaches, the base parsing models are either independently trained and combined at parsing time using voting (Sagae and Lavie, 2006; Hall et al., 2007; Attardi and Dell'Orletta, 2009; Surdeanu and Manning, 2010), or their training is integrated, e.g. using stacking (Nivre and McDonald, 2008; Attardi and Dell'Orletta, 2009; Surdeanu and Manning, 2010). Section 4.1 provides more precise details

about voting and stacking techniques used in dependency parsing.

Our work is distinguished from the aforementioned works in two dimensions. Firstly, we combine various *graph-based* models, constructed using different clustering annotations. Our model is a discriminative analog to the product of randomized models for generative phrase-structure parsing in (Petrov, 2010). Secondly, we do exact inference on the shared hypothesis space of the base models. This is in contrast to previous works which combine the best parse trees suggested by the individual base-models to generate a final parse tree, i.e. a two-phase inference scheme.

1.4 Contributions of the Thesis

The main contributions of this thesis can be summarized thus:

- We combine together multiple word representations based on the different clustering methods in order to improve discriminative dependency parsing in the MSTParser framework (McDonald, Crammer, and Pereira, 2005).
- We provide an ensemble method for combining diverse clustering algorithms that is the discriminative parsing analog to the generative product of experts model for parsing described in (Petrov, 2010).

The two contributions together improve unlabeled dependency accuracy from 90.82% to 92.46% on Section 23 of the Penn Treebank, and we see consistent improvements across all our test sets. In addition to testing on the Wall Street Journal Treebank data, we evaluated our ensemble model in the domain adaptation setting. We show that combining diverse clustering methods can help with the appearance of unknown words in the new domain: on Switchboard data we improve accuracy from 75.23% to 77.23%, and on the Brown corpus we improve accuracy from 84.69% to 86.43%.

1.5 Outline of the Thesis

The remainder of this thesis is structured as follows.

Chapter 2 provides necessary background on dependency structure, inference algorithms and parameter estimation methods.

Chapter 3 presents three different clustering methods to find word clusters and describes

the extended feature set that is used to discriminatively train a dependency parser.
Chapter 4 describes our ensemble model for combining diverse clustering-based models.
Chapter 5 concludes by summarizing the thesis and providing ideas for future work.

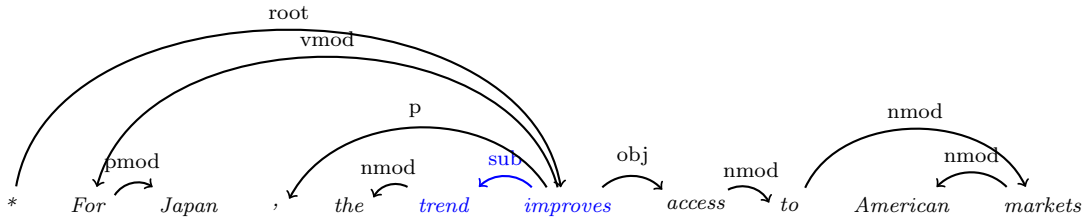
Chapter 2

Background

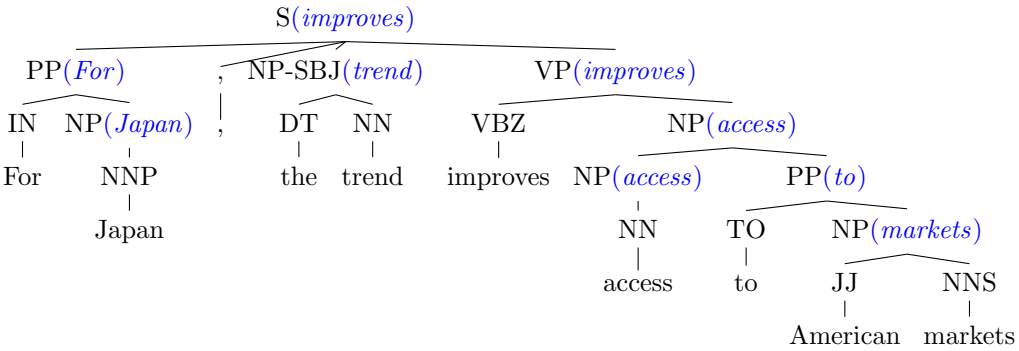
Before we move to the main body of the thesis, we first provide some basic information about dependency trees and the related inference and learning algorithms which are going to be used throughout this thesis. The book by Kübler, McDonald, and Nivre (2009) is a good introduction to this topic.

2.1 Dependency Trees

Dependency grammar formalizes syntactic structure by introducing a directed tree in which nodes correspond to the words, and arcs indicate *head-modifier* dependency relations (Mel'čuk, 1987). Figure 2.1a shows an example sentence and its corresponding dependency tree. The dependencies are shown by arcs pointing from head to modifier and each *label* on the dependency arc represents the *dependency type*. For example noun *trend* is a modifier of the verb *improves* with the dependency type *subject* (sub). The dependency tree shown in figure 2.1a is an example of a *labeled* dependency tree. However the dependency tree can only represent head-modifier dependencies and omit the labels. We refer to this type of dependency trees as *unlabeled* dependency trees. Parsing using unlabeled dependency trees (unlabeled dependency parsing) is more convenient to describe. In addition, we can recover the dependency labels as a post-processing step when doing unlabeled parsing (Mcdonald, Lerman, and Pereira, 2006). Therefore, we focus on unlabeled dependency parsing for the rest of the thesis. Another property of the dependency tree in figure 2.1a is that for any dependency edge *(h, m)* in the tree, h is an ancestor of all the words between h and m. In other words, there is no *crossing* dependency arc in the tree. This type of tree is known



(a) Dependency tree.



(b) Lexicalized phrase structure tree

Figure 2.1: Dependency and phrase structure tree for the example sentence *For Japan, the trend improves access to American markets*

as a *projective* dependency tree, in contrast with *non-projective* trees in which we can have crossing edges in the dependency tree. For the English language, we can analyze most of the sentences through projective dependency parsing efficiently[1]. However, for languages with freer word order such as Czech, German and Dutch non-projective dependency arcs are more common.

2.1.1 Relation Between Phrase Structure and Dependency Structure Representations

Dependency structure is different from phrase structure in the type of relations it represents. Dependency structure shows the relation between the *words* while phrase structure captures the relation between the *phrases*. Moreover, the labels in dependency structure group the

[1]Although for certain sentences, non-projective trees are preferable. For example for sentence *I saw my friend yesterday who was my roommate for years*, we have (saw, yesterday) and (friend, was) as crossing dependency arcs.

words by functional categories such as subject (sub) or object (obj) whereas phrase structures group the phrases by structural categories like noun phrase (NP) or verb phrase (VP). In order to convert the phrase structure into the dependency structure, we first apply a set of deterministic head rules to determine the head of each constituent (Collins, 1999; Yamada and Matsumoto, 2003). Figure 2.1b shows the head component of each phrase. Once we know the head for each constituent, we can define other words as the modifiers of that head. For example, we can infer from figure 2.1b that "*For*", ",", "*trend*" are the modifiers of the verb "*improves*".

2.2 Discriminative Dependency Parsing

In discriminative dependency parsing, we can formalize the dependency problem in the form of *structured linear models* (McDonald, Crammer, and Pereira, 2005) as follows:

$$\text{PARSE}(\mathbf{s}) = \arg\max_{\mathbf{t}\in\mathcal{T}(\mathbf{s})} \mathbf{w}\cdot\mathbf{f}(\mathbf{s},t) \tag{2.1}$$

where we are searching for the highest scoring parse tree t for the sentence s from the space of dependency trees $\mathcal{T}(\mathbf{s})$. We use a linear scoring function in which we score a parse tree t as the weighted sum of parse features. $\mathbf{f} : (\mathbf{s}, t) \rightarrow \mathbb{R}^d$ shows the d-dimensional feature vector representation of the event that t is the syntactic analysis for the sentence s and w is the related weight vector.

Arc-factored Models

Since the space of possible dependency trees $\mathcal{T}(\mathbf{s})$ grows exponentially with the size of the sentence, in order to be able to perform the maximization in equation 2.1 efficiently, we assume that we can decompose f into smaller representations which only depend on the bounded subset of t. In other words, we assume that we can *factor* the dependency tree into smaller *parts* r and therefore restrict the features representation as follows:

$$\text{PARSE}(\mathbf{s}) = \arg\max_{\mathbf{t}\in\mathcal{T}(\mathbf{s})} \sum_{r\in\mathbf{t}} \mathbf{w}\cdot\mathbf{f}(\mathbf{s},r) \tag{2.2}$$

In this notation, we provide local feature representations that only depend on the factored part of t, r. In the simplest case of factorization, we consider the parts to be the individual

head-modifier dependency arcs (h, m). In this setting, we can restate the equation 2.2 as follows:

$$\text{PARSE}(\mathbf{s}) = \arg\max_{\mathbf{t}\in\mathcal{T}(\mathbf{s})} \sum_{(h,m)\in\mathbf{t}} \mathbf{w}\cdot\mathbf{f}(\mathbf{s},h,m) \tag{2.3}$$

This type of factorization leads to *first-order* parsing model (McDonald, Crammer, and Pereira, 2005).

In the higher-order models, the parts consist of arcs together with some context, e.g. the parent or the sister arcs (McDonald and Pereira, 2006; Carreras, 2007; Koo and Collins, 2010). For example, in case of having sister arcs (h,si) and (h,m), we can rewrite the equation 2.2 as:

$$\text{PARSE}(\mathbf{s}) = \arg\max_{\mathbf{t}\in\mathcal{T}(\mathbf{s})} \sum_{(h,si,m)\in\mathbf{t}} \mathbf{w}\cdot\mathbf{f}(\mathbf{s},h,si,m) \tag{2.4}$$

2.3 Arc-factored Parsing Algorithms

Arc-factored parsing algorithms are based on the correspondence between the dependency trees and *spanning trees* (West, 2001). For a given sentence $s = s_0 s_1 ... s_n$ where s_0 is the root symbol, consider a graph $G = (V, E)$ in which $s_0, s_1, ..., s_n$ are the vertices in V and there is an edge (s_i, s_j) between all pair of words and from the root s_0 to every word with weight $score(i, j)$. Finding the maximum spanning tree (MST) in graph G rooted at s_0 is equivalent to finding the highest scoring dependency tree for sentence s.

2.3.1 Chu-Liu-Edmonds Algorithm

In case of non-projective dependency parsing, we can take advantage of existing MST algorithms for directed graphs. One such algorithm is the Chu-Liu-Edmonds algorithm (Chu and Liu, 1965; Edmonds, 1967). We are going to explain this algorithm through an example sentence *John loves traveling*. In order to find the MST for the graph shown in figure 2.2a, first we determine the highest scoring incoming edge for each word as illustrated in figure 2.2b. if the corresponding graph does not contain any cycles, we have found the MST. But in case that it introduces cycles, we *contract* the cycle into a *pseudo-node* and recalculate the weights. This is shown in figure 2.2c. We have shown the contraction of nodes *John* and *loves* as *John—loves*. We call this pseudo-node as v_c. The edge from v_c to *traveling* is 20 since it has the highest score among the edges from the cycle to traveling.

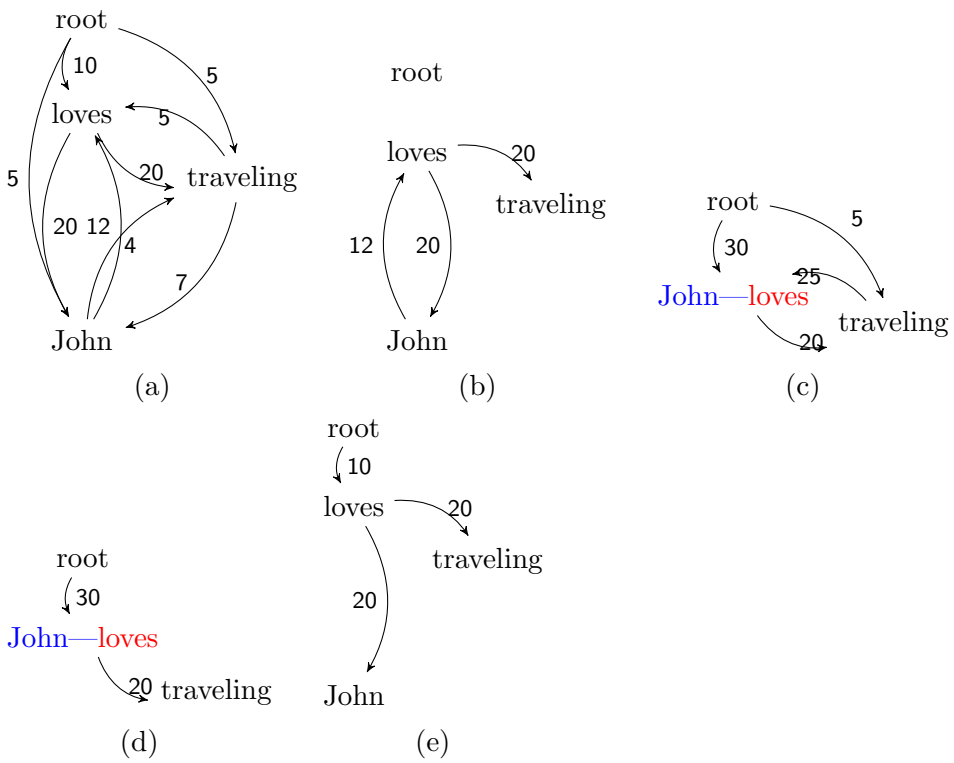


Figure 2.2: Different steps of Chu-Liu-Edmonds algorithm (Chu and Liu, 1965; Edmonds, 1967)

The edge from *traveling* to v_c is 25 because it has the highest scoring tree originating from *traveling* and including the nodes in v_c. the edge from *root* to v_c is going to be 30 similarly. We can call this algorithm recursively to get the MST of the graph but we also need to keep track of the endpoints of the edges coming in/out of the v_c. By applying the first step to the obtained graph, we will get the graph shown in figure 2.2d. Since this graph does not include any cycles, we have reached the MST of the graph. The original MST, has all the edges in the tree from *root* to v_c which are from *root* to *loves* and *loves* to *John*. This results in the MST shown in figure 2.2e.

2.3.2 Eisner Algorithm

Projective (nested) dependency parsers are related to context free parsers in the sense that the ideas in algorithms for parsing context free grammars can be used to parse projective dependency trees. One such algorithm is Cocke-Kasami-Younger (CKY) (Younger, 1967)

which is a bottom-up dynamic programming algorithm. Eisner (1996) provided an efficient algorithm for parsing projective dependency trees, by taking advantage of the observation that a word can collect its left and right modifiers independent of each other. This is illustrated in figure 2.3. We can break the construction of dependency arc from a given word w_i to w_j by considering the left and right subgraphs of the trees headed by w_i and w_j as shown in figure 2.3a. Let E[s][e][d][c] be the dynamic programming table which keeps

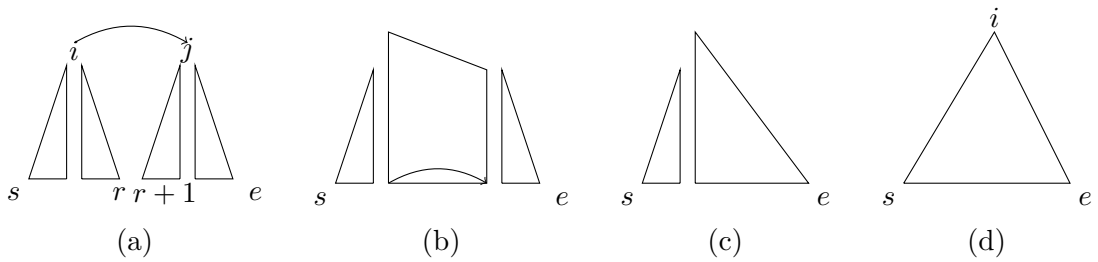


Figure 2.3: The dynamic-programming structures and derivations of the Eisner algorithm (Eisner, 1996). Complete spans are shown as triangles and incomplete spans as trapezoids

the score of the best sub-tree spanning from w_s to w_e with the dependency arc direction d (1=→, 0=←) and type of dynamic programming structure c (1=complete; no further modifiers, 0= not complete). The algorithm fills the table in a bottom-up fashion similar to CKY algorithm by initializing all length-one sub-graphs (E[s][s][d][c]) to zero and then considering spans of increasing length. The pseudo-code for the Eisner algorithm is shown in algorithm 1. In order to find the highest scoring incomplete right sub-tree, we find the index r in which joining the two complete subtrees with addition of constructing dependency arc leads to the highest score as shown in figure 2.3a, 2.3b. In order to find the best score for a complete right sub-tree, we need to find the index that leads to highest score by combining the incomplete and complete sub-tree as shown in figure 2.3b, 2.3c. Since the root is located at left side of the sentence, the highest scoring tree for the entire sentence is stored at E[0][n][1][1]. The complexity of the algorithm is $O(n^3)$ as the size of the table is $O(n^2)$ and for each entry in the table we need to go through different positions for index r which is $O(n)$.

McDonald and Pereira (2006) show that the dynamic programming structure can be changed to adjust to *sibling* factorization that decomposes each tree into sibling parts. In order to do so, a new type of dynamic programming structure is added: *sibling* item. The main idea is to first collect the pairs of adjacent modifiers consecutively before attaching

Algorithm 1 Pseudo-code for Eisner parsing algorithm (Eisner, 1996)

```
Initialization:T= E[s][s][d][c] = 0 ∀s, d, c
for ℓ : 1 ... n do
  for s : 1 ... n do
    e = s + ℓ
    if e > n then break
    E[s][e][0][0] = max_{s≤r<e}(E[s][r][1][1] + E[r + 1][e][0][1] + s(e, s))
    E[s][e][1][0] = max_{s≤r<e}(E[s][r][1][1] + E[r + 1][e][0][1] + s(s, e))
    E[s][e][0][1] = max_{s≤r<e}(E[s][r][0][1] + E[r][e][0][0])
    E[s][e][1][1] = max_{s<r≤e}(E[s][r][1][0] + E[r][e][1][1])
  end for
end for
```

them to their parent. This is shown in figure 2.4. The complexity of the algorithm is still $O(n^3)$.

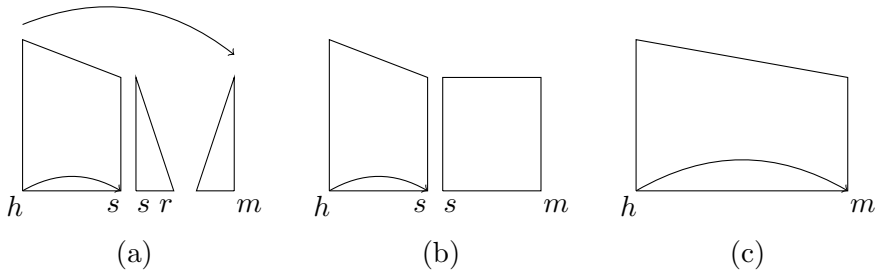


Figure 2.4: Extension of Eisner algorithm for second order sibling factorization (McDonald and Pereira, 2006). h constructs a dependency arc to m by having the information that the last modifier of h was s. This is done by creating a sibling item in (b)

2.4 Parameter Estimation

In order to train the dependency parser in the factored structured linear model provided in section 2.2, we need to estimate the parameter values w related to each feature. In this section, we explain two online methods for learning the parameter values w in a structured linear model.

2.4.1 Perceptron for Structured Output

Structured perceptron is a variant of the classic perceptron algorithm (Rosenblatt, 1958) that was first introduced by Collins for sequential classification problems (Collins, 2002). Algorithm 2 shows the pseudo-code for the algorithm. Initially the parameter vector is set to $\overrightarrow{0}$. Then in the series of N iterations, the algorithm parses an example from the training data and compares the predicted parse tree with the gold-parse tree. In case that there was a *mistake*, the algorithm updates the w by the difference between the gold feature vector and the predicted feature vector. The output of the algorithm is the *average* of the weight vectors after each iteration.

The factored representation of the features and the fact the algorithm only updates the weights when there is a mistake, makes the weight vector to be sparse since in many cases most of the parse tree can be correct. Therefore, using a sparse data structure gives the opportunity to take advantage of large number of features (Koo, 2010). Moreover, averaging the parameters has shown to be effective in terms of improving the performance since it reduces the overfitting (Collins, 2002).

Algorithm 2 Perceptron Algorithm for Structured Output (Collins, 2002)

Input: Training examples $\{(x_t, y_t)\}_{t=1}^{T}$, N: Number of iterations
Initialization: $\mathbf{w} = \overrightarrow{0}$, $v = \overrightarrow{0}$
for n : 1 ... N **do**
  for t : 1 ... T **do**
    $y' = \arg\max_{y'} \mathbf{w} \cdot \mathbf{f}(x_t, y')$
    if $y' \neq y_t$ **then**
      $\mathbf{w} = \mathbf{w} + \mathbf{f}(x_t, y_t) - \mathbf{f}(x_t, y')$
    end if
    $v = v + \mathbf{w}$
  end for
end for
$v = v/NT$

2.4.2 Margin Infused Relaxed Algorithm

One disadvantage of the perceptron algorithm is that it does not optimize any notion of classification margin which has shown to be effective in reducing generalization error (Boser, Guyon, and Vapnik, 1992). The margin infused relaxed algorithm (MIRA) provides an

online algorithm to minimize the weight vector based on a set of *margin constraints* that make sure the advantage in favor of the correct parse against any of other parses to be at least the amount of mismatch between the two of them. Algorithm 3 provides pseudo-code for MIRA. Each update in MIRA is *conservative* in the sense that there is no change in weight vector when the set of constraints are satisfied. If some constraints do not hold, the algorithm makes the smallest weight change such that it satisfies the margin constraints. The set of margin constraints keep the score of the correct parse above the score of incorrect ones by at least the amount of *loss* function. In the case of dependency parsing, McDonald, Crammer, and Pereira (2005) define the loss function to be the number of words with incorrect predicted heads for them (Hamming loss). Using Hamming loss as the loss function has this advantage that it is directly related to the evaluation metric for dependency parsing which is the percentage of correct predicted heads for the words in the sentence.

Since the objective function is quadratic in w and the margin constraints are linear, this quadratic programming problem can be solved using Hildreth's algorithm (Censor and Zenios, 1997). Properties of MIRA in terms of convergence is investigated in (Crammer and Singer, 2003; Crammer et al., 2006).

Algorithm 3 MIRA Algorithm for Dependency Parsing (McDonald, Crammer, and Pereira, 2005)

Input : Training examples $\{(x_t, y_t)\}_{t=1}^{T}$, N: Number of iterations
$w_0 = 0, v = 0, i = 0$
for n : 1 ... N **do**
  for t : 1 ... T **do**
    $w^{(i+1)} = \arg\min_{w'} ||w' - w^{(i)}||$
    s.t.
    $s(x_t, y_t) - s(x_t, y') \geq L(y_t, y')$ w.r.t w'
    $\forall y' \in parses\,(x_t)$
    $v = v + w^{(i+1)}$
    $i = i + 1$
  end for
end for
$w = v/NT$

One problem with algorithm 3 is that for a given input there are exponentially (to the length of the input) many parse trees and therefore exponentially many margin constraints. In order to make the algorithm tractable, McDonald, Crammer, and Pereira (2005) relax

the optimization by providing the margin constraints only for the $k - best$ parse trees as shown in algorithm 4. For dependency parsing, it is shown that even with small values of k the algorithm works comparably well (Ryan, 2006).

Algorithm 4 k-best MIRA Algorithm for Dependency Parsing (McDonald, Crammer, and Pereira, 2005)

Input: Training examples $\{(x_t, y_t)\}_{t=1}^{T}$, N: Number of iterations
$w_0 = 0, v = 0, i = 0$
for n : 1 ... N **do**
  for t : 1 ... T **do**
    $w^{(i+1)} = \arg\min_{w'} ||w' - w^{(i)}||$
    s.t.
    $s(x_t, y_t) - s(x_t, y') \geq L(y_t, y')$ w.r.t w'
    $\forall y' \in best_k\left(x_t; w^{(i)}\right)$
    $v = v + w^{(i+1)}$
    $i = i + 1$
  end for
end for
$w = v/NT$

MIRA has shown to be a good framework for dependency parsing due to its accuracy, efficiency and simplicity (Ryan, 2006).

2.5 Graph-based vs Transition-based Dependency Parsing

The model that we described so far is a form of *graph-based* model. Another type of model can be described as *transition-based* model. Transition-based models are different in terms of learning, inference and feature representation from the graph-based models. In transition-based parsing we learn a model that scores transitions from one parser state to the next one based on the parse history, as opposed to a graph-based model in which we learn a model for scoring possible dependency graphs. In transition-based models, parsing is done by greedily selecting the highest scoring transition among the parser states whereas in graph-based models we use exact inference algorithms to search for the highest scoring graph. The main advantage of transition-based models is their rich feature representations based on the history of parser decisions in contrast with the features in graph-based model that were restricted to a limited number of dependency arcs.

Throughout this thesis we focus on graph-based models in MSTParser framework (McDonald, Crammer, and Pereira, 2005). We work with both first-order and second-order models, we train the models using MIRA, and we use the (Eisner, 1996) algorithm for inference.

Chapter 3

Multiple Word Representations

3.1 Introduction

Lexicalized features have shown to be beneficial in resolving ambiguous relations in both generative and discriminative parsing (Collins, 1999; Charniak, 2000; Ryan, 2006) but care must be taken when using these features due to the sparsity problems. Koo (2010) illustrates the sparsity problem by plotting the distribution of frequencies of the head-modifier bigrams from held-out data in training data set. As it is shown in figure 3.1, most of the bigrams in held-out data occur quite infrequently in training data and using bigram features based on these low-frequency bigrams may not be fruitful. One approach to combat sparsity problem is by incorporating *word clusters* as features. Word clusters capture lexical information needed for resolving ambiguous relations and at the same time exist at a coarser level than words to help alleviate the sparse data problem. Using word clusters has been shown to be successful in NLP tasks such as named-entity recognition (Miller, Guinness, and Zamanian, 2004) and dependency parsing (Koo, Carreras, and Collins, 2008). We extend the framework of (Koo, Carreras, and Collins, 2008) for dependency parsing which uses a single clustering method for semi-supervised learning and make use of multiple diverse clustering methods. In this chapter we explain three different types of clustering methods used for extracting cluster identifiers: 1) Brown et al. (1992) clustering algorithm (Section 3.2), 2) HMM state splitting (Section 3.3), and 3) Syntactic clustering using split non-terminals from the PCFG-based Berkeley parser (Section 3.4). We also explain the cluster-based feature design in Section 3.5.

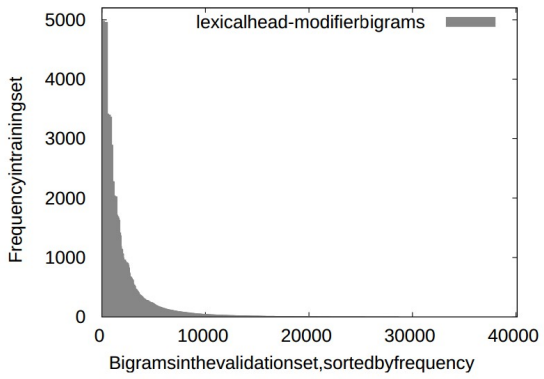


Figure 3.1: The frequency, in the English training corpus, of head-modifier bigrams encountered in the English held-out development corpus (Koo, 2010)

3.2 Brown Clustering Algorithm

Brown et al. (1992) provides a greedy bottom-up word clustering algorithm to obtain a hierarchical clustering of words. The algorithm takes a sequence of words $\omega = w_1^n$ as input and generates a binary tree in which the leaves are the words and the internal nodes are considered as clusters containing the words in that subtree. Initially each word is considered to have its own cluster. The algorithm then repeatedly merges the two clusters that cause the least decrease in likelihood of the input text according to the *class based bigram language model* defined on the clusters as shown in figure 3.2. The maximum likelihood of the model parameters are estimated with empirical counts. In this manner words are clustered such that the cluster of previous word $C(w_{i-1}) = c_{i-1}$ is the most predictive of the cluster of the current word $C(w_i) = c_i$:

$$L(\omega, C) = \prod_{i=1}^{n} p(c_i|c_{i-1})\, p(w_i|c_i) \tag{3.1}$$

Equation 3.1 can be rewritten in terms of mutual information between adjacent clusters (Liang, 2005).

If we keep track of the pair-wise merges, we can obtain a binary tree whose leaves are words and each word can have a unique identification based on its path from the root. By assigning 0/1 bit values to left/right branches respectively, we can specify the paths as bit-strings as depicted in Figure 3.3. We can obtain the word clusters by selecting the nodes at

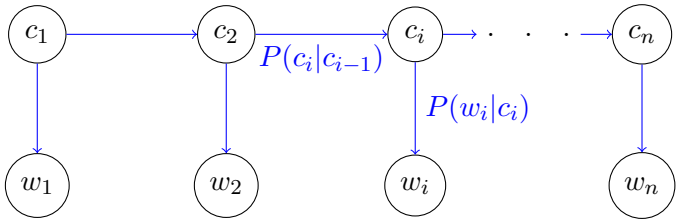


Figure 3.2: Class-based bigram language model

certain depths from the root. For example, if we choose the nodes at depth two of the tree in figure 3.3, we would get {*boss, leader, chief*}, {*September, October*}, {*walk*} and {*escape, run*} as word clusters.

In order to implement the algorithm such that it can be practical considering the size of vocabulary, the algorithm considers a maximum number of possible clusters K as follows: at first it sorts the words in the order of decreasing frequency and put each of the K most frequent words in its own cluster. Then for each subsequent word, the algorithm creates a new cluster and when the total number of clusters becomes greater than K, the algorithm merges the two clusters, in the same way as described before, to reduce the total number of clusters to K. This algorithm runs in $O(VK^2 + T)$ where V is the vocabulary size and T is the length of the text (Liang, 2005).

By choosing different depths in the binary tree we can obtain clusterings of different granularities. This is illustrated in figure 3.4 where the words are clustered using Brown algorithm with $K = 512$. Clusterings in the first row of figure 3.4 are obtained from full bit-strings while the clusterings in the second row are derived from considering 4-bit prefix of cluster identifiers. We can observe that the full bit-strings provide finer-grained clusterings in comparison with short bit-string prefixes. As an example, the clustering identifier 001100111 consists of past-participate verbs whereas the identifier 0011 contains different types of verbs.

3.3 HMM State Splitting

A closer look into equation 3.1 reveals that the underlying generative model can be viewed as a Hidden Markov Model (HMM), where the clusters are the HMM's hidden states. The greedy algorithm provided in (Brown et al., 1992) gives a *hard* partitioning of the words into K clusters by constraining the emission probabilities, i.e. each word belong to just one state

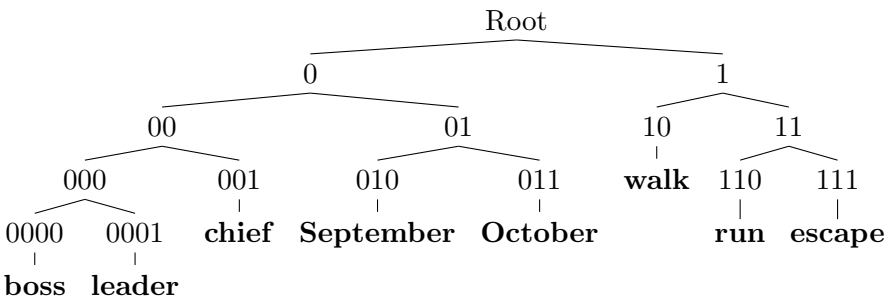


Figure 3.3: An example of a Brown word-cluster hierarchy

0011001111 emphasized discovered thought felt suggested ...	**1001**111111 chairman treasurer founder ...	**1011**0111 September October November January August ...
0011 think were said is found ...	**1001** chairman CEO result company transaction ...	**1011** March yesterday later earlier September ...

Figure 3.4: Clusterings of different granularities. The clusterings in the first row are full bit-strings whereas the clusterings in second row are obtained from some of the nodes at depth 4 from the root.

with non-zero emission probability. We can relax the hard clustering constraint to produce a *soft* clustering by maximizing the likelihood function using the EM algorithm. Given a sentence $\omega = w_1^n$ and its initial coarse annotation c_1^n, let $a_x \in a = c_i$ be the latent cluster for c_i and $b_y \in b = c_{i+1}$ be the latent cluster for c_{i+1}. The forward, $\alpha_{i+1}(b_y) = p\left(w_1^{i+1}, b_y\right)$

and backward, $\beta_i(a_x) = p\left(w_{i+1}^n | a_x\right)$ probabilities can be computed recursively as follows:

$$\begin{aligned} \alpha_{i+1}(b_y) &= \sum_x \alpha_i(a_x)\, p(b_y|a_x)\, p(w_{i+1}|b_y) \\ \beta_i(a_x) &= \sum_y \beta_{i+1}(b_y)\, p(b_y|a_x)\, p(w_{i+1}|b_y) \end{aligned} \tag{3.2}$$

In the Expectation step, the posterior probabilities can be computed as:

$$\begin{aligned} p(a_x, b_y|\omega) &\propto \alpha_i(a_x)\, \beta_{i+1}(b_y)\, p(b_y|a_x) \\ p(a_x, w_i|\omega) &\propto \alpha_i(a_x)\, \beta_i(a_x) \end{aligned} \tag{3.3}$$

In the Maximization step, the above posterior probabilities are used as weighted observations for updating the transition and emission probabilities:

$$\begin{aligned} p(b_y|a_x) &= \frac{\#(a_x, b_y)}{\sum_{b_y} \#(a_x, b_y)} \\ p(w|a_x) &= \frac{\#(a_x, w)}{\sum_w \#(a_x, w)} \end{aligned} \tag{3.4}$$

We employ a hierarchical split-and-merge method to gradually and adaptively add latent annotations in places where they would produce the greatest increase in training likelihood. In each split-merge round, we first split each HMM state into two sub-states, and initialize the parameters of the sub-states by adding a small random noise to the emission and transition probabilities of the parent state to break the symmetry. We then use EM to compute the maximum likelihood estimates for this doubled HMM. Afterwards, we trim the large HMM by merging back half of those state splits which make the least loss to the data likelihood. Petrov et al. (2006) employed split-merge PCFGs for parsing and Huang, Eidelman, and Harper (2009) have used split-merge HMMs for POS-tagging. In both of these works, the split-merge mechanism is used to relax the strong independence assumptions imposed by the original model.

The choice of the initial annotation is important, since it injects the bias into the model towards learning the aspects of the data we want to capture. We make use of coarse clusters obtained from the Brown algorithm for the initial annotation. This informs the model to capture sentence-specific semantic information about words. POS tags could have been used as the initial coarse annotations; however, they are used as a feature in our dependency model. Based on some initial experiments, we believe refined-POS tags do not inject much

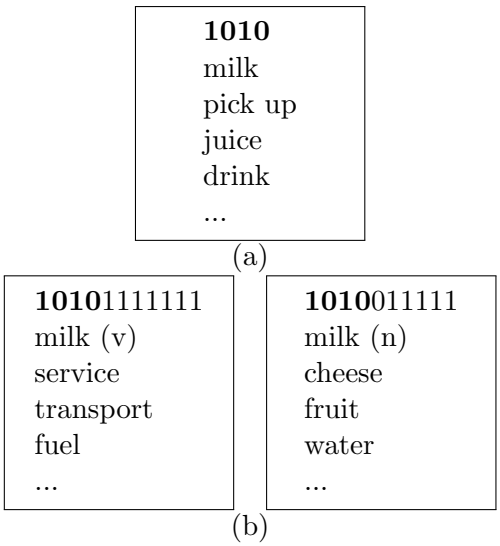


Figure 3.5: An example of hard clustering of Brown as opposed to soft clustering of HMM. (a) Hard clustering of *milk* in Brown clustering. (b) different clusterings for the word *milk* in HMM clustering.

information into the parsing model above the original POS tags.

HMM clustering provides a *context-specific* cluster for each word in which the same word can receive different clusters based on the context as opposed to Brown clustering where every word is exactly assigned one cluster regardless of the context. This is illustrated in figure 3.5. The word *milk* can refer to a noun (from cow) or verb (e.g., get advantage). Figure 3.5a shows that in hard clustering of Brown with maximum number of clusters $K = 64$, milk is assigned to cluster 1010, whereas in HMM clustering, with this Brown clustering as initial annotation, milk is assigned to different clusters based on its usage as verb or noun in the context.

3.4 Syntactic clustering

Our two other clusterings are extracted from the split non-terminals obtained from the PCFG-based Berkeley parser (Petrov et al., 2006). Split non-terminals from the Berkeley parser output are converted into cluster identifiers in two different ways.

Syn-High. Head percolation rules are used to label each non-terminal in the parse such that each non-terminal has a unique daughter labeled as head. Each word is assigned a cluster identifier which is defined as the parent split non-terminal of that word if it is not marked as head, else if the parent is marked as head we recursively check its parent until we reach the unique split non-terminal that is not marked as head. This recursion terminates at the start symbol TOP.

Syn-Low. The split POS tags for each word are used as an alternate word representation.

```
(TOP
  (S-14
     (PP-2 (IN-1 For)
           (NP-10 (NNP-19 Japan)))
     (,-0 ,)
     (NP-18 (DT-15 the) (NN-23 trend))
     (VP-6 (VBZ-1 improves)
           (NP-24 (NN-13 access))
           (PP-14 (TO-0 to)
                  (NP-9 (JJ-31 American)
                        (NNS-25 markets))))))
```

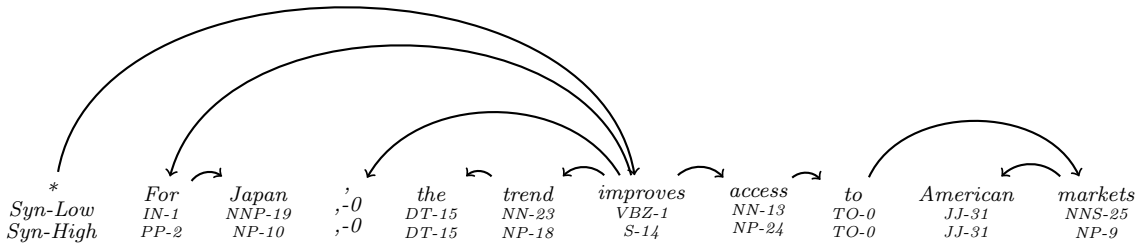


Figure 3.6: Dependency tree with cluster identifiers obtained from the split non-terminals from the Berkeley parser output. The first row under the words are the split POS tags (Syn-Low), the second row are the split bracketing tags (Syn-High).

For the example Berkeley parser output shown above, the resulting word representations and dependency tree is shown in Figure 3.6. If we group all the head-words in the training data that project up to split non-terminal NP-24 then we get a cluster: *loss, time, profit,*

earnings, performance, rating, ... which are head words of the noun phrases that appear as direct object of verbs like *improve*.

3.5 Feature Design

A key factor in the success of discriminative dependency parsing is the ease and flexibility which this framework offers to incorporate informative features. We make use of two sets of features in the feature mapping. Our baseline features capture information about the lexical items and their part of speech tags, as defined in (McDonald, Crammer, and Pereira, 2005). The baseline features include *Uni-gram* and *Bi-gram* features as well as *in Between POS Features* and *Surrounding Word POS Features*. The *Uni-gram* features contain features capturing information about either head or modifier(e.g. (mw, mpos)), while the *Bi-gram* features incorporate features for combination of head and modifier (e.g. (hw, mw)). *In between POS Features* not only include the POS of head and modifier, but also hold the POS of the word in between them in the form of a POS Tri-gram. *Surrounding word POS Features* provide information about the POS of the neighboring words of the edge in the form of a POS 4-gram. Following (Koo, Carreras, and Collins, 2008) we augment those features with *binned distance features* which determines if the distance between head and modifier is more than 2, 5, 10, 20, 30 or 40 words. We also added the *back-off* version of *Surrounding Word POS Features* in which one of the POS tags was removed.

Figure 3.7 provides a schematic view of the baseline feature templates as well as some examples of features for a sample sentence.

Following (Koo, Carreras, and Collins, 2008), we use word cluster identifiers as a source of an additional set of features. The clusters inject long distance syntactic or semantic information into the model (in contrast with the use of POS tags in the baseline) and help alleviate the sparse data problem for complex features that include n-grams. The cluster-based features we use are similar to (Koo, Carreras, and Collins, 2008). We define short bit-length (4 and 6 bit) clusters to capture information similar to POS tag and full bit-length clusters to retrieve information regarding word form. Figure 3.8 illustrates examples of cluster-based features. The cluster-based features mimic the template structure of baseline features illustrated in figure 3.7 but they take advantage of word clusters instead of POS tags. The features that use POS information exclusively are replaced by features that use word clusters. So we will have a set of features that use only cluster information and some

Example Sentence:
While worry grows about big Japanese investments in the U.S., Japan's big trading companies are rapidly increasing their stake in America's smaller business.
Example Edge:
(are,increasing)

(a)	(b)
(mw,mpos) (hpos,mpos)	(increasing VBG*&RA&1) (VBP VBG*&RA&1)
(hposA,mposA) (hpos,mw)	(V V*&RA&1)(VBP increasing*&RA&1)
(hposA,mw) (hw,mpos)	(V increasing*&RA&1)(are VBG*&RA&1)
(hw,mposA) (hw,hpos)	(are V*&RA&1)(are VBP*&RA&1)
(hw,hposA) (hw,mw)	(are V*&RA&1)(are increasing*&RA&1)
(hposA,mw,mposA)	(V increasing V*&RA&1)
(hpos,mw,mpos)	(VBP increasing VBG*&RA&1)
(hw,hpos,mpos)	(are VBP VBG*&RA&1)
(hw,hposA,mposA)	(are V V*&RA&1)
...	...
(hpos-1,hpos,mpos-1,mpos)	(NNS VBP RB VBG*&RA&1)
(hpos-1,hpos,mpos)	(NNS VBP VBG*&RA&1)
(hpos-1,hpos,mpos,mpos+1)	(NNS VBP VBG PRP$*&RA&1)
(hpos-1,hpos,mpos+1)	(NNS VBP PRP$*&RA&1)
(hpos,mpos,mpos+1)	(VBP VBG PRP$*&RA&1)
...	...
(mpos,spos) (mw,sw)	((VBG_RB_0_RA)(increasing_rapidly_RA)
(mw,spos) (mpos,sw)	(increasing_RB_RA)(VBG_rapidly_RA)
(hpos,spos,mpos)	(VBP_RB_VBG_LA)
...	...

Figure 3.7: (a) Example of baseline feature templates. each of the tuples provides a class of indicator features. Abbreviations: mw: modifier word; mpos: modifier POS tag; mposA: modifier coarse POS tags; hw: head word; hpos: head POS tag; hposA: head coarse POS tags; hpos-1, hpos+1: neighboring POS tags around the head word; sw: sibling word; spos: sibling POS tag. (b) Examples of baseline features for the given edge. The numbers that appear in some of the feature examples represent the binned distance. LA and RA indicate left and right arc respectively.

hybrid features that involve word clusters as well as POS tags similar to (Koo, Carreras, and Collins, 2008).

We note a few differences between our feature set and the feature design in (Koo, Carreras, and Collins, 2008). First, we do not use the *hybrid* features involving word clusters

Example Sentence:
While worry grows about big Japanese investments in the U.S., Japan's big trading companies are rapidly increasing their stake in America's smaller business.
Example Edge:
(are,increasing)

(a)	(b)
(hc) (mc)	(0101111110)(010101111100110)
(hc4) (mc4)	(0101)(0101)
...	...
(hc,mc)	(01011111101 010101111100110*&RA&1)
(hc4,mc4)(hc6,mc6)	(0101 0101&RA&1)(010111 010101&RA&1)
(hpos,mc4)(hpos,mc6)	(VBP 0101*&RA&1) (VBP 010101*&RA&1)
(hposA,mc4)(hposA,mc6)	(V 0101*&RA&1)(V 010101*&RA&1)
(hc4,mpos)(hc6,mpos)	(0101 VBG*&RA&1)(010111 VBG*&RA&1)
...	...
(hpos,mc,mpos)	(VBP 010101111100110 VBG*&RA&1)
(hpos,mc4,mpos)	VBP 0101 VBG*&RA&1)
(hc,hpos,mpos)	(01011111101 VBP VBG*&RA&1)
(hc4,hpos,mpos)	(0101 VBP VBG*&RA&1)
...	...
(hc4,sc4,mc4)	(0101_0010_0101_0_LA)
(hpos,sc4,mc4)	(VBP_0010_0101_0_LA)
(hpos,spos,mc4)	(VBP_RB_0101_0_LA)
(hc4,spos,mpos)	(0101_RB_VBG_0_LA)
(hc4,spos,mc4)	(0101_RB_0101_0_LA)
(hc6,sc6,mc6)	(010111_001011_010101_0_LA)
...	...

Figure 3.8: (a) Example of cluster feature templates. Abbreviations: hc4: 4-bit head cluster ; h6: 6-bit head cluster; hc: full-length head cluster; mc4,mc6,mc : similar for modifier; sc4, sc6, sc: similar for sibling. (b) Examples of cluster-based features for the given edge.

and word forms (e.g., (hw, hc)). Second, since our model is based on (McDonald, Crammer, and Pereira, 2005), it uses sibling interactions whereas in (Koo, Carreras, and Collins, 2008)'s feature design, the second order features are the same as (Carreras, 2007) which also take advantage of grandparent interactions.[1]

[1]Terry Koo was kind enough to share the source code for the (Koo, Carreras, and Collins, 2008) paper with us which allowed us to make a detailed comparison of our features with theirs. Since the features do not exactly match, the accuracy numbers we get are also slightly divergent from those in their paper.

Number of features	
4BitBrown	9670600
6BitBrown	11090398
FullBrown	23189514
JointBrown	28266519
HMM1	37441988
HMM2	37612766
HMM3	37632042
HMM4	37415959
HMM5	37608222
JointHMM	160888450

Table 3.1: Total number of features for each model

The joint models take the union of the cluster-based feature sets along with the baseline features in order to train the model. For the JointBrown, we use the union of 4-bit, 6-bit and full length Brown clusters and similarly for the JointHMM, we concatenate the 5 HMM clusters. Table 3.1 provides the total number of features in each of the individual models.

3.6 Experiments

In our experiments, we use the MSTParser framework (McDonald, Crammer, and Pereira, 2005) and add cluster-based features to the feature design. We work with both first-order and second-order models, train the models using MIRA, and we use the (Eisner, 1996) algorithm for inference.

Data The experiments are done on the English Penn Treebank (PTB), using standard head-percolation rules (Yamada and Matsumoto, 2003) to convert the phrase structure into dependency trees. We split the PTB into a training set (Sections 2-21), a development set (Section 22), and test sets (Sections 0, 1, 23, and 24). All of our experiments in this setting match previous work (Yamada and Matsumoto, 2003; McDonald, Crammer, and Pereira, 2005; Koo, Carreras, and Collins, 2008). POS tags for the development and test data were assigned by MXPOST (Ratnaparkhi, 1996), where the tagger was trained on the entire training corpus. To generate part of speech tags for the training data, we used 20-way jackknifing, i.e. we tagged each fold with the tagger trained on the other 19 folds.

Clusters The Brown algorithm word clusters are derived using Percy Liang's implementation[2] which we ran on the BLLIP corpus (Charniak et al., 2000) which contains ∼43M words of Wall Street Journal text.[3] This produces a hierarchical clustering over the words which is then sliced at a certain height to obtain the clusters.

The Split HMM clusters are also derived on the BLLIP corpus. We annotate sentences initially with the word clusters coming from the Brown clustering algorithm, and then do 6 rounds of split-and-merge cycles to refine these coarse annotations. Since the algorithm uses randomization, we get different clusterings if we use various randomization seeds. Petrov (2010) uses an analogous randomization initialization in order to obtain an ensemble of PCFGs.

The sentence-specific syntactic word clusters are derived from the parse trees using the Berkeley parser[4], which generates phrase-structure parse trees with split syntactic categories. To generate parse trees for development and test data, the parser is trained on the Section 2-21 of the Penn Treebank training data to learn a PCFG with latent annotations using split-merge operations for 5 iterations. To generate parse trees for the training data, we used 20-way jackknifing i.e. we parsed each fold with the parser trained on the other 19 folds.

3.6.1 Empirical Results

Table 3.2 presents our results. The baseline (first column of the table) does not use any cluster-based features, and the next models use cluster-based features using different clustering algorithms. The 4bitBrown, 6bitBrown and FullBrown models use Brown clustering algorithm with specified bit-string lengths (Section 3.2). The JointBrown model takes the union of feature sets with 4-bit, 6-bit and full length prefix. Since different runs of the split-merge HMM with various seeds can result in different clusterings, in AvgHMM we have averaged its dependency parsing results over 5 random runs. We use the Brown algorithm with $K = 512$ for the size of the initial coarse clustering from which the split-merge HMM starts refining (Section 3.3). The JointHMM specifies the joint model of the 5 HMMs. Syn-Low and Syn-High use syntactic clusters described in Section 3.4.

[2] *http://cs.stanford.edu/∼pliang/software/brown-cluster-1.2.zip*

[3] Sentences of the Penn Treebank were excluded from the text used for the clustering.

[4] *code.google.com/p/berkeleyparser*

First order features

Sec	Individual Models								
	Baseline	4BitBrown	6BitBrown	FullBrown	JointBrown	AvgHMM	JointHMM	Syn-Low	Syn-High
00	89.61	90.03	90.19	90.20	90.43	90.54±.08	90.52	90.01	89.97
	34.68	35.67	35.88	36.35	36.71	37.50±.51	37.44	34.42	34.94
01	90.44	91.05	91.27	91.22	91.46	91.59±.07	91.55	90.89	90.76
	36.36	38.22	39.57	39.52	39.12	39.73±.27	39.62	35.66	36.56
23	90.02	90.66	90.84	90.70	91.27	91.35±.02	91.30	90.46	90.35
	34.13	36.78	37.15	37.15	38.77	39.51±.27	39.43	36.95	35.00
24	88.84	89.49	89.47	89.79	90.10	90.03±.06	90.16	89.44	89.40
	30.85	31.44	31.97	33.08	34.72	33.57±.46	35.16	32.49	31.22

Second order features

Sec	Individual Models								
	Baseline	4BitBrown	6BitBrown	FullBrown	JointBrown	AvgHMM	JointHMM	Syn-Low	Syn-High
00	90.34	90.98	91.10	91.02	91.16	91.26±.09	91.27	90.89	90.59
	38.02	40.83	40.36	41.30	40.68	41.31±.13	41.35	38.80	39.16
01	91.48	91.86	92.05	92.06	92.02	92.23±.07	92.22	91.95	91.72
	41.48	43.64	43.19	43.49	43.99	43.54±.34	43.69	42.24	41.28
23	90.82	91.62	91.66	91.59	91.81	91.92±.08	91.90	91.31	91.21
	39.18	42.58	42.66	41.83	42.46	43.22±.53	43.16	40.84	39.97
24	89.87	90.36	90.50	90.53	90.68	90.85±.14	90.78	90.28	90.31
	35.53	36.65	37.62	38.36	37.77	38.15±.33	38.28	37.32	35.61

Table 3.2: For each test section and model, the number in the first/second row is the unlabeled-accuracy/unlabeled-complete-correct. See the text for more explanation about the description of individual models.

We can observe from all of the individual cluster-based models that cluster-based features help improve the accuracy over the baseline in both first order and second order models. Among different clustering methods, HMM clustering provides the best results. We can also notice that HMM clustering outperforms Brown clustering which indicates the advantage of soft clustering of words over hard clustering. The syntactic clustering of the words (Syn-High, Syn-Low) does not seem to be as informative as Brown and HMM clustering. This may be due to the fact that POS categories already capture a good deal of information about the syntactic role of each word, which may overlap with information that are provide by syntactic clusters.

3.7 Summary of the Chapter

In this chapter we provided multiple word representations based on different clustering algorithms to build multiple discriminative dependency parsing models in the MSTParser framework (McDonald, Crammer, and Pereira, 2005). The Brown algorithm provides a hierarchical clustering of the words based on contextual similarity. The HMM state split approach combines the hierarchical characteristics of the Brown clustering with the context-sensitive nature of clusterings based on HMMs. Split non-terminals from PCFG-based Berkeley parser provide syntactic cluster identifiers. All of these diverse clustering-based parsers outperform the baseline parser which does not make use of cluster-based features.

Chapter 4

Ensemble Model

4.1 Introduction

The idea of *ensemble learning* (Dietterich, 2002) is to employ multiple, mutually informative statistical learners and combine their predictions. For the task of parsing, various ensemble techniques have been used. One technique is using *co-training* algorithms, which has shown to be a promising technique for statistical parsing (Sarkar, 2001; Steedman et al., 2003; Hwa et al., 2003). In the statistical approaches to parsing based on the co-training framework, we choose two (or more) different parsers as our *views* of the classification problem. We build separate *models* for each of the parsers and train the models using the labeled examples. At each iteration of the co-training algorithm, we select a set of sentences from the unlabeled examples and employ the models to parse each of the sentences. We then choose a subset of sentences that are parsed with high confidence and add them to the training examples of the other parser. We iterate this procedure until the unlabeled data is treated completely.

Other techniques for system combination include *voting* and *stacking* that have shown to lead to significant improvements in statistical parsing. In voting approach, independently trained models are combined at parsing time, generating a parse tree which is supported by majority of the base systems. Henderson and Brill (1999) take advantage of voting when combining parsers in their *parse hybridization* approach in which they include a constituent appearing in the output of the majority of the parsers in their hypothesized parse. In case of dependency parsing, when selecting the head attachment for each word based on the majority of votes, the output may not result in a well-defined dependency parse tree (it may introduce cycles). In order to ensure the correctness of parse tree, two approaches have

been provided. Zeman and Žabokrtský (2005) provide a greedy algorithm in which at each step during voting, the dependencies that introduce cycles are avoided and if this leads to a situation where there are no allowed dependencies for a word, the existing dependency structure is withdrawn and the parse tree provided by the best parser is selected instead. They observe small degradation in terms of accuracy when this constraint is enforced. Sagae and Lavie (2006) provide another approach in which they reparse the dependency graph which is weighted by the number of votes by using maximum spanning tree algorithm to obtain the optimal dependency parse tree. More concretely, they built a graph in which each word corresponds to a node and weighted directed edges are created corresponding to the dependency arcs in each of the base parse trees. After the graph is built, they reparse the sentence using the existing parsing algorithms (Eisner, 1996; Edmonds, 1967; Chu and Liu, 1965). In their experiments, they observe promising results by combining variations of transition-based dependency parsers with a graph-based dependency parser. Hall et al. (2007) use the proposed methodology by (Sagae and Lavie, 2006) to combine multiple transition-based dependency parsers (using MaltParser (Nivre, 2007)) and they observe improvements over the single Malt.

In the stacking approach, the output of one or more parsers is incorporated as features at training time in order to learn from the prediction of base models. McDonald and Nivre (2007) integrate the transition-based and graph-based models using stacking techniques in which they extend the feature vector for the *base* model by a number of features from the *guide* model. In other words, during training, the *guided* model has access to both the gold dependency tree and the dependency tree predicted by guide model. They perform experiments with both MSTParser and MaltParser as guide model for the other one. Their experimental results show improvements over the base model. Attardi and Dell'Orletta (2009) also use the stacking technique to combine transition-based parsers at training time. They do right-to-left parsing by exploiting features extracted from the output of a left-to-right parser. They also provide a linear time approximate algorithm to combine parsers at parsing time.

In this chapter, we provide an ensemble model, with a different approach than the aforementioned techniques, to combine a collection of diverse and accurate models into a more powerful model.

4.2 Ensemble Model

In Chapter 3 we presented different syntactic or semantic cluster representations. We built the base models based on provided clustering algorithms (Section 3.2, Section 3.4 and Section 3.3) which are incorporated into the model via cluster-based features. The ensemble parsing model that we are providing here is a linear combination of the base models (Haffari, Razavi, and Sarkar, 2011):

$$\text{PARSE}(\mathbf{s}) = \arg\max_{\mathbf{t}\in\mathcal{T}(\mathbf{s})} \sum_k \alpha_k \sum_{r\in\mathbf{t}} \mathbf{w}_k \cdot \mathbf{f}_k(\mathbf{s}, r) \tag{4.1}$$

where α_k is the weight of the kth base model, and each base model has its own feature mapping $\mathbf{f}_k(.)$ based on its clustering annotation of the sentences. The probabilistic motivation for this framework is the product of experts model for parsing (Petrov, 2010). More concretely, we are searching for the tree t that maximizes:

$$P(\mathbf{t}|\mathbf{s}, \mathbf{w}_1, \mathbf{w}_2, ..., \mathbf{w}_k) = \prod_k P(\mathbf{t}|\mathbf{s}, \mathbf{w}_k) \tag{4.2}$$

making the assumption that the individual models are conditionally independent. Then $P(\mathbf{t}|\mathbf{s}, \mathbf{w}_k)$ can be defined as follows:

$$P(\mathbf{t}|\mathbf{s}, \mathbf{w}_k) = \frac{exp(\mathbf{w}_k \cdot \mathbf{f}_k(\mathbf{s}, \mathbf{t}))}{\sum_{t'} exp(\mathbf{w}_k \cdot \mathbf{f}_k(\mathbf{s}, t'))} \tag{4.3}$$

Now, the inference for the best tree amounts to the arg max in equation (4.1) in which each model k is weighted by α_k.

Feature sets of the base parsing models could be concatenated together to train one joint model in a discriminative parsing approach, but we argue that this is not the ideal way to combine word representations based on diverse set of clusterings. We compare the ensemble model with the joint model in which we take the union of all of the feature sets and train a joint discriminative parsing model. The ensemble model outperformed the joint model in almost all of our experiments (Section 4.4.2). Compared to a joint model over all the diverse clustering features, our ensemble method avoids the large memory requirements (during training) of concatenated feature vectors for multiple word representations. It is also more scalable since we can incrementally add a large number of clustering algorithms into the ensemble without an expensive re-training step over the entire training data (e.g., The training step for a joint model consisting of 5 HMM clustering annotations took about

47 hours whereas the training step for each of the models based on 5 HMM clustering annotations was done in parallel which took about 14 hours).

In the next section we inspect different strategies for setting the ensemble weights α_k.

4.3 Setting Ensemble Model Weights for Discriminative Dependency Parsing

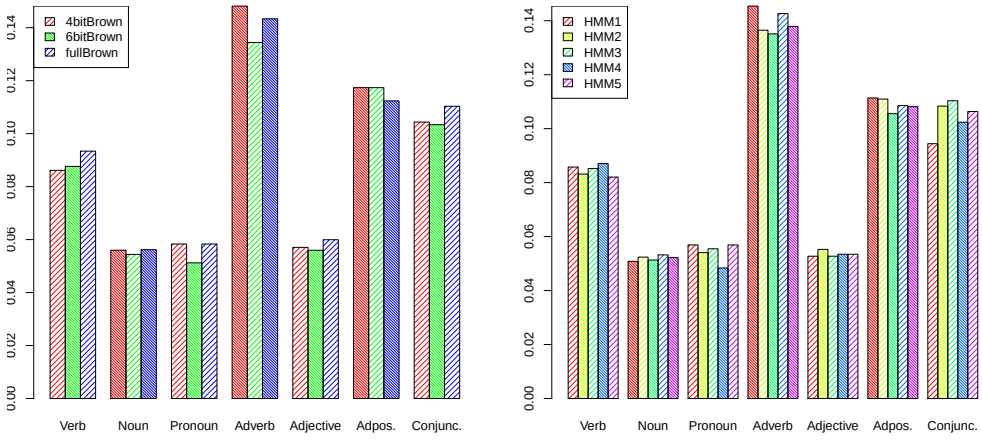
4.3.1 Uniform Setting

One approach for choosing the model weights is the uniform setting in which all the model weights are uniformly set to 1. This approach is in compliance with the reasons provided in (Petrov, 2010) which states that when combining the classifiers with comparable accuracy in the product of experts model, the weighting does not have significant contribution to the overall performance. This is also consistent with the results provided in previous works on the product model (Smith, 2005) in which in case of having classifiers with similar quality, *Logarithm of Opinion Pools for CRF* (LOP-CRF) with uniform weights performs comparable to the one with trained weights. We can see similar results in the study about ensemble models for dependency parsing (Surdeanu and Manning, 2010) which indicates that weighting strategies for voting when reparsing the candidate dependencies is not important.

4.3.2 Learning the Model Weights

Another approach for setting the model weights is through learning the model weights. For this matter, we take a closer look at the parse errors for each of the base models and the ensemble to understand better the contribution of each model to the ensemble. For each dependent to a head dependency, Figure 4.1 shows the error rate for each dependent grouped by a coarse POS tag for different models based on Brown and HMM state splitting clustering algorithms. Each model behaves differently on a different modifier POS category and this property gives us the intuition that it may be fruitful to learn the model weights over the dependency arcs to benefit from each model's strength on a specific modifier POS category. In order to do so, we first rewrite the equation 4.1 as follows:

$$\text{PARSE}(\mathbf{s}) = \arg\max_{\mathbf{t}\in\mathcal{T}(\mathbf{s})} \sum_{r\in\mathbf{t}} \sum_{k} \alpha_k(\mathbf{w}_k \cdot \mathbf{f}_k(\mathbf{s}, r)) \tag{4.4}$$



(a) (b)

Figure 4.1: Error rate of the head attachment for different types of modifier categories on Section 22 of PTB. (a) Brown clustering algorithm. (b) HMM state splitting.

Now we can define the features corresponding to model weights over the dependency arcs $(head, modifier)$ in different ways:

1. based on the granularity of head and modifier POS tags:

   a. fine grained POS tags : The Penn Treebank POS tagset

   b. coarse grained POS tags :

      - first letter of POS tags in Penn Treebank POS tagset
      - few cross-linguistically identifiable categories, including Noun, Pronoun, Verb, Adjective, Adverb, Adposition, Conjunction (c.f. (McDonald and Nivre, 2007))

2. based on elements in dependency arc (head, modifier)

   - both head and modifier
   - modifier only

Table 4.1 shows the PTB tagset as well as the coarse grained POS tags.

In the following sub-sections, we study different strategies for learning the model weights based on different feature definitions and training techniques for them.

1. CC	17. POS	33. WDT
2. CD	18. PRP	34. WP
3. DT	19. PP	35. WP
4. EX	20. RB	36. WRB
5. FW	21. RBR	37. #
6. IN	22. RBS	38. $
7. JJ	23. RP	39. .
8. JJR	24. SYM	40. ,
9. JJS	25. TO	41. :
10. LS	26. UH	42. (
11. MD	27. VB	43.)
12. NN	28. VBD	44. ”
13. NNS	29. VBG	45. ‘
14. NNP	30. VBN	46. ”
15. NNPS	31. VBP	47. ’
16. PDT	32. VBZ	48. ”

(a)

1. C	17. W
2. D	18. #
3. E	19. $
4. F	20. .
5. I	21. ,
6. J	22. :
7. L	23. (
8. M	24.)
9. N	25. ”
10.P	26. ‘
11.R	27. ”
12.S	28. ’
13.T	29. ”
14.U	
15.V	
16.P	

(b)

1. Noun: NN*
2. Verb: VB*, MD
3. Adverb: RB*, WRB
4. Adjective: JJ*
5. Pronoun: PP, WDT, WP
6. Adposition: TO, IN
7. Conjunction: CC

(c)

Table 4.1: Different POS tag categorizations. (a) PTB POS tag set (b), (c) coarse POS tag sets

Learning the Model Weights Using Partial Scores on Dependency Arcs

if we take a closer look at the equation 4.4, we can interpret the model weights α as the *parameters* and the partial scores $\mathbf{w}_k \cdot \mathbf{f}_k(\mathbf{s}, r)$ as *feature values*. More precisely, we can define the feature vectors based on different POS categories mentioned in table 4.1 for either *(head, modifier)* or *modifier* for the base models. Figure 4.2 shows four different templates for the features sets. The feature values are the partial scores on the dev set for the given dependency arc. We can use the MIRA algorithm in the MSTParser framework in order to learn the model weights on the dev set.

Learning the Model Weights Using Binary Indicator Features

One of the drawbacks of setting the feature values to the partial scores for the dependency arc is the following scenario: In cases where there is a high value associated with a dependency arc by mistake, using the partial score as the feature value can exacerbate the situation. In order to avoid this scenario, another approach can be used in which we define binary indicator features as follows:

- Use each of the base models to find the dependency parse trees on the dev set
- Define the indicator features that fire when a base model has predicted the dependency arc correctly

Example Sentence:
While worry grows about big Japanese investments in the U.S., Japan's big trading companies are rapidly increasing their stake in America's smaller business.
Example Edge:
(are,increasing)

1. (hpos,mpos,model)
2. (hposA,mposA,model)
3. (mpos,model)
4. (mposA, model)

(a)

1. (VBP VBG 0) (VBP VBG 1) ... (VBP VBG k-1)
2. (V V 0) (V V 1) ... (V V k-1)
3. (VBG 0) (VBG 1) ... (VBG k-1)
4. (V 0) (V 1) ... (V k-1)

(b)

Figure 4.2: (a) Four different feature templates when learning model weighs based on partial scores. Each of the tuples provides a class of features. Abbreviations: mw: modifier word; mpos: modifier POS tag; mposA: modifier coarse POS tags; hw: head word; hpos: head POS tag; hposA: head coarse POS tag; k : number of base models. (b) Examples of features for the given dependency arc.

Figure 4.3 shows different templates for the feature sets. MIRA algorithm is used in order to learn model weights on the dev set.

When using the POS categories defined in table 4.1 in the feature set, we can modify MIRA such that it updates the weights only based on one POS category to see the effect of each modifier category in improving the accuracy. Algorithm 5 shows the modified version of MIRA algorithm.

Algorithm 5 MIRA Algorithm for Learning Model Weights

Input : Training examples $\{(x_t, y_t)\}_{t=1}^{T}$, N: Number of iterations
$w_0 = 1, v = 0, i = 0$
for n : 1 ... N **do**
  for t : 1 ... T **do**
    $min||w^{(i+1)} - w^{(i)}||$
    s.t.
    $1.s(x_t, y_t) - s(x_t, y') \geq L(y_t, y')$
    $y' = 1best\left(x_t; w^{(i)}\right)$
    2. only update the weights for the specified modifier category
    $v = v + w^{(i+1)}$
    $i = i + 1$
  end for
end for
$w = v/NT$

Example Sentence:
While worry grows about big Japanese investments in the U.S., Japan's big trading companies are rapidly increasing their stake in America's smaller business.
Example Edge:
(are, increasing)

1. (hpos,mpos,model)
2. (hposA,mposA,model)
3. (cahpos,campos,model)
4. (mpos,model)
5. (mposA, model)
6. (campos, model)

(a)

1. (VBP VBG 0) (VBP VBG 1) ... (VBP VBG k-1)
2. (V V 0) (V V 1) ... (V V k-1)
3. (Verb Verb 0) (Verb Verb 1) ... (Verb Verb k-1)
4. (VBG 0) (VBG 1) ... (VBG k-1)
5. (V 0) (V 1) ... (V k-1)
6. (Verb 0) (Verb 1) ... (Verb k-1)

(b)

Figure 4.3: (a) Example of different possible feature templates when learning model weights based on binary indicator features. each of the tuples provides a class of indicator features. Abbreviations: campos: categorized modifier POS tag; cahpos: categorized head POS tag; k : number of base models. (b) Examples of features for the given dependency arc.

4.4 Experiments

4.4.1 Experimental Setup

We conduct experiments to evaluate the performance of the provided ensemble method on two scenarios: in-domain and out-of-domain settings. In the following, we first describe our data conditions.

Data For Experiments with Uniform Model Weights: The settings for in-domain experiments are the same as experimental settings provided in Section 3.6.

The out-of-domain experiments are done on the Brown (Kučera and Francis, 1967) and SwitchBoard (Godfrey, Holliman, and McDaniel, 1992) corpora. We filtered these two corpora to those sentences which have at least one unknown word with respect to the Penn Treebank training data for our in-domain experiments and whose length is at least 5 words. Using standard head-percolation rules, we convert the gold phrase-structure parse trees of the selected sentences into dependency trees. We use MXPOST trained on the PTB training data to annotate these sentences with POS tags. The size/average sentence length of the resulting test sets for the SwitchBoard and Brown corpora are 2720/13 and 4489/20, respectively.

Data For Experiments with Trained Model Weights: For the in-domain experiments, we use Section 22 as dev set for training the model weights based on partial scores (Section 4.3.2) and test on Section 23 of PTB. We use section 0 to train the model weights using binary indicator features (Section 4.3.2) and test on Section 22 and 23 of PTB. For out-of-domain experiments, we split the Brown and SwitchBoard corpus into 2 segments as dev set and test set. For Brown corpus the dev/test sets contain 2000/2489 sentences respectively. The SwitchBoard corpus contains 1720/1000 sentences in its dev/test set. The rest of data conditions are consistent with the experiments with uniform model weights.

4.4.2 Empirical Results in In-domain Experiments

For the in-domain experiments, we train our dependency parsing models on the PTB training data.

In-domain Results with Uniform Model Weights

Table 4.2 presents our results with all the model weights uniformly set to 1. First part of this table contains the baseline and the base models as mentioned in Chapter 3. The second part of Table 4.2 shows the results for our ensemble models; the columns present the results for the ensemble of three/three/five/eight/ten base models, respectively. As Table 4.2 shows, the ensemble models outperform the individual models in all cases.

We can also observe that the ensemble is outperforming the joint model in almost all cases. The ensemble models outperform all of the individual models and do so very consistently across both first-order and second-order dependency models. This improvement in accuracy is well illustrated in Figure 4.5. It is also interesting to see that the ensemble of 5 HMM-based parsing models, which does not use any syntactic clustering, is highly competitive. It outperforms the ensemble of JointBrown+Syn-Low+Syn-High in almost all of the cases.

Table 4.4 lists the accuracy of the ensemble model on Section 23 of the PTB together with the state-of-the-art graph-based second-order dependency parser and some of the relevant results from related works. The ensemble model behaves comparably to most of other dependency parsing models. Since our features do not exactly match the (Koo, Carreras, and Collins, 2008), the accuracy numbers we get are also slightly divergent from those in their paper.

First order features

Sec	Individual Models									Ensembles				
	Baseline	4BitBrown	6BitBrown	FullBrown	JointBrown	AvgHMM	JointHMM	Syn-Low	Syn-High	3Brown	JointBrown+Syn-Low+Syn-High	5HMM	5HMM+3Brown	5HMM+3Brown+Syn-(Low, High)
00	89.61	90.03	90.19	90.20	90.43	90.54±.08	90.52	90.01	89.97	90.56	90.83	90.91	90.97	**91.15**
	34.68	35.67	35.88	36.35	36.71	37.50±.51	37.44	34.42	34.94	37.5	37.96	39.06	38.54	**39.53**
01	90.44	91.05	91.27	91.22	91.46	91.59±.07	91.55	90.89	90.76	91.53	91.75	92.00	91.94	**92.08**
	36.36	38.22	39.57	39.52	39.12	39.73±.27	39.62	35.66	36.56	40.58	40.23	40.78	41.53	**41.53**
23	90.02	90.66	90.84	90.70	91.27	91.35±.02	91.30	90.46	90.35	91.12	91.20	91.62	91.55	**91.64**
	34.13	36.78	37.15	37.15	38.77	39.51±.27	39.43	36.95	35.00	38.73	38.98	40.51	40.47	**40.72**
24	88.84	89.49	89.47	89.79	90.10	90.03±.06	90.16	89.44	89.40	90.17	90.46	90.42	90.45	**90.61**
	30.85	31.44	31.97	33.08	34.72	33.57±.46	35.16	32.49	31.22	34.49	35.61	34.86	35.83	**36.05**

Second order features

Sec	Individual Models									Ensembles				
	Baseline	4BitBrown	6BitBrown	FullBrown	JointBrown	AvgHMM	JointHMM	Syn-Low	Syn-High	3Brown	JointBrown+Syn-Low+Syn-High	5HMM	5HMM+3Brown	5HMM+3Brown+Syn-(Low, High)
00	90.34	90.98	91.10	91.02	91.16	91.26±.09	91.27	90.89	90.59	91.37	91.53	91.53	91.59	**91.74**
	38.02	40.83	40.36	41.30	40.68	41.31±.13	41.35	38.80	39.16	41.82	41.87	41.82	**42.65**	42.55
01	91.48	91.86	92.05	92.06	92.02	92.23±.07	92.22	91.95	91.72	92.27	92.50	92.51	92.60	**92.68**
	41.48	43.64	43.19	43.49	43.99	43.54±.34	43.69	42.24	41.28	45.15	45.05	44.75	**45.75**	45.70
23	90.82	91.62	91.66	91.59	91.81	91.92±.08	91.90	91.31	91.21	91.99	92.02	92.26	92.37	**92.46**
	39.18	42.58	42.66	41.83	42.46	43.22±.53	43.16	40.84	39.97	44.11	43.78	44.15	44.98	**45.48**
24	89.87	90.36	90.50	90.53	90.68	90.85±.14	90.78	90.28	90.31	90.96	91.19	91.31	91.32	**91.55**
	35.53	36.65	37.62	38.36	37.77	38.15±.33	38.28	37.32	35.61	39.03	39.85	39.85	40.74	**41.11**

Table 4.2: For each test section and model, the number in the first/second row is the unlabeled-accuracy/unlabeled-complete-correct. See the text for more explanation about the description of individual and ensemble models. (3Brown is the ensemble of 4BitBrown, 6BitBrown and FullLengthBrown)

Sec	$K = 16$			$K = 64$			$K = 512$		
	JointBrown	AvgHMM	5HMM	JointBrown	AvgHMM	5HMM	JointBrown	AvgHMM	5HMM
00	90.59	91.02±.15	**91.42**	90.94	91.28±.03	**91.62**	91.16	91.26±.09	**91.53**
	39.84	40.44±.83	**41.77**	40.78	41.89±.43	**42.55**	40.68	41.31±.13	**41.82**
01	91.74	91.96±.08	**92.33**	91.97	92.15±.07	**92.43**	92.02	92.23±.07	**92.51**
	41.94	42.28±.62	**44.45**	43.59	43.44±.35	**44.90**	43.99	43.54±.34	**44.75**
23	91.33	91.49±.16	**92.05**	91.49	91.93±.07	**92.33**	91.81	91.92±.08	**92.26**
	41.92	41.48±.46	**43.37**	41.13	43.50±.52	**45.19**	42.46	43.22±.53	**44.15**
24	90.54	90.39±.11	**90.89**	90.60	90.76±.11	**91.17**	90.68	90.85±.14	**91.31**
	37.62	37.20±.95	**38.21**	38.21	38.13±.72	**39.40**	37.77	38.15±.33	**39.85**

Table 4.3: For different size K of initial coarse annotations, comparing the ensemble of five split-merge HMM models vs the individual models and the Baseline.

Parser Accuracies		
	Model	UAS
Baselines	McDonald and Pereira (2006)	91.5
	Koo, Carreras, and Collins (2008)-Standard	92.00
	Koo, Carreras, and Collins (2008)-Semi Supervised	93.16
Combined Parser	Sagae and Lavie (2006)	92.7
Ensemble	3Brown+5HMM+Syn-(Low, High)	92.46

Figure 4.4: Comparing Unlabeled Attachment Score (UAS) of the ensemble model with the state-of-the-art graph-based second-order dependency parser and some of the relevant works on Section 23 of PTB.

We also investigated the effect of the size for the initial coarse clustering, from which the split-merge HMM starts the annotation refinement. The rationale for this comparison is to show that the selective splits obtained by running the split-merge HMM provide better clusters than simply taking a more fine-grained clustering directly from the (Brown et al., 1992) algorithm. We annotate the sentences with $K \in \{16, 64, 512\}$ coarse clusters obtained from the (Brown et al., 1992) algorithm, and then run five random split-merge HMMs. Table 4.3 presents the parsing results. The performance of the Brown algorithm improves as K is increased. However, the ensemble of HMMs using the selective split-merge training

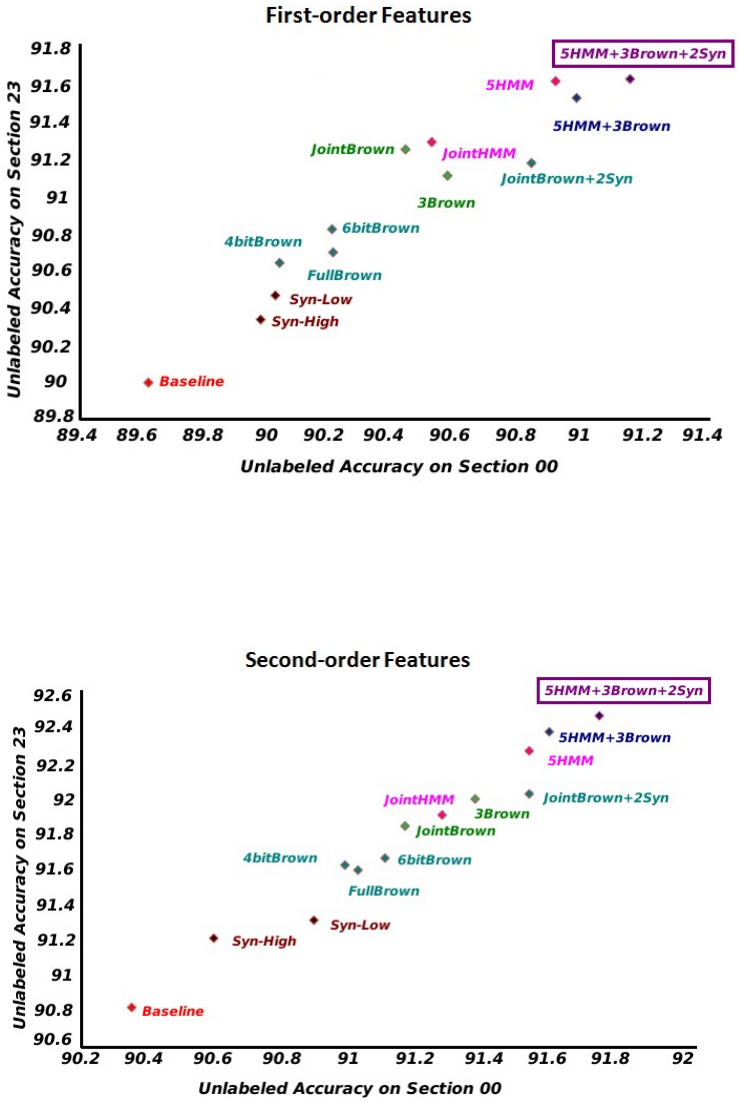


Figure 4.5: The unlabeled accuracy for individual and ensemble models. The x and y axis show the accuracy on Section 00 and 23 of PTB respectively. (a) First order features. (b) Second order features.

always does better. Although the ensemble of 5 HMMs outperforms the individual models for each K, the performance of the ensemble model seems to degrade as we increase K in some test sets, e.g. for Section 23 the performance drops from 92.33% to 92.26% as K is increased from 64 to 512. One explanation is that the diversity among the individual HMM clusterings becomes negligible as we increase K. Therefore, the performance of the resulting ensemble model may degrade since the success of an ensemble model highly correlates with the diversity among the individual models.

In-domain Results with Trained Model Weights

In order to observe the effect of learning model weights, we perform experiments on the 3 Brown ensemble model using first order feature vectors. Table 4.4 shows our results with the model weights learned based on the partial scores (Section 4.3.2). We study different POS categorization scenarios as our features set based on either both the head and modifier for each dependency arc or just the modifier. The model weights are trained on Section 22 of PTB. It seems that using POS of both head and modifier leads to better results compared to other scenarios (improvement from 91.12% to 91.13%). However, learning the model weights based on partial scores does not seem to be effective to the overall accuracy.[1]

Table 4.5 presents our results based on different scenarios explained in Section 4.3.2 using binary indicator features. The results for training the model weights with the modified version of MIRA (Algorithm 5) are shown as *MIRA-Modified* in the table. For each POS category specified in the table, each number represents the unlabeled accuracy when applying only the updated weight based on that POS category. We can observe from the table that *Verb*, *Adjective* and *Conjunction* POS categories seem to be effective in improving the accuracy on Section 22. More specifically, the improvement in unlabeled complete-correct accuracy is significant in some cases (e.g., from 40.55% to 40.96% in Verb column). On the other hand, for Section 23 of PTB, exploiting updated model weights based on Pronoun category seem to be helpful in terms of accuracy. Table 4.6 shows the results when using the updated weights based on different combinations of POS categories. Similar to the results provided in table 4.5 we can see some notable improvements on Section 22. Using combinations of POS categories seems to be helpful especially in improving complete-correct

[1]We observed degradation in accuracy when increasing the number of iterations in MIRA algorithm in some cases which can be due to overfitting.

accuracy (e.g., the complete-correct accuracy for Verb/Pr is 40.78%/40.49% respectively while for Verb+Pr is 40.90%).

Section	Ensemble-Uniform	POS	Coarse POS	Modifier-POS	Modifier-Coarse POS
23	91.12	**91.13**	91.12	91.11	91.10
	38.73	**38.73**	38.73	38.52	38.60

Table 4.4: The effect of learning the model weights based on partial scores from dev set on Section 23 of PTB.

Training Algorithm	Section	Ensemble-Uniform	Verb	Noun	Adjective	Adverb	Adposition	Conjunction	Pronoun
MIRA	22	91.31	**91.38**	91.27	91.31	91.30	91.31	91.31	91.32
		40.55	**40.78**	40.72	40.55	40.43	40.37	40.61	40.49
	23	91.12	91.13	91.00	**91.13**	91.12	91.10	91.11	**91.14**
		38.73	38.69	38.69	**38.77**	38.73	38.44	38.73	**38.77**
MIRA-Modified	22	91.31	**91.33**	91.27	**91.33**	91.31	91.12	**91.32**	91.32
		40.55	**40.96**	40.72	**40.61**	40.20	39.78	**40.61**	40.49
	23	91.12	91.04	91.08	91.10	91.12	90.97	91.09	**91.14**
		38.73	38.77	38.52	38.48	38.81	37.11	38.73	**38.77**

Table 4.5: The effect of using different modifier POS categories on the accuracy of the ensemble Brown on Section 22 and 23 of PTB using two different learning strategies.

4.4.3 Empirical Results in Out-of-domain Experiments

For the out-of-domain experiments, we train our dependency parsing models on the PTB training data, and test on the Brown and SwitchBoard test sets. The SwitchBoard corpus contains phone conversations which are radically different from the news articles that comprise the Penn Treebank corpus, while the sections of the Brown corpus we used in our experiments are taken from the press reviews genre/domain which is closer to the Treebank text in terms of vocabulary and syntax. Hence, compared to the Brown corpus, the SwitchBoard corpus has a larger domain divergence to the WSJ.

Ensemble Model	Section	Ensemble-Uniform	Verb+Pr	Verb+Conj	Noun+Conj
3Brown	22	91.31	**91.32**	91.31	91.23
		40.55	**40.90**	**40.96**	**40.78**

Table 4.6: The effect of using different combinations of modifier POS categories on Section 22 of PTB.

The intuition behind the benefit of an ensemble of 5 HMMS for dependency in out-of-domain experiments is well illustrated in figures 4.6 and 4.7. We selected two words *stuff* and *got* which are frequent words in the SwitchBoard corpus that are uncommon in the WSJ corpus. Figure 4.6 shows the set of words that are clustered with *stuff* in each of the 5 HMM models. Diversity of the words that are clustered with *stuff* in each model shows that using the ensemble of HMMs can lead to a better model.

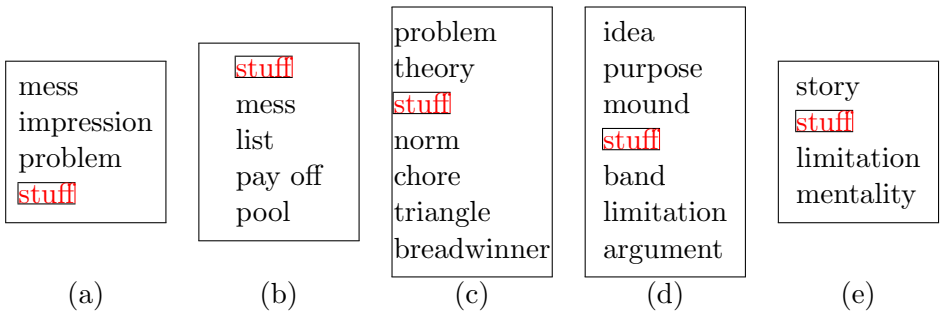


Figure 4.6: The word *stuff* is clustered with diverse set of words in each of the HMM models. figures a-e correspond to clusterings in HMM1-HMM5 respectively.

Out-of-domain Results with Uniform Model Weights

Table 4.7 presents our results with the model weights uniformly set to 1. In all cases, we see a consistent improvement in the performance of the ensemble model compared to the individual models and the baseline. As expected, because of the higher domain divergence of the SwitchBoard compared to the Brown corpus, the ensemble performance improvement for the SwitchBoard is more than that for the Brown in Table 4.7.

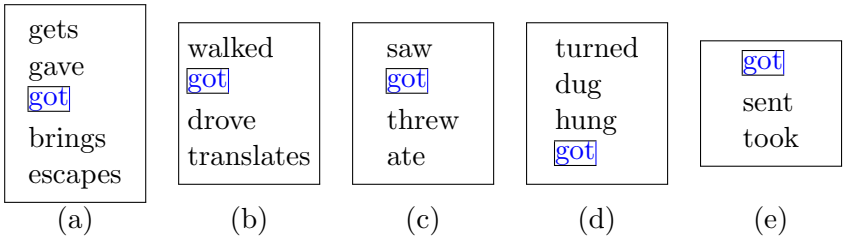


Figure 4.7: The word *got* is clustered with diverse set of words in each of the HMM models . figures a-e correspond to clusterings in HMM1-HMM5 respectively.

corpus	order	Baseline	JointBrown	4bitBrown	6bitBrown	FullBrown	AvgHMM	3Brown	5HMM
SwBd	1st	74.96	76.56	76.02	75.86	76.18	76.71 ±.10	76.42	**77.06**
		30.58	31.98	30.62	31.39	31.58	32.07 ±.35	31.72	**32.83**
	2nd	75.23	77.08	75.97	76.47	76.58	76.88 ±.18	76.88	**77.23**
		31.8	33.12	32.05	33.05	33.30	33.10 ± .33	**33.75**	33.45
Brown	1st	84.01	85.16	84.74	85.13	85.06	85.23 ±.13	85.45	**85.59**
		26.68	28.60	28.55	29.16	28.84	29.10 ±.31	**29.76**	29.62
	2nd	84.69	85.88	85.43	85.82	85.92	86.00 ± .08	86.20	**86.43**
		29.27	31.90	30.69	31.47	31.78	32.02 ± .44	32.41	**32.65**

Table 4.7: The Baseline (no cluster) vs individual cluster-based models as well as an Ensemble of 3 Browns and 5 HMMs for the SwitchBoard and Brown corpora.

We repeated the out-of-domain experiments with gold POS tags to investigate how much of the divergence between domains is due to errors made in the POS tagger. We saw a significant improvement in the parsing performance of the ensembles[2] from 77.06/77.23 to 81.55/81.80 for the first/second order models; this is almost 20% error reduction. Further error reduction $\sim 37\%$ is needed to reach the performance level of the in-domain parsers (i.e. around 90), which we believe can be achieved by capturing mainly the syntactic divergence between the SwitchBoard and WSJ domains.

[2]We observed similar improvements for other base models.

Out-of-domain Results with trained Model Weights

To observe the impact of learning model weights on out-of-domain data, we perform experiments on the ensemble model of 3 Browns and 5 HMMs using first order feature vectors. Table 4.8 shows the results on Brown corpus. For both of the ensemble models, it seems that employing the updated weights based on *Adjective* and *Adposition* is more helpful than others. But the improvements in accuracy are not considerable.

Table 4.9 demonstrates the results on the SwitchBoard corpus. We can see that in both of the ensemble models, using the updated weights based on *Adposition* is more fruitful than the others. The improvements in terms of accuracy are slightly more noticeable than the results provided in table 4.8 (from 76.48% to 76.62% in Adposition column). The results provided in both of the tables reveals that using the modified version of MIRA can lead to better results for out-of-domain experiments in most of the cases.

Our experimental results on learning the model weights in both in-domain and out-of-domain scenarios reveal that even though learning the model weights can lead to improvements in some cases, it does not have an important contribution to the overall accuracy. This is similar to the observation by (Zeman and Žabokrtský, 2005) who tried context sensitive voting in which the contexts (such as morphological tags) were trained on the dev set. They found that there is not enough predictive power in the context information to outperform simple voting.[3]

4.4.4 Error Analysis

To better understand the contribution of each model to the ensemble, we take a closer look at the parsing errors for each model and the ensemble. For each dependent to a head dependency, we compute the error rate of each modifier grouped by a categorized POS tag (as illustrated in table 4.1). We also calculate the F-score (harmonic mean of precision and recall) of each of the different models for various dependency lengths, where the length of a dependency from word w_i to word w_j is equal to $|i - j|$. Precision is the percentage of dependency arcs in the predicted graph that are at a distance of $|i - j|$ and are in the gold graph. Recall is the percentage of dependency arcs in the gold standard graph that are at a distance of $|i - j|$ and were predicted. Figure 4.8 shows the error rate and F-score of the

[3]We also tried using the accuracy of each parser model on dev set as the model weight similar to (Sagae and Lavie, 2006). We observed negligible improvements in terms of accuracy (from 91.12% to 91.13%)

Training Algorithm	order	Ensemble-Uniform	Verb	Noun	Adjective	Adverb	Adposition	Pronoun
MIRA	3Brown	85.04	84.55	84.89	85.03	84.98	85.01	85.01
		28.28	27.19	27.80	28.20	28.16	28.16	28.22
	5HMM	85.12	84.36	84.98	**85.14**	85.07	**85.14**	85.12
		28.36	26.11	27.76	28.32	28.28	28.28	28.32
MIRA-Modified	3Brown	85.04	84.93	84.92	**85.05**	84.99	85.01	85.03
		28.28	28.20	27.88	28.28	28.24	28.16	**28.29**
	5HMM	85.12	85.03	85.01	**85.15**	85.08	**85.17**	85.12
		28.36	27.76	28.12	28.32	28.32	28.32	28.28

Table 4.8: The effect of using different modifier POS categories on the accuracy of the ensemble Brown and ensemble HMM on Brown corpus using two different learning strategies.

Training Algorithm	Ensemble Model	Ensemble-Uniform	Verb	Noun	Adjective	Adverb	Adposition	Pronoun	Conjunction
MIRA	3Brown	75.79	74.52	75.66	75.69	75.76	**75.92**	75.71	75.66
		27.00	26.0	26.7	26.7	26.9	**27.0**	26.8	26.8
	5HMM	76.48	74.99	76.30	76.36	76.32	**76.58**	76.44	76.30
		28.4	26	27.8	28.2	27.9	28.2	28.2	28.2
MIRA-Modified	3Brown	75.79	74.94	75.76	75.73	75.76	**75.95**	74.82	75.70
		27	26.5	26.8	26.9	26.9	26.7	26.3	26.8
	5HMM	76.48	75.37	76.33	76.38	76.33	**76.62**	76.48	76.36
		28.4	27.2	28.2	28.2	28.0	**28.4**	28.3	28.3

Table 4.9: The effect of using different modifier POS categories on the accuracy of the ensemble Brown and ensemble HMM on SwitchBoard corpus using two different learning strategies.

different base and ensemble models respectively.

We see that different models are experts on different dependency lengths; For instance, for the ensemble of 3 Browns, 6bitBrown and FullBrown are experts at dependency of length 9 and 11 respectively, while the ensemble model can always combine their expertise and do

better at each length. This observation holds for other ensemble models in Figure 4.8.

Now let us investigate the type of errors made for the out-of-domain experiments. Figure 4.9 plots the head attachment error rate for each grammatical category and the F-score for different dependency lengths for the 3Brown on SwitchBoard corpus and for the 5HMM on Brown and SwitchBoard corpus. The ensemble models 3Brown and 5HMM behave similar to the in-domain scenario on SwitchBoard and Brown corpus respectively. The ensemble model of 5 HMMs on the SwitchBoard corpus however behaves somehow differently. Interestingly, the average of five HMMs and their ensemble perform worse than the Brown clustering for Pronoun, Verb, and Adverb. This means that the initial coarse annotation turns out to be just right for these categories, and as a result over-splitting them using the HMM hurts the parsing performance.

4.5 Summary of the Chapter

In this chapter we provided an ensemble of different dependency parsing models, each model corresponding to a different word clustering annotation. We have shown that either using slightly different clustering annotations from the same clustering method (e.g., 3Brown, 5HMM) or using different clustering annotations from different clustering methods (e.g., JointBrown+Syn-Low+Syn-High) results in a more powerful model. The ensemble model obtains consistent improvements in unlabeled dependency parsing, e.g. from 90.82% to 92.46% for Sec. 23 of the Penn Treebank (Significant 17% error reduction, reducing the number of errors from 5200 to 4272). We also investigated the effect of learning the model weights on the performance of the dependency parser. Our results show that simple uniform ensemble model performs essentially as well as other more complex models. Our error analysis has revealed that each parsing model is an expert in capturing different dependency lengths, and the ensemble model can always combine their expertise and do better at each dependency length.

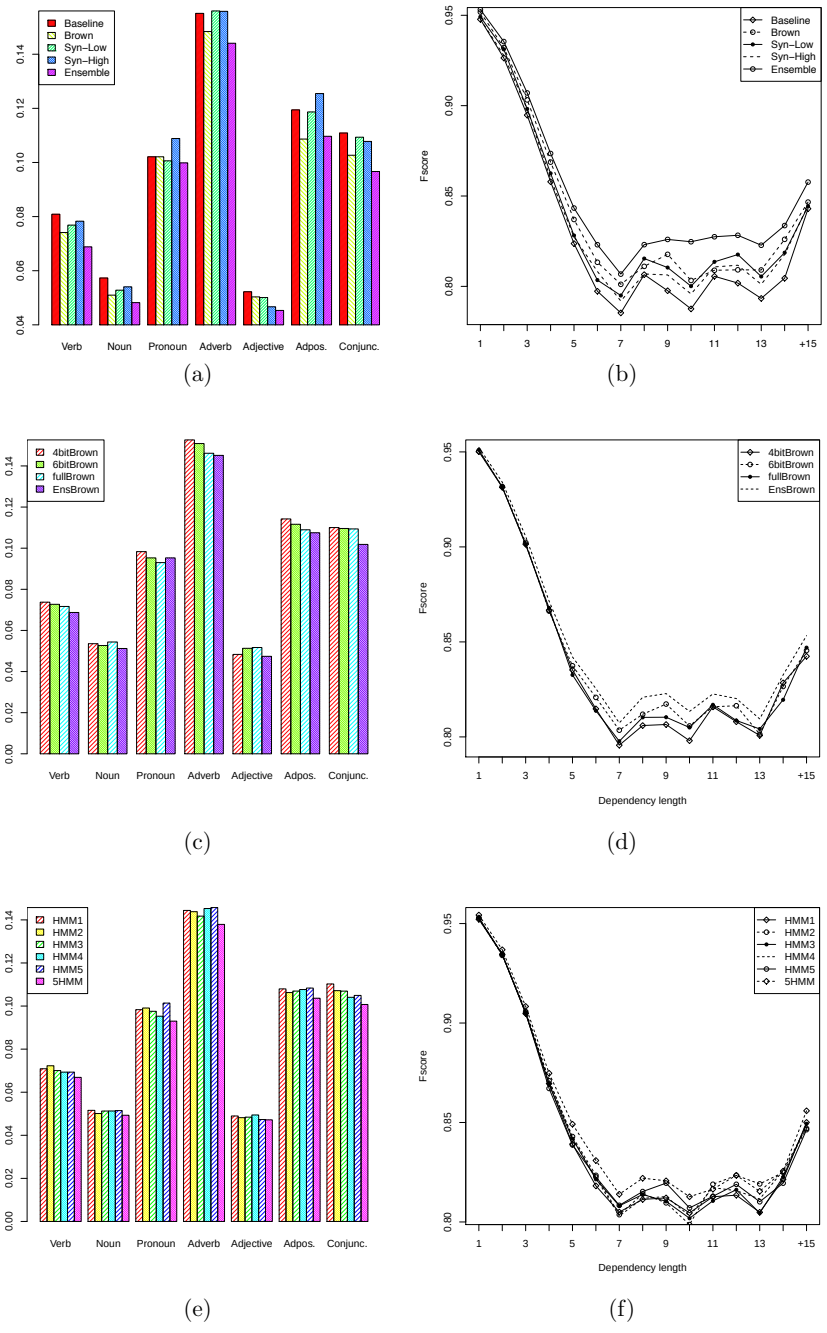


Figure 4.8: Error Analysis of different Ensemble models on in-domain experiments. First column shows the error rates of the head attachment for different types of modifier categories. the second column represents F-scores for each dependency length.

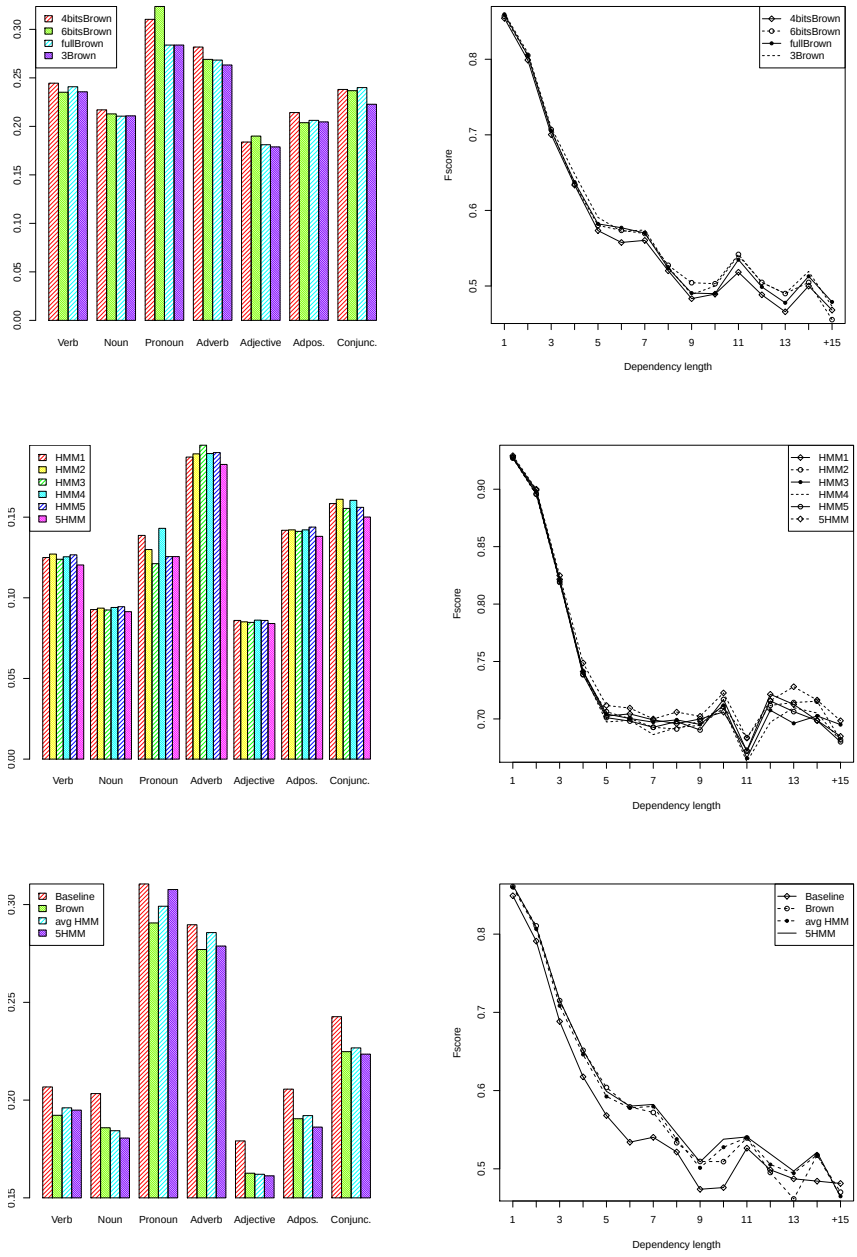


Figure 4.9: Error Analysis of different Ensemble models on out-of-domain experiments. First column shows the error rates of the head attachment for different types of modifier categories. the second column represents F-scores for each dependency length.

Chapter 5

Conclusion and Future Work

This chapter provides some brief concluding points and discusses topics for future research directions.

5.1 Thesis Summary

In this thesis, we provided multiple word representations based on three different clustering methods: The Brown clustering algorithm (Section 3.2), the split-merge HMM (Section 3.3) and the split non-terminals from Berkeley parser (Section 3.4). We then provided an ensemble model for combining the word representations together in the MSTParser framework (Chapter 4). We have shown that an ensemble of different dependency parsing models results in a more powerful model which obtains consistent significant improvements in unlabeled dependency parsing, e.g. from 90.82% to 92.46% for Section 23 of the Penn Treebank. Furthermore, we have shown the strength of our ensemble model in the domain adaptation scenario: on Switchboard data we improve accuracy from 75.23% to 77.23%, and on the Brown corpus we improve accuracy from 84.69% to 86.43%. We also investigated the effect of learning the model weights for the ensemble model and our experimental results show that the simple unweighted ensemble model performs essentially as well as other more complex models. Our error analysis has revealed that each parsing model is an expert in capturing different dependency lengths, and the ensemble model can always combine their expertise and do better at each dependency length. We can incrementally add a large number models using different clustering algorithms, and our results show increased improvement in accuracy when more models are added into the ensemble.

5.2 Future Work

The works in this thesis can be extended in different aspects:

- *Enhancing the existing clustering methods*: In Brown clustering algorithm we used bit-strings of fixed lengths. Instead of cutting the hierarchy at a fixed depth (4 and 6), we can exploit techniques that can choose to use shorter or longer prefixes at different points in the hierarchy, in order to provide a more balanced hierarchy.

- *Trying other approaches for clustering or combining the parsers*: It would be interesting to use other word representations such as distributed word representations (vector of reals) as opposed to clustering identifiers and combine both kinds of representations for dependency parsing. Using multiple word representations has shown to be successful for other NLP tasks such as chunking and named entity recognition (Turian, Ratinov, and Bengio, 2010).

  Also, it seems intriguing to try other approaches for combining the parse models and compare it with our ensemble model. One such approach can be using the Matrix Tree Theorem for computing the dependency arc marginals (Koo et al., 2007) under each of the models and taking their product.

- *Experimenting on other languages*: Our experiments were done on English Treebank. It would be interesting to see how the ensemble model performs on other languages.

- *Domain adaptation scenario*: Further improvement in the domain adaptation scenario should be achievable by designing models which aim to better capture the syntactic divergence between the source and target domains.

References

Attardi, G. and F. Dell'Orletta. 2009. Reverse revision and linear tree combination for dependency parsing. In *Proceedings of NAACL-HLT*.

Bartlett, P., B. Taskar, M. Collins, and D. Mcallester. 2004. Exponentiated gradient algorithms for large-margin structured classification. In *Proceedings of NIPS*.

Boser, B., I. Guyon, and V. Vapnik. 1992. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, COLT '92, pages 144–152, New York, NY, USA. ACM.

Breiman, L. 1996. Bagging predictors. *Mach. Learn.*, 24.

Brown, P. F., P. V. deSouza, R. L. Mercer, T. J. Watson, V. J. Della Pietra, and J. C. Lai. 1992. Class-based n-gram models of natural language. *Computational Linguistics*, 18(4).

Carreras, X. 2007. Experiments with a higher-order projective dependency parser. In *Proceedings of EMNLP-CoNLL Shared Task*.

Censor, Y. and S. Zenios. 1997. *Parallel Optimization: Theory, Algorithms and Applications*. Oxford University Press.

Charniak, E. 2000. A maximum-entropy-inspired parser. In *Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference*, NAACL 2000, pages 132–139, Stroudsburg, PA, USA. Association for Computational Linguistics.

Charniak, E., D. Blaheta, N. Ge, K. Hall, and M. Johnson. 2000. *BLLIP 1987-89 WSJ Corpus Release 1, LDC No. LDC2000T43*, Linguistic Data Consortium.

Chu, Y. J. and T. H. Liu. 1965. On the shortest arborescence of a directed graph. *Science Sinica*, 14:1396–1400.

Collins, M. 1999. *Head-driven statistical models for natural language parsing*. Ph.D. thesis, Philadelphia, PA, USA. AAI9926110.

Collins, M. 2002. Discriminative training methods for hidden markov models: theory and experiments with perceptron algorithms. In *Proceedings of EMNLP*.

Crammer, K., O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer. 2006. Online passive-aggressive algorithms. *J. Mach. Learn. Res.*, 7:551–585, December.

Crammer, K. and Y. Singer. 2003. Ultraconservative online algorithms for multiclass problems. *J. Mach. Learn. Res.*, 3:951–991.

Culotta, A. 2004. Dependency tree kernels for relation extraction. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL-04*, pages 423–429.

Dietterich, T. 2002. *Ensemble learning.* In The Handbook of Brain Theory and Neural Networks, Second Edition.

Ding, Y. and M. Palmer. 2005. Machine translation using probabilistic synchronous dependency insertion grammars.

Edmonds, J. 1967. Optimum branchings. *Research of the National Bureau of Standards*, 71B:233–240.

Eisner, J. 1996. Three new probabilistic models for dependency parsing: an exploration. In *COLING.*

Eisner, J. 2000. Bilexical grammars and their cubic-time parsing algorithms. In Harry Bunt and Anton Nijholt, editors, *Advances in Probabilistic and Other Parsing Technologies.* Kluwer Academic Publishers, October, pages 29–62.

Finkel, J. and C. Manning. 2009. Nested named entity recognition. In *EMNLP*, pages 141–150.

Fishel, M. and J. Nivre. 2009. Voting and stacking in data-driven dependency parsing. In *Proceedings of the 17th Nordic Conference on Computational Linguistics NODALIDA'2009*, pages 219–222, Odense, Denmark.

Godfrey, J., E. Holliman, and J. McDaniel. 1992. Switchboard: Telephone speech corpus for research development. In *Proceedings of IEEE ICASSP.*

Haffari, G., M. Razavi, and A. Sarkar. 2011. An ensemble model that combines syntactic and semantic clustering for discriminative dependency parsing. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers - Volume 2*, HLT '11, Stroudsburg, PA, USA. Association for Computational Linguistics.

Hall, J., J. Nilsson, J. Nivre, G. Eryigit, B. Megyesi, M. Nilsson, and M. Saers. 2007. Single malt or blended? a study in multilingual parser optimization. In *Proceedings of CoNLL Shared Task.*

Henderson, J. and E. Brill. 1999. Exploiting diversity in natural language processing: Combining parsers. In *Proceedings of the Fourth Conference on Empirical Methods in Natural Language Processing*, pages 187–194.

Hermjakob, U. 2001. Parsing and question classification for question answering. In *Proceedings of the Workshop on Open-Domain Question Answering at ACL-2001.*

Huang, L. 2006. Statistical syntax-directed translation with extended domain of locality. In *Proceedings AMTA 2006*, pages 66–73.

Huang, Z., V. Eidelman, and M. Harper. 2009. Improving Simple Bigram HMM Part-of-Speech Tagger by Latent Annotation and Self-Training. In *Proceedings of NAACL-HLT.*

Hwa, R., M. Osborne, A. Sarkar, and M. Steedman. 2003. Corrected co-training for statistical parsers. In *In ICML-03 Workshop on the Continuum from Labeled to Unlabeled Data in Machine Learning and Data Mining*, pages 95–102.

Koo, T. 2010. *Advances in discriminative dependency parsing*. Ph.D. thesis, Cambridge, MA, USA. AAI0822900.

Koo, T., X. Carreras, and M. Collins. 2008. Simple semi-supervised dependency parsing. In *Proceedings of ACL/HLT*.

Koo, T. and M. Collins. 2010. Efficient third-order dependency parsers. In *Proceedings of ACL*.

Koo, T., A. Globerson, X. Carreras, and M. Collins. 2007. Structured prediction models via the matrix-tree theorem. In *In EMNLP-CoNLL*.

Koo, T., A. Rush, M. Collins, T. Jaakkola, and D. Sontag. 2010. Dual decomposition for parsing with non-projective head automata. In *Proceedings of EMNLP*.

Kübler, S., R. McDonald, and J. Nivre. 2009. *Dependency parsing*. Morgan and Claypool Publishers.

Kučera, H. and W. Nelson Francis. 1967. *Computational analysis of present-day American English*. Brown University Press.

Liang, P. 2005. Semi-supervised learning for natural language. Master's thesis, Massachusetts Institute of Technology.

Marcus, M., M. Marcinkiewicz, and B. Santorini. 1993. Building a large annotated corpus of english: the penn treebank. *Computational Linguistics*, 19(2):313–330, June.

McDonald, R., K. Crammer, and F. Pereira. 2005. Online large-margin training of dependency parsers. In *Proceedings of ACL*.

Mcdonald, R., K. Lerman, and F. Pereira. 2006. Multilingual dependency analysis with a two-stage discriminative parser. In *Proceedings of the Conference on Computational Natural Language Learning (CoNLL*, pages 216–220.

McDonald, R. and J. Nivre. 2007. Characterizing the errors of data-driven dependency parsing models. In *Proceedings of EMNLP-CONLL*.

McDonald, R. and F. Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *Proceedings of EACL*.

McDonald, R., F. Pereira, K. Ribarov, and J. Hajič. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, HLT '05, pages 523–530, Stroudsburg, PA, USA. Association for Computational Linguistics.

Mel'čuk, I. 1987. *Dependency syntax: theory and practice*. State University of New York Press.

Miller, S., J. Guinness, and A. Zamanian. 2004. Name tagging with word clusters and discriminative training. In Daniel Marcu Susan Dumais and Salim Roukos, editors, *HLT-NAACL 2004: Main Proceedings*, pages 337–342, Boston, Massachusetts, USA, May 2 - May 7. Association for Computational Linguistics.

Nivre, J. 2007. Incremental Non-Projective 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 396–403, Rochester, New York, April. Association for Computational Linguistics.

Nivre, J. 2008. Algorithms for deterministic incremental dependency parsing. *Computational Linguistics*, 34.

Nivre, J. and J. Hall. 2005. Maltparser: A language-independent system for data-driven dependency parsing. In *Proceedings of the Fourth Workshop on Treebanks and Linguistic Theories*, pages 13–95.

Nivre, J., J. Hall, S. Kübler, R. McDonald, J. Nilsson, S. Riedel, and D. Yuret. 2007. The CoNLL 2007 shared task on dependency parsing. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, pages 915–932, Prague, Czech Republic, June. Association for Computational Linguistics.

Nivre, J. and R. McDonald. 2008. Integrating graph-based and transition-based dependency parsers. In *Proceedings of ACL*.

Petrov, S. 2010. Products of random latent variable grammars. In *Proceedings of NAACL-HLT*.

Petrov, S., L. Barrett, R. Thibaux, and D. Klein. 2006. Learning accurate, compact, and interpretable tree annotation. In *Proceedings COLING-ACL*.

Ratnaparkhi, A. 1996. A maximum entropy model for part-of-speech tagging. In *Proceedings of EMNLP*.

Rosenblatt, F. 1958. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408.

Ryan, M. 2006. *Discriminative learning and spanning tree algorithms for dependency parsing*. Ph.D. thesis, Philadelphia, PA, USA. AAI3225503.

Sagae, K. and A. Lavie. 2006. Parser combination by reparsing. In *Proceedings of NAACL-HLT*.

Sarkar, A. 2001. Applying co-training methods to statistical parsing.

Schapire, R. 1999. A short introduction to boosting. In *Proceedings of IJCAI*.

Smith, A. 2005. Logarithmic opinion pools for conditional random fields. In *In ACL*, pages 18–25.

Smith, D. A. and J. Eisner. 2008. Dependency parsing by belief propagation. In *Proceedings of EMNLP*.

Soricut, R. and D. Marcu. 2003. Sentence level discourse parsing using syntactic and lexical information. In *HLT-NAACL*.

Steedman, M., R. Hwa, S. Clark, M. Osborne, A. Sarkar, J. Hockenmaier, P. Ruhlen, S. Baker, and J. Crim. 2003. Example selection for bootstrapping statistical parsers. In *Proceedings of HLT-NAACL*.

Surdeanu, M. and C. Manning. 2010. Ensemble models for dependency parsing: Cheap and good? In *Proceedings of NAACL*.

Täckström, O., R. McDonald, and J. Uszkoreit. 2012. Cross-lingual word clusters for direct transfer of linguistic structure. In *HLT-NAACL*, pages 477–487.

Turian, J., L. Ratinov, and Y. Bengio. 2010. Word representations: a simple and general method for semi-supervised learning. In *Proceedings of ACL*.

West, D. 2001. *Introduction to Graph Theory*. Prentice Hall, 2nd editoin.

Yamada, H. and Y. Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *Proceedings of IWPT*.

Younger, D. 1967. Recognition and parsing of context-free languages in time n^3. *Information and Control*, 10(2):189–208, February.

Zeman, D. and Z. Žabokrtský. 2005. Improving parsing accuracy by combining diverse dependency parsers. In *Proceedings of the Ninth International Workshop on Parsing Technology*, Parsing '05, pages 171–178, Stroudsburg, PA, USA. Association for Computational Linguistics.