Advancements to Hindi Dependency Parsing: Semantic Information, Ensembling and $\mathcal{PQE}$

Thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Science (by Research)
in
Computational Linguistics

by

Naman Jain
201025028
naman.jain@research.iiit.ac.in

International Institute of Information Technology
Hyderabad - 500 032, INDIA
May 2016

International Institute of Information Technology
Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled *"Advancements to Hindi Dependency Parsing: Semantic Information, Ensembling and $\mathcal{PQE}$"* by Naman Jain, has been carried out under my supervision and is not submitted elsewhere for a degree.

_____ _____

Date Adviser: Prof. Dipti Misra Sharma

To my Parents...

Acknowledgments

First and foremost, I would like to thank my adviser Prof. Dipti Misra Sharma for showing complete faith and confidence in me and giving me enough space and encouragement to innovate and excel. Without her constant support and motivation this work would not have been possible. Her deep linguistic insights helped me at every point throughout my academic learning. She had not only taught me about linguistics but also helped me in exploring different aspects of life. I will always remember the long discussions I had with her even on topics outside academics.

A special mention to my other teachers Late Prof. Lakshmi Bai, Prof. Rajeev Sangal, Dr. Radhika Mamidi, Dr. Soma Paul and Dr. Prashanth Mannem, who had taught me everything I know about linguistics. I thank them for all the valuable discussions on variety of topics over these years.

I am highly grateful to my senior, mentor and friend Sambhav Jain, who helped me in understanding the computational aspects of linguistics. Most of my work took shape under his mentorship. His valuable feedbacks on all my ideas gave due direction to them. This thesis would not have been possible without him. I would also like to thank my senior and friend Riyaz Ahmad Bhat who taught me the very basics of doing research. He helped me a lot in improving my writing skills and discussing various ideas about parsing. A special thanks to other members of our lab's parsing team Aniruddha Tammewar and Bhasha Agrawal for helping me in learning various algorithms, computational tools and coding practices.

I am grateful to the vibrant and stimulating LTRC lab environment, which instigated several active discussions to spring up new ideas and scrutinizing the existing ones. Some contributors deserving special mention: Rishabh Srivastava, Rahul Sharma, Ankush Soni, Kunal Sachdeva, Karan Singla, Kshitij Mishra, Vandan Mujadia, Pruthwik Mishra, Maaz Anwar Nomani, Himanshu Sharma, Himani Chowdhary, Pratyusha Kuncham, Arpita Batra, Urmi Ghosh. I also thank, Rambabu, Srinivas Rao, Satish and Kumarswamy for making administrative matters run smoothly.

A special acknowledgment to my friends: Rahul Bansal, Rohit Laddha, Mohammed Moiz, Shoaib Khan, Rohan Gupta, Nikhil Goyal, Ishan Rastogi and Aman Shah for giving me constant breaks form the monotonous life via movie breaks, outings and dinners, without whom this work would have been finished much earlier.

Finally, I would like to thank my mother, father, sister and all my family members for their unconditional love and support. My mother and father are the unsung heroes behind everything that I have ever achieved.

Abstract

Natural Language Processing (NLP) is a challenging field in the area of artificial intelligence and computational linguistics. In simplistic terms, it can be defined as processing a natural language in any form either speech or written text. Though extensive research has been done for many of the languages in the world, Indian languages are still lagging behind in the race.

Processing of any natural language requires analysis to be done at multiple levels like word-level, phrase-level, sentence-level, semantic-level and higher levels of pragmatic and discourse. In this work we are presenting our efforts of making new advancements at the sentence level, which in linguistics terms, is regarded as *Syntactic Parsing*. Syntactic parsing involves establishing relations between different words of a sentence to convey the possible meaning.

Indian languages are morphologically rich and exhibit free-word order (MoR-FWO). Dependency Parsing, a type of syntactic parsing is better suited to such languages. Our efforts start from delivering a state-of-the-art Hindi Dependency Parsing system through the platform provided in the form of a shared task (Sharma et al., 2012). We employed a data-driven transition-based statistical system (Malt Parser), trained on Hindi Dependency Treebank (Bhatt et al., 2009; Palmer et al., 2009). Error-analysis performed in the task, helped us to target the problems in a more specific manner.

In the next phase, to target some of the problems like case ambiguity, data sparsity, lack of case marker, etc., we aided the process of dependency parsing by enriching the training model with semantic information. The information is extracted automatically from a rich lexical resource, Hindi WordNet (Narayan et al., 2002). Learning from the insights obtained in this advancement process, we moved to another well-established approach, Ensembling. Ensembling works on the principle of exploiting diversity of multiple parsing systems and combining their strengths to improve the parsing performance. We explored two ensembling approaches namely, re-parsing algorithms and word-by-word voting, using six different weighting strategies to combine six algorithmic variants of Malt parser. Improvements had been observed in the second approach.

After establishing a systematic comparison between both the techniques of ensembling, we obtained the lead to search for better weighting strategy to improve ensembling. The search ended with *Parse Quality Estimation* ($\mathcal{PQE}$) score. Adapting the work done in the past, we extended this functionality for our purpose of performing ensembling. The approach, which has failed earlier, has now shown improvements. Further, we also expanded the scope of $\mathcal{PQE}$ score for dependency arcs (attachment) to capture confusion made by the oracle in the parsing system. The functionality had also been extended

for Joint prediction of both arcs and labels simultaneously. To prove the efficacy of the approach, we implemented several real-world applications of $\mathcal{PQE}$ score. Finally, we proposed a robust evaluation framework in terms of *Domain Adaptability* (DA) and *Inter-Language Portability* (ILP), to better judge the effectiveness of Hindi Dependency Parser. During this evaluation process, using the property of portability, we also built dependency parsers for two of the Dravidian languages: Tamil and Telugu, which can be integrated easily in real world NLP systems like Machine Translation.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Natural Language Processing (NLP), as the name suggests, is a field of processing natural (human) languages, not by human beings but by building artificially intelligent machines which can perform this task. The task can be performed in three ways: (1) What we humans do with our brains (2) Human-Computer Interaction, and (3) Strong Artificial Intelligence i.e. making computers as intelligent as humans. Since the last option seems difficult to achieve in a near future, the main focus of the research community is towards the second option. For more than half a century, efforts have been taken to build systems where machines can complement human capabilities.

Natural languages have many components like speech, writing, gestures, etc. Processing all of these characteristics is quite different from one another. While speech requires the processing of sounds, the gestures require processing of body movements. Our focus in this dissertation is on processing written natural language text which is one of the most important and widely used component of any natural language.

Processing of a written text is analogous to a hierarchical processing which involves word level analysis at ground level to discourse level analysis at the top-most level. There is an intermediate level between this bottom to top hierarchy where the analysis is done at sentences level after analyzing words and phrases/clauses. Though all the levels have their due importance in the whole processing of natural language, the complexity of processing the natural language increases as one moves in the hierarchy from bottom to top. Much efforts have been taken in analyzing the bottom-most levels and significant success has been achieved in building highly accurate systems like Part-of-Speech (POS) Taggers, Morph Analyzers at word level analysis and Chunkers, Clause Identifier at Phrase/Clause/Chunk [1] level analysis. A huge success has already been achieved in building syntactic parsers to analyze sentences for many languages like English, Spanish, Chinese etc. but same development has not been seen for Indian languages. Though sincere efforts have been taken in the past and the results have been tremendous, still there is a long path ahead for Indian languages to be at par with other languages. The fact

[1] Though all three: phrase, clause and chunk are defined differently but for simplicity can be clubbed together at a broad level as phrase

that Indian languages account for more than 20% of the world's population, the need to work for Indian languages becomes all more important.

Syntactic parsing utilizes information gained at lower levels of text analysis, to establish relationships, which can be both syntactic (grammatical) or syntactico-semantic (partially conveying meaning also), between words (tokens) of a sentence to convey a possible sense, which may or may not have a relevance in the real world. It has been already established that syntactic parsing has huge applications like machine translation (Xu et al., 2009), natural language interface (Damljanovic et al., 2010), question answering (Wang et al., 2007), natural language generation (Wann et al., 2009), text summarization (Owczarzak, 2009), sentiment analysis (Meena and Prabhakar, 2007) and all such applications which involve processing of natural language text. Thus the aim of research community is to build and develop high quality parsers which can guarantee the reliability of resultant relations (syntactic relations) between words of a sentence to be further used in multiple real-world applications.

Syntactic Parsing establishes the representational relations pertaining to a grammatical framework like phrase structure grammar, dependency grammar, categorical grammar, etc. In general, syntactic parsing is of two types:-

1. **Constituent Parsing:** It explicitly represents phrases (non-terminal nodes: Noun Phrase, Verb Phrase, etc.), structural categories (non-terminal labels like Part-of-Speech) and possibly some functional categories (grammatical functions) i.e. using Phrase Structure Grammar (PSG).

2. **Dependency Parsing:** It explicitly establishes relationships between words as functional categories (directed labels) and possibly some structural categories as head-dependent relations (directed arcs) i.e. using Dependency Grammar (DG).

Figure 1.1 and 1.2 respectively shows a simplified phrase structure and dependency structure for the following example.

Example: *The girl eats an apple.*

Learning DG over PSG

- Dependency Parsing is more *straightforward* as it labels each token w_{i} with w_{j} in a sentence.

- Dependency Parsing directly encodes predicate-argument structure and thus fragments are directly *interpretable*.

- It has been suggested that *morphologically rich and free word order (MoR-FWO)* languages can be handled more efficiently using the dependency based framework than the constituency based one (Hudson, 1984; Shieber, 1987; Melćuk, 1988; Bharati et al., 1995). The use of dependency formalism for various NLP/CL tasks, especially for parsing, has increased many folds since the last decade. Consequently, most of the parsers for morphologically rich and free word order languages are dependency based (Tsarfaty et al., 2013).

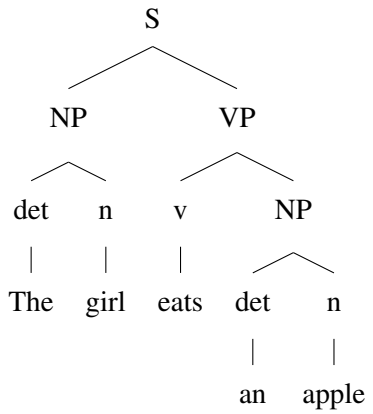


Figure 1.1 Example of Constituent Based Representation.

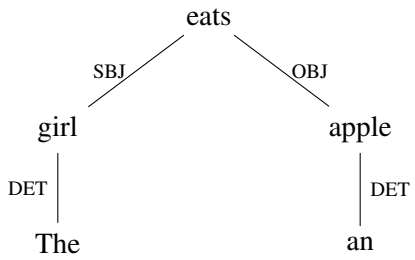


Figure 1.2 Example of Dependency Based Representation.

As already mentioned above, a lot of research has been done in the area of syntactic parsing for most commonly used languages of the world like English, Chinese, etc. (Collins et al., 1999; Henderson and Brill, 1999; Kingsbury et al., 2002; Montemagni et al., 2003; Klein and Manning, 2004; Xiong et al., 2005; Hwa et al., 2005; McClosky et al., 2006b; Buchholz and Marsi, 2006; Sagae and Lavie, 2006; Nilsson et al., 2007; Hall et al., 2007; Sagae and Tsujii, 2007; Zeman and Resnik, 2008; Kübler et al., 2009; Zhang et al., 2009; Fossum and Knight, 2009; Surdeanu and Manning, 2010; Agirre et al., 2011; Tsarfaty et al., 2013). Many of the NLP approaches like constituent parsing, leveraging semantic information to syntactic parsing, ensembling of various individual parsers, parse quality estimation, automatic parse error detection, human-aided systems, domain adaptation, inter-language portability and many more, which have been proved beneficial for such languages, are either employed partially for Hindi or never explored. Even approaches like constituent parsing is not suitable for Hindi due to its MoR-FWO property, which has been successfully exploited for English and other languages. In addition, there are multiple challenges like absence of case marker, case ambiguity, a relatively large

dependency tag-set, issue of non-projectivity and long-distance dependencies, lack of resources, etc., which are more Hindi-specific and require respective Hindi-specific treatment.

Through our dissertation we are providing an elaborate study of some of above mentioned NLP approaches for dependency parsing for Hindi (the most spoken Indian language with 380 million users [2]), to address the language-specific issues. In addition, to further improving the present state-of-the-art dependency parser for Hindi, we are also exploring the new dimensions in the area of dependency parsing for Hindi and proposing methodologies which can be adapted for other languages, in particular to other Indian languages.

I. Key Contributions of the Thesis

The following are the key contributions of this thesis :-

Dependency Parser for Hindi

Hindi has been most widely explored Indian language but still it is not at par with other majorly spoken world languages, especially English. With the scope of still improving the existing syntactic analysis (Parsing), we came up with the State-of-the-art Dependency Parser for Hindi in the form of a *Shared Task for Hindi Dependency Parsing, MTPIL* organized in *COLING 2012* (Singla et al., 2012). Our system performed best with respect to Gold Standard Track and second best in Automatic Standard Track.

Advancements to Dependency Parsing for Hindi

This is the most significant contribution of this thesis and also act as the central part of the same. To further advance the Dependency Parser for Hindi, while also addressing some of the existing challenges, we worked in the following three directions:

1. **Automatic Extraction of Semantic Information from Hindi WordNet:** Continuing the development of state-of-the-art dependency parser for Hindi through Malt parser, we used the semantic information to aid dependency parsing and address issues like case ambiguity, lack of case marker, etc.

2. **Ensembling:** We combined multiple transition-based data-driven statistical parsers (base parsers) to complement one-another's strength and thus advanced the dependency parsing. The combination had been governed using different weighting strategies based on different contexts at inference time. We used two different approaches to combine base parsers.

[2] https://en.wikipedia.org/wiki/List_of_languages_by_total_number_of_speakers

3. $\mathcal{PQE}$: Search for better weighting strategy, to improve ensembling, landed us with $\mathcal{PQE}$. It also presented us with the opportunity to improve the already available functionality of the $\mathcal{PQE}$ in Malt Parser. Earlier, the score was only accounting for those parser actions which result in label-prediction. Thus, we made efforts to also include those parser actions which contribute to an edge-formation and such actions were combined using five distinct strategies, giving us five distinct systems for drawing a better and more accurate picture of $\mathcal{PQE}$. The modified functionality of obtaining $\mathcal{PQE}$ score was also extended to Joint Model of Prediction, where parser actions responsible for both edge-formation and label-prediction, are executed simultaneously (Jain et al., 2015).

Multiple Applications based on $\mathcal{PQE}$

This is another key focus areas of this thesis where we explored a wide variety of applications using the $\mathcal{PQE}$ score. The broad context in which $\mathcal{PQE}$ score can be exploited includes Automatic Parse Error Detection, Improving level of Dependency Parsing in an Human-Aided or Semi-Autonomous (Jain et al., 2013a), Minimizing the cost of validation in Treebank Expansion (Jain et al., 2013b), Hybrid systems of Rules and Statistics, Generation of Treebanks for Less Resource Languages, etc. We explored some of these applications in great detail while proposing mechanisms for others, which could aid the concerned research and industrial community in developing the required modules.

Evaluation Framework to check the Robustness of Dependency Parser

To check the effectiveness and robustness of the Hindi Dependency Parser, exclusively over other domains and its algorithmic functionalities over other languages, we designed an evaluation framework. Measures had also been taken to come up with a generic framework which could not only be helpful in evaluating dependency parsers for other languages but the same methodology could also be applied to develop similar evaluation frameworks for other NLP tools.

Dependency Parsers for Dravidian Languages: Tamil and Telugu

In the course of designing the evaluation framework for Hindi Dependency Parser, we also developed the dependency parsing modules for Tamil and Telugu using portability mechanism. These have already been integrated in a real-world application (Machine Translation system). With these, we tried to provide an initial platform for the NLP research community to further research on and address the language specific complexities in the evolution of the more developed forms of the respective parsers.

II. Outline

The dissertation is organized into seven chapters and a brief outline of each is as follows:-

Chapter 2:

This chapter presents our first step towards our exploratory journey in the field of Dependency Parsing. It starts with our participation in a Shared Task on Hindi Dependency Parsing and delivering the State-of-the-Art Dependency Parser for Hindi. Along with the discussion of the system, the chapter also sets a line of action for future exploration.

Chapter 3:

The chapter deals with the first step towards advancing dependency parsing for Hindi. It includes automatic extraction of semantic information in the form of concept hierarchies corresponding to lexical items, present in Hindi WordNet. Further, it also shows, how using the semantic information, parsing accuracy can be improved along with a discussion on the extent of its impact on the parsing performance.

Chapter 4:

In this chapter, we employ two ensembling approaches at inference time i.e., re-parsing algorithm and word-by-word voting. Experiments involved in combining six-base parsers, are governed using different weighting strategies based on different contexts. A systematic comparison between both the approaches is also made to present a better picture of ensembling for Hindi. In addition to improving the parsing performance, ensembling also enrich us with new insights.

Chapter 5:

Based on the insights gained during ensembling, we explore other methodologies to compute a better quality score for parsing. $\mathcal{PQE}$ score computed by Jain and Agrawal (2013) is taken as a much better score to perform the task of Ensembling. Next, we take up the task of enlarging the scope of $\mathcal{PQE}$ score by also making it able to furnish scores capturing oracle confusion for edge-formation parser actions in addition to the previously computed confusion score corresponding to label-prediction only. Further, we also extend the functionality from single Branched Model of Prediction to Joint Model of Prediction for Malt Parser.

Chapter 6:

The sixth chapter provides a detailed account of our work with respect to a variety of applications based on $\mathcal{PQE}$ score and establish the credibility of the same. In addition to an elaborative discussion on some of the applications, the chapter also talks about some of the highly potent usage of $\mathcal{PQE}$ score.

Chapter 7:

The chapter prescribes a generic methodology while evaluating the robustness of our Dependency Parsing tool for Hindi on the lines of its effectiveness of Domain Adaptability and Inter-Language Portability for other Indian languages. Using the mechanism of Portability, we port our Hindi Dependency Parser for Tamil and Telugu, thus develop Dependency Parsers for the corresponding Dravidian Languages.

Chapter 8:

Being final chapter, it concludes our dissertation with a brief summary of the key points along with the possible extension of the utility of our work and its applications in various NLP tasks.

Chapter 2

Initial Exploration to Dependency Parsing

This chapter is the seed of our dissertation. The first step in our research journey was participating in a Shared Task [1] on Hindi Dependency Parsing, organized in COLING-2012 under MTPIL (Sharma et al., 2012). In past various such tasks have been organized in the area of Dependency Parsing. Some of the most popular ones which revolutionized the research in this field are Nilsson et al. (2007) and Buchholz and Marsi (2006) for many of the most spoken languages in the world. Similar shared tasks for Indian Languages Dependency Parsing and in particular for Hindi are Husain (2009) and Husain et al. (2010). The purpose of all such shared tasks is to promote natural language processing applications and evaluate them in a standard setting. These shared tasks helped in bringing some of the most important dependency parsing tools like Malt parser [2] and MST parser [3] into limelight.

While participating in the COLING-2012 shared task, we found out that there remains a huge potential with tools like Malt parser and others, that could be further utilized to improve the then state-of-the-art data-driven Hindi dependency parser. We performed several experiments using such tools and tried out with different settings available with them. This not only helped us in building new state-of-the-art Hindi dependency parsing system but also laid the foundation for our future journey. [4]

2.1 Shared Task: Two-stage Approach for Hindi Dependency Parsing using Malt Parser

We have already mentioned that Hindi is a morphologically rich and relatively free-word order language(MoR-FWO). Parsing is a challenging task for such MoR-FWO languages like Turkish, Basque, Czech, Arabic, etc. because of their non-configurability (Hall et al., 2007). It has been suggested that

[1] A shared task is a kind of competition where different teams participate to compete for a particular task. The participating teams are evaluated and compared in a systematic way.

[2] `http://www.maltparser.org/`

[3] `http://sourceforge.net/projects/mstparser/`

[4] The work has been published as Singla et al. (2012)

these kind of languages can be represented better using dependency framework rather than constituent framework (Hudson, 1984; Shieber, 1987; Melćuk, 1988; Bharati et al., 1995).

Previous efforts on parsing MoR-FWO languages include Nivre et al. (2007), Hall et al. (2010), McDonald and Nivre (2007), etc. In ICON 2010, best results were obtained by Kosaraju et al. (2010), who used Malt parser with SVM classifier for labeling and using local morph-syntactic, chunk and automatic semantic information as features. Ambati et al. (2010a) had explored two-stage approach of parsing Hindi. They splitted the data into two parts namely, interChunks and intraChunks [5]. The interChunk part of the data contains only dependency relations between chunkheads of the sentences while the intraChunk part has the dependency relations between the tokens of a chunk. The dependency labels for interChunk and intraChunk are disjoint. This approach helped in avoiding intraChunk relations to be marked as interChunk relations and vice-versa. Following this approach, we explored different parsing algorithm parameters and learner algorithm settings of Malt parser. Though we also performed pilot experiments with MST parser, found out that Malt parser performed better than MST parser and thus decided to explore it explicitly.

2.1.1 Data

A subset of the dependency annotated Hindi Treebank (HTB ver-0.5) was released as part of the Hindi Parsing Shared Task-2012 (HPST-2012). Morphological analysis, however, had not been validated for errors and inconsistencies. It was released for two evaluation tracks (gold standard and automatic). In the gold standard track, the input to the system consisted of lexical items with gold standard morphological analysis, part-of-speech tags, chunks and the additional features listed above. In the automatic track, the input to the system contained only the lexical items and the part-of-speech tags from an automatic tagger. Some sentences had been discarded due to presence of errors in the data. Table 2.1 shows the training, development and testing data sizes for Hindi. For the testing phase of the contest, the parser was trained on the entire released data (training + development).

Type	Sentence Count	Token Count	Avg. Sentence Length
Training	$12,041$ [6]	$268,093$	22.27
Development	$1,233$	$26,416$	21.42
Testing	$1,828$	$39,775$	21.87

Table 2.1 Hindi Treebank Statistics

[5] A chunk is a set of adjacent words which are in dependency relation with each other, and are connected to the rest of the words by a single incoming arc.

[6] We found 3 sentences to be erroneous so the total number of sentences used in training is $12,038$

2.1.2 Experiments and Results

In our experiments we had used freely available Malt parser (version 1.6.1) (Nivre et al., 2007). In this section we give an account of experiments performed in a series. Each experiment focuses on choosing the best option for a certain parameter/feature keeping the other parameter/feature fixed. In the subsequently following experiments the best parameter chosen from previous experiment is retained. This approach of experimenting was a brute-force way to search for best parameters. Due to time constraints in shared task, we could not explore many of the features in depth but this helped us to atleast get familiar with Malt parser. After the shared task, we started a systematic approach of studying some of the most important parameters individually which formed the basis of our future research journey.

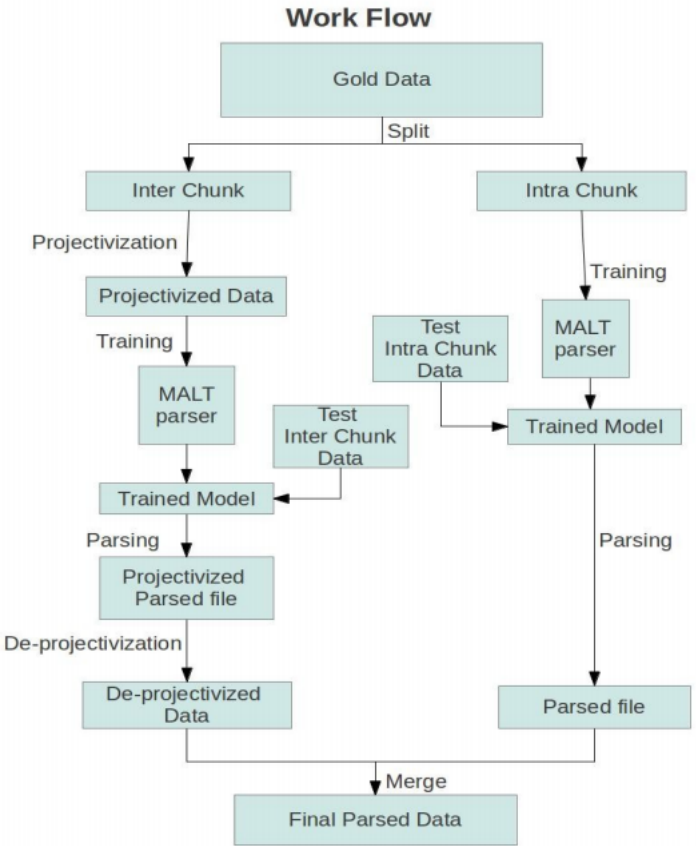


Figure 2.1 Work Flow of the Set-up

2.1.2.1 Feature Model

Feature model is the template, which governs the learning from the given training data. We explored various configurations for feature model using insight from previously used feature models from similar

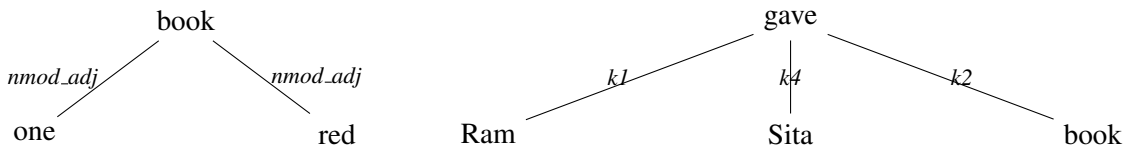


Figure 2.2 Example of intraChunk and interChunk Dependency Representations

tasks. We observed feature model used by Kosaraju et al. (2010) performed best and thus adopted it for our experiments. [7].

2.1.2.2 Two-stage (inter-intra chunk) Approach

Every sentence in the data-set is divided into chunks.

(1) राम ने सीता को एक लाल किताब दी ।
(raam-*Erg*) (sita-*dat.*) (one red book) (gave)
'Ram gave a red book to Sita.'

In the above example, the sentence is divided in 4 chunks marked by brackets. The dependency labels for intraChunk relations are different from that of interChunk, forming two disjoint sets of dependency labels: interChunk labels (*k1*, *k2*, *k7*, etc.) and intraChunk labels (*nmod_adj*, *lwg_psp*, *mod*, etc.) [8].

Theoretically, the parent of a non-chunkhead token should be a chunkhead or non-chunkhead token of the same chunk and the relation should be labeled with an intraChunk label. In (1), the non-chunkhead tokens एक and लाल are connected with the chunkhead token किताब and the relation label *nmod_adj* is an intraChunk label. The parent of the chunkhead must also be a chunkhead from another chunk. The relation should be marked with interChunk label.In (1), the chunkhead tokens राम, सीता, किताब and दी are attached with a chunkhead token and the dependency relation labels are interChunk labels. However, in the training data, there were few noisy cases where intraChunk relations werre marked as interChunk and vice-versa. Such instances were very less in number and hence were ignored.

Dividing the data into inter- and intra-chunk, all the above constraints would be automatically handled. In the resulting intraChunk data, since the chunks that formed, would behave as an individual sentence, the parser would not be able to make an inappropriate arc.

2.1.2.3 Experiment with Projectivity

Malt parser has a default constraint to give only projective output. However, in the training data we found approximately 1.1% arcs to be non-projective. To address the non-projectivity in data, we

[7]The feature template is mentioned in Appendix B

[8]To know more about dependency tag-set refer to `http://ltrc.iiit.ac.in/MachineTrans/research/tb/final_guidelines-ver2.pdf`

used *pseudo-projective algorithm* as proposed by Nivre and Nilsson (2005). We only incorporated the pseudo-projective algorithm in case of interChunk data as in intraChunk data, we found the arcs to be always projective. There are three options available with the pseudo-projective algorithm in Malt parser. We performed 6 intermediary experiments on all the three options and got some interesting results.

Pseudo-projective algorithm replaces all the non-projective arcs in the input data to projective arcs by applying a lifting operation. The lifts are encoded in the dependency labels of the lifted arcs. In order to apply an inverse transformation to recover the underlying (non projective) dependency graph, there is a need to encode information about lifting operations in arc-labels. The encoding scheme can be varied according to **marking_strategy** and there are currently five of them: **none**, **baseline**, **head**, **path** and **head+path** (Nivre and Nilsson, 2005). We performed intermediary experiments separately on each of them and observed that **head** option gave the best results. This option projectivizes input data with **head** encoding for dependency labels.

Secondly, there is an option called **covered_root**, which is mainly used for handling dangling punctuations. This option has five values: **none**, **ignore**, **left**, **right** and **head**. In our intermediary experiments, we found that **ignore** performed better than others.

On the basis of the lifting order, there are two ways to lift the non-projective arcs namely, **shortest** and **deepest**. In the **deepest** lifting order, most deeply nested non-projective arc is lifted first, not the shortest one. In our experiments we found that **deepest** option had no effect in increasing the parsing accuracy rather a slight decrement was observed in the accuracy as compared to **shortest** option.

2.1.2.4 Experiment with Features

We tried to experiment with different types of features that could be used in **FEATS** column in the CoNLL-X format [9]. We considered four ways: 1) without any information in FEATS column, 2) only *tam*[10] and *vib*[11] information, 3) *tam* and *vib* along with chunkType and, 4) with all the information present by default. The best results were obtained using option 4 i.e. complete information. This experimentaiton could only be performed for the Gold track as such information about the features was not provided for Automatic track. This was the major reason for the difference in the parsing accuracies between gold and auto data. As mentioned above, this was a brute-force method of searching most contributing features for dependency parsing. Realizing that such features had great impact on the parsing accuracy, our first task after completing shared task was to study the features in depth, as discussed in the next Chapter (Chapter 3).

2.1.2.5 Experiment with Algorithms

Kosaraju et al. (2010) has shown that **nivre_eager** algorithm results in the best accuracy for Hindi. Our intermediary experiments also supported the same. We also explored the **root-handling** option

[9] http://ilk.uvt.nl/conll/

[10] Tense-Aspect-Modality

[11] Vibhakti (post-position)

which could be any of **normal**, **strict** or **relaxed** arguments. Our experiments showed that **relaxed** option gave the best accuracy. In relaxed option, root dependents are not attached during parsing (attached with default label afterwards) and reduction of unattached tokens is permissible. Though many experiments could not have been performed, at a later stage working on the lines of new approaches like Ensembling i.e. combining strengths of different models, we explored the algorithmic diversity in great detail (Chapter 4).

2.1.2.6 Experiment with Prediction Strategy

There are three types of **prediction strategies** available in Malt parser:

- **combined** (default): Combines the prediction of the transition and the arc label.
- **sequential**: Predicts the transition and continues to predict the arc label if the transition requires an arc label.
- **branching**: Predicts the transition and if the transition does not require any arc label then the non determinism is resolved. But if the predicted transition requires an arc label then the parser continues to predict the arc label.

We performed several experiments with the above options and found that using **branching** there was an increase in the parsing accuracy.

2.1.2.7 Experiment with SVM settings

In our experiments, we used the *LIBSVM* learner algorithm following the SVM settings ($s0t1d2g0.12$-$r0.3n0.5m100c0.7e0.5$) in experiments reported by Kosaraju et al. (2010) for Hindi. These settings gave better results over the default SVM settings.

2.1.2.8 Results

We had trained Malt parser separately using Gold and Auto training data. For gold data, we trained two models, one for interChunk data with all settings obtained in the above experiments and other for intraChunk data with all the above settings except **branching** and **projectivization**. For both the states, we used the same algorithm **'nivre_eager'** and learner ***'LIBSVM'***. In the final evaluation, the system demonstrated LAS (Label Attachment Score) was **90.99%**, UAS (Unlabled Attachment Score) **95.87%** and LA (Label Accuracy or Label Score (LS)) **92.58%**. For Automatic track, we couldnt split the data in two parts as the information, on which the data was divided, was missing in the testing files. Except two-stage approach, all the other settings were exactly similar as for the gold data. The final LAS was **83.91%**, UAS **91.70%** and LS **86.77%**.

Here the evaluation was done only on the basis of numbers like LAS, UAS and LS. But to get a better picture about the quality of a parsed output tree, we worked in introducing another measure,

correct label	system output label	frequency
pof	*k2*	139
k1	*k2*	123
k2	*pof*	112
k7	*k7p*	95
k7p	*k7*	88

Table 2.2 Top 5 most frequent errors

$\mathcal{PQE}$, in later part of our thesis (will be discussed in Chapter 5). Also for a robust analysis for tools like dependency parser, we also proposed an evaluation framework in terms of Domain Adaptability and Inter-Language Portability (will be discussed in Chapter 7).

2.1.3 Error Analysis

The most frequent errors that the parser made contained the confusion between marking of *k2*, *k1* and *pof* dependencies. The confusion between *k1* and *k2* is because of the absence of the case markers for disambiguation. As *pof* is the verbal form of noun, it is even difficult for humans to disambiguate between *pof* and *k2*. The confusion between *k7* and *k7p* is also frequent because of their closeness in terms of sense. Some of these errors can be handled by post-processing algorithm. [12]

2.2 Summary

In this chapter, we experimented with different parameters, both previously explored and unexplored of the data-driven Malt parser along with the two-stage pre-processing approach to build a high quality dependency parser for Hindi. Though the system achieved best accuracy in Gold Standard Track and second best in Automatic Standard Track in the Sharing Task as floated in MTPIL (Sharma et al., 2012), still there left many more options which can be explored in future.

2.3 Conclusion

Malt parser, a generic data-driven algorithmic platform for Dependency Parsing has many more functionalities to explore than what we have touched upon. But exploring the features alone could make

[12] A more detailed discussion, about the challenges that are faced while parsing Hindi under Dependency Grammar formalism, is available in the next chapter, which motivated us to address some of these challenges using semantic information

the feature-template biased towards those sentence constructions present in the training-data. Further, it is not a cost-effective and time-effective strategy to explore the functionalities in a brute-force manner. To divert from that methodology, we chose to aid dependency parsing, particularly interChunk parsing, by studying some of above explored parameters in depth. The first among them is utilizing the strength of feature set (FEATS) as the next step of our research journey.

Chapter 3

Advancements to Dependency Parsing: Semantic Information from Hindi WordNet

In the last chapter, we have seen that there are many different aspects related to dependency parsing in using Malt parser like learning algorithms, features, parsing algorithms, etc., which play an important role in the data-driven statistical dependency parser. In this chapter we are exploring one of those aspects. We will present our efforts towards incorporating external knowledge from Hindi WordNet to aid dependency parsing in the form of semantic information as a feature set. [1]

3.1 Background and Challenges

Hindi is an Indo-Aryan language with richer morphology as compared to English. It exerts a relatively free word order with SOV (Subject-Object-Verb) being the default configuration. Due to the flexible word order, dependency representations are preferred over constituency for its syntactic analysis (Bharati and Sangal, 1993). The dependency representations do not constrain the order of words in a sentence and thus are better suited for flexible ordering of words. The dependency grammar formalism, used for Hindi, is *Computational Paninian Framework* (CPG) (Begum et al., 2008; Bharati et al., 2009c). The dependency relations in CPG formalism are closer to semantics and hence they are also denoted as *syntactico-semantic* relations.

The most important feature explored for dependency parsing is 'case clitics' that largely governs the relations nominals bear with their heads. Several efforts in past, on parsing Hindi, have greatly benefited by utilizing these clitics as a feature (Ambati et al., 2010a; Ambati et al., 2010b). However, case markers and case roles do not have a one-to-one mapping as each case marker is distributed over a number of case roles. Among the six case markers only Ergative case marker is unambiguous (Mohanan, 1994). Although case markers are good indicators of the relation a nominal bears in a sentence, their ambiguous nature bar their ability in effectively identifying the role of a nominal while parsing. Consider the examples from (1a-e), the instrumental *se* is extremely ambiguous. It can mark the instrumental

[1] The work has been published as Jain et al. (2013c)

adjuncts as in (1a), source expressions as in (1b), material as in (1c), comitatives as in (1d), and causes as in (1e).

(1a) मोहन ने चाबी से ताला खोला ।
Mohan-*Erg* key-*Inst* lock-*Nom* open
'Mohan opened the lock with a key.'

(1b) गीता ने दिल्ली से सामान मंगवाया ।
Geeta-*Erg* Delhi-*Inst* luggage-*Nom* procure
'Geeta procured the luggage from Delhi.'

(1c) मूर्तिकार ने पत्थर से मूर्ति बनायी ।
sculptor-*Erg* stone-*Inst* idol-*Nom* make
'The sculptor made an idol out of stone.'

(1d) राम की श्याम से बात हुई ।
Ram-*Gen* Shyaam-*Inst* talk-*Nom* happen
'Ram spoke to Shyaam.'

(1e) बारिश से कई फसलें तबाह हो गयीं ।
rain-*Inst* many crops-*Nom* destroy happen-*Perf*
'Many crops were destroyed due to the rain.'

Not all instances of a nominal in Hindi are case marked, as shown in Table 3.1. In appropriate contexts, a nominal can also bear a nominative case which is morphologically null (henceforth referred as unmarked nominal). It is possible, in fact quite frequent, to have more than one unmarked nominal within a single clause and due to the relative free word order, the movement can result in different surface configurations.

(2a) चिड़िया दाना चुग रही है ।
bird-*Nom* grain-*Nom* peck-*Prog*

(2b) दाना चिड़िया चुग रही है ।
grain-*Nom* bird-*Nom* peck-*Prog*
'A bird is pecking grain.'

A conventional parser has no cues for the disambiguation of instrumental case marker se in examples (1a-e) and similarly, in example (2a-b), it is hard for the parser to know whether 'bird' or 'grain' is the agent of the action 'peck'. Apart from **lexical** and **structural ambiguity**, there are also **data sparsity** and **out of vocabulary** (OOV) problems when parsing out-of-domain text. Traditionally, syntactic parsing has largely been limited to the use of only a few lexical features. Features like POS-tags are

	Patient-Unmarked	Patient-Marked
Agent-Unmarked	1276	741
Agent-Marked	5373	966

Table 3.1 Co-occurrence of Marked and Unmarked verb arguments in Hindi Dependency Treebank. *Source*: training-set, shared task MTPIL 2012

way too coarse to provide deep information valuable for syntactic parsing. So in order to assist the parser for better judgments, we need to complement the morphology somehow.

3.2 Exploring Semantic Information in Hindi WordNet for Hindi Dependency Parsing

Last decade has witnessed several efforts towards developing robust data driven dependency parsing techniques (Kübler et al., 2009). The efforts, in turn, initiated a parallel drive for building dependency annotated treebanks (Tsarfaty et al., 2013), which serve as a data source for training data driven dependency parsers. The annotations are often multi-layered and furnish information on part-of-speech category of word forms, their morphological features, related word groups and the syntactic relations. The availability of such rich resources have considerably improved the parsing performance of syntactic parsers (Collins et al., 1999). However, the error analysis studies carried out on these parsers later revealed that certain syntactic relations are difficult to deduce and disambiguate with the syntactic information available in the annotated treebanks.

The need for richer information invoked several efforts in the direction of annotating higher order linguistic information in treebanks. It was felt that semantics can be leveraged for syntactic disambiguation and thus semantic annotation was performed in syntactic treebanks to complement the morpho-syntactic annotations (Kingsbury et al., 2002; Montemagni et al., 2003). Fujita et al. (2007) and MacKinlay et al. (2012) illustrated that semantic annotation delivers a significant improvement in parsing, confirming the hypothesis that semantics can assist syntactic analysis.

Among Indian languages, notable efforts on using semantic information in dependency parsing are on *Hindi*. Bharati et al. (2008) illustrated that mere *animacy* (human, non-human and inanimate) of a nominal significantly improves the accuracy of the parser. Later studies on extending such information with finer semantic distinctions like *time*, *place*, *abstract* reconfirmed the substantial role of semantics in syntactic parsing (Ambati et al., 2009). These studies are carried out on a data-set with hand annotated semantics. Although these studies provide deep insights on the role of semantics in parsing, they

are limited in application as such information can not be automatically generated while parsing new sentences.

Thus, we make an effort to supply the aforementioned semantic information by employing concept hierarchy available in Hindi WordNet (henceforth HWN).

3.2.1 Related Work

Attempts have been made to utilize hand annotated semantic information for *constituency parsing* (Fujita et al., 2007; MacKinlay et al., 2012) as well as *dependency parsing* (Øvrelid and Nivre, 2007; Bharati et al., 2008; Ambati et al., 2009). However, acquiring such information for new sentences remains a challenge. This leads us to the exploration of lexical databases and ontologies for accessing semantic information useful for parsing. Xiong et al. (2005) used two lexical resources *HowNet*[2](Dong and Dong, 2000) and *TongYiCi CiLin* (Mei and Gao, 1996) for parsing Penn Chinese Treebank (Xue et al., 2002). Agirre et al. (2008) demonstrated that semantic classes obtained from English WordNet (Miller, 1995) help to obtain significant improvements in both PP attachment and PCFG parsing. Similarly, for dependency parsing, Agirre et al. (2011) utilized the English WordNet semantic classes and improved parsing accuracies.

3.2.2 Hindi WordNet and Concept Ontologies

Hindi WordNet is a lexical database developed on the lines of English WordNet, under the Indo WordNet project (Narayan et al., 2002). For each lexical item, Hindi WordNet defines a synset which enlists its synonyms. Further, each synset is mapped to a concept ontology. The concept ontology is a hierarchical organization of concepts like entities, actions etc. which defines the semantic properties of lexical items of a given synset. The ontology consists of around 200 different concepts. The lexical item is the leaf node in this hierarchical construct. As we move up the hierarchy, the specific semantic aspects of a given lexical item are unraveled. The hierarchy terminates, immediately after capturing the syntactic category of a word, at the *TOP* node. The *TOP* acts as a $root$, holding the hierarchies of all the lexical items listed in HWN. Figure 3.1 illustrates a typical hierarchy in this ontology, where *Ape* is the most explanatory node. As we move up, it becomes more and more generic. Further, the relations between different synsets are captured based on the following paradigms :

- Semantic (hypernymy, hyponymy, meronymy etc.)
- Lexical (antonymy, synonymy etc.)
- Gradience (size, quality, manner etc.).

[2]http://www.keenage.com

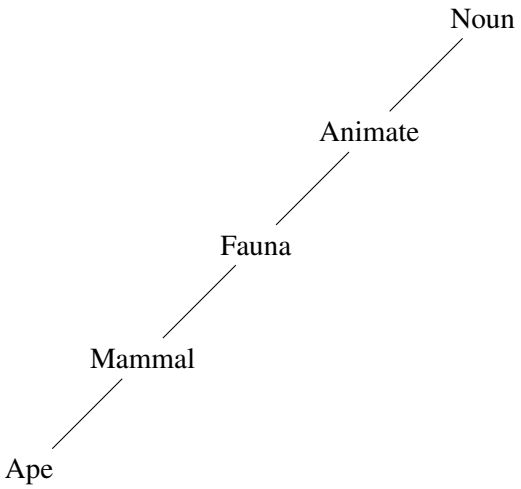


Figure 3.1 Sample Hierarchy of Concepts in Hindi WordNet

3.2.3 Hindi Dependency Treebank

In this section, we give an overview of Hindi Treebank (HTB ver-0.51) (Bhatt et al., 2009; Palmer et al., 2009) a part of which was released for Hindi Dependency Parsing shared task, MTPIL, (Sharma et al., 2012). It is a multi-layered dependency treebank with morphological, part-of-speech and dependency annotations based on the CPG. In the dependency annotation, relations are mainly verb-centric. The relation that holds between a verb and its arguments is called a '*karaka*' relation. Besides *karaka* relations, dependency relations also exist between nouns (genitives), between nouns and their modifiers (adjectival modification, relativization), between verbs and their modifiers (adverbial modification including subordination). CPG provides an essentially syntactico-semantic dependency annotation, incorporating *karaka* (e.g., agent, theme, etc.), *non-karaka* (e.g. possession, purpose) and other (part-of) relations. A complete tag-set of dependency relations based on CPG can be found in (Bharati et al., 2009c). The ones starting with '*k*' are largely Paninian *karaka* relations, and are assigned to the arguments of a verb. The data is released in two formats, SSF (Bharati et al., 2007) and CoNLL-X[3] formats (details in Table 3.2). It has also been released in UTF-8 encoding and roman readable WX[4] notation. We are using the CoNLL-X format and UTF-8 encoding.

3.2.4 Incorporating Knowledge from Concept Ontologies

In this section, we present our approach to incorporate semantic knowledge from HWN into the parsing model. We transform the hierarchical information in the concept ontology listed in HWN, into

[3] `http://ilk.uvt.nl/conll/\#dataformat`

[4] `http://sanskrit.inria.fr/DATA/wx.html`

[5] A chunk is a set of adjacent words which are in dependency relation with each other, and are connected to the rest of the words by a single incoming arc.

Type	Sentence Count	Token Count	Chunk[5]Count
Training	12,038	268,009	142,445
Development	1,233	26,416	13,945
Testing	1,828	39,775	21,165

Table 3.2 Statistics of Data Sets used for experiments

a string feature (henceforth WN feature) for all the tokens in our data. Given a lexical item, we extract the information using its syntactic category from the ontological hierarchy corresponding to the most appropriate sense selected. In the following, we discuss in detail the selection and incorporation of this information with the challenges posed.

3.2.4.1 Feature Extraction

In this section, we explore the extraction of features from HWN corresponding to the lexical items in our data. We also address the issues like sense selection and coverage.

Sense Selection: Attributed to the phenomenon of lexical ambiguity, a lexical item can have senses varying across different contexts. Although HWN lists all the possible senses of a lexical item, to choose the contextually appropriate sense is a challenging task. Here, we discuss our approach to select the sense of a lexical item best suited in a given context.

- $Category\ Based\ Sense\ Selection$: Consider a word $chaat$, it can either mean 'lick' or 'snacks'. The former corresponds to a verb while the latter is a nominal as depicted in Figure 3.2. The syntactic category of a lexical item provides an initial cue for the sense selection. Among the varied senses, we filter out the senses that do not fall into its syntactic category.

- $Intra-Category\ Sense\ Selection$: As a matter of fact, words are ambiguous not only across different syntactic categories but also within same category as depicted in Figure 3.3. Once the senses of a lexical item are filtered based on its syntactic category, within category senses, if many, are investigated for the best sense based on the following strategies:

  - $First\ Sense$: Among the varied senses, we select the first sense listed in HWN corresponding to the POS-tag of a given lexical item. The choice is motivated by our observation that the senses of a lexical item are ordered in the descending order of their frequencies of usage i.e., the first sense listed in HWN is the predominant sense of a given lexical item.

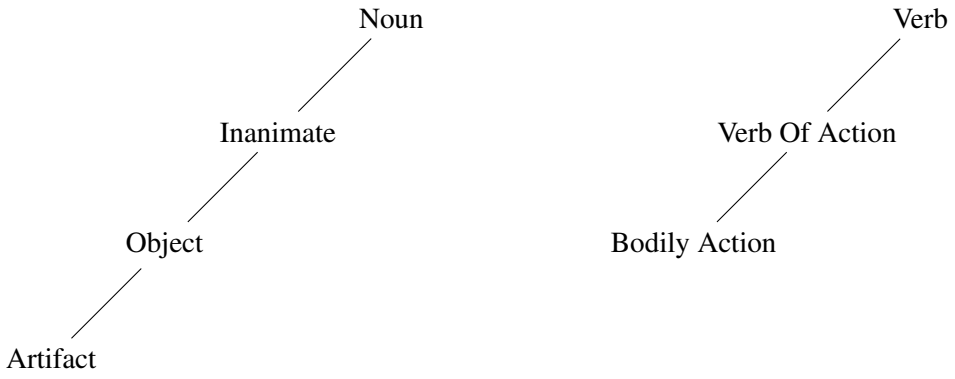


Figure 3.2 Nominal and Verb Sense of $chaat$

- WSD: Although first sense captures the predominant usage of a lexical item, it is inappropriate for its other infrequent usages. We, therefore, need to pick the contextually appropriate sense of a lexical item. To this end, we exercise Extended Lesk, a classical word sense disambiguation (WSD) algorithm (Banerjee and Pedersen, 2003).

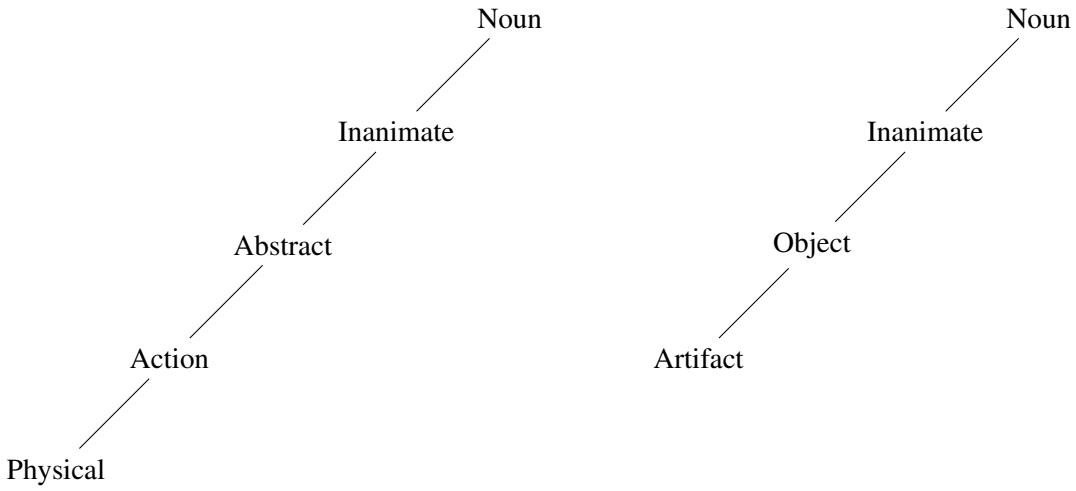


Figure 3.3 Two senses for the nominal $chaat$

Numeric Expressions: As is obvious, no lexical resource can have an exhaustive coverage because of the evolving nature of human language. In the context of HWN, the problem further intensifies as it restricts the entry to only words of open class syntactic categories. Apart from that, it also has a limited coverage for numeric expressions as these expressions belong to an infinite set. Numerals can be used in wide range of senses. Apart from their simple ordinal or cardinal usages, they can also be

used as nominals in expressions like time and measurement. In their adjectival sense, WN features can be extracted corresponding to the head word they modify e.g., the temporal sense of an expression $10\ saal$ can be identified by the head word $saal$ 'year'. However, to identify the temporal sense of a numeral, used as nominal, like 2013 is challenging. We use a numeric-expression recognizer, built in-house, to identify measurement and temporal expressions. The tool makes use of regular expressions and cue words. Once identified, we assign them an appropriate HWN ontological hierarchy which either corresponds to *time*, *measurement* or *number*.

Complex Predicate as a Feature: Complex predicates (CPs, also known as complex verbs) are highly frequent in South Asian languages (Mohanan, 1997). They occur in the form of $nominal + verb$ combinations (called conjunct verbs) and $verb + verb$ combinations (called compound verbs). For example, in (5), 'शरण लेना' (refuge take) is a complex predicate, composed of a nominal 'शरण' and a light verb 'लेना'. The constituents of a complex predicate are related by a dependency relation *pof* in HDT. In Hindi dependency parsing, the major chunk of parse errors is attributed to the low learnability of complex predicates (Husain and Agrawal, 2012). Begum et al. (2011) addressed the identification of these expressions using some linguistic rules. Fortunately, HWN has listed a finite set of these expressions in its database (Chakrabarti et al., 2007). We first extract the multi word expressions listed in HWN if the last word in the expression is a verb. Then, from the list, only 2-word expressions are selected and treated as complex predicates. Instead of adding WN features to the nominal of a complex predicate, we assign a separate CP tag to it. The semantics of light verbs is, however, kept as such.

3.2.4.2 Feature Design

After the extraction of WN features, we explore possibilities of their design and incorporation in the parsing framework, as follows.

Grouping Similar Features: We observed that few concept ontological lineages are semantically similar. For example, the six lineages depicted below address the notion of time.

- *Time*
- *Descriptive* → *Time*
- *Inanimate* → *Abstract* → *Time*
- *Inanimate* → *Abstract* → *Time* → *Period*
- *Inanimate* → *Abstract* → *Time* → *Season*
- *Inanimate* → *Abstract* → *Time* → *Mythological Period*

Since our focus is on adding representative semantic features which can assist parsing, we believe that such divergences should be grouped together. In the listed example, first, second and the last four differ in terms of their origin and belong to different branches in the hierarchy. Thus they can not be grouped by optimal depth selection (described later in *Ontology Depth* and requires a manual scrutiny. We studied the possible lineages in the concept ontology and performed merging wherever necessary, furnishing a semantically well diverse set of concept lineages.

Split Vs Conjoined: The concept lineage, derived for a word from HWN concept ontology, contains diverse concepts at each level of the lineage. The choice of using each of these concepts as independent features or the complete lineage as a single feature demands exploration. In the context of parsing, each independent concept from the lineage can potentially capture a specific aspect of syntax, depending on the fineness of the concept. The down side of this proposition is the increase in the feature dimensions, as each level adds a new dimension in the feature space. Whereas, using the complete lineage as a single feature does not add any additional dimension in the feature space but captures only a specific concept. This trade off is difficult to comprehend on theoretical grounds, hence we explore both choices of feature design in our experiments.

Ontology Depth: Hindi WordNet concept ontology furnishes a 'generalization hierarchy' for a lexical item, where the specificity of concepts increases as we move down the hierarchy. It may look intuitive to use fully expanded concept lineage, as it contains more detailed description of the lexical unit. However, opting for a highly fine-grained concept lineage leads to the problem of sparseness. It becomes less and less probable to find ample training examples as the feature becomes more fine-grained. At the same time, too much generalization is also unrewarding since the richer information is cast away in the excessive coarser lineage. This calls for measures to obtain an optimal depth of concept lineage for each lexical item. On one hand it should be generalized enough to give significant examples of its respective type while on the other hand, it should be fine enough to capture the rich ontological concept associated with the lexical unit. In order to quantify the trade-off, we resort to statistical correlation measures and employed *Gini Coefficient* (Gini, 1912). We computed the coefficient against all possible concept lineages in the training set and set a threshold. The lineages that fall below the threshold are generalized till they are above the threshold. For example, in Figure 3.1 the concept ape is suppressed to give the lineage till $mammal$ only. So, in future, if a word gives the lineage as in Figure 3.1, it will be replaced with its one level up generalization i.e. *Animate* $\rightarrow$ *Fauna* $\rightarrow$ *Mammal*.

3.2.5 Experiments and Results

In our experiments, we focus on establishing dependency relations between the chunkheads, which we henceforth denote as *inter-chunk* parsing. The relations between the tokens of a chunk (*intra-chunk* dependencies) are not considered for experimentation. In example (3), dotted line shows an *intra-chunk*

relation while the bold lines show *inter-chunk* dependency relations[6]. The decision is motivated by the fact that the *intra-chunk* dependencies can easily be predicated automatically using a finite set of rules (Kosaraju et al., 2012). Moreover we also observed the high learnability of *intra-chunk* relations from an initial experiment. We found the accuracies of *intra-chunk* dependencies to be more than 99.00% for both Labeled Attachment (LAS) and Unlabeled Attachment (UAS).

In this section, we present our parsing experiments incorporating the features extracted from HWN, as discussed in § 3.2.4. First, we setup our baseline parser followed by the detailed discussion on the impact of the individual features, extracted from HWN, on the overall parsing performance.

We setup our baseline parser on the lines of (Singla et al., 2012) with minor modifications in the parser *feature model*. We employ Malt parser version-1.7[7] (Nivre et al., 2007) and Nivre's *Arc Eager* algorithm for all our experiments reported in this work. All the results, reported, are evaluated using *eval07.pl*[8]. We use MTPIL (Sharma et al., 2012) dependency parsing shared task data described in § 3.2.3. Among the features available in the FEATS column of the CoNLL format data, we only consider *Tense, Aspect, Modality (tam)* and *postpositions (vib)* while training the baseline parser. Other columns like POS, LEMMA, etc. are used as such. After the baseline, the parsing framework is further enriched with the semantic features extracted from HWN to address the problems raised in § 3.1. These features are added in the FEATS column of the data, separated by '|'. In a pilot experiment, split form of features, as discussed in § 3.2.4.2, is found to perform better than conjoined form, which motivated us to use WN feature in split form in all our experiments. The experimentation proceeds in the order as listed in Table 3.3, which also presents the consolidated results of our parsing experiments using the MTPIL training and testing sets. In order to see the impact of semantic information on data sparsity, we split the MTPIL training set into datasets of different sizes. We experiment with 6 data sets of different sizes. The results are produced on MTPIL test set and are plotted on Graph (Figure 3.4). The increase in LS and LAS, as the training size decreases, show the impact of semantic information on data sparsity. The improvement of 1.1 (LAS), by introducing semantics, upon reducing the training examples to 1000, implies that semantics can address the data sparsity and OOV problems, when working with out-of-domain text.

Next, we discuss the impact of WN features on the accuracy of our parsing results produced on datasets of different sizes:

- $Sense\ Selection$: As discussed in § 3.2.4.1, we performed two experiments to extract the WN features corresponding to the most appropriate sense of a lexical item. In the first experiment, the first sense of each lexical item is selected, while in the second, WSD is used to pick the contextually most appropriate sense. These features, corresponding to the chosen sense, are coupled with the features already present in the baseline. As depicted in Graph (Figure 3.4), there is an average increase of 0.38 (LAS) on all datasets using the first sense strategy from the baseline. However,

[6] k1: Doer, k1s: Noun Complement, k5: Source, k7p: Place, k7t: Time, pof: part-of (complex predicate), lwg__psp: local-word-group post-position

[7] `http://www.maltparser.org/download.html`

[8] `http://nextens.uvt.nl/depparse-wiki/SoftwarePage/\#eval07.pl`

	Experiments	LAS(%)	UAS(%)	LS(%)
E1	Baseline	83.69	92.43	86.58
E2	E1 + First Sense	83.78	92.4	86.73
E3	E1 + WSD (Extended Lesk)	83.6	92.34	86.57
E4	E2 + Numeric Expressions & Grouping	**83.88**	**92.45**	**86.87**
E5	E4 + Ontological Depth	83.84	92.4	86.79
E6	E4 + Complex Predicate	83.75	92.39	86.72
E7	E5 + E6 (Complex Predicate + Ontological Depth)	83.74	92.39	85.7

Table 3.3 Results of Parsing Experiments

using WSD, the accuracy decreased across all datasets. As is obvious, the fall in accuracy can be attributed to the wrong sense selection. The problem can be addressed by using better WSD algorithms for Hindi.

- $Numeric\ Expressions\ and\ Grouping$: As discussed in § 3.2.4.1, numeric expressions and sense grouping increase the coverage of HWN. This obvious reason is clearly depicted in the improvement in parsing results as shown in Table 3.3. More the number of lexical items with semantic information available in the data, more will be its impact on the parsing.

- $Depth\ of\ Information$: The optimality of feature coarseness is put to test in this experiment. This experiment is run on numeric expression data with feature pruning done as described in § 3.2.4.2. An increment of average 0.03% LAS across datasets is observed from the previous experiment. In the test set, there are only a few cases that are updated by choosing an optimal lineage depth which explains the minimal increase in accuracy.

- $Complex\ Predicate$: As pointed in (Begum et al., 2011), addressing the low learnability of complex predicates can improve the parsing results. The improvements are particularly seen in the core arguments of a verb. The similar syntactic distribution of adjectival or nominal element of a complex predicate and the syntactic arguments of a verb, particularly $objects$, make these expressions highly ambiguous. Identifying these expressions beforehand, as suggested in (Begum et al., 2011), improves the parsing performance. The incorporation of this crucial information from HWN is rewarding as we achieve an improvement of $\sim 0.4\%$ in LAS on a dataset of $1,000$ sentences.

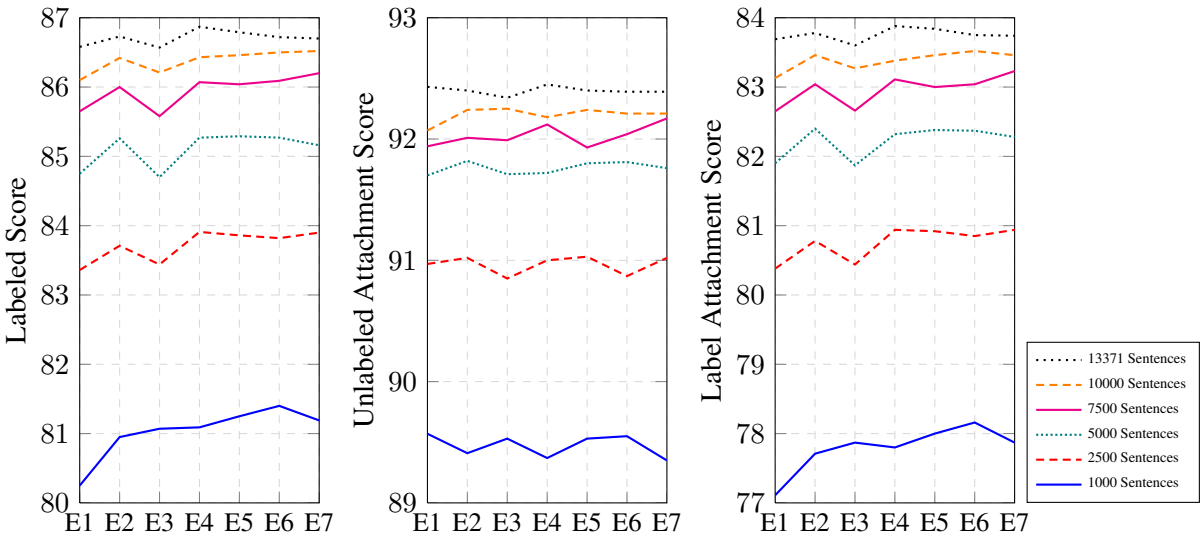
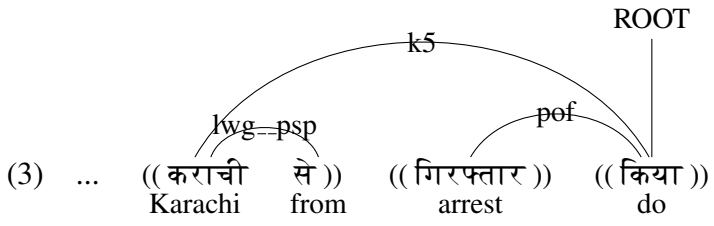


Figure 3.4 Impact of WN features on different data sizes

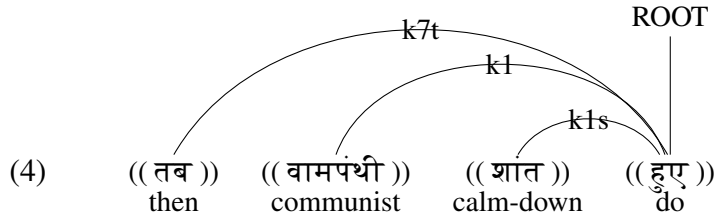
3.2.6 Discussion

In this section, we discuss further, how well the issues, raised in § 3.1, are handled by the incorporation of semantic information in the parsing framework of Hindi. In § 3.1, we stated that ambiguities in morphological cases in Hindi bar their efficient exploitation while parsing. Also we noted that unmarked nominals may as well affect the performance of a parser. So we propose semantics as a complementing information that can fill these gaps. Below we discuss whether semantic information has bridged these gaps or not.

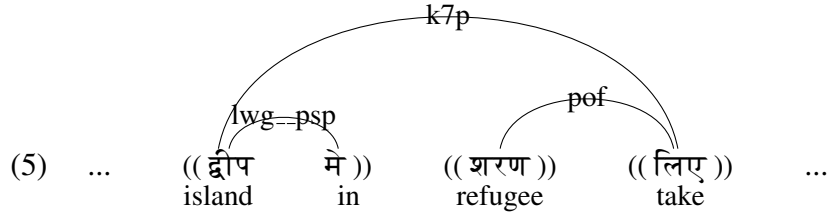
- *Case Ambiguity*: Incorporating the semantics from HWN, to help disambiguate the confusion present in a case marker, has improved parsing accuracy. Particularly, confusion among the roles of concrete *vs* abstract time and place, and direct *vs* indirect object relations, have been removed. In example 3, the dependency relation between nodes $Karachi$ and do has been corrected from $k2$ 'Theme' to $k5$ 'Source'. The post-position $from$ can either mark a theme or a source relation. Semantics has removed this confusion.



- *Lack of Case Marker*: In absence of case marking, lexical semantics acted as a complementing information. The improvement has been, as observed during error analysis, particularly for agents and patients. Thus, semantics can be seen here as pseudo case markers. This is clearly visible from the example (4). The dependency relation between the nodes *then* and *do* has been corrected to $k7t$ 'time of action' from $k1$ 'subject'.



- *Complex Predicates*: As we have discussed above, complex predicates are identified using HWN, so that the similar syntactic distributions of verb arguments and the nominal or adjectival part of a CP can be disambiguated. Identifying the complex predicates, has turned to be rewarding. As was expected, the prior identification of CPs has significantly improved the joint identification of label and attachment. The system, trained on $1,000$ sentences, has shown an improvement of 0.34% (LAS) and $\sim 0.2\%$ (UAS) by prior identification of complex predicates. The confusion, that has been removed, is among the arguments of a verb and the nominal part of the CP i.e., between agent, patient vs nominal, adjectival part of CP. In example below, baseline incorrectly identifies $refuge$ as an argument of verb $take$. '$refuge\ take$' is a complex predicate, which is correctly identified upon incorporation of complex predicates in our parsing module.



3.3 Summary

We tried to address some of the shortcomings, as observed in previous chapter, while exploring lexical resources, Hindi WordNet in our case, to discover features which can complement the available

morpho-syntactic features conventionally explored for parsing. We find concept ontology, available in HWN, quite resourceful in furnishing features which can essentially break syntactic ambiguity, resulting in better accuracies for parsing. The semantic information is extracted from Hindi WordNet in an automatic, analyzed and informed fashion. Though it did not increase the LAS quite significantly over a large amount of training data, significantly benefited a smaller size of training data. It also helped in enriching the Hindi Dependency Treebank with lexical and semantic information, which could be further exploited as per the users' needs in areas like Question-Answering systems, NLIDB systems and in other areas as well.

3.4 Conclusion

We chose to aid dependency parsing, particularly interChunk parsing, by using semantic information, which provided us with many new insights in solving problems like Case Ambiguity, Lack of Case Markers and also the trade-off between the size of training data and the complementarity of semantic information. Still it did not improve the parsing accuracy (LAS) to a significant level on a large size of training data. Now the journey takes a turn to explore a well-established approach i.e. Ensembling, to exploit different algorithmic variations provided in Malt parser.

Chapter 4

Advancements to Dependency Parsing: Ensembling in Hindi

In the last chapter we explored how semantic information, as extracted from a rich lexical and semantic resource Hindi WordNet, could leverage the parsing accuracy. Simultaneously, we also noted that as the size of the training data increases, the importance of the information starts getting sidelined (cf. Figure 3.4). In such a scenario, it becomes a tedious task to further improve the parsing accuracy. It requires too much efforts and perhaps a bag of approaches each along with its own strength and diverse nature, rather than focusing on a single approach (as in our case a single parse tree), to raise the bar even a little higher. In this chapter we are exploring one such path in the form of a well established approach, **Ensembling**, to make some advancement in the process of dependency parsing.

4.1 Ensembling

Ensembling, more commonly known as a combination technique, is an approach based on integrating different tools, designed for the same task, in order to effectively complement the strengths of one another, to develop a high quality system. Last decade has witnessed an increasing adaptation of ensembling for syntactic parsing over multiple languages (Hall et al., 2007; Martins et al., 2008). The aim of any ensembling methodology, in context of syntactic parsing, is to exploit diversity captured by different parsers individually. For the first time, this property has been explored by Henderson and Brill (1999). They employed this technique for constituent parsing and achieved significant improvements in parsing accuracies. Due to its increasing utility, ensembling has also been used extensively for dependency parsing (Zeman and Žabokrtskỳ, 2005; Sagae and Lavie, 2006; Sagae and Tsujii, 2007; Hall et al., 2007; Reichart and Rappoport, 2007; Nivre and McDonald, 2008; Martins et al., 2008; Zhang et al., 2009), perhaps more than constituent parsing (Fossum and Knight, 2009).

The combination approaches, proposed for the parsing of syntactic dependencies, can generally be classified into two categories:

1. **Parser Combination during Training Time:** In this model of ensembling, base parsers are integrated at the learning time, e.g. using *Stacking* (Nivre and McDonald, 2008; Martins et al., 2008; Attardi and Dell'Orletta, 2009). Nivre and McDonald (2008) first experimented this approach by

integratiing two base dependency parsers at learning time using a setup, which they referred as *guided parsing*. They integrated graph-based (MST parser (McDonald and Pereira, 2006)) and transition-based (Malt parser (Nivre et al., 2007)) dependency parsers to achieve improvements in parsing accuracy for 5 out of 12 languages, they experimented with. Later, Martins et al. (2008) introduced the generalized framework for *stacking* for dependency parsing, as a way of extending a parsing model's feature space. They proposed stacked learning as a way of incorporating non-local features into dependency parsing.

2. **Parser Combination at Inference Time:** Here, ensembling is done by combining independently-trained models only at parsing time (Henderson and Brill, 1999; Sagae and Lavie, 2006; Sagae and Tsujii, 2007; Hall et al., 2007; Zhang et al., 2009; Fossum and Knight, 2009; Surdeanu and Manning, 2010). The correctness of the final dependency tree is ensured by: (i) selecting entire trees proposed by base parsers (Henderson and Brill, 1999) or (ii) re-parsing the pool of dependencies proposed by base models (Sagae and Lavie, 2006). The parser combination techniques proposed by Sagae and Lavie (2006) have not only been proved successful for multilingual parsing (Hall et al., 2007) but also for domain adaptation (Sagae and Tsujii, 2007).

Ensembling For Indian Languages: Kolachina et al. (2010)'s, was the first attempt at exploring ensembling approaches such as *blending*, to parse Indian languages in ICON 2009 Shared Task (Husain et al., 2010). In their system, they tried to exploit the fact that blending is only successful when more diversity is added. Though the fact is completely true, as we have also observed above, they experimented in a brute-force way to add this diversity. In their goal of increasing diversity in base parsers, they blended Single Malt systems with feature models, used even for the non-configurational languages other than Hindi, from the CoNLL Shared task. Except for Bangla, this approach failed in bringing any improvements for either Hindi or Telugu. Furthermore, they, like many others, explored only CPOS-weighted dependencies in their ensemble models. In the same shared task, another ensembling system (Zeman, 2009) was explored using three different base parsers namely, Malt parser, MST parser and DZ parser [1]. Zeman (2009) used UAS score of each base parser to weight votes, obtained by the same parser. Though he experimented with ensembling in different ways, the work could not provide any new insights. Also his work was not concentrated in seeking a detailed view of ensembling for Hindi and even the error analysis was done on the development data, which itself was used to tune the weights (UAS).

Ensembling for Indian languages (Bangla, Hindi and Telugu) via. *Stacking*, has been explored to some extent, by Kolachina (2012). In a stacked parsing architecture, a level-1 parser is trained not only on a manually-annotated treebank but also on the output of another level-0 parser (or multiple level-0/base parsers). He experimented with, stacking a graph-based data-driven dependency parser (MST parser (McDonald and Pereira, 2006)) as a level-1 parser on 3 level-0 dependency parsers namely,

[1] `http://ufal.mff.cuni.cz/~zeman/projekty/parser/`

transition-based data-driven Malt parser (Nivre et al., 2007) [2], MST parser [3] and grammar-driven Constraint-Based Hybrid Parser (CBHP) (Bharati et al., 2009a; Bharati et al., 2009b; Husain, 2011) [4], CBHP was based on the framework of Bharati and Sangal (1993). Both, *MST+MST* and *MST+Malt* stacked models, performed marginally better as compared to the baseline systems available at that time for all the three languages. The third model *MST+CBHP*, which could be employed only for Hindi, stacked both coarse-grained [5] and fine-grained [6] MST models on top of a coarse-grained CBHP model. Though *MST+CBHP* stacking performed significantly better than the baseline coarse-grained CBHP system for both the cases, it could hardly beat the already available fine-grained baseline dependency system for Hindi.

Kukkadapu et al. (2012) is the most recent work explored ensembling in dependency parsing for Hindi. They explored two different methods i.e. simple voting (Zeman, 2009) and blending method (Sagae and Lavie, 2006), to combine three different base parsers namely, Malt parser, MST parser and TurboParser [7] (Martins et al., 2010) in MTPIL Shared Task on Hindi Dependency Parsing (Sharma et al., 2012). For the first time Turbo parser was used to parse Hindi sentence constructions and it did perform better than all other systems in automatic track. Though they achieved best results on automatic track, not by ensemble system of three base parsers rather using only Turbo parser. This proved to be one of the most important contributions of their work. Their simple voting system did outperform all other single base parsers on gold standard text but could not beat the best system which was based on using only Malt parser (Singla et al., 2012). This clearly shows that previous works have not been successful in their efforts of utilizing ensemble systems to aid dependency parsing for Hindi.

Kolachina et al. (2010), Zeman (2009) and Kukkadapu et al. (2012) only explored ensembling for Hindi dependency parsing in brief. Though some serious effort was done by Kolachina (2012) via stacking, all the three stacked systems explored could not prove beneficial for either of the languages. Thus, the pre-conceived best stacked combination of *MaltParser+MST*, which provided best results for some languages (Nivre and McDonald, 2008; Martins et al., 2008), failed to make any impact on Indian languages' dependency parsing. Although Kolachina (2012) also experimented with the other ensembling approach i.e. ensembling during runtime but only for English (not for Hindi) and reported best parsing accuracies for English at that time. In addition, Surdeanu and Manning (2010) had already reported that ensemble systems have huge potential to improve the parsing accuracy and building such systems can be both *cheap* and *simple*. All these results and observations motivated us to explore the not-so-well explored path of employing ensembling techniques for Hindi at inference time using both re-parsing algorithms and word-by-word voting (simple voting) techniques.

[2] http://www.maltparser.org/
[3] http://sourceforge.net/projects/mstparser/
[4] http://researchweb.iiit.ac.in/samar/tools/GH-CBP-Hindi-release_version-1.6.tgz
[5] Dependency Parsing with a maximum of 26 dependency labels
[6] Dependency Parsing with more than 60 dependency labels
[7] http://www.cs.cmu.edu/~ark/TurboParser/

4.1.1 Ensembling for Hindi at Inference time: Reparsing Algorithms

Our ensembling system is based on the methodology proposed by Sagae and Lavie (2006). Given the output dependency graphs DG_p ($1 \leq p \leq P$) of P different base parsers for an input sentence S, we construct a weighted directed graph $DG = (V, E, s)$, where $E = \{e : e \in DG_p, \forall p \in P\}$, $V = \{v : v \in S\}$ and $s : E \rightarrow \mathbb{R}$ is a cost (or weight) function defined on E. Each edge e is weighted by a score $s(e)$ reflecting its popularity among the P parsers. A Directed Spanning Tree (DST) of DG, rooted at r ($r \in V$), is a subgraph T of DG, such that undirected version of T, is a tree and T contains a directed path from r to any other vertex (nodes or words) in V. The cost $s(T)$ of directed spanning tree T, is the sum of the costs of its edges, i.e. $s(T) = \sum_{e \in T} s(e)$. The output of the ensemble system for S will be the **Minimum Directed Spanning Tree** ($MDST$) of DG, rooted at r (or node 0), which is a directed spanning tree of minimum cost i.e., $s(MDST) = min\{s(T) : T \in DG\}$. $MDST$ can be extracted using *Chu-Liu-Edmonds* algorithm (Chu and Liu, 1965; Edmonds, 1967). McDonald et al. (2005) and Hall et al. (2007) have used Maximum Spanning Tree (MST), instead of $MDST$, to maximize the cost of the resultant tree by using parser accuracies (or confidence scores), achieved by different base parsers. The cost (or weight) function, we use, to score each edge e is $s(e) = min\{(w_p^c e_p)_{p=1}^P\}$, where w_p^c is the weight assigned to each parser p for the word context c [8] of the dependent of e, and e_p is 1 if $e \in DG_p$ otherwise 0.

Our weight w_p^c, is slightly different from what Hall et al. (2007) have used since we are considering edges weighted with confusion scores. We are measuring the popularity of an edge e, by its tendency to be less error prone. In other words, each edge e has been assigned score as the confusion made by the parser p in the word context c while predicting the dependency relation. Hence, the task of our ensemble system is to output $MDST$ by minimizing the total confusion of the resultant tree. We capture this confusion as $w_p^c = (100 - avg(Accuracy_p)^c)$ [9], where $avg(Accuracy_p)^c$ is the averaged Accuracy [10] of parser p. Also, unlike others (Hall et al., 2007), we are not adding the scores of those edges which are directed in the same direction with different weights between two nodes (words). Instead, we select the edge with least score so as to minimize the total cost of $MDST$. In future we would like to explore different methods of using the weighting function in much more detailed and exhaustive manner, including the one used by Hall et al. (2007).

Setup: We are using six base parsing models as six different variants of Malt parser. The six Malt parser variants are built by varying the parsing algorithms but not in the same way as done by Hall et al. (2007) who used 3 algorithms in 2 directions (right-to-left and left-to-right). Instead, we are using six different algorithms in only one direction i.e., left-to-right, to utilize the diversity of algorithms in particular. All these algorithms belong to two families of algorithms out of the three families as

[8] A context is any basis which can be used to differentiate or categorize a word like Part-of-Speech tag, the label of dependency a word has with its parent or any other context.

[9] Instead of directly using avg(Accuracy) to weight an edge, we are using $(100 - avg(Accuracy))$ to aid the ensembling process in later part of the thesis, where we will employ a better technique of calculating confusion score

[10] Accuracy can be either LAS, LS or Recall depending on the context c

provided with Malt parser. The six algorithms we used in our experiments are: Nivre Arc Eager (AE), Nivre Arc Standard (AS), Covington Projective (CP), Covington Non-Projective (CNP), Planar (P) and 2-Planar (2P) (Nivre, 2003; Nivre, 2004; Covington, 2001; Gómez-Rodríguez and Nivre, 2010). AE, AS, P & 2P belonging to Nivre family, are linear-time ($\Theta(n)$) and CP & CNP belonging to Covington family are quadratic ($\Theta(n^2)$) [11].

Malt parser system has three group of parameters (as already mentioned before): parsing algorithm parameters, feature model [12] parameters and learning algorithm parameters. We are building base parsers by varying the algorithmic parameters only and not other parameters. Corresponding to all the algorithms, feature models are adjusted with respect to the most obvious differences in parsing strategy (e.g. by deleting features that could never be informative for a given parser or adding those which can utilize the rich morphological information such as *Tense, Aspect, Modality* (tam) and *postpositions* (vib) available for Hindi). For learning algorithms, we did not make any changes in learning algorithm (SVM) settings and used the same settings as used previously (cf. § 2.1.2.7).

In our experiments, we used the same data set as used in earlier experiments (cf. § 3.2.3). Table 4.1 shows the performance of our six algorithmically varied base parsers in terms of LAS, UAS and LS on the test data.

Parser	LAS	UAS	LS
Malt$_{AE}$	**83.69**	**92.43**	**86.58**
Malt$_{AS}$	83.46	92.03	86.47
Malt$_{CNP}$	82.41	91.49	85.40
Malt$_{CP}$	82.31	91.31	85.37
Malt$_{2P}$	81.69	90.27	85.34
Malt$_{P}$	81.58	90.06	85.29

Table 4.1 LAS, UAS and LS for the base models. The parsers are listed in descending order of LAS obtained on test data

[11] To know more about the algorithms refer to `http://www.maltparser.org/userguide.html#parsingalg`

[12] Malt parser use a history-based feature model for predicting the next parsing action. Each feature of this model is an attribute of a token, where attribute can be any of the symbolic attributes in the CoNLL format (cf. § 2.1.2.1)

4.1.1.1 Experiments and Results

Experiments for ensembling have been done combining the six base parsers taken x at a time. To further increase the diversity among ensemble models, we considered six different contexts (c) to calculate the weight (w_p^c) and then use it to assign a score ($s(e)$) to each edge (e) of the output tree from the parser (p). Many of these contexts are guided by the contexts used by Hall et al. (2007) and Surdeanu and Manning (2010).

Calculation of weights based on different contexts: We are using accuracies corresponding to 6 different contexts to obtain the weight. The accuracies are extracted by training base parsers on 90% of the training data and the remaining 10% is used to tune those accuracies. In other words, from the evaluation of the parsed development data, the accuracies are extracted corresponding to the contexts. Hall et al. (2007) has used only a single context CPOS in their weight function and Surdeanu and Manning (2010) has used 5 different weighting strategies. We have used all of their strategies in our ensemble models except sentence length. The six different contexts used are the following:

1. *Unweighted:* No context is used and all the dependencies are assigned an equal weight of 1 unit. This is very important to consider as shown in Surdeanu and Manning (2010).

2. *CPOS:* Coarse Part-of-Speech (CPOS) tag of the dependent of an edge is used as the context and the averaged accuracy taken in terms of the recall of CPOS, which can also be termed as LAS of base models for CPOS, is used as the weight in the score function.

3. *LAS:* LAS is taken as the weight considering each parser as a context in itself.

4. *Label of Dependency (drel):* We used dependency label as the basis to differently weight the dependency relations and corresponding LS score is taken as the weight.

5. *Label of Dependency (DRel):* Though the context is same as in $drel$ i.e., dependency label marked to an edge, the weight is corresponding LAS score. The difference is that LS only covers the accuracy in terms of labels but LAS covers both correct labels and attachments (or edges) thus providing a relatively better scenario.

6. *Dependency Length:* The distance between a parent and the child in a dependency relation is used as the context. This distance is split into 5 bins as: (i) to_root (or 0): denotes that the word has no parent and it is the root of the sentence (ii) 1 (iii) 2 (iv) $3-6$ and (v) $7-$ .... We have categorized distance into 5 categories because the evaluation file generated by *eval07.pl* (Nilsson and Nivre, 2008) [13], provides recall for these categories only.

As an important note, it should be mentioned explicitly that when the ensemble models are used to re-parse the development data during dry-run, they were bound to be overly optimistic due to the fact

[13]The evaluation script of the CoNLL-X shared task, containing minor modications as compared to eval.pl

that the same development data was used to tune the above six types of weights, which were used further in building ensemble models.

Results: Using the above 6 scoring techniques, votes by different parsers have been weighted and different ensemble systems have been built by combining x parsers at a time. Table 4.2 shows the results of the best combinations of x base parsers at a time, which achieved best accuracy among respective combinations, using each of the weighting strategy.

4.1.1.2 Discussion

Our results seem similar to the previous works (Kolachina et al., 2010; Zeman, 2009; Kukkadapu et al., 2012). *Ensemble*$_2^{LAS}$ performed best among all the ensemble models. But instead of improving parsing accuracy, it kept the accuracy exactly same while all other systems decreased LAS score from baseline. This is due to the reason that *Ensemble*$_2^{LAS}$ used LAS as the context to weight the votes and thus maintained at least that much score. Though *Ensemble*$_2^{DRel}$ did increase LS marginally (0.04%) but overall decreased LAS by 0.02% from baseline. The results on development data were positive but as we have already mentioned they were over-optimistic, so they can not display a complete picture. One of the observations that we noticed is that as the number of base parsers increases (in combination), the accuracy decreases from the baseline. One of the possible reasons could be the lack of diversity in the base parsers as compared to Hall et al. (2007) and Surdeanu and Manning (2010), who, though did not use as many algorithms as we did, made the variations in the algorithms themselves by using two different models for each algorithm to parse the text in both forward and backward directions. Another point mentioned in Hall et al. (2007) was that the systems did not seem to perform as better for free word order (FWO) languages as compared to other languages. Since Hindi is a free word order language, the observation does co-relate with our results too. Another possible reason can be that several shortcuts such as optimizing learning algorithms' parameters for speed rather than accuracy and extrapolation instead of proper tuning for the algorithms have not been fully explored due to time constraints. This strongly suggests that there is a probable chance to improve the performance of six-base parsers separately by tuning parameters, which eventually will help in raising the standard of ensemble models.

Another important insight taken from the above experiments is that, though not even a single system did outperform the baseline system, the ones which were closest to the baseline, were all weighted systems and not unweighted one. Thus working towards improving weighting strategy, may perhaps improve the parsing accuracy.

Since ensembling via re-parsing algorithm technique, lacked in making any significant advancement to the dependency parsing scores, we started searching of other ensembling techniques through which

[14]To know about which combinations (Ensembled Systems) of base parsers as mentioned in Table 4.2 performed best, among all possible systems when any x number of parsers are ensembled, refer to Appendix C

	Unweighted			Weighted by CPOS of modifier			Weighted by LAS		
Best-Ensemble[14]	**LAS**	**UAS**	**LS**	**LAS**	**UAS**	**LS**	**LAS**	**UAS**	**LS**
Ensemble$_2$	83.60	92.22	86.44	83.57	92.23	86.45	**83.69**	**92.43**	86.58
Ensemble$_3$	82.57	91.20	85.73	83.13	92.13	85.73	83.26	92.04	86.31
Ensemble$_4$	82.28	90.84	85.51	82.28	90.84	85.51	82.61	91.33	85.92
Ensemble$_5$	81.80	90.32	85.11	81.80	90.32	85.11	81.22	90.40	84.58
Ensemble$_6$	81.50	89.97	84.94	81.50	89.97	84.94	80.69	89.91	84.28

	Weighted by label of Dependency (LS)			Weighted by label of Dependency (LAS)			Weighted by dependency length		
	LAS	**UAS**	**LS**	**LAS**	**UAS**	**LS**	**LAS**	**UAS**	**LS**
Ensemble$_2$	83.68	92.23	86.61	83.67	92.23	**86.62**	83.56	92.21	86.46
Ensemble$_3$	83.07	91.75	85.86	82.85	91.60	85.82	83.02	91.65	86.07
Ensemble$_4$	82.70	91.31	85.76	82.54	91.23	85.73	82.54	91.24	85.73
Ensemble$_5$	81.33	89.85	85.13	81.33	89.97	84.98	81.39	90.55	84.64
Ensemble$_6$	80.91	89.41	84.89	80.87	89.50	84.67	80.93	90.14	84.34

Table 4.2 Scores of best combination models based on re-parsing algorithms using different voting strategies. Ensemble$_x$ denotes that combination of base parsers taken x at a time.

ensembling could be done at inference time while also improving the accuracy. Though Sagae and Lavie (2006) clearly mentioned that the most simplest approach to do ensembling could be by using word-by-word voting technique but they themselves did not experiment with it. The reason mentioned was that the experiments of Zeman and Žabokrtskỳ (2005), which though performed better using this technique, dropped the performance sharply when constraint of formation of well-formed dependency trees [15] was applied. But later the work of Surdeanu and Manning (2010) claimed that cheap and simple ensemble models could be generated even using simple word-by-word voting by proving it in the context of dependency parsing for English. Also Kukkadapu et al. (2012) showed that word-by-word voting improves accuracy for Hindi. So taking motivation from this reasoning, we also tried to implement word-by-word voting technique of ensembling. Similar to re-parsing algorithm technique for Hindi, this technique has also not been explored earlier in detail. Furthermore, it can also help us to frame a systematic comparison between both the approaches of ensembling to lay directions for future work. In addition, experimenting this approach can also provide a holistic picture of ensembling for Hindi as after implementing it, all the three best approaches of ensembling, i.e. re-parsing (Re-parsing Algorithm and Word-by-Word voting), and Stacking (Kolachina, 2012) would have been explored.

4.1.2 Ensembling for Hindi at Inference time: Word-by-Word Voting

Word-by-Word Voting is one of the most common approaches of integrating independently-trained models at parsing time, to assign each dependency relation a score based on the number of votes it receives from the base parsers. But since the goal of ensembling is to capture more and more diversity through different base parsers, their strength could be captured by using different weighting strategy for voting. Also as we have already shown above that weighting matters for Hindi, we are using the same six weighting strategies that we have used in the previous technique of ensembling. The set-up involving six base parsers based on six different algorithms would also be same. This will help us in making a systematic comparison between both the ensembling techniques. Further as mentioned above, the exploration is also motivated by the bold claims made by Surdeanu and Manning (2010). Table 4.3 reports the results from all the experiments combining six different base parsers taken x at a time, over all the unweighted and weighted voting strategies.

4.1.2.1 Discussion

It is clearly evident from Table 4.3 itself that word-by-word voting increases the performance of ensemble models as claimed by Surdeanu and Manning (2010). Thus, ensembling for Hindi via word-by-word voting re-establishes their approach for Hindi language also. Though unweighted systems per-

[15] A dependency tree which does not contain multiple roots, zero roots or cycles

[16] To know more about which combinations (or ensemble systems) of base parsers, as mentioned in Table 4.3, performed best, among all possible systems when any x number of parsers taken at a time, refer to Appendix C

Best-Ensemble[16]	Unweighted			Weighted by CPOS of modifier			Weighted by LAS		
	LAS	**UAS**	**LS**	**LAS**	**UAS**	**LS**	**LAS**	**UAS**	**LS**
Ensemble$_2$	83.03	92.23	86.42	83.55	92.29	86.53	83.69	92.43	86.58
Ensemble$_3$	84.02	92.71	86.88	84.06	92.78	86.92	84.06	92.72	86.87
Ensemble$_4$	83.88	92.73	86.91	83.95	92.75	86.87	84.10	92.78	86.95
Ensemble$_5$	84.01	92.68	86.91	84.10	92.73	**87.01**	84.14	92.77	**87.01**
Ensemble$_6$	83.77	92.60	86.79	84.01	92.74	86.94	84.13	92.77	87.00

	Weighted by label of Dependency (LS)			Weighted by label of Dependency (LAS)			Weighted by dependency length		
	LAS	**UAS**	**LS**	**LAS**	**UAS**	**LS**	**LAS**	**UAS**	**LS**
Ensemble$_2$	83.65	92.21	86.61	83.63	92.19	86.62	83.55	92.26	86.46
Ensemble$_3$	84.04	92.68	86.87	84.06	92.68	86.91	84.03	92.72	86.85
Ensemble$_4$	**84.16**	92.73	87.00	84.15	92.72	87.00	84.09	**92.79**	86.93
Ensemble$_5$	84.04	92.62	86.90	83.96	92.59	86.81	84.02	92.68	86.89
Ensemble$_6$	84.04	92.64	86.88	84.01	92.58	86.90	84.06	92.70	86.89

Table 4.3 Scores of best combination models using different voting strategies. The combined trees are assembled using a word-by-word voting scheme.

formed better than individual baseline system, they could not beat the weighted systems. *Ensemble*$_4^{drel}$, the best weighted ensemble model, outperformed the best unweighted model *Ensemble*$_3^{unweighted}$ by a margin of 0.14%. Even there are many more weighted ensemble models which performed better than unweighted ones either marginally or significantly. In addition to improving parsing accuracy as compared to re-parsing algorithm ensembling technique, word-by-word voting is also cheap in terms of time-complexity.

Effect of Single and Ensemble systems on dependencies in terms of distance: McDonald and Nivre (2007) mentioned that a single base (Malt) parser tends to suffer from two problems: (1) Low precision on dependencies originating in the artificial root node due to fragmented dependency trees [17], and (2) Error propagation due to the deterministic parsing strategy, typically affecting long dependencies more than short ones.

Hall et al. (2007) clarified that the first problem is more severe using ensemble models as compared to a single base parser. They showed that precision for root dependents from a blended output is lower than from a single malt output. But this is not true in our case for Hindi. In fact the results are opposite. Table 4.4 shows that for *root* dependents, where recall is improved by 1.43%, precision on the other hand remained unchanged (96.26%), which gives a net increment of ~ 0.35 in F_1 score. The precision remained unchanged because the ensemble system predicted more fragmented trees (27) but at the same time also corrected more parse trees (26) as compared to best performing single base parser (Malt$_{AE}$). The same fact has also been pointed out by others (Surdeanu and Manning, 2010; Sagae and Lavie, 2006). Though ensemble systems improve accuracy, they do not guarantee well-formed trees [18]. In fact, the resulting graph may not even be connected. But when such errors were calculated for our experiments, we found out that the percentage of badly-formed dependency trees is only $\sim 2.8\%$ $(51/1828)$ of testing data (in-domain) in the parser output by *Ensemble*$_4^{drel}$, which is not as high for Hindi as reported by Surdeanu and Manning (2010) for English. Thus, it is not correct to say that ensembling increases the severity of fragmented dependency trees, at least not for Hindi. As suggested by Surdeanu and Manning (2010), to prevent the additional 27 badly-formed trees while keeping 26 well-formed trees remain intact, search for better re-parsing algorithms (which failed in our case) should be continued.

The second problem of error propagation in single base parsing systems also improved which is similar to Hall et al. (2007). Except decrement of 0.08 in F_1 score for dependencies with distance *two*, all other dependencies irrespective of distance, showed positive results with a maximum improvement of 0.42 in F_1 score in long dependencies ($7+$). This clearly shows that effect of error propagation is mitigated to some extent, using an ensembling system even though all the base parsers are deterministic.

Though the second approach of ensembling (word-by-word voting) performs better than any of the previous ensembling methodologies, a detailed analysis of both the approaches is necessary to completely understand the positives and negatives of both the approaches.

[17] A dependency tree where there are multiple roots or a forest instead of a single tree

[18] A dependency tree which does not contain multiple roots, zero roots or cycles

	root		1		2		3-6		7+	
Parser	**R**	**P**	**R**	**P**	**R**	**P**	**R**	**P**	**R**	**P**
Malt$_{AE}$	97.26	**96.26**	**97.24**	96.37	**90.66**	**90.76**	**89.84**	91.74	88.91	**89.36**
Ensemble$_4^{drel}$ (Word-by-Word)	**98.69**	**96.26**	97.03	**96.72**	90.59	90.52	89.51	**92.46**	**91.51**	88.48
Δ F_1	$+0.35$		$+0.04$		-0.08		$+0.09$		$+0.42$	

Table 4.4 Recall (R), Precision (P) and Change in F_1 score (Δ) of State-of-the-art Hindi Dependency Parser and Best performing Ensemble Model for dependencies of different length (root = dependents of root node, regardless of length).

4.1.3 Systematic Analysis of Reparsing Algorithms and Word-by-Word Voting

Since many ensembling techniques had already been explored in past for multiple languages using Malt parser (Hall et al., 2007; Surdeanu and Manning, 2010), they helped us in employing ensembling for Hindi dependency parsing. But a systematic comparison similar to Surdeanu and Manning (2010) is necessary to present the real picture of ensembling for Hindi. In this context, we can surely gain from the significant work done by Surdeanu and Manning (2010), in raising various questions about ensembling, in general, and then answering them in the context of dependency parsing, in particular, for Hindi. We pose the same questions to our approach of *Ensembling* and try to address some of the issues raised in Surdeanu and Manning (2010). This will help not only in providing a systematic analysis about our approach but may also prove useful in finding new directions to be explored in future.

Q1. When combining models at parsing time, what is the best scoring model for candidate dependencies during re-parsing? Can a meta-classifier improve over unsupervised voting?

A1. Surdeanu and Manning (2010) showed that weighting strategies do not have an important contribution in overall performance of English dependency parsing as only one weighted model could outperform the unweighted voting model and that too only marginally. This observation is in contrast to our first technique of ensembling to some extent. Though unweighted model, instead of increasing accuracy, degrade it, all other weighted models also performed badly. But one observation that is insighful is that, some of the weighted models despite of decreasing accuracy, performed marginally better than the unweighted model. This suggests that weighting can not be discarded altogether and need to be explored further to capture any chance of improving parsing accuracy.

The importance of weighting over unweighted becomes even more important for the second technique of ensembling (word-by-word voting), when compared with Surdeanu and Manning

(2010)'s work. Though unweighted system improved the parsing accuracy from the baseline, out of 25 weighted models that we employed, 15 completely outperformed the unweighted one and 3 were at par with the unweighted one (difference $= 0.01\%$). Also to achieve this feat, unlike Surdeanu and Manning (2010) where 6 base parsers were required, it just require 4 base parsers for us and on the top of that, all the ensemble models generated combining 3-base parsers clearly outperformed the best unweighted model. This completely justifies that weighting matters for Hindi, irrespective of the type of ensembling done at inference time, and new scoring models should be explored to further improve the accuracy.

Another point made by Surdeanu and Manning (2010) was that re-scoring candidate dependencies through supervised meta-classifiers, which can combine evidence from multiple contexts, is also not an effective methodology for improving parsing accuracy. To prove this, Surdeanu and Manning (2010) implemented a L2-regularized logistic regression classifier using contexts as features. Although we did not test this claim for Hindi using the same classifier, but since **weighting does matter for Hindi**, we could not neglect the importance of scoring function altogether. Thus, we decided to explore a new kind of weighing mechanism as discussed in the next chapter.

While experimenting, we made an important but striking observation. During first technique of ensembling, we found that our best ensemble model was designed combining only 2 base parsers and as we increased the number of base parsers, the accuracy instead of increasing, which is generally observed, decreased drastically even way below baseline. This raised serious questions about the diversity of our base parsers and also the ensembling approach, we employed. But when we implemented the second word-by-word voting-based ensembling, we did not observe the same phenomenon. First, all the ensemble models performed above the baseline except five. Second, out of 5 top-most performing models, 3 were created by the combination of 4 base parsers. This provided us with different insights: (1) Perhaps re-parsing algorithms perform better when used with small number of base parsers but those base parsers must be the best among all (2) In word-by-word voting, there exits an upper limit to combine base parsers to improve the accuracy and just by increasing the number of base parsers in a brute-force way, the accuracy can not continue to increase. This can also be verified from the results reported in Surdeanu and Manning (2010) that, ensemble model combining 6 base parsers performed better than combining 7 in all the different voting strategies. (3) To further raise the performance of an ensemble model, there is a need to increase the diversity among the base parsers, even more than just using different variants of a single approach. Other successfully claimed parsers including MST and TurboParser as explored by Kukkadapu et al. (2012), must also be explored in future to increase diversity, but in an improved and informed fashion.

The most important take away from our ensembling experiments, which we will use to explore more in the area of dependency parsing for Hindi, is whether improving weighting can make any improvement in parsing accuracy or not. The pursuit of better weighting techniques will lead us to $\mathcal{PQE}$ (cf. Chapter 5) in next step of our journey.

Q2. Are (potentially expensive) re-parsing strategies justified for English? What percentage of trees are not well-formed if one switches to a light word-by-word voting scheme?

A2. Surdeanu and Manning (2010) asserted that linear-time re-parsing algorithms guarantee well-formed dependency trees without losing performance significantly. The suggestion to use re-parsing algorithms is driven by the fact that word-by-word voting strategy, although performs better than other approaches, but only in terms of hard-core numbers (LAS) and also create the problem of badly-formed trees. But such bold claim can not be made from our side due to two reasons: (1) linear-time re-parsing algorithms have never been explored for Hindi in ensembling before so we could not compare its performance with more complex-time algorithms such as Chu-Liu-Edmonds (Chu and Liu, 1965; Edmonds, 1967), which we have used in our case (2) Also we could not guarantee that badly-formed tress will not be generated by using linear-time algorithms due to lack of any such information. But it will be worth to explore re-parsing algorithm-based ensembling using linear-time algorithms in future. Also it may aid in selecting better re-parsing algorithm in terms of trade-off between time-complexity and accuracy. Moreover, there is a strong possibility that linear-time or any other algorithm can improve parsing accuracy via re-parsing algorithm ensembling as our first technique failed to make any impact.

Q3. How important is the integration of base parsers at learning time?

A3. Surdeanu and Manning (2010) undoubtedly proved that simple ensembling done at inference time performed significantly better than ensembling at learning time. Perhaps we can not agree more on this answer among all the other 4 answers they had provided. We have already discussed above that ensembling at learning time as explored by Kolachina (2012) for Hindi and other Indian languages could not perform. Our ensembling models, which integrated base parsers at run-time, did improve the parsing accuracy. Now a clear comparison between both the approaches exists, which also reinforces the bold claims made by Surdeanu and Manning (2010)

Q4. How do ensemble models compare against state-of-the-art supervised parsers?

A4. Though the simple, cheap and unweighted ensemble models of Surdeanu and Manning (2010) could not beat the then state-of-the-art dependency parsing system for English, their model, without using any higher-order features, stood second. While employing ensembling for Hindi dependency parsing, our model performed better than the state-of-the-art Hindi dependency parsing system (cf. Chapter 2). There has been seen an improvement of 0.47% in LAS, 0.36% in UAS and 0.43% in LS over the baseline system using $\textit{Ensemble}_4^{drel}$. Though the improvements are not as significant as it should be (results on development data), but they are important in the following ways: (1) Unlike Surdeanu and Manning (2010), ensembling did improve parsing accuracy from the baseline and that too using a weighted model. Thus, establishing ensembling to be an effective strategy in dependency parsing for Hindi and also emphasized the importance of weighting in ensembling. (2) Ensembling, as employed by us, opens a new chapter of dependency parsing

strategies which can be further improved for Hindi (like diversifying the base parsers, learning parameters' optimization in base parsers, effective weighting strategies, etc.) and can also be employed easily for other Indian languages. (3) We have also observed in Chapter 3 that as the training size increases, the impact of semantic information decreases in making significant advancements in statistical terms. Though a similar scenario in ensembling is highly probable, the relative improvement from baseline is more using ensembling than aiding dependency parsing by incorporating semantic information.

4.1.4 Limitations of Ensembling

Ensembling has been proved beneficial for dependency parsing in Hindi, in particular, word-by-word voting technique. But with all the positives and insights we have gained, there are some negatives and limitations also attached to it. Discussion over them is also necessary to completely justify our next task in the chain of **Advancements of Dependency Parsing**.

Overfitting: It has been clearly pointed out in Hall et al. (2007) that weights (cf. § 4.1.1.1) could not be tuned on held-out data (development data) for the dry-run, to find out the best ensemble system to be used later for testing. They had observed an average drop of 1% in case of Blended parser from the results obtained on dry-run as compared to test data. On the other hand using Single Malt parser, the test results over multiple languages were very close to the dry-run results. This suggests that Single Malt prevented the over-fitting of tuned parameters on development data which was not possible with Blended Parser. In our case, the drop observed from the dry-run results, made the ensembling technique modeled using re-parsing algorithm, to be seen as a failure by even decreasing instead of increasing the parsing accuracy from the baseline system [19]. This strongly suggests to devise a weighting strategy which does not need beforehand tuning of weights on development data so as to prevent the problem of over-fitting and must be dynamic in nature unlike static weights based on different contexts.

Number of Base Parsers: As we move down in Tables 4.2 and 4.3, the number of base parsers in the combination increases from 2 to 6. But the same does not co-relate with the accuracies of the corresponding ensemble systems. In case of re-parsing algorithm, the more the parsers, the less the accuracy. And, in case of word-by-word voting, though by increasing base parsers accuracy increases, only upto a certain point. This suggests that base parsers with very low accuracy ($Malt_P$ and $Malt_{2P}$) impact the best-performing models ($Malt_{AE}$ and $Malt_{AS}$) in a more negative sense than positive. The low-performing systems even decreases the overall accuracy and this is more severe in re-parsing algorithm technique. Thus before including more base parsers in an ensembling system to increase diversity, it is important to first evaluate whether any possibility exists to improve the parsing accuracy by combining

[19] When parameters were tuned on testing data itself and used in re-parsing algorithm ensembling to parse the same data, the LAS score increased from baseline by 0.08%

small number of best-performing base parsers. In addition, this will also save the time spent on training poor-performing base parsers.

4.2 Summary

To make advancement in dependency parsing for Hindi, we employed two different ensembling approaches at inference time: re-parsing algorithm and word-by-word voting, using different variants of Malt parser in terms of algorithms which are combined using six different weighting mechanisms. Though the former did not make any impact on the parsing accuracy, the latter improved LAS score by a margin of $\sim 0.5\%$ over baseline system. The improvement seems only marginal but we have obtained a number of important insights, while employing ensembling via re-parsing algorithm and word-by-word voting, to decide our future course of research.

4.3 Future work and Conclusion

Due to lack of adequate resources and time constraints, we could not explore the entire parameter space for Malt parser or experimented with other dependency parsers like MST parser and TurboParser. In future, we would like to experiment with all the individually best-performing dependency parsers for Hindi and also including more variants of Malt parser to enrich the diversity among base parsers, the foundational principle of ensembling. Though we have explored six different weighting mechanisms, a lot more remains to try. Hence, for next step of our exploratory journey, we are putting more focus on one of the weighting strategies in the form of $\mathcal{PQE}$, which is motivated by the fact that ensembling for Hindi notably depends on the type of weighting, in addition to different ensembling techniques.

Chapter 5

Advancements to Dependency Parsing: $\mathcal{PQE}$

In the last chapter, we have discussed about different ensembling approaches and showed the importance of weighting votes and the improvement in parsing accuracy due to ensembling. In this chapter we are continuing our journey in search of a better weighting mechanism which could cater some of the problems, if not all, like over-fitting, without lossing the points in LAS. $\mathcal{PQE}$ or *Parse Quality Estimation* is the result of our search.

5.1 $\mathcal{PQE}$

$\mathcal{PQE}$ or *Parse Quality Estimation*, as the name suggests, quantifies the quality of a parse structure. It is a kind of prediction task, where the output parse tree is also accompanied with a quality score (or confusion score). $\mathcal{PQE}$ estimates the extent of correctness or wrongness of a parse tree. Pursuit of increasing only LAS numbers over a testing set to evaluate the performance of a dependency parser is not the ultimate goal of parsing. The quality of an individual parse tree also plays a significant role in real-world applications like Machine Translation (Galley and Manning, 2009). Popel et al. (2011) also pointed out that an incorrect parse tree hurts the accuracy of an MT system. In addition, the search for the best parameters and models to develop a dependency parser, using development set as a tuning set for dry-run, makes the dependency parser bias towards the training data. Thus try to limit the parser not to perform for out-of-domain, which is a real-world situation. Moreover, testing data, most of the times is but a small part of large training data. Hence, evaluation on testing data may not provide a thorough picture of parsing accuracy. Thus, to better present the picture of a parse tree, there is a need to quantify the quality of the parse tree.

It seems that posterior probability of the parsed output to compute confusion score, can be one of the solutions in the age of statistical data-driven parsers. But some of the most popular algorithms like Support Vector Machines (SVM) (Cortes and Vapnik, 1995) and Margin Infused Relaxed Algorithm (MIRA) (Crammer and Singer, 2003) do not employ probabilistic modeling techniques. A relevant work in this direction is done by Mejer and Crammer (2012). They proposed methods to estimate the confidence or correctness of a resultant dependency parse structure in a graph-based scenario using

MST parser. Though it is a welcome step in this direction, not for us due to our constraints. In previous chapter we had modeled our six base parsers as the algorithmic variants of Malt parser to perform the ensembling experiments. So to compare our new results with previous ones, we have to find such a functionality in Malt parser. Fortunately, we found that such a functionality has been implemented by Jain and Agrawal (2013) for Malt parser. Jain and Agrawal (2013) proposed an approach to dynamically compute an entropy-based confusion score corresponding to a dependency label, which is assigned in run-time during parsing. Since the calculation of the score is dynamic, it differentiates the work with other conventional methods. We also found two hindrances in using their work directly to perform our ensembling experiments:

1. Jain and Agrawal (2013)'s system only provides confusion score corresponding to the arc/dependency labels marked to an edge/arc (analogous to LS score). Though it would have been better if such a functionality is also available for arcs/edges (like UAS) or jointly for both arcs and labels (similar to LAS), but since our best ensembling model (Ensemble$_4^{drel}$ (word-by-word voting)) also used weights corresponding to LS score of dependency labels, the confusion score is not all that bad for our experiments.

2. The second hindrance was unavailability of $\mathcal{PQE}$ functionality for all the Malt parser algorithms that have been used in our ensembling task in previous chapter. Jain and Agrawal (2013) has hinted that since the confusion score is computed at the oracle level (learning algorithm), which is independent of the parsing algorithm, it can be extended easily to other algorithms.

Using the same approach as mentioned in Jain and Agrawal (2013), we integrated the given $\mathcal{PQE}$ functionality corresponding to each of the dependency labels over all the algorithms, which we have to use in our ensembling experiments:

1. Nivre Arc Standard (Nivre, 2003; Nivre, 2004)

2. Covington Projective and Covington Non-Projective (Covington, 2001)

3. Planar and 2-Planer (Gómez-Rodríguez and Nivre, 2010)

5.1.1 Ensembling using $\mathcal{PQE}$

We have already mentioned (cf. § 4.1.3) that the search of $\mathcal{PQE}$-like score is motivated by the fact that since one of the weighting mechanisms (using context of *drel*) performed better in ensembling via word-by-word voting, there was a need to explore more in this direction. We are using confusion score ($1 - \mathcal{PQE}$) as the weight (w_p^c) to be assigned to an edge (e) of a parse tree resulted from a base parser p. The new experiments are performed using the same re-parsing algorithm and word-by-word voting as mentioned in § 4.1.1 and 4.1.2 respectively, and the same *set-up* of six algorithmic variants of Malt parser as base parsers. Since $\mathcal{PQE}$ is calculated corresponding to the dependency label that is

marked for an edge, it is similar to LS in the terms that the current functionality of $\mathcal{PQE}$ only estimates the quality corresponding to a dependency label and not the dependency itself[1].

Results: Table 5.1 shows the LAS, UAS and LS scores using both the ensembling techniques with confusion score ($1 - \mathcal{PQE}$) as the weight in score function ($s(e)$). Here again the word-by-word voting (*Ensemble*$_4^{PQE}$) performed better than re-parsing algorithm methodology (*Ensemble*$_2^{PQE}$), but now only with a marginal difference of 0.08% than earlier 0.47%. Confusion score captured by the oracle (learning algorithm of base parser) proved beneficial for re-parsing algorithm ensembling as unlike earlier, where the ensembling technique failed to make any impact on dependency parsing (there has been an improvement of 0.31% in LAS). This is a strong evidence that $\mathcal{PQE}$-like weighting strategies can aid dependency parsing using ensembling. When we relook the efficacy of the score in word-by-word voting ensembling technique, there has been a decrease in LAS by 0.08% (cf. Table 5.2). Though the decrement is marginal and even now the results are better than the previously used unweighted ensembling models, but losing some points using another weighting strategy raises question on the effectiveness of $\mathcal{PQE}$ (or confusion score) in ensembling.

	Re-parsing Algorithm			**Word-by-Word Voting**		
Best-Ensemble [2]	**LAS**	**UAS**	**LS**	**LAS**	**UAS**	**LS**
Ensemble$_2$	84.00	92.60	**86.82**	84.00	92.60	**86.82**
Ensemble$_3$	83.62	92.08	86.65	84.06	**92.65**	86.75
Ensemble$_4$	83.33	91.85	86.37	**84.08**	92.62	86.76
Ensemble$_5$	82.49	90.96	85.68	83.98	92.50	86.70
Ensemble$_6$	82.19	90.62	85.56	83.90	92.44	86.57

Table 5.1 LAS, UAS and LS for $\mathcal{PQE}$-based Ensemble systems taken x at a time

[1]This will be our motivation for expanding the confusion score for both *Attachment* (or dependency) and *Joint* prediction of Label and Attachment in Malt parser (cf. § 5.1.2)

[2]To know about which combinations (Ensemble Systems) using $\mathcal{PQE}$ as the weighting strategy, performed best, among all possible systems when any x number of parsers are ensembled, refer to Appendix D

Parser	LAS	UAS	LS
Malt$_{AE}$	83.69	92.43	86.58
Ensemble$_4^{drel}$ (Word-by-Word voting)	**84.16**	**92.73**	**87.00**
Ensemble$_4^{PQE}$ (Word-by-Word voting)	84.08	92.62	86.76

Table 5.2 Comparison of Best Systems without ensembling and with & without $\mathcal{PQE}$ ensembling

Discussion: Before jumping to the conclusion that $\mathcal{PQE}$ is not an effective weighting strategy in word-by-word ensembling only because it decreased accuracy from the previous best-performing ensemble model (*Ensemble*$_4^{drel}$ *(word-by-word)*), we need to consider the following points:

1. *Ensemble*$_4^{PQE}$ performed marginally poorer than *Ensemble*$_4^{drel}$ (0.08% less) in word-by-word voting technique of ensembling.

2. *Ensemble*$_2^{PQE}$ performed better than *Ensemble*$_2^{LAS}$ (0.31% better) in re-parsing algorithm ensembling.

3. The $\mathcal{PQE}$-weighting strategy generate scores in run-time unlike contextual weights, which are first tuned on held-out data set (development data) i.e., no problem of over-fitting (Hall et al., 2007) as mentioned in § 4.1.4.

4. $\mathcal{PQE}$ carries no biasness towards the genre or quality of development set unlike in previous scenario. Thus can perform better on out-of-domain data, which was a severe problem in ensembling as mentioned in Surdeanu and Manning (2010)

5. It provides an insight that only increasing number of base parsers to improve the parsing accuracy is not important, but the quality of the resultant parse tree is also simultaneously important which matters more in real-time applications.

6. Another point to note is that ensembling via word-by-word voting required 4 base parsers to achieve the best results. Though using $\mathcal{PQE}$ also needs the 4 base parsers to perform best but in ensembling via re-parsing algorithm technique, the increment in paring accuracy is observed combining only 2 base parsers without losing significant points in accuracy. Thus the trade-off between parsing accuracy and number of base parsers can not be neglected altogether.

7. Surdeanu and Manning (2010) also justified to pay a small penalty in terms of losing some points in parsing accuracy to guarantee a well-formed tree, which is guaranteed in re-parsing algorithm ensembling.

Another important observation is related to the effect of $\mathcal{PQE}$-based ensemble models on the distance of the dependencies. Table 5.3 shows the Recall (R), Precision (P) and change in F_1 score (Δ) of the best-performing baseline parser (*Malt*$_{AE}$) and the improvement over it by using both with and without $\mathcal{PQE}$-based ensemble systems. Though $\mathcal{PQE}$-based ensemble also improved dependencies of all those lengths as improved by using word-by-word voting ensembling, it outperformed all the previous ensemble models in dependencies of length $3-6$ ($(\Delta\ F_1)^{\mathcal{PQE}}_{3-6}$) and $7+$ ($(\Delta\ F_1)^{\mathcal{PQE}}_{7+}$). This clearly confirms that $\mathcal{PQE}$ score has utility in ensembling, particularly in correcting higher dependencies.

Furthermore, both with and without $\mathcal{PQE}$-based re-parsing algorithm ensemble systems have been negatively impacted by using low-performing algorithms. As the number of base parsers increases, the accuracy of ensemble model decreases (cf. Table 5.1 & 4.2). But this is not true in case of word-by-word voting ensembling technique. Even as the number of base parser increases, the accuracy increases upto a point (upto to combining 4 base parsers) and then starts decreasing. This suggests that only best-performing single base parsers in combination and that too with an effective weighting strategy like $\mathcal{PQE}$ score can outperform the baseline system during ensembling via re-parsing algorithm for Hindi. Also for word-by-word voting ensembling, weighting matters but number of parsers also matters equivalently.

One of the properties that need special mention for future work, in addition to our experiments on ensembling, is the use of *arc-directionality*. It has been implemented by Kolachina et al. (2010) and reported better results, particularly for Hindi. This feature is also utilized by Hall et al. (2007) and Surdeanu and Manning (2010) making it an important aspect altogether.

	root		**1**		**2**		**3-6**		**7+**	
Parser	**R**	**P**	**R**	**P**	**R**	**P**	**R**	**P**	**R**	**P**
Malt$_{AE}$	97.26	**96.26**	**97.24**	96.37	**90.66**	**90.76**	**89.84**	91.74	88.91	**89.36**
Ensemble$_4^{drel}$ (Word-by-Word)	98.69	**96.26**	97.03	**96.72**	90.59	90.52	89.51	92.46	91.51	88.48
Ensemble$_4^{PQE}$ (Word-by-Word)	**98.80**	94.11	96.92	96.70	90.34	90.27	89.14	**93.05**	**92.08**	88.54
$(\Delta\ F_1)^{drel}$ $(F_1^{drel} - F_1^{baseline})$	$+0.35$		$+0.04$		-0.08		$+0.09$		$+0.42$	
$(\Delta\ F_1)^{PQE}$ $(F_1^{PQE} - F_1^{baseline})$	$+0.18$		$+0.003$		-0.20		$+0.16$		$+0.57$	

Table 5.3 Recall (R), Precision (P) and Change in F_1 (Δ) Score of State-of-the-art Hindi Dependency Parser and best Ensemble systems with & without using $\mathcal{PQE}$ for dependencies of different length (root = dependents of root node, regardless of length).

5.1.2 Generating $\mathcal{PQE}$ for Attachment and Joint model

As already mentioned above, Malt parser (Nivre et al., 2007) has been explored by Jain and Agrawal (2013) to capture confusion encountered by the oracle in predicting arc labels. But it lacked capturing the complete picture of the parsed output as it does not capture the confusion encountered by the oracle while predicting arcs in a dependency parsed structure. Given the fact that each paradigm (transition or graph based) has its own strengths and weaknesses (Zhang and Clark, 2008), it motivated us to explore a transition-based parser for employing oracle confusion during either prediction of arc-formation or joint prediction of arc-formations and arc-labels for $\mathcal{PQE}$, which has not been explored to the best of our knowledge. [3]

5.1.2.1 Oracle Confusion for $\mathcal{PQE}$

Malt parser outputs a single best parse by greedily choosing parsing actions, advocated by an oracle trained on the training data. In a typed dependency framework, the parser performs two distinct kinds of actions to form a dependency tree: formation of directed arcs (or edges) between two words (or vertices) (henceforth *attachment*) and assignment of arc labels (or dependency labels) (henceforth *label*) to previously formed directed arcs. Malt parser provides a choice[4] to train separate oracles for the above two kinds of actions or a single oracle that jointly predicts both attachment and label for a pair of words.

In case of separate oracles for attachment and label prediction, the parser first starts with querying the attachment oracle for an appropriate parser action. In Nivre's algorithm (Nivre, 2003; Nivre, 2004), there are four possible parsing actions namely, *Shift* (S), *Reduce* (R), *Left Arc* (LA) and *Right Arc* (RA). Here LA and RA are arc-forming[5] parser actions, while S and R are non-arc-forming[6] parsing actions. The attachment oracle is queried until an arc-forming action is returned, which signifies an arc formation. Next, the label oracle is queried for an arc label, as per the given context, which is then associated with the recently formed arc. In joint oracle, the arc-forming actions are concatenated with possible labels. Now, if an arc-forming parser action is returned it also has a label concatenated with it (for eg. LA ~*nsubj*).

5.1.2.2 Calculating Entropy for Parser Actions

The confusion score of each attachment in a parse tree can be derived from the entropies of the parser actions that resulted in arc formation. A parser action can result either in arc-forming or non-arc-forming category and accordingly corresponding entropy is denoted as H_{arc} or $H_{non-arc}$. Based on the

[3] The task of extending the PQE functionality for attachment and joint model is a combined effort done by me (Naman Jain) and Sambhav Jain as a team and it has been published as Jain et al. (2015)

[4] `http://www.maltparser.org/userguide.html\#predstrate`

[5] Parsing action results in arc formation either in right-to-left or left-to-right direction in separate oracle. In case of a joint oracle, in addition to arc formation, also delivers the label corresponding to the resultant arc

[6] Parsing action does not result in forming an arc neither does it predicts a label, instead governs a state change in the arrangement of tokens or switches in data structures like stack of partially processed tokens and queue of remaining input tokens.

lines of Jain and Agrawal (2013), uncertainty or confusion score of each attachment is quantitatively determined by *entropy(H)* using the following formula:

$$H_{PA} = -\sum_{i=1}^{n} p_i \log(p_i) \tag{5.1}$$

where, n denotes the total number of possible candidates for parser action (henceforth *PA*), and p_i denotes the membership posterior probability corresponding to i^{th} candidate. The higher the entropy, the more uncertain the oracle is about the prediction.

5.1.2.3 Confusion Score for Tree Edges

Transforming uncertainty into confusion score for tree edges is not straightforward. The restricting factor being the presence of non-arc-forming parser actions i.e. S and R. Since these transitions are not decomposed over the tree edges, the oracle confusion associated with them can not be delineated to any specific edge. For example, at a given state during parsing, oracle will need to decide if it should perform an LA, RA, R or S action. This decision will not only influence the single edge immediately added by the LA or RA action, but also influence other future edges. It should also be noticed that, though S or R action will not add any edge, yet will have a complex effect on the set of edges that could or could not be added in future. Thus, the entropy of parser actions, resulting in non-arc-forming decisions should be considered together with the entropy of an arc-forming action while computing the uncertainty of an arc (edge) formation. However, it must be noted that label prediction in case of separate oracles does not have a dependency on any of the previous parser actions, and thus entropy of the labeling parser action is directly attributed as the confusion score for that label.

$$Confusion\ Score_{Attachment_i} = f(H_{arc_i}, H_{non-arc_{i-1}}, H_{non-arc_{i-2}}...) \tag{5.2}$$

For confusion score of attachments, we resort to following approaches to derive a function for combining the non-arc decisions with the arc-decisions:

Assuming Independence from Vicinity : Before moving to complex mechanisms of combining H_{arc} and $H_{non-arc}$ entropies, it is worth to consider the approximation that the confusion of an i^{th} attachment action only depends on H_{arc_i}.

$$Confusion\ Score_{Attachment_i} = H_{arc_i} \tag{5.3}$$

Decay Factor : Adhering to the fact that transition-based parser makes local and greedy decisions, we apply *principal of locality* and propose that parser actions in immediate vicinity have larger contribution in the confusion score as to a bit previous ones. Figure 5.1 describes the derivation of the algorithm to compute the confusion score of i^{th} attachment action in totality which takes a fraction of the remaining uncertainty at every step.

Figure 5.1 Algorithm to Compute the Confusion Score in totality

Total Confusion Score (uncertainty) due to all *PA*s can be **1.0** i.e., 100% wrong

The contribution from the *arc* forming *PA* $(arc_i) = \beta_0 \times H_{arc_i}$

where $\beta_0 = 1$

Remaining Uncertainty $= 1 - H_{arc_i}$

The contribution from previous *non-arc* forming *PA* ($(i-1)^{\text{th}}$ *PA*) (if present)

$= (1 - H_{arc_i}) \times \beta_1 \times H_{non-arc_{i-1}}$

where β_1 is an additional decay parameter.

Similarly, contribution from 2^{nd} previous *non-arc* forming *PA* or *(i-2)th PA*

$=$ New Remaining Uncertainty $\times \beta_2 \times H_{non-arc_{i-2}}$

$= (((1 - H_{arc_i}) - (1 - H_{arc_i}) \times \beta_1 \times H_{non-arc_{i-1}}) \times \beta_2 \times H_{non-arc_{i-2}}$

$= (1 - H_{arc_i})(1 - \beta_1.H_{non-arc_{i-1}}) \times \beta_2 \times H_{non-arc_{i-2}}$

from j^{th} previous *non-arc* forming *PA* or *(i-j)th* PA

Contribution$_j = (1 - H_{arc_i}).(1 - \beta_1.H_{non-arc_{i-1}})...(1 - \beta_{j-1}.H_{non-arc_{i-(j-1)}}) \times \beta_j \times H_{non-arc_{i-j}}$

Contribution$_j = (1 - H_{arc_i}) \times \left(\prod_{j=1}^{j-1} (1 - \beta_j H_{non-arc_{i-j}}) \right) \times \beta_j \times H_{non-arc_{i-j}}$, for $j \geq 1$

Consolidating the contribution as,

Confusion Score$_{Attachment_i} = \sum_{j=0}^{k}$ Contribution$_j$

where,

Contribution$_0 = (1 - H_{arc_i})$,

k is total number of previous *non-arc* forming *PA*s corresponding to i^{th} *arc* forming *PA*

Now, confusion score corresponding to an arc, depends on the entropy of a parser action contributing to the arc-formation and previous actions, but multiplied by a factor known as **Decay factor** β_i corresponding to i^{th} previous parser action. Since β_i is unknown in the calculation of confusion score, we establish different functional behaviors to calculate it:

1. **Linear Decay** : The parameter adhere to a linear decay and subsequently can be calculated as:

$$y(x) = mx + c$$
$$\beta_x = mx + c$$

Since β_0 = 1,

$$\beta_x = 1 + mx$$

Let m = -t

$$\beta_x = 1 - tx$$

where,

$$0 \leq t \leq 1, \text{to accommodate decaying characteristic}$$
$$x = 0, 1, 2, 3..k$$

2. **Polynomial Decay** : The parameter follows a polynomial decay as :

$$y(x) = mx^t + c$$
$$\beta_x = mx^t + c$$

Since β_0 = 1,

$$\beta_x = 1 + mx^t$$

Let m = -α

$$\beta_x = 1 - \alpha x^t$$

where,

$$0 \leq \alpha, t \leq 1, \text{to accommodate decaying characteristic}$$
$$x = 0, 1, 2, 3..k$$

3. **Exponential Decay** : The parameter follows an exponential decay as :

$$y(x) = \alpha^{mx} + c$$
$$\beta_x = \alpha^{mx} + c$$

Since β_0 = 1,

$$\beta_x = \alpha^{mx}$$

Let $\alpha^m = e^{-t}$

$$\beta_x = e^{-tx}$$

where,

$$0 \leq t \leq 1, \text{to accommodate decaying characteristic}$$
$$x = 0, 1, 2, 3..k$$

The parameters α and t are tuned on the development set. Confusion scores are calculated for each possible value of parameters iteratively and in each iteration, edges are sorted in decreasing order of their confusion scores, followed by book-keeping the number of incorrectly parsed edges in top *'K'* entries. *'K'* is total number of incorrectly parsed edges in the development set. The corresponding values which prioritize maximum errors are chosen as parameters. As illustrated in Figure 5.4, α is chosen to be 0.72 as it prioritize maximum errors (156/161).

5.1.2.4 Complex Association : Regression Analysis

We work with an intuitive assumption of a diminishing contribution from previous *PA*s. However, the actual relation may be more complex. So, we do away with the assumption of decaying contribution of previous *PA*s.

$$\text{Confusion Score}_{Attachment_i} = f(H_{arc_i}, H_{non-arc_{i-1}}, H_{non-arc_{i-2}}....)$$

In order to establish a function that best fits the scenario to furnish confusion scores, we utilize regression analysis. We used a development set to train a *SVM* regression model with entropy of an arc-forming action and all prior actions' entropies as features. If an arc has been parsed incorrectly we keep the corresponding confusion score value as 1.0, otherwise it is kept 0.0 in the data for training the regression model.

The above five configurations give us five distinct systems to compute *confusion scores* for attachments. We denote them respectively as $\mathcal{S}_{independent}$, $\mathcal{S}_{linear}$, $\mathcal{S}_{polynomial}$, $\mathcal{S}_{exponential}$ and $\mathcal{S}_{regression}$. All the systems give confusion scores between 0.0 and 1.0. $\mathcal{S}_{regression}$ utilizes a curve fitting approach, therefore, by default, may give values slightly less than 0.0 or marginally greater than 1.0. We incorporated an intermediate normalization step for adjusting the values between 0.0 to 1.0.

α=.01	α=.02	...	...	α=0.72	...	...	α=0.98	α=0.99
e#134*(0.998)	e#045*(0.984)	...	...	...	...	...	e#091*(0.983)	e#063 (0.991)
e#045*(0.987)	e#134*(0.978)	...	...	...	...	...	e#063 (0.977)	e#065*(0.982)
e#037 (0.973)	e#009*(0.969)	...	...	...	...	...	e#074 (0.968)	e#091*(0.978)
e#009*(0.964)	e#065*(0.962)	...	...	...	...	...	e#011 (0.961)	e#074 (0.966)
e#129 (0.961)	e#096 (0.961)	...	...	...	...	...	e#081 (0.957)	e#096 (0.952)
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
e#074 (0.791)	e#091*(0.728)	...	...	...	...	...	e#009*(0.814)	e#111 (0.828)
e#111 (0.785)	e#063 (0.712)	...	...	...	...	...	e#134*(0.804)	e#045*(0.821)
#Error = 42/161	46/161	.	.	156/161	.	.	32/161	32/161

Table 5.4 Choosing parameter α based on the ability to prioritize errors. 'e#N' (Confusion Score): Edge Index 'N' with Confusion Score in brackets; * denotes an actual incorrect edge. '#Error': Total number of incorrect edges correctly prioritized out of total incorrect edges actually present.

5.1.3 $\mathcal{PQE}$ Score capturing Oracle Confusion

The confusion scores computed from aforementioned approaches are indicators of parse quality. To estimate the quality of an arc (or confidence score) on the scale of 0 to 1, the confusion score can be subtracted from 1.

$$\mathcal{PQE}_{system} = 1.0 - \mathcal{S}_{system} \tag{5.4}$$

5.1.3.1 Correlation between $\mathcal{PQE}$ Scores and Actual Accuracy

To validate the authenticity of our proposed $\mathcal{PQE}$ scores, we plot them against the actual accuracy of the edges which depicts a positive correlation between the quantities as shown in Figure 5.2. The edges are grouped according to the value of their $\mathcal{PQE}$ scores. 10 bins are used dividing the $\mathcal{PQE}$ range of [0.0,1.0] into equal size interval of 0.1. Bin indexed i contains edges with $\mathcal{PQE}$ score value in the range $[\frac{i-1}{10}, \frac{i}{10}]$ where i =1..10. From each of these bins, fraction of correctly parsed edges, c_i, is computed. Mejer and Crammer (2010) acknowledged that the value of computed frequency (c_i) should be about center value of bin i i.e. $\frac{2i-1}{20}$, as it indicates that the proposed score is a good estimator of the

actual frequency of the correct edges. Thus, best performance is obtained when a line corresponding to a method is close to the line $y = x$ in Figure 5.2.

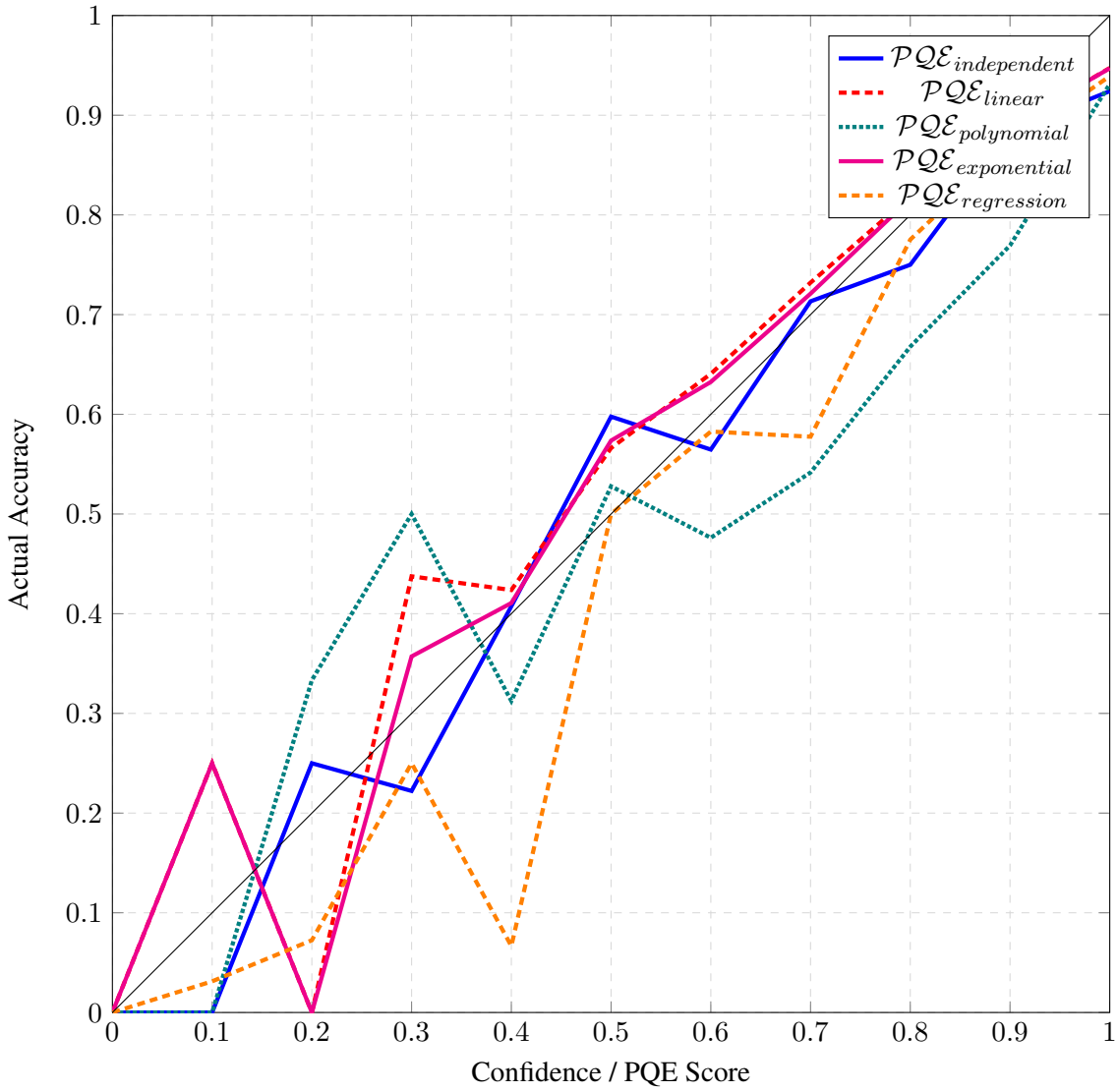


Figure 5.2 Predicted vs Actual Accuracy comparison between different PQE systems for English attachments

5.1.3.2 Comparison with Mejer and Crammer (2012)

We compared our approach with (Mejer and Crammer, 2012), for English attachments. MST parser, along with the confidence score, is trained for the CoNLL 2007 English data, so that a fair comparison

can be performed. We found that $\mathcal{PQE}_{exponential}$ closely matches (Mejer and Crammer, 2012) for scores ≥ 0.3, while for the rest of the range, a bit better.

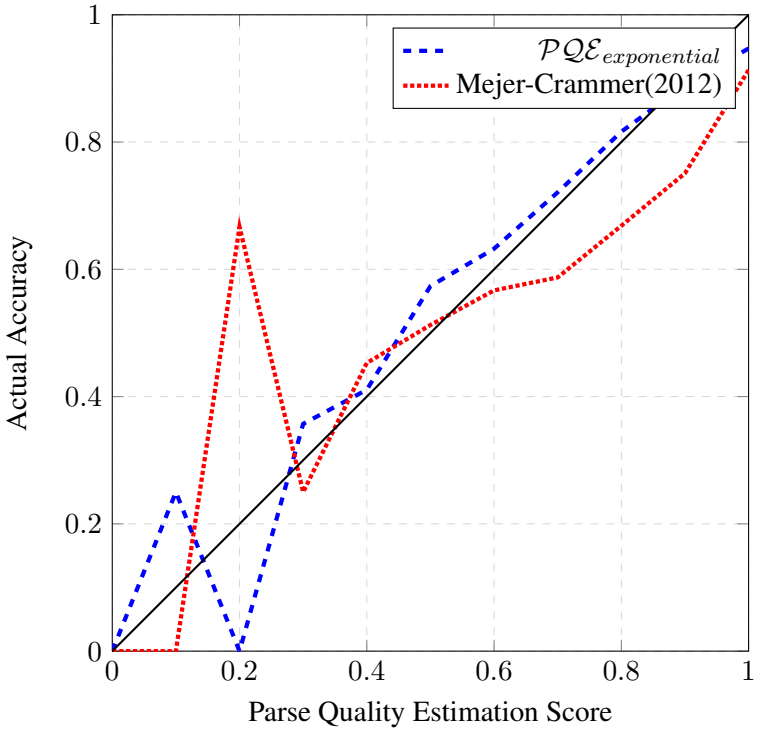


Figure 5.3 Predicted vs Actual Accuracy comparison between our $\mathcal{PQE}$ system and Mejer and Crammer (2012) for English attachments

5.1.4 Extendibility of $\mathcal{PQE}$ Score

The calculation of $\mathcal{PQE}$ score, in a dynamic manner, for all the possibilities like label, attachment and attachment+label, can be easily extended to all the algorithms and also to calculate the net $\mathcal{PQE}$ score of a parse tree for full parse quality. Though we have already extended the $\mathcal{PQE}$ functionality corresponding to labels for our ensembling experiments as described above, $\mathcal{PQE}$ score corresponding to attachment and joint prediction can also be extended to all the algorithms in a similar manner.

5.1.4.1 Extension to Full Parse Quality

The $\mathcal{PQE}$ score in the proposed approach is calculated separately for each arc/edge. Similarly, it can also be defined for a full parse. Calculation of $\mathcal{PQE}$ score for a full parse can be obtained by taking an average over the $\mathcal{PQE}$ scores for individual edges in the parse tree. Other measures like average of worst five arcs, score of worst arc itself, etc. can also be adopted as per application's demand. This

enables visualization for quality of the complete parse tree, which is apt in a scenario of large collection of parse trees, such as treebanks and also for applications like Active Learning (Tang et al., 2002; Hwa, 2004).

5.1.4.2 Extendibility to Algorithms

Malt parser supports three families of parsing algorithms:

1. **Nivre** : Nivre Arc Eager, Nivre Arc Standard (Nivre, 2003; Nivre, 2004), Planar, 2-Planer (Gómez-Rodríguez and Nivre, 2010)

2. **Covington** : Covington Projective and Covington Non-Projective (Covington, 2001)

3. **Stack** : Stack Projective, Stack Eager, Stack Lazy (Nivre et al., 2009; Nivre, 2009)

Since our method executes at the oracle level, it does not require any special treatment for algorithmic choice made while running MaltParser. Also, pseudo projective transformation (Nivre and Nilsson, 2005) for our approach is an extrinsic process, so non-projectivity does not perturb our methodology.

5.1.5 Implementation over Malt parser

We integrated our proposed approach with Malt parser without altering its functionality. The implementation of the second method in Wu et al. (2004), is incorporated in *MaltLibsvmModel.java* [7] which borrows the relevant code from Java implementation of *libsvm* package [8]. A stored model, now, can also render class membership probabilities along with the votes [9]. We did not alter the parsing methodology which uses votes to make the decision on parsing action. Entropy is calculated using *logarithm* with a base equal to the number of existing classes. The calculated entropy is emitted out along with the parser action.

The entropies are combined for tree edges by separate system developed in *python* and is outside Malt parser (in future will be integrated inside). This system does tuning of parameter given developed sets and also prepare a regression model for $\mathcal{PQE}_{regressionl}$ (*'C'* for the regression is however, tuned externally using *grid.py*) [10]. It finally furnishes the desired $\mathcal{PQE}$ score based on the user desired configuration for each node of a parsed tree produced by Malt parser.

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

[8] `http://www.csie.ntu.edu.tw/~cjlin/libsvm/`

[9] These class membership probabilities and votes will play a significant role in many of the applications associated with $\mathcal{PQE}$

[10] `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/#grid_parameter_search_for_regression`

5.2 Summary

In this chapter, we seek the aid of $\mathcal{PQE}$ score as an effective weighting strategy while employing ensembling to improve Hindi dependency parsing. The functionality of the score to be furnished for each label, has already been integrated into Malt parser (the platform used to build Hindi Dependency Parser) by Jain and Agrawal (2013). This score improved the re-parsing algorithm ensembling while losing some points in word-by-word voting technique. Since the score was only available to capture the confusion corresponding to a label, we expanded its scope to also cover attachment i.e., arc/dependency prediction and in a joint model of attachment and label prediction using five different functions.

5.3 Conclusion and Future Work

Currently, we have two different methodologies to dynamically calculate the $\mathcal{PQE}$ score for both attachment and label separately, and also in a joint manner. The first question that needs to be answered in future is whether combining the separate oracles for attachment and labels can be compared with the joint prediction and which one is better than other and more effective. Another work for future is to use confusion score corresponding to the attachment and joint prediction of attachment and label in both the techniques of ensembling. We have already seen the importance of $\mathcal{PQE}$ in ensembling, but it has much more potential in various other applications, which is the next station of our research journey.

Chapter 6

Implementation of Applications of Parse Quality Estimation

Parse Quality Estimation ($\mathcal{PQE}$) score that has been discussed in § 5.1, can be used in multiple tasks, besides predicting the correctness of a parse tree. In this chapter we present our efforts for implementing many such applications and also briefing on some other applications.

6.1 Automatic Parse Error Detection

Despite huge efforts in creating a treebank, it is nearly impossible to create a treebank which is free from all errors. Also, when a parsing system predicts the output, the errors are not uniform over the test file and some sentences can be parsed better than others. To identify such errors in an automated way, we use techniques like parse error detection. This can help a machine or a human expert to advance the correction process in the later stage.

Automatic error detection aims to efficiently determine and flag incorrectly predicted edges. The edges exhibiting high confusion scores are also highly probable to be incorrect, as the oracle is uncertain in its decision. Using this insight, an attachment or label is flagged as potential error if its confusion score is above a pre-calculated threshold (θ). In this task we have focused on identifying errors in two possible configurations i.e. attachment incorrectness and combined attachment & label incorrectness (either label or attachment is incorrect). [1]

6.1.1 Data and Experimental Setup

We conducted experiments on 18 languages[2], using data from CoNLL-X (Buchholz and Marsi, 2006), CoNLL 2007 (Nilsson et al., 2007) and MTPIL COLING 2012 (Sharma et al., 2012) shared tasks on dependency parsing. We employ Malt parser version-1.7[3] (Nivre et al., 2007). We carried out experiments on the systems proposed in Nivre et al. (2006), Hall et al. (2007) and Singla et al. (2012),

[1] The work has been published as Jain et al. (2015)

[2] Included all the languages in CoNLL-X and CoNLL 2007 shared task, except German and Czech due to unavailability of resources.

[3] `http://www.maltparser.org/download.html`

which are individually, the best performing Malt parser based systems, in their respective shared tasks. Best performing Malt parser based systems for all the languages use Arc-Eager mode of Nivre's algorithm[4] (Nivre, 2003; Nivre et al., 2004), except Chinese which uses Arc-Standard mode. All the results, reported here, are on the official test sets.

6.1.2 Identifying Optimal Threshold(θ)

Threshold (θ) is a crucial parameter in our experimental setup. An optimal θ is chosen by making use of the development set. Corresponding to each of the iteratively increasing candidate values for θ, from minimum to maximum, the incorrect edges are flagged and $Precision$, $Recall$ & $F\text{-}score$ (Manning et al., 2008) are calculated. The value asserting the maximum $F\text{-}score$, is chosen as threshold θ. For simplicity, we used balanced $F\text{-}score$, i.e. $F_1\text{-}score$. However, as per the application and available resources, a relevant F_β can be chosen to maximize the yield on the input effort.

$$F_\beta = (1 + \beta^2) \times \frac{precision \times recall}{(\beta^2 \times precision) + recall} \tag{6.1}$$

Figure 6.1 depicts $Precision$, $Recall$ and $F_1\text{-}score$ corresponding to each candidate value of θ for Hungarian development data. The maximum $F_1\text{-}score$ is attained at 0.36, which is thus taken as the θ for Hungarian. [5]

Since, CoNLL-X and CoNLL 2007 datasets did not provide development sets, we holdout 10% sentences of each training set as development data, using random sampling stratified on sentence length. The remaining training data is utilized to train a parser model, which parse the development set. This development set is again partitioned into two subsets $\mathcal{D}_1$ and $\mathcal{D}_2$, respectively utilized for parameter tuning (§ 5.1.2.3) (or training regression model (§ 5.1.2.4)) and threshold selection (§ 6.1.2). However, the final training is performed on the entire training data and evaluation on the test set.

6.1.3 System and Baseline

We constructed two baselines for each configuration and language in our experiments. The first, *Baseline-I*, adopts a naive methodology of randomly selecting and marking errors. The number of errors to be marked, is derived from the evaluation[6] on $\mathcal{D}_1$. The second, *Baseline-II*, assigns a confusion score to arc, equal to percentage of error of respective coarse POS-tag, as per the evaluation on $\mathcal{D}_1$. Five systems to predict the confusion score of an arc are created as discussed in § 5.1.2.3 and 5.1.2.4 for 18 languages.

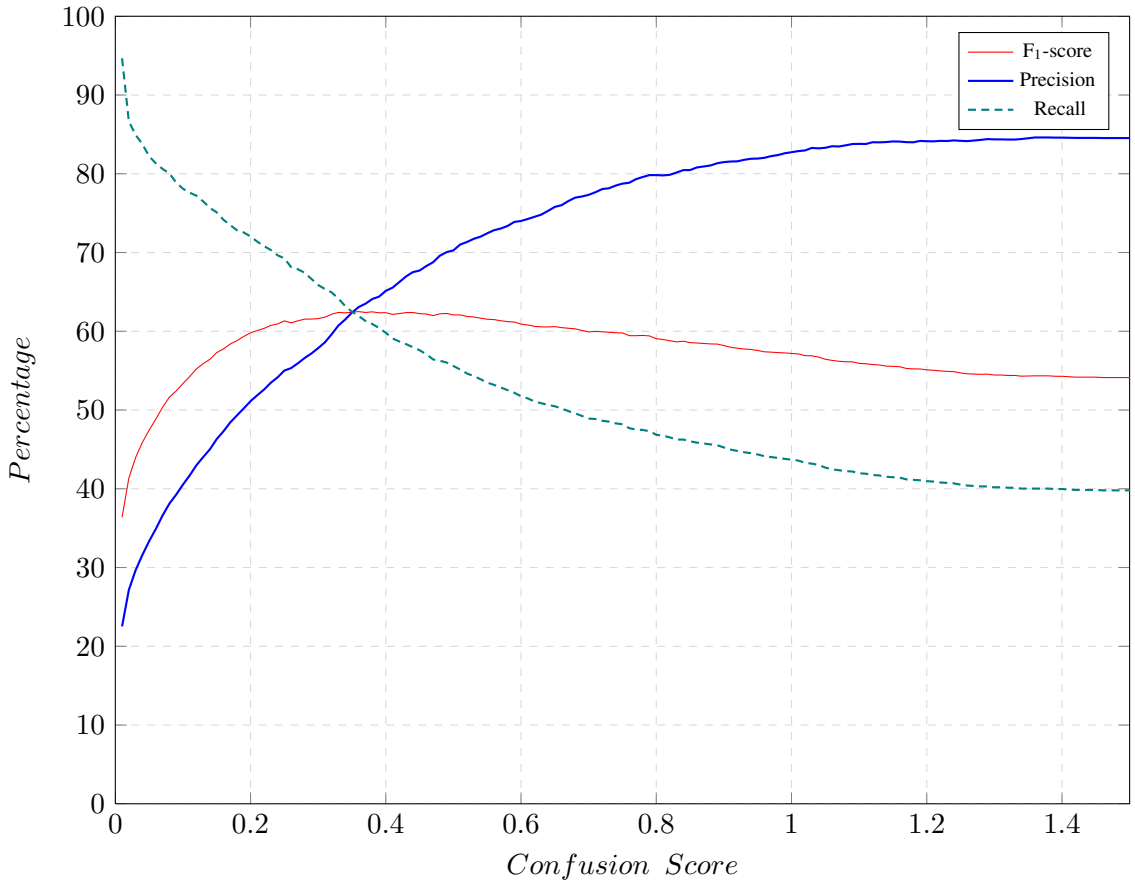


Figure 6.1 Precision, Recall and F_1-score for various values of confusion score on 'Hungarian' development set.

		Average						English					
	Measure	**F**	**P**	**R**	**EDI-1**	**EDI-5**	**EDI-10**	**F**	**P**	**R**	**EDI-1**	**EDI-5**	**EDI-10**
Attachment	**Baseline-I**	14.79	12.28	19.06	0.77	3.84	7.68	14.15	12.39	16.5	0.89	4.45	8.9
	Baseline-II	33.58	23.75	62.74	2.49	10.82	20.99	32.55	23.02	55.52	3.08	13.49	22.55
	Independent	39.69	33.63	52.64	2.15	10.40	24.56	46.68	47.02	46.34	6.16	24.53	38.88
	Linear	47.01	40.48	59.39	4.67	21.08	36.60	47.73	49.54	46.05	6.61	26.41	40.2
	Polynomial	45.82	38.31	61.32	4.66	19.77	34.69	47.65	48.73	46.63	6.61	**26.56**	**40.34**
	Exponential	**47.62**	40.54	60.33	**4.76**	**21.12**	**37.76**	**48.73**	45.67	52.22	6.61	**26.56**	**40.34**
	Regression	32.36	31.78	56.51	4.07	16.83	29.44	46.41	46.47	46.34	**7.17**	24.53	39.74
Attachment+Label	**Baseline-I**	21.47	18.00	28.80	0.78	3.92	7.84	14.6	12.8	16.99	0.83	4.14	8.28
	Baseline-II	34.06	21.38	99.96	1.11	5.51	10.74	26.53	15.29	100	2.9	14.18	23.49
	Independent	54.98	49.22	64.03	**3.97**	17.56	32.12	51.42	60.28	44.84	6.03	**27.19**	**42.35**
	Linear	55.99	49.39	65.49	3.88	17.76	32.67	**52.78**	52.24	53.33	6.03	26.34	40.94
	Polynomial	54.16	47.01	65.80	3.83	17.37	31.51	50.95	49.45	52.55	6.03	25.11	39.21
	Exponential	**56.20**	49.44	66.00	3.95	**17.79**	**32.80**	52.49	50.06	55.16	6.03	26.68	41.22
	Regression	42.48	41.19	62.66	3.81	16.18	28.96	36.02	44.36	30.33	**6.27**	20.52	29.93

Table 6.1 Average results over 18 languages and results for English for automatic error detection task.

EDI $x\%$ edges= Error detected on inspecting top $x\%$ of total edges.

6.1.4 Results and Discussion

Table 6.1 exhibits the results obtained for automatic error detection. We present the F_1-$score$, $Precision$ and $Recall$ obtained over 18 languages in the task, with detailed results for **English**. The detailed results for other languages are presented in Appendix A.

EDI: To efficiently capture the efficacy of our approach, another metric EDI (Error Detected on Inspecting top $x\%$ of total edges) is presented which corresponds to the percentage of errors detected by inspecting 1%, 5% and 10% of total edges. The metric portrays a more precise view of the effort required to correctly identify parsing errors (often through manual validation).

Our experiments indicate that confusion score has a dependency on previous actions of parser. The $\mathcal{PQE}_{independent}$ (§ 5.1.2.3) assumption is not found to perform well as comparison to other strategies of combining entropies. $\mathcal{PQE}_{exponential}$ is found to perform best, but marginally better than $\mathcal{PQE}_{linear}$ and $\mathcal{PQE}_{polynomial}$, for attachment error prediction. For joint prediction of attachments and labels, $\mathcal{PQE}_{independent}$, $\mathcal{PQE}_{linear}$, $\mathcal{PQE}_{polynomial}$ and $\mathcal{PQE}_{exponential}$ show comparable results. The regression based system $\mathcal{PQE}_{regression}$, however is not found to match up with the accuracies of these systems except the baselines. Best results are obtained for Portuguese while for **Hindi**, results are not very different from *Baseline-II*.

A comparison with (Mejer and Crammer, 2012) over English, indicates, our results ($\mathcal{PQE}_{polynomial}$ for Attachment) at par with them; 6.6% vs 7.17% (in ours) for 1% edge inspection, 27% vs 26.56% (in ours) for 5% and 46% vs 40.34% (in ours) for 10% edge inspection. This is only a tentative comparison since they reported results on Penn Treebank (55K words) while we use English data from CoNLL 2007 shared task (5K).

6.2 Effective Parsing for Human-Aided NLP systems

Despite extensive advancements in parsing research, it is observed that parsers perform clumsily when incorporated in NLP applications (Kolachina and Kolachina, 2012). The approaches addressing the shortcomings in the past have adopted building further high quality parsers with domain adaptations (Blitzer et al., 2006; McClosky et al., 2006a). However, it is practically impossible to account for all the domains and build an ideal universal parser. This has been a major reason for exploring Human-Aided NLP systems such as Human-Aided Machine Translation (HAMT) (Rao et al., 1998), which aims at providing quality output with guided minimal human intervention for crucial decisions. [7]

The practical impact of parsing errors, at application's end, may be more severe as depicted by the accuracy of a parser. Popel (2011), in the context of Machine Translation (MT), pointed out that an acutely incorrect parse can degrade the quality of output.

[4] Nivre's Algorithm has two different modes namely, Arc-Eager and Arc-Standard

[5] We are performing this experiment for 18 languages and demonstrating calculation of threshold for Hungarian. For other languages, similar process is used to obtain optimal threshold

[6] `http://nextens.uvt.nl/depparse-wiki/SoftwarePage/\#eval07.pl`

[7] The work has been published as Jain et al. (2013a)

We explore human assisted automated parsing, with human intervention limited to only those cases which are difficult for the statistical model (oracle) to disambiguate. Our focus has been to guide and minimize human intervention in terms of effort and time. The scheme involves running a data driven dependency parser and later involving a human validation step for the probably incorrect decisions which are also identified and flagged by our system.

Minimizing the human intervention calls for automatic identification of ambiguous cases as we have discussed in § 6.1. We have employed $\mathcal{PQE}$ score to capture uncertainty exerted by the parser oracle and later flag the highly uncertain predictions. To further assist human decision, we provide *k* probable alternatives in the order of their likelihood. In all, the approach comprises of the following two steps:-

- Identification of probable incorrect predictions.
- Selection of *k*-best alternates.

6.2.1 Background and Motivation

Earlier attempts on Hindi dependency parsing (Ambati et al., 2010a), demonstrate that UAS[8] is greater than LS[9] by $\sim 6\%$, which is reconfirmed by our baseline parser (later described in § 6.2.4), where UAS is 6.22% more than LS. The UAS in our baseline is well above 90% (92.44%) while the LS is still 86.21%. This drives us to focus on improving LS, to boost the overall accuracy(LAS[10]) of the parser.

Dependency annotation scheme, followed in Hindi Dependency Treebank (Bhatt et al., 2009), consists a tag-set of ~ 95 dependency labels [11] which is comparatively larger than the tag-set for other languages[12], like Arabic(~ 10), English(~ 55), German(~ 45), etc. This apparently is a major reason behind the observed gap between LS and UAS for Hindi parsing. One of the frequent labeling errors that the parser makes, is observed to be between closely related dependency tags, for eg. $k7$ (abstract location) and $k7p$ (physical location) are often interchangeably marked (Singla et al., 2012). We have reasons to believe that such a decision is comparatively tougher for an automatic parser to disambiguate than a human validator.

In the past, annotation process has been benefited from techniques like Active Learning (Osborne and Baldridge, 2004), where unannotated instances exhibiting high confusion can be prioritized for manual annotation. However, in Active Learning, the annotators or validators generally have no information about the potentially wrong sub-parts of a parse and thus full parse needs to be validated. Even if the the annotators are guided to smaller components (as in Sassano and Kurohashi (2010)), the potentially correct alternates are not endowed. In our approach, the validator is informed about the edges which are likely to be incorrect and, to further assist the correction, k-best potential label replacements are also

[8]UAS = Unlabeled Attachment Score

[9]LS = Label Accuracy Score

[10]LAS = Labeled Attachment Score

[11]To know more about the dependency tag-set refer to Bharati et al. (2009c)

[12]As observed in the CoNLL-X and CoNLL 2007 data sets for the shared tasks on dependency parsing.

furnished. So, effectively just partial corrections are required and only in worst cases (when a correction triggers correction for other nodes also), a full sentence needs to be analyzed. Though the efforts saved in our process are tough to be quantified, the following example provides a fair idea of efficacy of our proposition. In Figure 6.2, second parse has information of the probable incorrect label and also has 2 options to correct the incorrect label to guide a human validator.

(1) ... जेलों में समस्या हल करने ...
... prisons in problems solve do ...
... solve problems in prisons ...

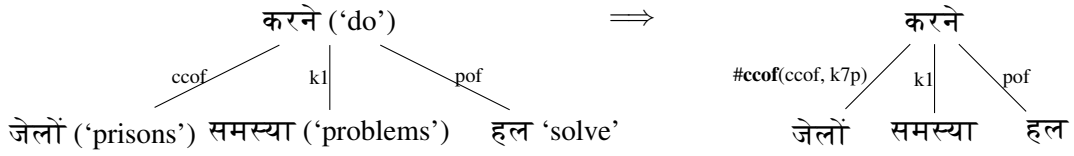


Figure 6.2 Example showing output from conventional parser vs output from our approach. Arc-label with '#' represents incorrect arc label (confusion score $> \theta$) along with 2-best probable arc labels.

6.2.2 Methodology

As discussed in Chapter 5, $\mathcal{PQE}$ score captures the confusion exerted by the parser's oracle while predicting a parser action and propagating it to the arc-label of the dependency tree. The quantification of confusion is done by calculating entropy with the class membership probabilities of the parser actions.

We obtained the confusion score for each arc-label in our data. Next, we obtained a threshold ($\theta = 0.137$) for which the maximum F_1-$score$ is observed for incorrect label identification on the development set (Figure 6.3). In Figure 6.2, the edge with the label '$ccof$' has been flagged (#) because the confusion score is greater than θ, which signifies that it is probably incorrect. The proposition is indeed correct as the correct label is '$k7p$' instead of '$ccof$'.

The additional details about the correctness of an arc-label, can duly indicate the cases where the probability of the arc-label to be incorrect is high. In our efforts to minimize the human intervention, we propose to subject the reviewer only to the cases where the confusion score is above θ. At this stage, the reviewer will be required to judge if the flagged label is indeed incorrect and if it is, then choose the corresponding correct label among all the remaining labels.

To further assist human decision, we provide *k* probable alternatives in the order of their likelihood as proposed by the oracle. The reason behind this hypothesis is that it is likely that the correct label exists among the top label candidates. This, potentially, can give quick alternates to the reviewer for choosing the correct label and thereby, speedup the review process.

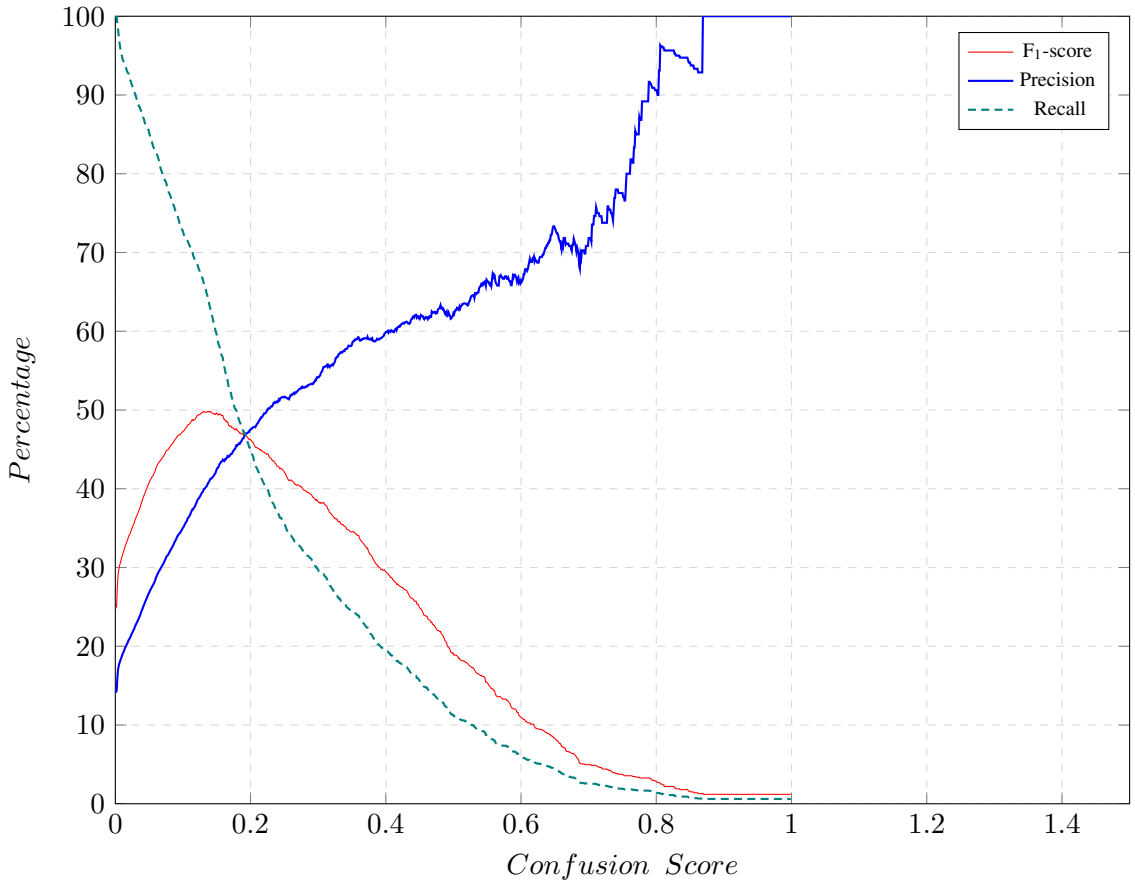


Figure 6.3 Precision, Recall and F_1-score for various values of confusion score on 'Hindi' development set.

6.2.3 *k*-Best Dependency Labels for the Flagged Arc-Labels

The likelihood of the arc-labels is obtained and ranked using the following three strategies:-

- $Voting$: The list of predicted labels, using voting mechanism, is sorted in decreasing order of the number of votes, obtained during classification. The label with maximum number of votes is emitted as the resultant dependency label in the output parse. Broadly, this can be viewed as predicting 1-best label using *voting* strategy which can easily be extended to predict *k*-best labels.

- $Probability$: The calculation of confusion scores demand class membership probabilities for arc-labels (refer § 6.4.1). The posterior probabilities for the candidate labels can also be alternatively used, to emit out the resultant dependency label. Similar to voting scheme, the labels are sorted in decreasing order of their probabilities. The sorted list of predicated labels may differ from that of *voting* mechanism, which motivated us to consider *probability* for choosing the *k*-best dependency labels.

- $Voting + Probability$: A tie can occur between two or more labels in the list of *k*-best candidate labels if their votes/posterior probabilities are same. However, the phenomenon is unlikely in case of probabilities due to the real valued nature calculated up-to 10 decimal places. On the other hand *votes* are integer-values ($\{0, ..., {}^{n}\mathrm{C}_2\}$, where *n* is number of labels) and are much more susceptible to ties. The tie in voting can be resolved using complement information from probabilities (and vice-versa).

6.2.4 Experiments

As mentioned earlier in Chapter 2, our focus is on correctly establishing dependency relations between the chunk[13] heads, which we refer as *inter-chunk* parsing. The relations between the tokens of a chunk (*intra-chunk* dependencies) are not considered for experimentation. The decision is driven by the fact that the *intra-chunk* dependencies can easily be predicted automatically using a finite set of rules (Kosaraju et al., 2012). Moreover, we also observed the high learning ability of *intra-chunk* relations by performing a pilot experiment. We found the accuracies of *intra-chunk* dependencies to be more than 99.00% for both LAS and UAS.

Each experiment assumes the availability of a human expert for validation of the machine parsed data, who, when queried for a potential incorrect edge label, responds with the correct edge label. The experiments aim to measure the assistance provided to human expert by our approach. We varied the list of *k*-best labels from $k = 1$ to $k = 5$.

We setup a baseline parser on the lines of Singla et al. (2012) with minor modifications in the parser *feature model*. We employ Malt parser version-1.7 and Nivre's Arc Eager algorithm for all our experiments reported in this work. All the results reported for overall parsing accuracy are evaluated using

[13] A chunk is a set of adjacent words which are in dependency relation with each other, and are connected to the rest of the words by a single incoming arc.

eval07.pl[14]. We use MTPIL (Sharma et al., 2012) dependency parsing shared task data. Among the features available in the FEATS column of the CoNLL format data, we only consider *Tense, Aspect, Modality (tam)*, *postpositions(vib)* and *chunkId*, while training the baseline parser. Other columns like POS, LEMMA, etc. are used as such.

In case of typed-dependency parsing, the accuracy can be LAS, UAS or LS. However, in our case, since we are focusing on the correct prediction of arc-labels the results are in terms of LS. In terms of strategies mentioned in § 6.2.3, the baseline system is generated using *Voting* strategy with $k = 1$. The LS is 86.21% as shown in Table 6.2.

6.2.5 Evaluation and Discussion

The evaluation of an interactive parse error correction is a complicated task due to intricate cognitive, physical and conditional factors associated with a human annotator. Since a human expert may not match the consistency of a machine, we have to resort to few compelling assumptions which would give us an idea of the approximate benefit or an upper bound of benefit from our proposed approach. We have assumed a perfect human oracle who always identifies incorrect label and picks the correct label from the available *k*-best list, if correct label is present in the list. The simulation of the perfect human oracle is done using the gold annotated data. It is also assumed that the decision of the correct label can be taken with the information of local context and the whole sentence is not reviewed (which is not always true in case of a human annotator). This gives the upper bound of the accuracies that can be reached with our approach. The validation of the results obtained by automatic evaluation is done by performing a separate human evaluation for 100 nodes with the highest confusion score.

In our dataset, we found out $\sim 23\%$ $(4,902$ edges) of total $(21,165)$ edges having confusion score above θ and thus marked as potentially incorrect arc-labels. Table 6.2 exhibits LS, improved by perfect human oracle, for *k*-best experiments, where, $k = 1$ to 5 on $\sim 23\%$ potentially incorrect identified arc-labels.

k	$\mathbf{LS_{Voting}}$ **(%)**	$\mathbf{LS_{Probability}}$ **(%)**	$\mathbf{LS_{Voting+Probability}}$ **(%)**
1	86.21	**86.35**	86.28
2	90.86	**90.96**	90.91
3	92.13	**92.24**	92.18
4	92.72	**92.86**	92.74
5	92.97	**93.16**	93.04

Table 6.2 *k*-Best improved LS on inspecting $\sim 23\%$ $(> \theta)$ edges.

[14] `http://nextens.uvt.nl/depparse-wiki/SoftwarePage/\#eval07.pl`

Table 6.2 also depicts that as the value of *k* increases, the label accuracy also increases. The best results are obtained with *Probability* scheme. There is a substantial increment in LS moving from 1-best to 2-best in all the schemes. The amount of gain, however, decreases with increase in *k*.

Ideally, to achieve maximum possible LS, all the edges should be reviewed. Table 6.3 confirms that if all the edges are reviewed, an LS of 93.18% to 96.57% is achievable for *k*, ranging over 2 to 5. But practically this would be too costly in terms of both time and effort. In order to economize, we decided to only review the cases which are probable enough to be incorrect. Confusion scores give a prioritized list of edges, which dictates the cases that should be dispatched first for review. To relate the review cost against LS gain, we present a metric AGI_x defined as:-

AGI_x :*"Accuracy Gain on Inspecting top $x\%$ of edges"* corresponds to the ratio of accuracy gain from baseline by inspecting top $x\%$ of total number of edges, when sorted in decreasing order of their *confusion score*. The metric takes into account the human effort that goes into the validation or revision, and thus gives a better overview of *ROI* (Return on Investment).

$$AGI_x = \frac{\text{Accuracy after validating top } x\% \text{ edges} - \text{Baseline accuracy}}{x}$$

From Table 6.2 and Table 6.3, we observed for $k = 2$ and *probability* scheme, that the improved LSs are 90.96% and 93.18% on inspecting 23% and 100% edges respectively. Although the latter is greater than the former by $\sim 2\%$, but this additional increment requires an extra inspection of $\sim 77\%$ additional edges, which is economically inviable. The fact is better captured in Table 6.4, where AGI_{23} subdues AGI_{100} for different values of *k* using different 'schemes'.

Further. to incorporate the fact that *'larger the candidate list more will be the human efforts required to pick the correct label'*, we also present the results of **AGI_x/k**, which can govern the choice of *k*, best suited in practice. While taking this into account, we assume that the human efforts are inversely proportional to *k*. Results for **AGI_{23}/k** on improved LS, over all the experiments are reported in Table 6.5.

k	**$\text{LS}_{\text{Voting}}$ (%)**	**$\text{LS}_{\text{Probability}}$ (%)**	**$\text{LS}_{\text{Voting+Probability}}$ (%)**
1	86.21	**86.35**	86.28
2	93.19	93.18	**93.26**
3	95.10	95.11	**95.14**
4	95.94	**96.06**	95.97
5	96.41	**96.57**	96.49

Table 6.3 *k*-Best improved LS on inspecting 100% edges.

As shown in Table 6.4, AGI_{23} increases with increase in the value of *k*, but it is practically inefficient to keep larger values of *k*. Optimal choice of *k* is observed to be 2 from the metric AGI_x/k (as shown in Table 6.5), where the maximum value for $\mathbf{AGI_{23}/k}$ is ~ 0.10 for all the 'schemes', which corresponds $k = 2$.

	Voting		**Probability**		**Voting+Probability**	
k	$\mathbf{AGI_{23}}$	$\mathbf{AGI_{100}}$	$\mathbf{AGI_{23}}$	$\mathbf{AGI_{100}}$	$\mathbf{AGI_{23}}$	$\mathbf{AGI_{100}}$
1	0.0000	0.0000	**0.0060**	0.0014	**0.0030**	0.0007
2	**0.2008**	0.0698	**0.2051**	0.0697	**0.2029**	0.0705
3	**0.2556**	0.0889	**0.2604**	0.0890	**0.2578**	0.0893
4	**0.2811**	0.0973	**0.2871**	0.0985	**0.2820**	0.0976
5	**0.2919**	0.1020	**0.3001**	0.1036	**0.2949**	0.1028

Table 6.4 $\boldsymbol{AGI_{23}}$ and $\boldsymbol{AGI_{100}}$ for $k = 1$ to 5

	$\mathbf{AGI_{23}/k}$	$\mathbf{AGI_{23}/k}$	$\mathbf{AGI_{23}/k}$
k	(Voting)	(Probability)	(Voting+Probability)
1	0.0000	0.0060	0.0030
2	**0.1004**	**0.1025**	**0.1015**
3	0.0852	0.0868	0.0859
4	0.0703	0.0718	0.0705
5	0.0584	0.0600	0.0590

Table 6.5 $\boldsymbol{AGI_{23}/k}$ for $k = 1$ to 5

From the above analysis, we can establish that with 2 probable alternatives, a perfect human oracle can increase the LS by 4.61%, inspecting top $\sim 23\%$ of the total number of edges. The corresponding increase in LAS is 4.14% (earlier 83.39% to now 87.53%).

The validation of the observation is done by a human expert who confirmed of the assistance from the above methodology over the default procedure. He was provided with 2-best alternatives for the top 100 edges that are obtained using *probability* scheme. The LS gain on his evaluation is approximately 10% which matches the expected gain.

6.3 Minimizing Validation Effort for Treebank Expansion

In § 6.2, it has been shown that how, with minimal human intervention, higher accuracies could be achieved on any kind of data. Similarly, it has also been proved that with only 2-best probable alternatives, a human expert can increase the labeled attachment score (LAS) by 4.14%, inspecting only $\sim 23\%$ of total edges. But this was, when we have to use human interference for improving the parsing accuracy. The same approach can also be applied, with some modification, for minimizing validation effort for treebank expansion from a seed data. Manual annotation is an expensive task as is evident from the fact that a treebank project usually takes 6 to 7 years to create a sizeable amount of parsed trees. For instance, $400K$-sized Hindi Dependency Treebank took $\sim$ 7 long years and $200K$-sized Urdu Dependency Treebank took $\sim$ 4 years. Further, in terms of human resources, it also requires dedicated human effort and competent skills. Thus, our approach can be most useful for low-resource languages.

Working on the same lines and using same methodology as mentioned in § 6.2.3 of providing the manual annotators with k-best alternatives, we are trying to demonstrate how much effort can be saved in terms of both time and resource. Since our goal is to expand treebanks and not to improve the parsing accuracy, we have to provide k-best alternatives for all the 100% edges[15] to expand the treebank with minimal effort. As reported in Table 6.3, using 5-best alternatives, one could improve the LS upto 96.57% on inspecting all 100% edges. It means a treebank accuracy can be attained as high as $\sim 96\%$, which is itself an achievement, considering the fact that between any two annotators, there hardly exists an inter-annotator agreement of 95% over a large size of data.

In our experiment, we wish to compute the assistance provided to a human validator by our approach. However, computation of an interactive parse correction system, is a complicated task due to intricate cognitive, physical and conditional factors associated with a human annotator. The extent of benefit will differ from annotator to annotator and thus quantifying the gain is a challenging task. We perform two kinds of experiments involving evaluation to illustrate the effectiveness of our approach.

The first experiment is an automatic evaluation task, which assumes a perfect reviewer who always identifies incorrect label and picks the correct label from the available *k*-best list, if correct label is present in the list. Though this would be an ideal scenario, it gives an upper bound of the accuracies that can be attained with our approach. The simulation of the perfect reviewer is done using the gold annotated data. It is also assumed that the decision of the correct label can be taken with the information of local context only and the whole sentence is not reviewed (which is not always true in case of a human annotator). Since the automatic evaluation has already been done once in § 6.2.5, we are not repeating the same experiment and only focusing on the second experiment.

[15]Due to time and resource constraints, for our experimentation, we would only provide pilot set of small sentences and that too for most error-prone dependency relations.

The second experiment is a human evaluation task, where two annotators are requested to review the arc labels, *with* and *without* additional information from our system. The gain is measured in terms of time saved with the additional information.[16]

6.3.1 Human Evaluation

	Annotator with info.		Time Taken		Expected Time		
Position	**of the k-best labels**	***k***	**A**	**B**	**A**	**B**	**Gain**
Top	A	2	10	15	19.69	-	**9.69**
Top	B	2	15	11	-	11.43	**0.43**
Top	A	3	15	13	17.06	-	**2.06**
Top	B	3	10	10	-	7.62	-2.38
Top	A	5	10	9	11.81	-	**1.81**
Top	B	5	10	8	-	7.62	-0.38
Middle	A	2	7	7	9.19	-	**2.19**
Middle	B	2	6	9	-	4.57	-4.43
Bottom	A	2	9	11	14.44	-	**5.44**
Bottom	B	2	7	7	-	5.33	-1.67

Table 6.6 Human evaluation with time gain due to the k-best labels. All time values are in minutes.

The human evaluation is performed with the assistance of two human experts[17], who have prior experience in the dependency annotation for Hindi. First, the relative proficiency of the annotators is estimated by taking the ratio of the time taken by each in annotating the labels for a small set of 20 sentences. The time taken by $Annotator\text{-}A$ and $Annotator\text{-}B$ is 105 and 80 minutes respectively and thus the relative annotation proficiency of annotators A to B is $105/80$. The disagreement between the two annotators is observed to be less than 3% and hence neglected for the relative proficiency calculation.

Next, both the annotators are provided with six sets of 10 parse trees each and asked to review a specific label (probably incorrect label) in each parse. The six sets are formed by picking parse trees corresponding to the nodes having maximum confusion scores. Evey parse tree, provided for review, already has the attachments and the labels annotated except for the label under review. Each set has same sentences but one version of the set also has the k-best labels corresponding to the label under

[16]The work has been published as Jain et al. (2013b)

[17]We would like to thank Ms. Preeti Pradhan and Ms. Nandini Upasani for helping us in the human evaluation process.

review, which is alternatively provided to each reviewer. During the review, time is noted down for each annotator in annotating the sentences in each set. The first six rows of Table 6.6 illustrate the time taken by each reviewer for these six sets. The value of k is taken to be 2,3 and 5 with two sets corresponding to each k. Among these two sets once the k-best labels are provided to $Annotator$-A and once to $Annotator$-B, as shown in the second column of Table 6.6.

The impact of the additional information provided in form of k-best labels is quantified in terms of the time gain. Time gain is the difference between the expected time, an annotator is supposed to take, and the time actually taken with the assistance of information provided as k-best labels. The expected time is estimated using the relative proficiency calculated earlier. For example, in the first set (first row of Table 6.6) $Annotator$-A is provided with the k-best label version of the set while $Annotator$-B has no such information in her version of the set. Time taken by $Annotator$-B in the review process is recorded to be 15 minutes. The expected time for $Annotator$-A can be computed with the relative proficiency which amounts to 19.69 minutes ($\frac{15\times105}{80}$). The gain in the time is 9.69 minutes, which is the difference between the expected time (19.69 minutes) and the actual time taken (10 minutes) by $Annotator$-A.

The results from first six sets illustrate maximum gain for $k = 2$. So, for $k = 2$, we further chose four more sets taken respectively from the middle and bottom of the top 23% nodes, having confusion score above θ. All the results are shown in Table 6.6.

In human evaluation, we observed cases where the gain is *negative* and it typically happens for $Annotator$-B. The discussion with the annotators concluded that if the annotator is highly skilled, the extent of aid from our approach may be less as they can directly recall the label by observing the parent and child nodes. However, they further added that the approach would certainly assist semi-proficient, less skilled and relatively novice annotators. The argument can be supported with the fact that $Annotator$-B is highly proficient as she took only 80 minutes in comparison to $Annotator$-A who took 105 minutes to annotate the same set of parse trees with dependency labels.

6.3.2 Discussion

It has been clearly shown that there is a time gain from our approach. Though the experiment is done only at demonstration level, it can always be used at large scale to prove the exact worth of the approach. Also to ease the task of validation, we are working on developing a real-time interactive interface, where our approach could be effectively utilized by the human annotators without much complexity of understanding the algorithmic functionality.

6.4 Hybridization of Rules and Statistics

Both rule-based systems and data-driven statistical systems have their weaknesses and strengths (Krivanek and Meurers, 2011). A combination of the strength of both the systems while complementing oth-

ers' weaknesses, can be proved beneficial to our task of dependency parsing (Zhang and Wang, 2009; Grella, 2015). Though our data-driven statistical Malt parser-based Hindi Dependency Parser performs well in terms of LAS and now with the modification of $\mathcal{PQE}$, it also furnishes a measure of the correctness, it makes some very basic errors, which have the potential to wrong the whole parse tree and thus handicap the application at user's end.

For example, one of the very basic rules of Hindi dependency parsing is related to two very important tags: *k1* and *k2*[18]. The rule is stated as: *"Root of any parse tree can never be linked to two or more words with the same tag either k1 or k2"*[19] (henceforth referred as $k1 - k2$ rule). Violation of this rule led to one of the major errors that we analyzed in § 2.1.3. Though semantic information from Hindi WordNet (§ 3.2) reduced such errors to some extent, it could not completely remove them. In a statistical system, it is not easy to find out as what mistakes had oracle made while learning the rules (or patterns) from the training data. But with the assistance of human-crafted rules, such instances can be corrected. Integrating such rules with $\mathcal{PQE}$ score can solve the error in an autonomous fashion in a real-time output, without any need of human intervention or if could not solve the error, then can at least enrich us with the specific leads to solve the error.

6.4.1 Methodology

Again, in this application of $\mathcal{PQE}$, we are exploiting the same k-best labels as we obtained in § 6.2.3 but in a completely different manner. Earlier, we were identifying the potential errors using an optimal threshold value (cf. § 6.1.2). But now the identification is based on the human-designed rules. Those parsed trees which do not conform with the rules, are flagged as potential errors using *identification* algorithm developed on the basis of rules. In a parallel process, as the incorrect trees are identified, the automatic *correction* algorithm starts working side-by-side. In this algorithm, lists consisting of k-best labels, corresponding to the "edges labeled wrongly", are retrieved from a set of all the lists corresponding to all the tokens for all the sentences and then will be used for automatic correction. By "edges labeled wrongly" we mean, all those edges which are in the circle of doubt of being erroneous.

To better understand the working of algorithm, let us take an example based on violation of above $k1-k2$ rule and its correction. In the first stage, i.e. **identification stage**, an output parse tree is resulted and there is a need to identify those edges which can be labeled potentially incorrect as:

> **Given: ROOT**[20] of a sentence is attached to 3 tokens with three edges, namely $E1$, $E2$ and $E3$ and the edges are marked with dependency labels as *k1*, *k1* and *k3*.

[18] For description of dependency labels, refer to Bharati et al. (2009c)

[19] This rule is as per the guidelines of the Dependency Annotation Scheme (cf. Bharati et al. (2009c))

Edges Marked Wrongly: Clearly, $k1 - k2$ rule has been violated. Edges which will be marked as potentially incorrect are $E1$ and $E2$. Though it is quite possible that one of them is correct, it is not with surety which one is correct. So both will fall in the circle of doubt.

Continuing with the same example, during **retrieval stage**, k-best lists are retrieved, where each k-best list is sorted in decreasing order based on a score (it can be either *voting*, *probability* or combination of *voting* and *probability* (*voting+probability*) as obtained in § 6.2.3 separately) as:

E1: (label_{11}:score_{11}; label_{12}:score_{12}; label_{13}:score_{13}; ... ; label_{1k}:score_{1k})

E2: (label_{21}:score_{21}; label_{22}:score_{22}; label_{23}:score_{23}; ... ; label_{2k}:score_{2k})

where, both label_{11} and label_{21} are $k1$ in our example

Now with $if...else$ conditions (rules/constraints) in the automatic **correction stage**, the algorithm works as:

if $\text{score}_{11} > \text{score}_{21}$ then **E1** = label_{11} and **E2** = label_{22}; exit

else if $\text{score}_{11} < \text{score}_{21}$ then **E1** = label_{12} and **E2** = label_{21}; exit

else $\text{score}_{11} = \text{score}_{21}$ then continue with next best labels until the programme exits

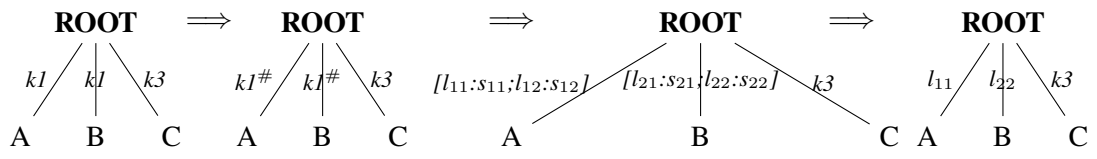


Figure 6.4 Schematic Working of Hybridized Algorithm (assuming $s_{11} > s_{21}$)

Figure 6.4 shows a schematic working of our hybridized algorithm. The above demonstrated example is just one particular case among many. Different hand-picked rules can be easily integrated with the

[20] In most of the types of sentences like intransitive, transitive, di-transitive, etc., which contain at least one verb, the main verb takes the role of ROOT or in co-ordinating sentences, the co-ordinating elements (and, or, etc.) act as ROOT

current functionality of Hindi Dependency Parser while keeping the core working of the **identification-retrieval-correction** algorithm same.

6.4.2 Discussion

Though instances with violation of $k1 - k2$ rule are very less as compared to the size of the Hindi Dependency Treebank and after correcting those instances, there is hardly any significant improvement on LAS via LS, the performance of the parser over a single sentence matters in a real-time scenario. As more and more such hand-crafted rules are integrated, there is every possible chance that quality of Hindi Dependency Parser will also improve.

Since all the steps including identification, retrieval of k-best lists and correction are done after the tree is parsed, it can also be termed as a **3 Stage Post-Processing Algorithm**, which is a hybridization of human-framed rules and the statistical scores obtained during computation of $\mathcal{PQE}$. Due to these stages in parsing, i.e., parsing using Hindi Dependency Parser and Post-Processing, the time-complexity of parsing increases. Now the whole process takes double time than a simple parser. The twice increase in the time-complexity is due to the furnishing of $\mathcal{PQE}$ score and post-processing step. This is one of the negative factors which deeply affect the trade-off between time-complexity and parsing accuracy. Analyzing this problem, we found out that this is due to the double efforts, Hindi Dependency Parser has to take. First, for predicting the parsed output and again to furnish the $\mathcal{PQE}$ scores. To make both the operations run parallely, we have already started working and hopefully, it will be solved in future.

6.5 Other Applications

6.5.1 Active Learning and Self Training

Active Learning includes querying unannotated sentences and ranking them based on the highest uncertainty they exert on an existing parser during parsing. The principle behind it is that the sentences parsed with low confidence, if annotated and included in training, contributes the most in reducing the uncertainty, among other unannotated sentences. Hwa (2004) in their work on sample selection for statistical constituency parsing, proposed an uncertainty score, measured in terms of entropy of a parse tree that a hypothesis grammar generate for a given sentence. Similar to them, $\mathcal{PQE}$ score can be used as $(1.0 - ConfusionScore)$ to measure the uncertainty for statistical dependency parsing. Modern day parsers, such as Malt parser, MST parser, are equipped with learning from partial annotated data. This give a possibility of extending a training data with partial but precise data which can be scrutinized based on $\mathcal{PQE}$ scores.

Similarly, Self Training (Bootstrapping) can also be used to improve an existing model of dependency parsing. Self training is a method of incorporating data using an existing model to build a new model. The existing model is first used to annotate unlabeled data and then a subset of the resultant

labeled data, treating as truth, combined with the actually annotated data to train new models. The acceptance of a part of the resultant labeled data as *Truth*, is based on the highest confidence of that part using a quality measure. The void of the quality measure can be filled with $\mathcal{PQE}$ to utilize this quality estimation score even in the exercise of self training to improve the existing models in a complete automatic manner.

6.5.2 Treebank Generation

Manual annotation of treebank with dependency information is a cumbersome task and quite susceptible to human errors. Therefore, it is usually followed by a validation step by a different annotator than the one who originally annotated the corpora. However, if the errors can be prioritized, say by utilizing the proposed $\mathcal{PQE}$ scores, major errors can be quickly eradicated by reviewing these suggested errors first. The modeling here has to be different from parse error identification as we no longer assume the annotated data as gold. A proposed methodology could be to employ cross-validation treating the hold-out set for parsing and the rest for training a parsing model, followed by a similar treatment as in parse error detection.

Any combination of approaches discussed in § 6.1, 6.2, 6.3 and 6.4, that can be proved useful for Treebank Generation, can be employed in an easy, effective and efficient manner. Along with the above positives, there is also a downside of the methodology. It can not identify an error uniformly distributed over the treebank like in case of domain adaptability or out-of-vocabulary. To address such kind of problems and many others, we are proposing a generic evaluation framework in terms of domain adaptability in the forthcoming chapter.

6.6 Summary

In this chapter we dealt with a variety of applications involving direct or indirect exploitation of $\mathcal{PQE}$ score, which not only improves the performance of a Dependency Parser but also helps in various tasks like Treebank Error Detection and Treebank Expansion. Discussions in this chapter opens a Pandora box full of new and interesting directions about Dependency Parsing, worth exploring in future.

Chapter 7

Generic Evaluation Framework: Domain Adaptability and Inter-Language Portability

In earlier chapters, we have discussed how dependency parsing can be aided by exploiting different algorithmic features, how semantic information can be used to solve different types of ambiguities and also how a quality score ($\mathcal{PQE}$) corresponding to a sentence construction provides an informed picture about the parse tree and help in developing multiple applications in real-time scenario. While doing all the work, we trained our supervised models on a given set of manually annotated training data and did tuning of all the variations and parameters on a development set, to report results on an unseen testing set. Thus, the Dependency Parser for Hindi is developed[1]. But one thing, that perhaps is missed in all the work, is the coverage of the parser on data sets other than the training genre or quality. Though the training data or in general the sentences present in Hindi Dependency Treebank (HDTB) [2] are extracted from multiple newspapers, assuming that a newspaper possibly covers many different domains of the universe, still it does not cover everything and there is a high probability that the parser, in its developmental stages, may get biased towards the quality of the training data. Thus, to complete the developmental stage of the parser and also to provide a holistic picture about the quality and utility of the parser, it is required to look into several other dimensions of parsing.

Dependency parser for any language or for that matter any NLP tool, should also be evaluated in terms of its robustness and comprehensiveness over different domains and genres, other than that of training-data. This evaluation can help in analyzing the strengths and weaknesses of the parser over a large variety of text and may also reveal about its coverage over the language and its usability for multiple purposes. This property can be termed as **Domain Adaptability**. In § 7.1, we will evaluate our Hindi Dependency Parser in terms of its capability to parse out-of-domain text.

Strength of an algorithmic structure of a data-driven dependency parser can also be tested against different languages. The same tool can be trained on different data sets of different languages and in a similar manner as the Hindi Dependency Parser is developed, dependency parsers for other languages

[1]Hindi Dependency Parser can be accessed at `http://ltrc.iiit.ac.in/showfile.php?filename=downloads/full_parser.php`

[2]`http://ltrc.iiit.ac.in/treebank_H2014/`

can be developed too. It is more probable that the methodology would work better on those languages which share the same grammar even partially, if not the same writing script. Put differently, languages belonging to the same language family[3] may be syntactically parsed using the same algorithm. But it is very unlikely that languages belonging to different language families share the same grammar. Even in that scenario, using the same algorithm, might help us to build very basic versions of the Dependency Parsers, which can be modified later in a more respective language specific way. While analyzing this dimension, we will term it **Inter-Language Portability**, as it involves porting Hindi Dependency Parser over other languages including both intra-family and inter-family.

Evaluation in terms of *Domain Adaptability* and *Inter-Language Portability* of Hindi Dependency Parser may enrich us with some new insights about the nature and working of the algorithmic framework used in Parsing. Throughout the thesis we have taken Hindi as the core and the central language to perform our experiments and discuss the obtained results and shortcomings. Thus, to test the *Domain Adaptability* of the parser, we considered the data of different domains to be also available in the same language i.e. Hindi in Devanagari [4] script. Evaluation of *Inter-Language Portability* is done separately for the following two cases:

1. **Intra-Family-Language Portability**: Languages share same grammar and also belong to the same language family but are written in different scripts.

2. **Inter-Family-Language Portability**: Languages share neither the same grammar nor the same script and also belong to different language families. [5]

7.1 Domain Adaptability

Developing a dependency parser for a particular language, trained on a particular treebank, may suffer from the problem of biasness as it could have become biased towards the quality and type of treebank data in a data-driven statistical system (Goldberg and Elhadad, 2010). There are many possibilities which may result in biasness like training data may not be an exhaustive set, dependency parser may parses better only those sentence constructions which belong to or are similar to the genre from which the training data has been collected. So it becomes the responsibility of researchers to check NLP tools like dependency parser on multiple domains to correctly evaluate the robustness of the same.

In this pursuit of robustness evaluation, we have taken data from four different domains namely, *Box-office*[6], *Cricket*, *Gadgets* and *Recipe*. It is a well accepted fact that no data-driven tool can be so

[3] A language family is a group of related languages which share a common ancestor, called the proto-language. For eg, Hindi and Urdu belong to same Indo-European language family.

[4] $Devanāgāri$, also called $Nāgāri$, is an alphasyllabary alphabet of India and Nepal.

[5] It is impossible that the grammar of languages belonging to two different language families do not share anything at all. It will be better to rephrase the above statement as: the divergences in corresponding grammars in case of inter-family languages will be much more as compared to languages belong to same language family. Due to clear demarcations we will refer the second option as not sharing any grammar.

[6] Bollywood Movies

robust as to cover all the possible domains in the same or impartial behavior unless the training data covers all the possible sentence constructions in the universal set of the language. But covering all the possible sentences in the training pool is itself not a realistic solution. Efforts have been taken in the past to build high quality parsers with domain adaptations (Gildea, 2001; Steedman et al., 2003; McClosky et al., 2006a; Blitzer et al., 2006; Petrov et al., 2010). Most of these works were surrounded around increasing tree size using various approaches like co-training (Steedman et al., 2003), re-ranking and self-training (McClosky et al., 2006a; Sagae, 2010) among others. At the same time one can also look on and work on the development side of the tool itself (in our case it is Dependency Parser) i.e. modifying or developing the tool in such a way that it may not give best results on any of the data set but better on all the domains simultaneously. In other words, A General Tool rather than A Master of a specific genre/domain.

7.1.1 Domain Data Statistics

As mentioned in 7.1, we are considering four different domains to observe the effect of Hindi Dependency Parser over them. The data was collected as a part of a project that was offered in Annual Summer Research Internship programme in LTRC [7], from various online websites offering information about the particular domains. The data was then cleaned in multiple stages and manually annotated by the participants of the project. It got enriched with multiple layers of information like POS-tag, morpho-syntactic features, chunk-head information and marking of dependency relations. Later, it got validated by the professional annotators [8] working in LTRC on different projects. Before using the data we also run some automatic checks while converting the SSF-format (Bharati et al., 2007) data into CoNLL[9] format using SSF-To-CoNLL Convertor [10]. Table 7.1 provides the data statistics for all the four domains in terms of the number of tokens (words), number of sentences, number of chunks (interChunks/chunk-heads) [11] and an estimate of the average length of the sentences.

7.1.2 Experiments and Results

Different kinds and levels of linguistic information corresponding to a token (or word) like POS-tag, morpho-syntactic information, etc. are used in the form of features in the task of parsing, to aid the overall process. In an ideal evaluation of Dependency Parser, all such linguistic information in the testing data (or in our case the domain data) has to be obtained in an automated manner i.e. separately using POS-Tagger for Hindi to mark POS-tag, using MorphAnalyzer to obtain morpho-syntactic information and other such similar information. Though we performed the POS-tagging and Chunking to analyze

[7] ltrc.iiit.ac.in

[8] We would like to thank Ms. Preeti Pradhan and Ms. Nandini Upasani for annotating the data.

[9] http://ilk.uvt.nl/conll/#dataformat

[10] https://github.com/riyazbhat/SSF-to-CONLL-Convertor

[11] A chunk is a set of adjacent words which are in dependency relation with each other, and are connected to the rest of the words by a single incoming arc.

Domain	#Tokens	#Chunks	Sentences	Avg. Sentence Length	Avg. Chunks in a Sentence
Box-Office	8605	3939	503	17.11	7.83
Cricket	10724	4952	542	19.79	9.14
Gadgets	9938	4348	529	18.79	8.22
Recipe	7610	4010	535	14.22	7.50

Table 7.1 Domain Data Statistics

their overall effect in error-cascading, but to capture the impact of parser alone, without impacting the parsing accuracy from the error-cascading effect (as happens in a sequential task), we use Gold information (manually annotated information).

We run our Hindi Dependency Parser, trained on the HDTB, on each of the domain test files separately. Table 7.2 shows the results obtained, where accuracies are reported in terms of the LAS scores. The other two columns show the results obtained after POS Tagging and Chunking using respective Tagger and Chunker, which are based on $CRF++$ (Lafferty et al., 2001)[12] package.

Domain	POS Tagging (%)	Chunking (%)	Parsing (LAS) (%)
Box-Office	87.09	92.36	75.99
Cricket	89.63	92.65	69.75
Gadgets	82.60	91.79	71.53
Recipe	83.73	89.79	65.67
Average	85.76	91.65	70.74

Table 7.2 POS-tagging, Chunking and Parsing Accuracies on Domain data

[12] `http://crfpp.googlecode.com/svn/trunk/doc/index.html?source=navbar`

7.1.3 Discussion

Domain	Tagging/Chunking (All Tokens)	Parsing (Chunk-heads)
Box-Office	17.98	12.92
Cricket	20.43	17.27
Gadgets	30.39	25.45
Recipe	27.69	33.58

Table 7.3 Domain OOV rates

If we compare the average accuracy of the Dependency Parser over different domains, it comes to 70.74% which is significantly less than the parsing accuracy over the testing data (83.69%). But the parser performs relatively better for *Box-Office* than other three domains and *Gadgets* & *Cricket* domains perform better than *Recipe* domain. *Recipe* domain performs least, which can be validated by the observation that it shares maximum OOV[13] rate with respect to chunk-heads used in parsing, among all the domains as mentioned in Table 7.3. Least OOV-rate for *Box-Office* domain also corresponds to the best performance of Dependency Parser over the domain.

Though the OOV rate for *Gadgets* domain is quite less as compared to *Cricket* domain, the parsing results speak otherwise with a difference of 1.78%. There could be multiple reasons behind this. For instance, sentence constructions in *Gadgets* domain may be more in sync with the training data. One thing that is clear from these experiments is that the absence of chunk-heads, which most of the times belong to open-class category words [14], impact heavily on the parsing behavior. The problem can potentially be handled using the semantic information which will classify the words belonging to same ontological hierarchy as we have used in Chapter 3 (§ 3.2.2), while exploiting semantic information from Hindi WordNet. In addition, WordNet also provides many other types of information like SynsetID corresponding to a group of similar words, which can also help to reduce the OOV rates from different domains.

[13] Out-Of-Vocabulary (OOV) Rate: Percentage of tokens that are not common in two different data sets

[14] Words can be categorized into two classes: Open and Closed. In open class, new words can be added when the need arises for instance, Nouns, Verbs, etc. On the other hand, words like prepositions, determiners, or conjunctions, have a finite sets of words which are never or rarely expanded and that too over long periods of time.

Domain	Inter-Annotator Agreement
Box-Office	0.68
Cricket	0.64
Gadgets	0.73
Recipe	0.57

Table 7.4 Inter-Annotator Agreement for Domain data

To explain the not-so-good results, we have also calculated the inter-annotator agreement [15] over the domain data. Table 7.4 shows the κ [16] (Galton, 1892; Smeeton, 1985; Cohen and others, 1960) value which denotes the qualitative aspect of annotated data. It is clear from the table that annotators agreed least when annotating the *Recipe* domain data. This observation provides ample evidence that the quality of the *Recipe* domain data is not good as compared to other three domains where there is more agreement. It is highly possible that the data contains many errors. The κ value for other three domains is also not much appreciable. Thus, even if our parser performed well enough on the domain data, the evaluation against the poor quality of annotated data would result in lower LAS scores. Application of $\mathcal{PQE}$ (cf. § 6.1) as mentioned in Chapter 6 can help in identifying the potential errors and thus, improve the quality of the treebanks. Ensembling of different parsers as implemented in § 4.1 can also be used on the similar lines of Annamaneni et al. (2013).

As a note, it should also be clarified that above measure of κ value can not raise any questions on the quality of Hindi treebank data which had also been annotated by the same annotators. HDTB has been developing for last seven years and also passed through multiple types of both automatic and manual validation stages. Thus it had acquired a reasonable quality. Even one of the reasons of poor quality of manually-annotated domain data is that it had been annotated by the annotators after a wide gap of ten months from the Hindi-Urdu Treebank Project [17] and thus it is quite possible that annotators might have lose the same efficiency and effectiveness as they had while annotating the HDTB.

Though Hindi Dependency Parser not performed at par with the testing data, it is justified as testing data is but a small part of the larger Hindi Dependency Treebank and thus the parser still carried biasness towards testing data. In addition, the inter-annotator agreement justify the results. Though for future it is worth exploring to modify the Hindi Dependency Parser to handle different domains in a better way. Even aiding the parser using the semantic information can also be integrated as a permanent functionality in Hindi Dependency Parser. Using Ensembling approach, as we have implemented in

[15] Inter-annotator agreement is a measure of how well two (or more) annotators can make the same annotation decision for a certain category

[16] κ measures inter-rater agreement for qualitative items and is generally thought to be more robust measure than simple percent agreement calculation, since κ measures agreement by chance

[17] `http://verbs.colorado.edu/hindiurdu/`

Chapter 4 on the lines of Sagae and Tsujii (2007), can also be explored in future. Even ensembling based on $\mathcal{PQE}$ as discussed in § 5.1.1, can be proved more effective as it prevents any kind of overfitting or biasness. Further, Human-Aided Parsing based on $\mathcal{PQE}$ (cf. § 6.2), is also a workable option to be explored to improve the results.

7.2 Inter-Language Portability

Dependency parsers can be developed using different strategies like data-driven or rule-based. But in a scenario where there exists thousands of languages, building an NLP tool from scratch and that also a dependency parser, which itself demands specific pre-requisites, is a time-consuming and an expensive task. In a world which is changing more rapidly than ever and with it, the increasing needs and demands of the consumers, there is a requirement to find intelligent methods which must be both time and cost-effective.

Though rule-based systems, as they evolve over time, have the potential to outperform the data-driven systems, to craft the rules for any language is a costly task both in terms of time and of expenses on qualified manpower. Further as we include new domains, we might need many new rules and the list of hand-picked rules could run upto thousands. Many a times, it becomes an obstacle to achieve the results in a speedy manner. Data-driven systems can solve this problem by utilizing the automatic platform to derive the rules. But they too need manually annotated 'data' to 'drive' the task. Both rule-based systems and data-driven systems require, in addition to actual annotation/rule framing, a substantial effort on treebank/grammar design, format specifications, tailoring of annotation guidelines, etc. The latter costs are rather constant no matter how small is the resulting corpus.

This raises a question whether it is ever possible to build a parser without a treebank (or a broad-coverage of formal grammar) of the target language. In past this problem has been addressed by Klein and Manning (2004), Hwa et al. (2005), McClosky et al. (2006b), Zeman and Resnik (2008) among others. Klein and Manning (2004) used a hybrid unsupervised approach, which combines a constituency and a dependency parser and achieved good results. Hwa et al. (2005) used the technique of parallel corpus for automatically obtaining word alignments in the target languages in addition of a limited-set of language-specific post-projection transformation rules. The work of Zeman and Resnik (2008) was based on McClosky et al. (2006b)'s re-ranking and self-training algorithm.

If somehow we could generate the data for training the parser, then the task becomes quite easy and the development of dependency parser becomes faster. Porting an already available tool to other languages could be one such solution to ease the problem of developing new dependency parsers (Chanev, 2005). Processes to port our Hindi Dependency Parser on an intra-family-language and inter-family-language are different and will work differently. Even in the past the same Malt parser has been used to parse multiple languages using different set of functionalities corresponding to languages (Buchholz and Marsi, 2006; Hall et al., 2007; Nilsson et al., 2007) and even performed best among all other parsers over some languages and second best over the remaining. We are using the same Malt parser but with

the modification of $\mathcal{PQE}$ score, that is used to build our Hindi Dependency Parser, for parsing other languages.

Virtuous Cycle of Parser Development: To solve the problem of unavailability of data, we are innovating a cyclic methodology, which involves both the generation of the treebank and the development of the dependency parser in a cyclic way.

The Cycle starts with the need of a set of seed[18] training data of the target language. After porting Hindi Dependency Parser for that language, the new parser could be trained on the seed data. The target language parser could now be used to parse the new unannotated raw sentences. Along with the system predicted dependency relations, the parser also furnishes $\mathcal{PQE}$ score. $\mathcal{PQE}$ score will be the driving force to run the cycle. Now using one of the applications of $\mathcal{PQE}$ as discussed in § 6.3, the treebank of the target language could be expanded. The approach of Treebank Generation can also be utilized along with Active Learning as mentioned in § 6.5.2 and § 6.5.1 respectively. While using the $\mathcal{PQE}$ score, the rules can also be framed and then the same can be used in a hybrid fashion to further improve the Dependency Parser (as discussed in § 6.4). Now the more advanced version of the parser, can again be utilized to parse the remaining sentences, to further expand the treebank and one can decide to stop the cycle if satisfied with the size of the treebank or reached to an acceptable level of performance of the Dependency Parser over the target language.

One of the things which might get missed in this whole discussion of the Virtuous Cycle, is the *Dependency Annotation Scheme*. Which scheme one should use for the target language: the Hindi Annotation Dependency Scheme (Begum et al., 2008) or design a new one? It has been observed that different languages are annotated using different schemes. To derive the dependency relations between the tokens of a sentence, a common but more comprehensible scheme is always welcome. The Paninian Grammar, has assumed such a place, at least, among Indian languages, if not worldwide (Bharati et al., 1995). The Panian grammar captures all the significant syntactico-semantic relations in a language. Also efforts like Universal Dependencies [19] (De Marneffe and Manning, 2008; McDonald et al., 2013; De Marneffe et al., 2014) are new solutions to ease the task of Parsing both mono-lingually and cross-lingually. Even in an extrement scenario, if some dependency relations, which are indispensable to capture important information in a language, are required, then instead of designing a complete new Dependency Annotation Scheme from scratch, one can always modify or add new relations in the existing schemes.

Before starting the discussion over the specific instances of Intra-Family-Language Portability and Inter-Family-Language Portability, some clarifications are required. Though the Virtuous Cycle is generic and applicable to both intra-family and inter-family languages, but there could be some specifications that might be required as per the behavior and properties of the language and its relation with Hindi. For an intra-family language, there are four possibilities:-

[18] A seed data unlike a Treebank, consists of very few manually annotated sentences of a particular language

[19] `https://lindat.mff.cuni.cz/repository/xmlui/handle/11234/1-1548`

- If the language (or dialect) is written in the same script and shares same grammar with Hindi, then the task becomes easier. For instance, hundreds of dialects like Khari Boli, Awadhi, Brij Bhasha, etc. share same grammar and script with Hindi.

- If the sibling language shares only grammar and not the script, which is again not so common, but possible like in case of Hindi and Urdu, then the cyclic methodology will not be required and using a transliterator (Srivastava and Bhat, 2013) can also ease the development of dependency parser. We are exploring this possibility in § 7.2.1.

- Another case with something common between two siblings is when both languages share same script but not the grammar. Since dependency parsing is a task where the word-order and the grammar matters most, the sharing of script can only help minimally by saving the effort of transliteration but even then the Virtuous Cycle needs to be followed. 11 of the official languages of India use same Devanagari script (Dongre and Mankar, 2011).

- The case in which the sibling language neither shares the grammar nor script, then the Virtuous Cycle of Parser Development cab be applicable without any modification. English and Hindi, both belong to Indo-European language family, but neither the grammar is same nor writing script.

In case of an inter-family-language, the divergences in sharing same grammar and script with Hindi will be much more as compared to intra-language family and the procedure can be directly referred to the Virtuous Cycle. This will be discussed in detail in § 7.2.2.

Since our task is to explore the portability behavior of Hindi Dependency Parser and not to create Treebanks, we are utilizing the treebanks available with us to experiment with portability behavior. For Intra-Family-Language Portability we are porting Hindi Dependency Parser to Urdu. Though a similar Urdu Dependency Parser (Bhat et al., 2012) based on Malt parser is already explored but not with the $\mathcal{PQE}$ functionality and definitely not using portability methodology. Similarly, for Inter-Family-Language Portability, we are applying our approach over the Dravidian Language Family, different than the Indo-Aryan Language Family.

7.2.1 Intra-Family-Language Portability

Under this head, we are exploring the case when two languages belonging to the same language family share the grammar but not the script. Since sharing grammar can aid the task of dependency parsing, the efforts required in implementing Virtuous Cycle can be saved and hence, this case needs separate attention.

We are considering Hindi and Urdu, as the language pair to perform the task of portability. Hindi and Urdu together constitute the third largest language spoken in the world[20]. They are two standardized registers of what has been called the *Hindustani* language, which belongs to the Indo-Aryan language

[20] http://www.ethnologue.com/statistics/sizeandhttps://en.wikipedia.org/wiki/List_of_languages_by_number_of_native_speakers

family. Masica (1993) explains that, while they are different languages officially, they are not even different dialects or sub-dialects in a linguistic sense; rather, they are different literary styles based on the same linguistically defined sub-dialect. He further explains that at the colloquial level, Hindi and Urdu are nearly identical, both in terms of core vocabulary and grammar. However, at formal and literary levels, vocabulary differences begin to loom much larger (Hindi drawing its higher lexicon from Sanskrit and Urdu from Persian and Arabic) to the point that where the two styles/languages become mutually intelligible. In written form, not only the vocabulary but the way Urdu and Hindi are written makes one believe that they are two separate languages. They are written in separate orthographies, Hindi being written in Devanagari, and Urdu in a modified Perso-Arabic script (Nastaleeq). Given these differences in script and vocabulary, Hindi and Urdu are socially and even officially considered two separate languages. These apparent divergences have also led to parallel efforts for resource creation and application building in computational linguistics. Hindi-Urdu treebanking is one such example, where the influence of differences in Hindi and Urdu text have led to the creation of separate treebanks for Hindi and Urdu (Bhatt et al., 2009; Xia et al., 2008). However, pursuing them separately in computational linguistics makes sense. If the two texts differ in form and vocabulary, they can not be processed with a single model unless the differences are accounted for and addressed.

Although study of the similarities and differences between Hindi and Urdu in linguistic sense requires a completely separate discussion and exploration (We have already started working in this direction with Bhat et al. (2014b) as the first step), but to demonstrate the portability behavior of a dependency parser in a comprehensible manner, we are experimenting with them. Through this experiment, we are trying to affirm, if, there is any possibility to handle both the languages at the parsing level using a single portable model.

7.2.1.1 Data

The experiments that need to be performed use data from both Hindi Treebank (HTB) and Urdu Treebank (UTB). In our earlier experiments we used partial HTB data that has been released during COLING, 2012 Shared Task (Sharma et al., 2012). But after that HTB has been expanded and more number of sentences have been added. The updated version of HTB can be accessed at `http://ltrc.iiit.ac.in/treebank_H2014/`. Both these treebanks are multi-layered and multi-representational (Bhatt et al., 2009; Palmer et al., 2009; Bhat et al., 2014a). They contain three layers of annotation namely **dependency structure (DS)** for annotation of modified-modifier relations, **PropBank-style annotation** for predicate-argument structure, and an independently motivated **phrase-structure annotation**. Each layer has its own framework, annotation scheme, and detailed annotation guidelines. [21]

Dependency Structure, the first layer in these treebanks involves dependency analysis based on the Paninian Grammatical framework (Bharati et al., 1995; Begum et al., 2008). *Pāṇini* was an Indian grammarian, who is credited with writing a comprehensive grammar of Sanskrit. The underlying theory

[21] `http://verbs.colorado.edu/hindiurdu/`

of his grammar provides a framework for the syntactico-semantic analysis of a sentence. The grammar treats a sentence as a series of modified-modifier relations where one of the elements (usually a verb) is the primary modifier. This brings it close to a dependency analysis model as propounded in Tesnière's Dependency Grammar (Tesnière, 1959). The complete statistics about HTB and UTB are provided in Table 7.5.

Language	Sentence Count	Token Count	Chunk[22] Count	Avg. Sentence Length
Hindi	20,783	434,586	142,445	20.91
Urdu	6,786	191,447	21,165	28.21

Table 7.5 Statistics of latest versions of HTB and UTB

7.2.1.2 Experiments and Results

To perform our experiments we divided both the treebanks into five parts using stratic sampling based on the sentence length, for five-fold cross-validation. Since we already have UTB, a manually annotated treebank for Urdu, we built a Urdu Dependency Parser based on the lines of Bhat et al. (2012) as the size of the data has increased (cf. Table 7.5). In five-fold cross-validation, we used four parts as the training data and the remaining part as the testing data (cf. Table 7.6). The average accuracies of five-fold experiments, obtained, are reported in Table 7.7 in terms of LAS.

	Training	Testing
Hindi	16,629	4,154
Urdu	5,432	1,354

Table 7.6 Training and Testing data for Hindi and Urdu

	Gold (in %)	Predicted (in %)
Hindi	84.62	81.89
Urdu	74.03	72.68

Table 7.7 LAS Score via Separate Parsers for Hindi and Urdu Dependency Parsing (Baseline Systems)

[22] A chunk is a set of adjacent words which are in dependency relation with each other, and are connected to the rest of the words by a single incoming arc.

The algorithm used in developing Hindi Dependency Parser, showed its effectiveness for Urdu also. Thus fulfilling the portability aspect. But this is only partial story. We could have also ported the HTB to parse Urdu sentences due to the similarities in their grammar. In other words, Hindi Dependency Parser, trained on HTB can also be used to parser Urdu constructions. The only problem is doing this experiment is that sentences used in training data and testing data (in this case the Urdu data) must be in same script. To bring both the training and testing data in a same script, we have the following three scripts:-

- Devanagari
- Persio-Arabic
- WX-notation[23]

To select one of the scripts, in which our parser can perform the best, we did some pilot experiments. First, we try to check the effect of script in POS-Tagging. Since POS-tagging is the first stage towards dependency parsing (as discussed in Chapter 1), its impact will also be significant in further stages due to error-cascading property in sequential tasks. We used a transliterator by Srivastava and Bhat (2013), to transliterate HTB training data present in Devanagari to Persio-Arabic and to transliterate UTB testing data[24] from Persio-Arabic to Devanagari. To bring both HTB training data and UTB testing data into WX-notation, we used WX-convertor [25] to normalize the datasets.

After bringing data sets into the same script, we built temporary POS-taggers using $CRF++$ (Lafferty et al., 2001)[26] package, trained on HTB, in all the 3 scripts and tested them on UTB testing data on all the 3 scripts correspondingly. We found out that the experiment performed in Devanagari script, resulted in better accuracy (70.65%) than in Persio-Arabic script (69.47%) and much better than in WX-notation (31.88%). So as per this experiment, it is confirmed that we need to select Devanagari script for future parsing experiments, as using this script minimizes our chances of error. To reinforce our observation that POS-tagging plays a significant role in error-cascading and the error in this stage must be minimized, we performed another experiment to measure the impact of POS-tagging in future tasks that use POS-tag as a pre-determined feature. We built intermediate Chunker also based on the CRF-algorithm which uses POS-tag as a feature while classifying the chunk-heads. This time, we single handedly used Devanagari script to perform the experiment. While using gold POS-tag (manually annotated POS-tag) as a feature, the results were significantly better than using the system predicted POS-tag with a margin of 18.84% ($91.81 - 72.97$). So it is undoubtable to select that script which results best POS-tagging accuracy in the first stage of natural language processing.

Finally the experiment of porting a HTB trained Dependency Parser to parse UTB testing data in Devanagari script is performed. In order to parse in more realistic scenario, we performed the exper-

[23] `http://sanskrit.inria.fr/DATA/wx.html`

[24] We took testing data as one of the five parts in the stratic-splitting of UTB

[25] `http://ltrc.iiit.ac.in/showfile.php?filename=downloads/FC-1.0/fc.html`

[26] `http://crfpp.googlecode.com/svn/trunk/doc/index.html?source=navbar`

iment using system predicted linguistic information like POS-tag and Chunk-tag. The obtained LAS score over Urdu testing data is 57.72%.

7.2.1.3 Discussion

If we compare the results of both, porting Hindi Dependency Parser over Urdu and porting HTB for Urdu Dependency Parser, there is a significant loss of accuracy. The latter, using predicted information as features resulted in 57.31% LAS as compared to former 72.68% with the loss of 15.37%. This might conclude that though porting Hindi Dependency Parser in algorithmic terms is quite effective, the same not true in porting HTB for Urdu Dependency Parser. This may further conclude that Hindi and Urdu can not be dealt together and must need separate treebanks and all similarly other NLP tools at different level of processing. But to completely understand the ineffectiveness, we need to consider several other factors which must have played their role in making HTB ineffective, handling Urdu sentences.

- **Effect of Transliterator:** Though using transliterator developed by Srivastava and Bhat (2013) was a good choice, while transliterating Urdu sentences written in Persio-Arabic into Devanagari script, there are multiple cases of errors and the accuracy can never be 100%. Perhaps this is one of the reasons why POS-tagger in Devanagari script performed slightly better than in Persio-Arabic script as transliterating HTB into Persio-Arabic consisting of more than $20,000$ sentences would add more error than transliterating $6,786$ sentences present in UTB.

- **Error Cascading:** We have already observed during our experiments that due to low accuracy at POS-tagging level, the accuracy of Chunking also dropped and both of these have contributed significantly in degrading the quality of linguistic information available in the form of features during parsing.

- **OOV Rates:** Though both Hindi and Urdu are nearly identical in terms of core vocabulary and grammar, but in terms of lexicon, there are appreciable differences. Hindi draws its lexicon majorly from Sanskrit, while Urdu from Persia and Arabic along with Indic languages. The OOV-rate for UTB testing data with respect to HTB training data is found to be 42.25%. This must be due to two reasons: (1) some tokens of Urdu text are wrongly transliterated and (2) many words must be genuinely absent. Further, one more observation that we found out was that OOV-rate was least when both datasets Hindi and Urdu are in Devanagari as compared to either Persio-Arabic or WX-notation. This is an additional proof of the right selection of the script.

- **Domain Problem:** The porting of Hindi Dependency Parser over Urdu, also bring another interesting insight that whether parsing Urdu using HTB can be termed as another domain problem and should be handled similar to that rather considering Urdu as a separate language. The comparison of OOV-rate in case of UTB testing data with Domain OOV-rates (cf. Table 7.3) and the corresponding parsing accuracies, clearly point out that there is a strong co-relation between OOV-rates and LAS scores. The more the OOV-rate, the less the parsing accuracy, which is also

an easy observation. Moreover, the less OOV-rate for Urdu, is also due to the transliteration error which is not the case with domains. Another supporting argument is, despite of using system predicted features like POS-tag and Chunk-tag in Urdu parsing experiment, its LAS is not as much lowered as compared to the domain parsing accuracies where we used manually annotated information, from the overall performance of Hindi Dependency Parser. More error cascading due to 70.65% POS-tagging accuracy as compared to an average of 85.76% in domains, also degraded the performance of Hindi Dependency Parser over Urdu testing data. Hence, all these observations strongly direct that the problem or ineffectiveness arose during porting HTB over Urdu, can be solved in the similar manner as with domains.

Porting Hindi Dependency Parser over Urdu has opened new doors to study the relationship between Hindi and Urdu in a new and separate manner so as to exploit it to save the cost, time and human resources. The work has already been started with first step to handle low OOV-rates as Bhat et al. (2014b), which can help in reducing error-cascading effect and improve the overall parsing performance.

7.2.2 Inter-Family-Language Portability

Among all the language families other than that Indo-Aryan to which Hindi belongs, we select Dravidian family. As after Hindi, it has the maximum number of speakers in the country [27]. Also, since the treebanks of these languages, though very small in size, are also available, it would be helpful to establish the efficacy of our approach of portability over inter-family languages. Though much work has been done in the early stages of language processing for these languages, dependency parsers do not exist at par even with Hindi. While demonstrating the applicability of our approach, we built the first-hand naive versions of dependency parser for **Tamil** and **Telugu**, which can be integrated in pipeline systems like Machine Translation (MT) and Natural Language Interface to Database (NLIDB).

7.2.2.1 Data

Under the project **Indian Languages-Indian Languages Machine Translation (IL-ILMT)**[28], the work has been done to create treebanks of multiple Indian languages and especially those which have maximum users in India. This project is being led by IIIT Hyderrabad in a consortium of 13 institutes, working in the area of Linguistics. The creation of *Tamil* and *Telugu* Treebanks is part of the project for the task of Machine Translation.

The Tamil Treebank is created by the team of Anna University (AU)[29]. Similarly, the task of creating Telugu Treebank has been assigned to University of Hyderabad (HCU)[30]. Treebank creation from

[27] https://en.wikipedia.org/wiki/List_of_languages_by_number_of_native_speakers_in_India

[28] http://ltrc.iiit.ac.in/MachineTrans/showfile.php?filename=projects/

[29] https://www.annauniv.edu/

[30] http://www.uohyd.ac.in/

scratch is a time-consuming and resource-intensive task as evident by the fact that creation of HTB itself took around 7 years and had been validated multiple times to improve its quality by reducing different kinds of errors. These treebanks have also been used in various shared tasks organized by different national and international conferences for building various NLP tools (Husain, 2009; Husain et al., 2010; Sharma et al., 2012). Though it might be possible that the quality of Tamil and Telugu Treebanks is not as good as compared to Hindi and Urdu[31], but they are at least in an appreciable state, enough to build advanced NLP tools like Dependency Parsers. The data statistics of Tamil and Telugu treebanks, which we are using for our portability experiments, are reported in Table 7.8.

Language	Sentence Count	Token Count	Chunk Count	Avg. Sentence Length
Tamil	876	7,520	4,577	8.59
Telugu	2,133	13,646	10,222	6.40

Table 7.8 Data Statistics of Tamil and Telugu treebanks

7.2.2.2 First-Hand Dependency Parsers

Though relatively much work has been done for Hindi dependency parsing, the same attention has not been paid towards other languages which also have significant number of users in the Southern part of the country. Earlier works for both of the Dravidian languages involve Constraint Based Hybrid Systems (Kesidi et al., 2010), Hybrid of Rule-based and Machine Learning (Ramasamy and Žabokrtskỳ, 2011; Sureka et al., 2014) or completely machine-based systems (DHIVYA, 2011). But there were no systems which could be easily integrated in a pipeline manner and also useful for real-world applications. Thus we ported Hindi Dependency Parser, which is build on an language-independent platform provided via Malt parser (Nivre et al., 2007), to build the first-hand dependency parsers. These first-hand dependency parser might not be as powerful and accurate as the Hindi Dependency Parser but undoubtedly it will be helpful for the research community to utilize them in other NLP tasks like Machine Translation and NLIDB systems. They have already been embedded in the current pipeline of IL-ILMT system, which itself speaks about the story of their success. Further, it is always welcome to improve their functionality using more language specific features in the form of hand-crafted rules or using approaches similar to Zeman and Resnik (2008), Smith and Eisner (2009) and Søgaard (2011)

We ported the algorithmic set-up of Hindi Dependency Parser over Tamil and Telugu. First, we splitted the corresponding treebanks in training and testing data in the ratio of $90 : 10$, using stratic sampling based on the length of the sentences. The Dependency Parsers are trained on the training data and the parsing accuracies are reported on the testing data as shown in Table 7.9.

[31] It took around ~ 4 years in creation and still improving

	LAS (in %)	UAS (in %)	LS (in %)
Tamil	64.04	84.26	70.21
Telugu	69.23	89.87	70.89

Table 7.9 Parsing Accuracies of Tamil and Telugu Dependency Parsers

7.2.2.3 Discussion

Though the parsing scores are not too encouraging as compared to Hindi and Urdu Dependency Parsers, these parsers can now be easily integrated in pipeline systems in a developer-friendly manner. In addition, another key feature with which these parsers are enriched, is the availability of $\mathcal{PQE}$ score, which has been proved very useful in improving Dependency Parsers and even in the expansion of Treebanks (cf. Chapter 6). Since we are neither native speakers of Dravidian languages nor experts in their grammar, it was a natural constraint for us to not being able to exploit other rich linguistic information as we have done for Hindi in the form of WordNet Ontologies (cf. § 3.2). Also, the same limitation hindered us in making the tool more robust. With the assistance and guidance of the linguistics experts of Dravidian languages, we are planning to improve the performance of respective Dependency Parsers as a future task. Also, since lexical resources like WordNet, exist for Tamil and Telugu (Rajendran et al., 2010) [32], they can be explored on the same lines as of Hindi.

7.3 Other Indian Languages

Parsing plays an important role in various NLP tasks like Machine Translation and others. For morphologically-rich and free-word-order (MoR-FWO) languages, dependency parsing is more suitable than constituent parsing. Our approaches like Domain Adaptability and Inter-Language Portability can be utilized for any language and more importantly for Indian languages. As as instance of it, porting Hindi Dependency Parser over other languages has already been started as a regular exercise. In IASNLP 2014 [33], many projects had been floated to explore dependency parsing for Marathi, Malayalam and other Indian languages. Work has already been started for Punjabi and Bangla as a part of phase-II of IL-ILMT project using same approaches. Treebanks for Malayalam and Kannada are in progress and as soon as they will be available to us, we will be able to deliver respective first-hand dependency parsers for the same. For low-resource languages (LRU), the Virtuous Cycle (cf.§ 7.2) can be exploited very efficiently, both, in the creation and expansion of treebanks and developing corresponding dependency parsers. Once dependency parsers for different languages are available, they can also be made more robust and comprehensive for other domains and genres.

[32] http://www.cfilt.iitb.ac.in/indowordnet/

[33] http://ltrc.iiit.ac.in/iasnlp2014/index.cgi?topic=projects

Chapter 8

Conclusion and Future Work

8.1 Summary

In this dissertation we have illustrated the significance of dependency parsing as a type of syntactic parsing in NLP and various efforts that have been taken up in past to build high quality dependency parsers for Hindi. We presented our experiments that led us to build the then state-of-the-art Hindi Dependency Parser through a shared task (Sharma et al., 2012). Since the experiments were performed in a brute-force method due to time constraints, we further tried to focus on some of the core features specific to Hindi language in depth, which we used later to effectively exploit the functionalities available with the Malt parser, a transition-based data-driven statistical parser.

To advance the process of dependency parsing for Hindi, first we studied the role of morphological and semantic features to address some of the problems like case ambiguity, lack of case marker, data sparsity, etc. We extracted semantic information in an automatic manner from Hindi WordNet, a rich lexical resource for Hindi. The semantic information that we had used, was present in Hindi WordNet as Hierarchical Concept Ontologies for each lexical item (word). We have used the hierarchical concept ontologies as a feature set in Malt parser to improve the parsing accuracy. Though due to large size of Hindi Treebank, the impact of such information is not fully realized on dependency parsing, it led to some interesting insights, which motivated us to move to another approach.

As a second approach to make advancements in Hindi Dependency Parsing, instead of using a single parser, we worked on combining the strength of multiple base parsers to improve the results. A technique known as Ensembling, has already been established effectively for multiple languages in past. We experimented with the diversity of base parsers that could have been captured by different algorithms in the form of six-algorithmic variants of Malt parser. The ensembling unlike earlier, has been performed at inference time for Hindi in two ways namely, re-parsing algorithms and word-by-word voting. The systems have been combined using six different weighting strategies. The latter performed better than the former and we have observed an improvement of $\sim 0.5\%$ in labeled attachment score (LAS). To provide a systematic comparison between both the approaches we have posed the same questions as used by Surdeanu and Manning (2010) to present an informed discussion on ensembling for Hindi.

Motivated by the observation that weighting does matter in ensembling for Hindi, we have studied our final approach to improve the parsing performance. We used an entropy-based dynamically computed score in the form of confusion that the oracle makes while predicting the dependencies. Adopting the methodology proposed in Jain and Agrawal (2013), we extended the Parse Quality Estimation ($\mathcal{PQE}$) functionality (corresponding to dependency labels) for all the remaining five algorithms not done by Jain and Agrawal (2013) and performed ensembling with new weighting strategy. The new weighting scheme improved the accuracy in ensembling via re-parsing algorithms methodology. In addition, we also extended the $\mathcal{PQE}$ functionality for dependency arcs and for joint prediction of both dependency arcs and labels.

To prove the efficacy of $\mathcal{PQE}$ score, we also implemented several applications using the same. We demonstrated how $\mathcal{PQE}$ can be used for Automatic Parse Error Detection, Improving parsing performance in semi-automatic way (Human-Aided NLP systems), Minimizing the cost of treebank validation during treebank expansion and how the score can complement the hand-crafted rules in a hybrid manner.

Finally, to put focus on robust evaluation of a dependency parser, we proposed a generic evaluation framework in terms of Domain Adaptation (DA) and Inter-Language Portability (ILP). Using data from four different domains, we evaluated the performance of the Hindi Dependency Parser. Similarly for Inter-Language Portability, we experimented with languages belonging to both intra-family languages and inter-family languages using Urdu, Tamil and Telugu.

8.2 Conclusion and Future Work

We have gained significant insights through our methods and also used different approaches for Hindi which either have not explored in past or explored minimally. Though real use of such systems are in real-world applications, there is still scope to increase the parsing accuracy while further reducing the time-complexity to make parsing useful in real-time applications. Further, the approaches mentioned in this thesis can be used effectively for low-resource languages.

Our dissertation provide many new directions that need to be explored in future. To name a few:

- Hindi WordNet has huge potential in providing multiple types of semantic information like hypernymy, hyponymy, etc. which can be exploited in the same manner as we did or in other ways.

- The same approach of introducing semantic information for dependency parsing can also be implemented for other languages, especially for low-resource languages.

- More diversity can be introduced in base parsers by using different dependency frameworks like MST parser, Turbo parser, etc. and also using different weighting strategies to make ensembling more effective. Also new techniques to combine multiple votes from multiple base parsers can be explored in future.

- $\mathcal{PQE}$ functionality corresponding to attachments and joint prediction can also be extended to other algorithms and later be used for ensembling, in addition to many applications.
- Hindi Dependency Parser, ca be made more robust by making is more suitable to different domains.
- Porting Hindi Dependency Parser to other Indian languages can be proved helpful in future to build new systems in a speedy and cost-effective manner.

Due to increasing penetration of social media in every sphere of life, there is a new challenge in the form of processing code-mixed and code-switched data. Though we also took an initial step in the form of Jain et al. (2014), this is just a first step and more needs to be done in the area of syntactic parsing to make it more useful.

Appendix A

Results: Automatic Parse Error Detection

		Swedish[¶]						Turkish[#]					
	Measure	**F**	**P**	**R**	**EDI-1**	**EDI-5**	**EDI-10**	**F**	**P**	**R**	**EDI-1**	**EDI-5**	**EDI-10**
Label	**Baseline-I**	10.90	8.98	13.86	0.76	3.78	7.57	13.55	10.30	19.81	0.74	3.68	7.36
	Baseline-II	35.09	29.13	44.11	0.00	0.00	28.82	38.27	25.93	73.06	3.58	17.88	35.76
	Independent	41.56	31.69	60.36	0.69	3.44	19.60	45.79	33.21	73.69	0.70	3.51	16.93
Attachment	**Baseline-I**	10.27	8.47	13.05	0.75	3.77	7.53	14.48	11.48	19.61	0.73	3.65	7.31
	Baseline-II	30.58	23.43	44.03	3.49	13.61	26.74	37.62	25.06	75.46	1.96	9.60	19.96
	Independent	32.32	24.94	45.91	0.73	3.63	20.73	38.89	26.93	69.96	0.63	3.13	12.50
	Linear	44.26	41.57	47.33	5.06	22.54	40.20	43.33	32.42	65.30	3.55	17.89	33.20
	Polynomial	44.47	39.09	51.57	6.29	24.15	41.09	30.11	18.53	80.25	3.55	9.91	16.54
	Exponential	45.19	40.60	50.94	6.29	24.15	41.09	44.98	34.27	65.44	3.55	17.82	33.37
	Regression	25.88	43.66	18.40	4.45	19.41	32.00	8.93	37.11	5.08	3.25	9.17	15.65
Attachment+Label	**Baseline-I**	14.83	12.66	17.90	0.77	3.84	7.68	20.49	16.64	26.68	0.75	3.77	7.53
	Baseline-II	28.33	16.50	100.00	1.00	5.00	10.01	36.19	22.09	100.00	1.00	5.00	10.00
	Independent	55.83	48.43	65.92	4.33	21.77	38.45	59.36	51.05	70.91	2.03	14.96	28.97
	Linear	55.41	50.62	61.20	4.36	21.36	38.40	59.66	52.26	69.51	2.03	15.06	29.02
	Polynomial	54.35	53.84	54.88	4.39	21.36	39.06	47.27	34.15	76.73	2.03	10.60	19.58
	Exponential	56.27	49.51	65.17	4.36	21.56	38.92	59.38	50.82	71.41	2.03	14.88	28.97
	Regression	54.21	44.63	69.02	4.39	21.82	37.98	56.76	41.57	89.47	2.37	14.76	29.74

		Slovene¶						Spanish¶					
	Measure	**F**	**P**	**R**	**EDI-1**	**EDI-5**	**EDI-10**	**F**	**P**	**R**	**EDI-1**	**EDI-5**	**EDI-10**
Label	**Baseline-I**	15.88	12.57	21.56	0.72	3.59	7.19	6.26	4.99	8.38	0.58	2.91	5.81
	Baseline-II	28.39	21.21	42.93	1.43	7.20	13.37	24.80	15.42	63.39	1.92	9.62	19.23
	Independent	54.75	45.13	69.59	5.72	19.97	34.64	38.02	29.78	52.56	1.83	15.92	39.35
Attachment	**Baseline-I**	23.97	19.55	30.95	0.87	4.34	8.67	16.44	13.57	20.87	0.82	4.09	8.18
	Baseline-II	39.93	27.16	75.36	1.48	9.61	16.10	37.87	27.93	58.79	2.05	10.25	20.51
	Independent	47.34	47.14	47.54	2.49	14.37	26.50	44.83	41.06	49.36	0.95	8.04	25.11
	Linear	47.73	38.63	62.46	2.44	14.10	25.64	53.43	45.93	63.88	4.09	19.43	35.33
	Polynomial	45.45	35.51	63.15	2.40	11.95	22.00	52.79	44.13	65.68	4.40	19.17	34.32
	Exponential	48.05	40.12	59.89	2.44	14.15	25.56	53.13	46.92	61.23	4.13	19.66	35.56
	Regression	37.63	24.70	78.97	2.67	11.77	20.18	20.24	51.29	12.61	3.33	14.93	28.54
Attachment+Label	**Baseline-I**	31.85	26.99	38.85	0.86	4.32	8.65	19.06	16.45	22.64	0.83	4.13	8.25
	Baseline-II	47.58	31.22	100.00	1.00	4.98	9.97	33.25	19.95	99.74	1.00	5.00	10.01
	Independent	65.12	63.55	66.77	3.20	15.86	27.69	52.12	50.29	54.10	3.87	16.52	30.45
	Linear	63.52	56.29	72.88	3.20	15.91	27.21	55.34	50.07	61.85	3.78	17.63	30.82
	Polynomial	61.43	52.55	73.93	3.20	15.62	26.41	55.68	48.68	65.02	3.58	16.99	31.32
	Exponential	64.28	59.99	69.22	3.20	15.91	27.61	55.93	48.72	65.64	3.85	17.31	30.67
	Regression	61.29	67.47	56.14	3.20	15.97	27.64	33.18	19.96	98.41	3.72	16.63	30.31

		Japanese¶						Portuguese¶					
	Measure	**F**	**P**	**R**	**EDI-1**	**EDI-5**	**EDI-10**	**F**	**P**	**R**	**EDI-1**	**EDI-5**	**EDI-10**
Label	**Baseline-I**	4.00	2.36	13.02	0.43	2.14	4.28	5.12	4.09	6.85	0.51	2.57	5.14
	Baseline-II	9.71	12.50	7.94	6.04	11.33	11.54	27.10	16.91	68.31	2.39	11.96	23.92
	Independent	25.99	15.63	77.14	1.98	9.89	19.77	50.78	52.90	48.82	10.89	38.93	57.97
Attachment	**Baseline-I**	3.58	2.12	11.38	0.37	1.87	3.73	9.67	7.88	12.54	0.67	3.37	6.74
	Baseline-II	13.98	8.01	54.77	5.86	13.42	12.83	31.87	26.44	40.09	2.34	11.71	23.42
	Independent	23.70	14.29	69.54	1.38	6.90	13.79	38.82	39.31	38.34	5.60	23.82	36.20
	Linear	25.82	16.43	60.31	3.31	16.57	30.47	47.04	40.72	55.69	6.84	26.26	43.28
	Polynomial	30.92	20.94	59.08	3.42	17.08	37.94	47.50	42.35	54.08	6.52	24.74	42.18
	Exponential	35.56	25.43	59.08	3.42	17.08	46.47	47.24	41.10	55.54	6.93	26.65	43.57
	Regression	14.76	9.19	37.54	1.99	10.77	18.21	20.90	11.68	99.27	4.99	18.90	35.13
Attachment+Label	**Baseline-I**	4.38	2.67	12.27	0.35	1.76	3.52	13.20	10.93	16.67	0.74	3.71	7.42
	Baseline-II	14.00	7.53	99.54	1.00	4.98	9.96	25.67	14.73	100.00	1.00	5.00	10.00
	Independent	45.45	32.35	76.39	4.00	19.99	44.92	49.26	42.88	57.87	5.77	21.75	34.73
	Linear	45.74	32.64	76.39	4.00	19.99	44.92	52.90	46.57	61.23	5.74	23.49	39.30
	Polynomial	38.28	26.19	71.06	4.00	19.99	41.07	52.08	46.31	59.49	5.82	23.54	39.30
	Exponential	45.45	32.35	76.39	4.00	19.99	44.92	52.58	46.76	60.07	5.84	23.27	39.61
	Regression	13.45	24.54	9.26	3.68	12.65	19.67	48.31	37.30	68.52	5.72	21.29	37.64

		Hungarian[#]						Italian[#]					
	Measure	**F**	**P**	**R**	**EDI-1**	**EDI-5**	**EDI-10**	**F**	**P**	**R**	**EDI-1**	**EDI-5**	**EDI-10**
Label	**Baseline-I**	11.37	9.13	15.06	0.68	3.41	6.82	8.09	6.29	11.33	0.59	2.93	5.86
	Baseline-II	32.85	23.44	54.83	2.80	14.01	22.36	22.52	14.98	45.34	9.32	46.58	15.81
	Independent	57.65	54.54	61.14	4.23	21.14	42.28	40.59	32.88	53.02	1.67	8.35	29.65
Attachment	**Baseline-I**	20.05	16.41	25.77	0.80	4.02	8.05	12.65	9.85	17.66	0.75	3.76	7.51
	Baseline-II	41.80	33.18	56.48	2.64	13.20	16.59	29.70	20.19	56.14	4.58	11.84	23.68
	Independent	50.54	47.29	54.27	2.77	13.87	27.75	40.19	33.88	49.40	1.37	6.84	25.23
	Linear	53.19	52.40	54.01	4.09	20.44	35.66	49.09	41.18	60.78	2.14	18.47	38.29
	Polynomial	46.89	43.01	51.54	4.09	20.44	30.90	48.21	38.81	63.62	2.10	19.15	39.50
	Exponential	52.78	52.96	52.60	4.09	20.44	35.59	48.01	38.39	64.07	2.10	19.15	39.49
	Regression	52.95	57.51	49.07	4.68	20.06	35.20	36.78	32.07	43.11	3.66	16.20	28.31
Attachment+Label	**Baseline-I**	23.97	19.70	30.59	0.82	4.10	8.20	14.41	11.61	19.01	0.70	3.49	6.98
	Baseline-II	38.75	24.03	100.00	1.00	5.00	10.00	28.50	16.62	100.00	1.00	4.99	9.98
	Independent	56.88	49.75	66.40	3.42	17.11	30.88	49.31	38.10	69.89	1.80	14.92	30.14
	Linear	56.73	51.75	62.78	3.42	17.11	30.91	53.88	46.34	64.34	1.80	14.57	32.10
	Polynomial	50.27	40.99	64.99	3.42	17.11	30.35	53.55	44.73	66.71	1.80	14.44	33.36
	Exponential	56.85	50.80	64.53	3.42	17.11	30.88	53.39	43.61	68.83	1.80	14.80	32.33
	Regression	47.85	33.40	84.36	3.87	16.71	31.06	14.08	36.27	8.74	2.12	11.27	22.48

		Greek[#]						Hindi[♠]					
	Measure	**F**	**P**	**R**	**EDI-1**	**EDI-5**	**EDI-10**	**F**	**P**	**R**	**EDI-1**	**EDI-5**	**EDI-10**
Label	**Baseline-I**	14.70	12.67	17.51	0.75	3.75	7.50	54.57	51.03	58.64	0.96	4.82	9.64
	Baseline-II	41.81	28.17	81.01	0.00	9.87	19.26	74.10	60.02	96.82	1.36	6.82	13.67
	Independent	54.60	51.42	58.20	4.94	20.73	36.41	78.02	74.06	82.42	1.79	8.50	16.62
Attachment	**Baseline-I**	17.31	15.31	19.91	0.80	4.00	8.01	39.55	34.53	46.27	1.12	5.62	11.23
	Baseline-II	45.21	32.82	72.58	2.61	13.01	26.03	48.62	32.22	98.99	2.01	10.07	20.13
	Independent	43.86	36.07	55.93	3.48	14.90	28.10	47.11	34.14	75.96	1.94	8.97	18.35
	Linear	52.40	45.84	61.15	3.73	17.50	31.80	50.17	33.61	98.94	1.89	8.71	17.87
	Polynomial	51.78	45.37	60.28	3.60	16.16	30.49	47.03	30.74	100.00	0.90	2.97	5.75
	Exponential	52.93	45.19	63.87	3.81	16.91	30.43	47.03	30.74	100.00	1.87	8.77	18.07
	Regression	45.44	44.56	46.35	3.95	15.64	28.57	50.84	36.28	84.92	2.20	9.72	19.18
Attachment+Label	**Baseline-I**	26.55	23.30	30.85	0.89	4.43	8.85	62.46	59.40	65.84	0.99	4.94	9.87
	Baseline-II	41.66	26.31	100.00	1.00	5.00	10.00	75.12	60.16	100.00	1.00	5.00	10.00
	Independent	61.24	53.77	71.12	3.72	14.82	28.03	77.05	67.49	89.77	1.58	7.86	15.00
	Linear	61.86	55.06	70.57	3.26	15.30	28.45	77.41	66.23	93.14	1.58	7.85	14.98
	Polynomial	61.33	54.56	70.02	3.34	15.02	27.29	77.78	66.10	94.46	1.58	7.50	14.09
	Exponential	62.13	54.51	72.23	3.48	14.96	27.89	77.01	67.33	89.95	1.58	7.86	14.99
	Regression	54.72	39.67	88.13	3.46	15.14	28.66	78.12	68.44	91.01	1.52	7.66	15.03

		Dutch[¶]						English[#]					
	Measure	**F**	**P**	**R**	**EDI-1**	**EDI-5**	**EDI-10**	**F**	**P**	**R**	**EDI-1**	**EDI-5**	**EDI-10**
Label	**Baseline-I**	12.45	10.11	16.21	0.62	3.11	6.23	9.78	8.31	11.87	0.69	3.43	6.87
	Baseline-II	32.73	31.50	34.07	2.54	12.68	25.36	27.93	20.99	41.74	3.86	17.94	20.08
	Independent	46.81	38.43	59.87	1.58	7.90	17.73	47.83	50.94	45.08	7.97	34.16	45.49
Attachment	**Baseline-I**	11.65	9.47	15.15	0.60	3.01	6.02	14.15	12.39	16.50	0.89	4.45	8.90
	Baseline-II	32.25	27.29	39.41	2.45	12.26	24.53	32.55	23.02	55.52	3.08	13.49	22.55
	Independent	44.49	37.94	53.76	1.62	8.10	18.13	46.68	47.02	46.34	6.16	24.53	38.88
	Linear	50.11	49.78	50.46	2.72	15.16	34.43	47.73	49.54	46.05	6.61	26.41	40.20
	Polynomial	50.66	43.18	61.28	2.72	16.07	34.11	47.65	48.73	46.63	6.61	25.17	38.28
	Exponential	51.03	45.72	57.74	2.72	15.28	34.88	48.73	45.67	52.22	6.61	26.56	40.34
	Regression	43.65	45.54	41.91	4.17	17.88	31.19	46.41	46.47	46.34	7.17	24.53	39.74
Attachment+Label	**Baseline-I**	16.94	14.04	21.33	0.72	3.62	7.24	14.60	12.80	16.99	0.83	4.14	8.28
	Baseline-II	32.49	19.39	100.00	1.00	5.00	10.00	26.53	15.29	100.00	2.90	14.18	23.49
	Independent	57.09	52.35	62.79	2.05	11.22	28.95	51.42	60.28	44.84	6.03	27.19	42.35
	Linear	56.37	49.93	64.73	2.05	11.23	29.10	52.78	52.24	53.33	6.03	26.34	40.94
	Polynomial	56.64	51.04	63.62	2.05	11.09	27.35	50.95	49.45	52.55	6.03	25.11	39.21
	Exponential	57.26	51.87	63.90	2.05	11.33	29.06	52.49	50.06	55.16	6.03	26.68	41.22
	Regression	19.74	59.81	11.82	3.13	15.62	30.35	36.02	44.36	30.33	6.27	20.52	29.93

		Chinese[#]						Danish[¶]					
	Measure	**F**	**P**	**R**	**EDI-1**	**EDI-5**	**EDI-10**	**F**	**P**	**R**	**EDI-1**	**EDI-5**	**EDI-10**
Label	**Baseline-I**	13.70	12.25	15.55	0.89	4.43	8.86	8.40	6.61	11.50	0.72	3.59	7.18
	Baseline-II	26.35	15.27	96.08	1.70	8.51	14.03	23.91	18.40	34.14	8.14	40.71	24.72
	Independent	33.05	24.12	52.52	0.59	2.93	5.87	39.43	31.06	53.99	0.56	2.81	30.60
Attachment	**Baseline-I**	8.79	8.21	9.47	0.68	3.40	6.80	10.58	8.40	14.31	0.82	4.09	8.18
	Baseline-II	23.40	13.54	86.04	1.49	7.47	15.03	25.65	16.33	59.73	0.00	0.00	23.98
	Independent	26.91	19.27	44.62	0.67	3.36	6.72	33.41	25.89	47.09	0.50	2.52	25.83
	Linear	34.85	29.92	41.73	4.46	19.35	30.82	46.55	39.79	56.07	7.47	28.73	45.45
	Polynomial	34.26	29.22	41.41	4.14	19.27	30.87	46.71	41.70	53.08	7.35	26.14	45.31
	Exponential	34.19	29.12	41.41	4.14	19.27	30.93	47.69	44.46	51.41	7.59	29.71	45.99
	Regression	21.52	12.06	99.84	1.73	12.16	24.31	18.58	10.29	95.67	3.69	20.02	34.66
Attachment+Label	**Baseline-I**	15.93	14.53	17.65	0.90	4.50	9.00	16.00	13.25	20.20	0.87	4.35	8.70
	Baseline-II	27.79	16.14	100.00	1.00	5.00	10.00	26.43	15.23	100.00	1.00	4.99	9.97
	Independent	44.54	40.19	49.94	4.65	17.48	30.26	55.91	50.00	63.41	4.79	22.04	38.69
	Linear	45.01	39.40	52.46	4.77	17.53	30.50	55.06	47.12	66.22	4.23	21.59	38.57
	Polynomial	42.64	33.68	58.10	3.67	17.28	27.02	55.87	50.36	62.74	4.87	22.42	40.05
	Exponential	44.62	39.94	50.54	4.65	17.53	30.38	56.24	49.70	64.76	5.01	22.30	40.06
	Regression	27.77	16.12	99.88	3.35	14.97	26.91	43.98	58.47	35.24	4.87	21.67	37.32

		Arabic[#]						Basque[#]					
	Measure	**F**	**P**	**R**	**EDI-1**	**EDI-5**	**EDI-10**	**F**	**P**	**R**	**EDI-1**	**EDI-5**	**EDI-10**
Label	**Baseline-I**	13.81	11.91	16.43	0.86	4.29	8.57	18.94	15.87	23.47	0.74	3.68	7.36
	Baseline-II	34.66	22.72	73.03	1.54	7.7	15.4	47.38	40.85	56.41	2.78	14.75	26.10
	Independent	**51.43**	41.85	66.71	**5.18**	**21.65**	**37.59**	53.11	45.49	63.80	3.51	9.99	24.51
Attachment	**Baseline-I**	17.92	15.42	21.4	0.92	4.59	9.19	19.74	16.50	24.58	0.75	3.74	7.48
	Baseline-II	39.9	28.27	67.79	3.12	9.88	19.77	45.43	30.76	86.87	1.13	12.66	23.19
	Independent	44.73	37.22	56.05	4.32	17.69	31.65	46.18	43.16	49.66	2.72	13.36	24.99
	Linear	**56.52**	49.27	66.28	5.04	21.88	38.83	45.26	34.99	64.06	3.28	14.37	25.01
	Polynomial	56.23	47.8	68.26	**5.39**	21.82	38.75	45.36	33.62	69.70	2.97	12.30	22.86
	Exponential	56.22	47.91	68.02	5.37	**21.99**	**39.02**	45.51	34.30	67.59	3.44	14.37	25.59
	Regression	42.01	31.96	61.28	3.37	15.49	26.94	40.95	33.06	53.79	3.19	11.00	19.65
Attachment+Label	**Baseline-I**	26.17	23.49	29.54	0.9	4.51	9.02	41.94	26.53	100.00	0.77	3.85	7.70
	Baseline-II	41.31	26.03	100	1.00	5.00	10.00	41.94	26.53	100.00	1.00	5.00	10.00
	Independent	57.27	47.33	72.49	**3.77**	14.69	25.38	53.57	46.79	62.66	3.20	11.49	22.34
	Linear	59.5	51.02	71.36	3.69	15.3	27.18	52.99	45.11	64.20	3.20	11.66	22.42
	Polynomial	59.47	51.25	70.84	**3.77**	**15.39**	27.14	52.52	46.78	59.86	3.20	11.63	21.81
	Exponential	**59.55**	51.21	71.14	3.69	15.38	**27.28**	53.13	45.17	64.48	3.20	11.77	22.20
	Regression	54.79	41.48	80.66	3.62	14	25.41	53.19	45.87	63.29	2.47	10.98	20.90

		Bulgarian[¶]						Catalan[#]					
	Measure	**F**	**P**	**R**	**EDI-1**	**EDI-5**	**EDI-10**	**F**	**P**	**R**	**EDI-1**	**EDI-5**	**EDI-10**
Label	**Baseline-I**	6.93	5.14	10.66	0.69	3.46	6.91	11.04	9.12	13.97	0.79	3.94	7.89
	Baseline-II	29.80	21.46	48.75	2.98	14.88	29.76	40.76	27.92	75.52	3.17	15.85	31.70
	Independent	35.43	24.82	61.90	0.30	1.50	26.52	66.55	70.60	62.93	8.65	39.52	62.15
Attachment	**Baseline-I**	7.13	5.06	12.06	0.71	3.55	7.10	8.33	6.49	11.66	0.76	3.79	7.58
	Baseline-II	22.24	15.37	40.19	2.00	9.98	19.97	25.85	16.40	61.07	2.53	12.64	22.77
	Independent	29.41	20.90	49.65	0.31	1.57	24.84	34.94	28.94	44.06	1.11	11.59	36.79
	Linear	48.47	41.67	57.92	9.05	36.22	58.45	50.15	45.00	56.64	8.32	35.42	53.87
	Polynomial	48.91	42.08	58.39	8.97	33.37	59.03	49.41	42.76	58.51	8.71	35.82	54.05
	Exponential	48.83	42.73	56.97	8.88	33.81	58.59	50.00	44.05	57.81	8.25	34.36	55.20
	Regression	39.22	36.04	43.03	6.78	27.55	46.68	15.78	8.56	100.00	7.95	27.77	45.72
Attachment+Label	**Baseline-I**	9.77	7.67	13.48	0.69	3.45	6.89	13.94	11.40	17.94	0.73	3.67	7.34
	Baseline-II	20.02	11.12	100.00	1.00	5.00	10.00	27.42	15.89	100.00	1.00	5.00	10.00
	Independent	51.12	44.37	60.30	7.34	26.85	45.17	46.98	47.04	46.93	5.91	19.50	35.78
	Linear	51.99	46.23	59.39	7.01	26.97	46.00	51.70	50.18	53.32	5.65	20.77	37.23
	Polynomial	53.22	44.16	66.97	6.78	26.30	46.14	51.68	52.31	51.07	5.66	21.10	36.40
	Exponential	53.24	47.19	61.06	7.31	26.94	46.20	51.86	50.35	53.45	5.65	20.64	37.21
	Regression	19.94	11.08	99.39	5.39	20.68	38.99	47.34	51.01	44.17	5.45	18.83	31.21

Appendix B

Feature Model

Tokens	Attributes					
	FORM	**LEMMA**	**CPOSTAG**	**POSTAG**	**FEATS**	**DEPREL**
S:Top	+	+	+	+	+	+
I:Next	+	+	+	+	+	
I:Next+1	+			+		
I:Next+2				+		
G:Leftmost dependent of Top			+	+		
G:Rightmost dependent of Top				+	+	
G:Next left (same-side) sibling of rightmost dependent of Top						+
G:Predecessor of Top in linear order of input string	+			+		
G:Leftmost dependent of Next	+		+	+		

Two Addtional Features:

- *Number of words* occuring between **Top** and **Next**, with discrete categories 0, 1, $2-4$ and $5-$.
- *Number of left dependents* of **Next**, with discrete categories 0, 1, $2-4$ and $5-$.

Appendix C

Best Ensemble Systems

Best ensemble systems during ensembling at inference time: Re-parsing Algorithms

	Best Combination		
Ensemble$_x$	**Unweighted**	**Weighted by CPOS of modifier**	**Weighted by LAS**
Ensemble$_2$	AE+AS		AE+CP
Ensemble$_3$	AE+P+2P		AE+CP+CNP
Ensemble$_4$	AE+AS+P+2P		AE+AS+CP+CNP
Ensemble$_5$	AE+CP+CNP+P+2P		
Ensemble$_6$	AE+AS+CP+CNP+P+2P		

	Weighted by label of dependency (LS)	**Weighted by label of dependency (LAS)**	**Weighted by dependency length**
Ensemble$_2$	AE+AS		
Ensemble$_3$	AE+CP+CNP		AE+AS+2P
Ensemble$_4$	AE+AS+CP+CNP		AE+AS+P+2P
Ensemble$_5$	AE+AS+CP+CNP+2P		
Ensemble$_6$	AE+AS+CP+CNP+P+2P		

Best ensemble systems during ensembling at inference time: Word-by-Word Voting

Ensemble$_x$	Best Combination		
	Unweighted	**Weighted by CPOS of modifier**	**Weighted by LAS**
Ensemble$_2$	AE+AS	AE+CP	
Ensemble$_3$	AE+AS+CNP	AE+CNP+P	AE+AS+P
Ensemble$_4$	AE+AS+CNP+P	AE+AS+CP+P	AE+AS+CNP+2P
Ensemble$_5$		AE+AS+CP+CNP+2P	
Ensemble$_6$		AE+AS+CP+CNP+P+2P	

	Weighted by label of dependency (LS)	**Weighted by label of dependency (LAS)**	**Weighted by dependency length**
Ensemble$_2$		AE+AS	
Ensemble$_3$		AE+AS+CNP	
Ensemble$_4$	AE+AS+CNP+P		AE+AS+CNP+2P
Ensemble$_5$	AE+AS+CP+CNP+2P	AE+AS+CP+CNP+P	AE+AS+CP+CNP+2P
Ensemble$_6$		AE+AS+CP+CNP+P+2P	

where,

Ensemble$_x$**:** denotes number of base parsers taken x at a time

AE: Arc-Eager (Nivre, 2003; Nivre, 2004)

AS: Arc-Standard (Nivre, 2003; Nivre, 2004)

CP: Covington Projective (Covington, 2001)

CNP: Covington Non-Projective (Covington, 2001)

P: Planar (Gómez-Rodríguez and Nivre, 2010)

2P: 2-Planar (Gómez-Rodríguez and Nivre, 2010)

Appendix D

Best PQE-based Ensemble Systems

	Best Combination	
Ensemble$_x$	**Re-parsing Algorithm**	**Word-by-Word Voting**
Ensemble$_2$	AE+AS	
Ensemble$_3$	AE+AS+CP	AE+AS+CNP
Ensemble$_4$	AE+AS+CP+CNP	AE+AS+CNP+2P
Ensemble$_5$	AE+CP+CNP+P+2P	AE+AS+CNP+P+2P
Ensemble$_6$	AE+AS+CP+CNP+P+2P	

where,

Ensemble$_x$**:** denotes best $\mathcal{PQE}$-based Ensemble system built by the combination of base parsers taken x at a time

AE: Arc-Eager (Nivre, 2003; Nivre, 2004)

AS: Arc-Standard (Nivre, 2003; Nivre, 2004)

CP: Covington Projective (Covington, 2001)

CNP: Covington Non-Projective (Covington, 2001)

P: Planar (Gómez-Rodríguez and Nivre, 2010)

2P: 2-Planar (Gómez-Rodríguez and Nivre, 2010)

Related Publications

1. Karan Singla, Aniruddha Tammewar, **Naman Jain**, and Sambhav Jain. 2012. Two-stage Approach for Hindi Dependency Parsing Using MaltParser. In *Proceedings of the Workshop on Machine Translation and Parsing in Indian Languages (MTPIL-2012), 24th International Conference on Computational Linguistics (COLING-2012), pages* $22 - 27$. *Mumbai, India.*

2. Sambhav Jain, **Naman Jain**, Riyaz Ahmad Bhat, Aniruddha Tammewar, and Dipti Misra Sharma. 2013. Exploring Semantic Information in Hindi WordNet for Hindi Dependency Parsing. In *Proceedings of the 6th International Joint Conference on Natural Language Processing (IJCNLP 2013), Nagoya, Japan.*

3. **Naman Jain**, Sambhav Jain, and Dipti Misra Sharma. 2013. In *Proceedings of The 13th International Conference on Parsing Technologies (IWPT 2013), pages* $141 - 146$*, Nara, Japan.*

4. **Naman Jain**, Sambhav Jain, and Dipti Misra Sharma. 2013. Minimizing Validation Effort for Treebank Expansion. In *The Twelfth Workshop on Treebanks and Linguistic Theories (TLT12), pages* 73*. Sofia, Bulgaria.*

5. Sambhav Jain, **Naman Jain**, Bhasha Agrawal, and Rajeev Sangal. 2015. Employing Oracle Confusion for Parse Quality Estimation. In *Computational Linguistics and Intelligent Text Processing, pages* $213 - 226$*, 16th International Conference, CICLing 2015, April 14-20, 2015, Proceedings, Cairo, Egypt.*

Other Publications

1. **Naman Jain** and Riyaz Ahmad Bhat. 2014. Language Identification in Code-Switching Scenario. In *Proceedings of The First Workshop on Computational Approaches to Code Switching, pages* $87 - 93$*, October 25, 2014, EMNLP-2014, Doha, Qatar*

2. Riyaz Ahmad Bhat, **Naman Jain**, Dipti Misra Sharma and Ashwini Vaidya, Martha Palmer, James Babani, Tafseer Ahmed and LTRC, IIIT-H. 2014. Adapting Predicate Frames for Urdu PropBanking. In *Proceedings of LT4CloseLang, EMNLP-2014, pages* 47*, Doha, Qatar.*

3. (Under Review) Riyaz Ahmad Bhat, Irshad Ahmad Bhat, **Naman Jain**, and Dipti Misra Sharma. 2015. Dependency Parsing of Hindi and Urdu Text. In *Transactions on Asian and Low-Resource Language Information Processing, TALLIP*.

References

[Agirre et al.2008] E. Agirre, T. Baldwin, and D. Martinez. 2008. Improving parsing and PP attachment performance with sense information. *Proceedings of ACL-08: HLT*, pages 317–325.

[Agirre et al.2011] E. Agirre, K. Bengoetxea, K. Gojenola, and J. Nivre. 2011. Improving dependency parsing with semantic classes. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics*, pages 699–703.

[Ambati et al.2009] Bharat Ram Ambati, Pujitha Gade, Chaitanya Gsk, and Samar Husain. 2009. Effect of minimal semantics on dependency parsing. In *Proceedings of the Student Research Workshop*, pages 1–5, Borovets, Bulgaria, September. Association for Computational Linguistics.

[Ambati et al.2010a] Bharat Ram Ambati, Samar Husain, Sambhav Jain, Dipti Misra Sharma, and Rajeev Sangal. 2010a. Two methods to incorporate local morphosyntactic features in Hindi dependency parsing. In *Proceedings of the NAACL HLT 2010 First Workshop on Statistical Parsing of Morphologically-Rich Languages*, pages 22–30. Association for Computational Linguistics.

[Ambati et al.2010b] B.R. Ambati, S. Husain, J. Nivre, and R. Sangal. 2010b. On the role of morphosyntactic features in Hindi dependency parsing. In *Proceedings of the NAACL HLT 2010 First Workshop on Statistical Parsing of Morphologically-Rich Languages*, pages 94–102. Association for Computational Linguistics.

[Annamaneni et al.2013] Narendra Annamaneni, Riyaz Ahmad Bhat, and Dipti Misra Sharma. 2013. Ensembling dependency parsers for treebank error detection. In *The Twelfth Workshop on Treebanks and Linguistic Theories (TLT12)*, page 1.

[Attardi and Dell'Orletta2009] Giuseppe Attardi and Felice Dell'Orletta. 2009. Reverse revision and linear tree combination for dependency parsing. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers*, pages 261–264. Association for Computational Linguistics.

[Banerjee and Pedersen2003] Satanjeev Banerjee and Ted Pedersen. 2003. Extended gloss overlaps as a measure of semantic relatedness. In *International Joint Conference on Artificial Intelligence*, volume 18, pages 805–810. LAWRENCE ERLBAUM ASSOCIATES LTD.

[Begum et al.2008] Rafiya Begum, Samar Husain, Arun Dhwaj, Dipti Misra Sharma, Lakshmi Bai, and Rajeev Sangal. 2008. Dependency annotation scheme for indian languages. In *IJCNLP*, pages 721–726. Citeseer.

[Begum et al.2011] Rafiya Begum, Karan Jindal, Ashish Jain, Samar Husain, and Dipti Misra Sharma. 2011. Identification of conjunct verbs in hindi and its effect on parsing accuracy. In *Computational Linguistics and Intelligent Text Processing*, pages 29–40. Springer.

[Bharati and Sangal1993] A. Bharati and R. Sangal. 1993. Parsing free word order languages in the Paninian framework. In *Proceedings of the 31st annual meeting on Association for Computational Linguistics*, pages 105–111. Association for Computational Linguistics.

[Bharati et al.1995] Akshar Bharati, Vineet Chaitanya, Rajeev Sangal, and KV Ramakrishnamacharyulu. 1995. *Natural language processing: a Paninian perspective*. Prentice-Hall of India New Delhi.

[Bharati et al.2007] A. Bharati, R. Sangal, and D.M. Sharma. 2007. Ssf: Shakti standard format guide. *Language Technologies Research Centre, International Institute of Information Technology, Hyderabad, India*, pages 1–25.

[Bharati et al.2008] A. Bharati, S. Husain, B. Ambati, S. Jain, D. Sharma, and R. Sangal. 2008. Two semantic features make all the difference in parsing accuracy. *Proc. of ICON*, 8.

[Bharati et al.2009a] Akshar Bharati, Samar Husain, Dipti Misra, and Rajeev Sangal. 2009a. Two stage constraint based hybrid approach to free word order language dependency parsing. In *Proceedings of the 11th International Conference on Parsing Technologies*, pages 77–80. Association for Computational Linguistics.

[Bharati et al.2009b] Akshar Bharati, Samar Husain, Meher Vijay, Kalyan Deepak, Dipti Misra Sharma, and Rajeev Sangal. 2009b. Constraint based hybrid approach to parsing indian languages. In *PACLIC*, pages 614–621.

[Bharati et al.2009c] Akshara Bharati, Dipti Misra Sharma, Samar Husain, Lakshmi Bai, Rafiya Begam, and Rajeev Sangal. 2009c. Anncorra: Treebanks for Indian Languages Guidelines for Annotating Hindi Treebank (version–2.0).

[Bhat et al.2012] Riyaz Ahmad Bhat, Sambhav Jain, and Dipti Misra Sharma. 2012. Experiments on dependency parsing of urdu. *Proceedings of TLT11*, pages 31–36.

[Bhat et al.2014a] Riyaz Ahmad Bhat, Rajesh Bhatt, Annahita Farudi, Prescott Klassen, Bhuvana Narasimhan, Martha Palmer, Owen Rambow, Dipti Misra Sharma, Ashwini Vaidya, Sri Ramagurumurthy Vishnu, et al. 2014a. The hindi/urdu treebank project. *Handbook of Linguistic Annotation*.

[Bhat et al.2014b] Riyaz Ahmad Bhat, Naman Jain, Dipti Misra Sharma, Ashwini Vaidya, Martha Palmer, James Babani, Tafseer Ahmed, and IIIT-H LTRC. 2014b. Adapting predicate frames for urdu propbanking. *LT4CloseLang 2014*, page 47.

[Bhatt et al.2009] R. Bhatt, B. Narasimhan, M. Palmer, O. Rambow, D.M. Sharma, and F. Xia. 2009. A multi-representational and multi-layered treebank for hindi/urdu. In *Proceedings of the Third Linguistic Annotation Workshop*, pages 186–189. Association for Computational Linguistics.

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

[Buchholz and Marsi2006] Sabine Buchholz and Erwin Marsi. 2006. CoNLL-X shared task on multilingual dependency parsing. In *Proceedings of the Tenth Conference on Computational Natural Language Learning*, pages 149–164. Association for Computational Linguistics.

[Chakrabarti et al.2007] Debasri Chakrabarti, Vaijayanthi Sarma, and Pushpak Bhattacharyya. 2007. Complex predicates in indian language wordnets. *Lexical Resources and Evaluation Journal*, 40(3-4).

[Chanev2005] Atanas Chanev. 2005. Portability of dependency parsing algorithms–an application for italian. In *Proceedings of the Fourth Workshop on Treebanks and Linguistic Theories (TLT)*, pages 29–40.

[Chu and Liu1965] Yoeng-Jin Chu and Tseng-Hong Liu. 1965. On the shortest arborescence of a directed graph. *Science Sinica*, 14(1396-1400):270.

[Cohen and others1960] Jacob Cohen et al. 1960. A coefficient of agreement for nominal scales. *Educational and psychological measurement*, 20(1):37–46.

[Collins et al.1999] Michael Collins, Lance Ramshaw, Jan Hajič, and Christoph Tillmann. 1999. A statistical parser for czech. In *Proceedings of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistics*, pages 505–512. Association for Computational Linguistics.

[Cortes and Vapnik1995] Corinna Cortes and Vladimir Vapnik. 1995. Support-vector networks. *Machine learning*, 20(3):273–297.

[Covington2001] Michael A Covington. 2001. A fundamental algorithm for dependency parsing. In *Proceedings of the 39th annual ACM southeast conference*, pages 95–102. Citeseer.

[Crammer and Singer2003] Koby Crammer and Yoram Singer. 2003. Ultraconservative online algorithms for multiclass problems. *The Journal of Machine Learning Research*, 3:951–991.

[Damljanovic et al.2010] Danica Damljanovic, Milan Agatonovic, and Hamish Cunningham. 2010. Natural language interfaces to ontologies: Combining syntactic analysis and ontology-based lookup through the user interaction. In *The Semantic Web: Research and Applications*, pages 106–120. Springer.

[De Marneffe and Manning2008] Marie-Catherine De Marneffe and Christopher D Manning. 2008. Stanford typed dependencies manual. Technical report, Technical report, Stanford University.

[De Marneffe et al.2014] Marie-Catherine De Marneffe, Timothy Dozat, Natalia Silveira, Katri Haverinen, Filip Ginter, Joakim Nivre, and Christopher D Manning. 2014. Universal stanford dependencies: A cross-linguistic typology. In *Proceedings of LREC*, pages 4585–4592.

[DHIVYA2011] R DHIVYA. 2011. Dependency parser for tamil using machine learning approach.

[Dong and Dong2000] Zhendong Dong and Qiang Dong. 2000. HowNet.

[Dongre and Mankar2011] Vikas J Dongre and Vijay H Mankar. 2011. A review of research on devnagari character recognition. *arXiv preprint arXiv:1101.2491*.

[Edmonds1967] Jack Edmonds. 1967. Optimum branchings. *Journal of Research of the National Bureau of Standards B*, 71(4):233–240.

[Fossum and Knight2009] Victoria Fossum and Kevin Knight. 2009. Combining constituent parsers. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers*, pages 253–256. Association for Computational Linguistics.

[Fujita et al.2007] Sanae Fujita, Francis Bond, Stephan Oepen, and Takaaki Tanaka. 2007. Exploiting Semantic Information for HPSG Parse Selection. In *ACL 2007 Workshop on Deep Linguistic Processing*, pages 25–32, Prague, Czech Republic, June. Association for Computational Linguistics.

[Galley and Manning2009] Michel Galley and Christopher D Manning. 2009. Quadratic-time dependency parsing for machine translation. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2-Volume 2*, pages 773–781. Association for Computational Linguistics.

[Galton1892] Francis Galton. 1892. *Finger prints*. Macmillan and Company.

[Gildea2001] Daniel Gildea. 2001. Corpus variation and parser performance. In *Proceedings of the 2001 Conference on Empirical Methods in Natural Language Processing*, pages 167–202.

[Gini1912] Corrado Gini. 1912. *Variabilità e mutabilità: contributo allo studio delle distribuzioni e delle relazioni statistiche.* Tipogr. di P. Cuppini.

[Goldberg and Elhadad2010] Yoav Goldberg and Michael Elhadad. 2010. Inspecting the structural biases of dependency parsing algorithms. In *Proceedings of the Fourteenth Conference on Computational Natural Language Learning*, pages 234–242. Association for Computational Linguistics.

[Gómez-Rodríguez and Nivre2010] Carlos Gómez-Rodríguez and Joakim Nivre. 2010. A transition-based parser for 2-planar dependency structures. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1492–1501. Association for Computational Linguistics.

[Grella2015] Matteo Grella. 2015. Notes about a more aware dependency parser. *arXiv preprint arXiv:1507.05630*.

[Hall et al.2007] Johan Hall, Jens Nilsson, Joakim Nivre, Gülşen Eryiğit, Beáta Megyesi, Mattias Nilsson, and Markus Saers. 2007. Single malt or blended? A study in multilingual parser optimization. *Proceedings of the CoNLL Shared Task of EMNLP-CoNLL 2007*, pages 933–939.

[Hall et al.2010] Johan Hall, Jens Nilsson, and Joakim Nivre. 2010. Single malt or blended? a study in multilingual parser optimization. In *Trends in Parsing Technology*, pages 19–33. Springer.

[Henderson and Brill1999] J. Henderson and E. Brill. 1999. Exploiting diversity in natural language processing: Combining parsers. In *in Proceedings of the Fourth Conference on Empirical Methods in Natural Language Processing*, pages 187–194.

[Hudson1984] Richard A Hudson. 1984. *Word grammar*. Blackwell Oxford.

[Husain and Agrawal2012] Samar Husain and Bhasha Agrawal. 2012. Analyzing Parser Errors to improve parsing accuracy and to inform tree banking decisions. *Linguistic Issues in Language Technology*, 7(1).

[Husain et al.2010] Samar Husain, Prashanth Mannem, Bharat Ram Ambati, and Phani Gadde. 2010. The icon-2010 tools contest on indian language dependency parsing. *Proceedings of ICON-2010 Tools Contest on Indian Language Dependency Parsing, ICON*, 10:1–8.

[Husain2009] Samar Husain. 2009. Dependency parsers for indian languages. *Proceedings of ICON09 NLP Tools Contest: Indian Language Dependency Parsing*.

[Husain2011] Samar Husain. 2011. *A Generalized Parsing Framework Based On Computational Paninian Grammar*. Ph.D. thesis, IIIT-Hyderabad.

[Hwa et al.2005] Rebecca Hwa, Philip Resnik, Amy Weinberg, Clara Cabezas, and Okan Kolak. 2005. Bootstrapping parsers via syntactic projection across parallel texts. *Natural language engineering*, 11(03):311–325.

[Hwa2004] Rebecca Hwa. 2004. Sample selection for statistical parsing. *Computational Linguistics*, 30(3):253–276.

[Jain and Agrawal2013] Sambhav Jain and Bhasha Agrawal. 2013. A dynamic confusion score for dependency arc labels. In *Proceedings of the Sixth International Joint Conference on Natural Language Processing*, pages 1237–1242, Nagoya, Japan, October. Asian Federation of Natural Language Processing.

[Jain et al.2013a] Naman Jain, Sambhav Jain, and Dipti Misra Sharma. 2013a. Effective parsing for human aided nlp systems. In *Proceedings of The 13th International Conference on Parsing Technologies (IWPT-2013)*, pages 141–146.

[Jain et al.2013b] Naman Jain, Sambhav Jain, and Dipti Misra Sharma. 2013b. Minimizing validation effort for treebank expansion. In *The Twelfth Workshop on Treebanks and Linguistic Theories (TLT12)*, page 73.

[Jain et al.2013c] Sambhav Jain, Naman Jain, Riyaz Bhat, Aniruddha Tammewar, and Dipti Misra Sharma. 2013c. Exploring semantic information in hindi wordnet for hindi dependency parsing. *Proceedings of the Sixth International Joint Conference on Natural Language Processing*.

[Jain et al.2014] Naman Jain, IIIT-H LTRC, and Riyaz Ahmad Bhat. 2014. Language identification in code-switching scenario. *EMNLP 2014*, page 87.

[Jain et al.2015] Sambhav Jain, Naman Jain, Bhasha Agrawal, and Rajeev Sangal. 2015. Employing oracle confusion for parse quality estimation. In *Computational Linguistics and Intelligent Text Processing*, pages 213–226. Springer.

[Kesidi et al.2010] Sruthilaya Reddy Kesidi, Prudhvi Kosaraju, Meher Vijay, and Samar Husain. 2010. A two stage constraint based hybrid dependency parser for telugu. *ICON*.

[Kingsbury et al.2002] Paul Kingsbury, Martha Palmer, and Mitch Marcus. 2002. Adding semantic annotation to the penn treebank. In *Proceedings of the Human Language Technology Conference*, pages 252–256. Citeseer.

[Klein and Manning2004] Dan Klein and Christopher D Manning. 2004. Corpus-based induction of syntactic structure: Models of dependency and constituency. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, page 478. Association for Computational Linguistics.

[Kolachina and Kolachina2012] Sudheer Kolachina and Prasanth Kolachina. 2012. Parsing any domain english text to conll dependencies. In *LREC*, pages 3873–3880.

[Kolachina et al.2010] Sudheer Kolachina, Prasanth Kolachina, Manish Agarwal, and Samar Husain. 2010. Experiments with malt parser for parsing indian languages. *Proc of ICON-2010 tools contest on Indian language dependency parsing. Kharagpur, India*.

[Kolachina2012] Sudheer Kolachina. 2012. *Non-local features in Syntactic Parsing*. Ph.D. thesis, International Institute of Information Technology Hyderabad.

[Kosaraju et al.2010] Prudhvi Kosaraju, Sruthilaya Reddy Kesidi, Vinay Bhargav Reddy Ainavolu, and Puneeth Kukkadapu. 2010. Experiments on indian language dependency parsing. *Proceedings of the ICON10 NLP Tools Contest: Indian Language Dependency Parsing*.

[Kosaraju et al.2012] Prudhvi Kosaraju, Samar Husain, Bharat Ram Ambati, Dipti Misra Sharma, and Rajeev Sangal. 2012. Intra-chunk dependency annotation: expanding Hindi inter-chunk annotated treebank. In *Proceedings of the Sixth Linguistic Annotation Workshop*, pages 49–56. Association for Computational Linguistics.

[Krivanek and Meurers2011] Julia Krivanek and Detmar Meurers. 2011. Comparing rule-based and data-driven dependency parsing of learner language. In *Proceedings of the Int. Conference on Dependency Linguistics (Depling 2011)*, pages 310–317.

[Kübler et al.2009] Sandra Kübler, Ryan McDonald, and Joakim Nivre. 2009. Dependency parsing. *Synthesis Lectures on Human Language Technologies*, 1(1):1–127.

[Kukkadapu et al.2012] Puneeth Kukkadapu, Deepak Kumar Malladi, and Aswarth Dara. 2012. Ensembling various dependency parsers: Adopting turbo parser for indian languages. In *24th International Conference on Computational Linguistics*, page 179.

[Lafferty et al.2001] John Lafferty, Andrew McCallum, and Fernando CN Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the International Conference on Machine Learning*, pages 282–289.

[MacKinlay et al.2012] Andrew MacKinlay, Rebecca Dridan, Diana McCarthy, and Timothy Baldwin. 2012. The effects of semantic annotations on precision parse ranking. In *Proceedings of the First Joint Conference on Lexical and Computational Semantics-Volume 1: Proceedings of the main conference and the shared task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation*, pages 228–236. Association for Computational Linguistics.

[Manning et al.2008] Christopher D Manning, Prabhakar Raghavan, and Hinrich Schütze. 2008. *Introduction to information retrieval*, volume 1. Cambridge University Press Cambridge.

[Martins et al.2008] André FT Martins, Dipanjan Das, Noah A Smith, and Eric P Xing. 2008. Stacking dependency parsers. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 157–166. Association for Computational Linguistics.

[Martins et al.2010] André FT Martins, Noah A Smith, Eric P Xing, Pedro MQ Aguiar, and Mário AT Figueiredo. 2010. Turbo parsers: Dependency parsing by approximate variational inference. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 34–44. Association for Computational Linguistics.

[Masica1993] Colin P Masica. 1993. *The Indo-Aryan Languages*. Cambridge University Press.

[McClosky et al.2006a] David McClosky, Eugene Charniak, and Mark Johnson. 2006a. Effective self-training for parsing. In *Proceedings of the main conference on human language technology conference of the North American Chapter of the Association of Computational Linguistics*, pages 152–159. Association for Computational Linguistics.

[McClosky et al.2006b] David McClosky, Eugene Charniak, and Mark Johnson. 2006b. Reranking and self-training for parser adaptation. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 337–344. Association for Computational Linguistics.

[McDonald and Nivre2007] Ryan T McDonald and Joakim Nivre. 2007. Characterizing the errors of data-driven dependency parsing models. In *EMNLP-CoNLL*, pages 122–131.

[McDonald and Pereira2006] Ryan McDonald and Fernando Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *Proceedings of EACL*, volume 6, pages 81–88.

[McDonald et al.2005] Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan 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*, pages 523–530. Association for Computational Linguistics.

[McDonald et al.2013] Ryan T McDonald, Joakim Nivre, Yvonne Quirmbach-Brundage, Yoav Goldberg, Dipanjan Das, Kuzman Ganchev, Keith B Hall, Slav Petrov, Hao Zhang, Oscar Täckström, et al. 2013. Universal dependency annotation for multilingual parsing. In *ACL (2)*, pages 92–97. Citeseer.

[Meena and Prabhakar2007] Arun Meena and TV Prabhakar. 2007. Sentence level sentiment analysis in the presence of conjuncts using linguistic analysis. In *Advances in Information Retrieval*, pages 573–580. Springer.

[Mei and Gao1996] Jia-ju Mei and Yunqi Gao. 1996. Tongyi cilin (a chinese thesaurus). *China: Shanghai Lexicographical Publishing House*.

[Mejer and Crammer2010] Avihai Mejer and Koby Crammer. 2010. Confidence in structured-prediction using confidence-weighted models. In *Proceedings of the 2010 conference on empirical methods in natural language processing*, pages 971–981. Association for Computational Linguistics.

[Mejer and Crammer2012] Avihai Mejer and Koby Crammer. 2012. Are you sure?: confidence in prediction of dependency tree edges. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 573–576. Association for Computational Linguistics.

[Melćuk1988] I.A. Melćuk. 1988. *Dependency syntax: theory and practice*. State University of New York Press.

[Miller1995] George A Miller. 1995. WordNet: a lexical database for English. *Communications of the ACM*, 38(11):39–41.

[Mohanan1994] Tara Mohanan. 1994. *Argument structure in Hindi*. Stanford Univ Center for the Study.

[Mohanan1997] Tara Mohanan. 1997. Multidimensionality of representation: Nv complex predicates in hindi. *Complex predicates*, pages 431–471.

[Montemagni et al.2003] Simonetta Montemagni, Francesco Barsotti, Marco Battista, Nicoletta Calzolari, Ornella Corazzari, Alessandro Lenci, Antonio Zampolli, Francesca Fanciulli, Maria Massetani, Remo Raffaelli, et al. 2003. Building the Italian syntactic-semantic treebank. In *Treebanks*, pages 189–210. Springer.

[Narayan et al.2002] Dipak Narayan, Debasri Chakrabarty, Prabhakar Pande, and Pushpak Bhattacharyya. 2002. An experience in building the indo wordnet-a wordnet for hindi. In *First International Conference on Global WordNet, Mysore, India*.

[Nilsson and Nivre2008] Jens Nilsson and Joakim Nivre. 2008. Malteval: an evaluation and visualization tool for dependency parsing. In *LREC*.

[Nilsson et al.2007] Jens Nilsson, Sebastian Riedel, and Deniz Yuret. 2007. The CoNLL 2007 shared task on dependency parsing. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL*, pages 915–932. sn.

[Nivre and McDonald2008] Joakim Nivre and Ryan T McDonald. 2008. Integrating graph-based and transition-based dependency parsers. In *ACL*, pages 950–958.

[Nivre and Nilsson2005] Joakim Nivre and Jens Nilsson. 2005. Pseudo-projective dependency parsing. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 99–106. Association for Computational Linguistics.

[Nivre et al.2004] Joakim Nivre, Johan Hall, and Jens Nilsson. 2004. Memory-based dependency parsing. In *Proceedings of CoNLL*, pages 49–56.

[Nivre et al.2006] Joakim Nivre, Johan Hall, Jens Nilsson, Gülşen Eryiğit, and Svetoslav Marinov. 2006. Labeled pseudo-projective dependency parsing with support vector machines. In *Proceedings of the Tenth Conference on Computational Natural Language Learning*, pages 221–225. Association for Computational Linguistics.

[Nivre et al.2007] Joakim Nivre, Johan Hall, Jens Nilsson, Atanas Chanev, Gülsen Eryigit, Sandra Kübler, Svetoslav Marinov, and Erwin Marsi. 2007. Maltparser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13(02):95–135.

[Nivre et al.2009] Joakim Nivre, Marco Kuhlmann, and Johan Hall. 2009. An improved oracle for dependency parsing with online reordering. In *Proceedings of the 11th international conference on parsing technologies*, pages 73–76. Association for Computational Linguistics.

[Nivre2003] Joakim Nivre. 2003. An efficient algorithm for projective dependency parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*. Citeseer.

[Nivre2004] Joakim Nivre. 2004. Incrementality in deterministic dependency parsing. In *Proceedings of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together*, pages 50–57. Association for Computational Linguistics.

[Nivre2009] Joakim Nivre. 2009. Non-projective dependency parsing in expected linear time. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1-Volume 1*, pages 351–359. Association for Computational Linguistics.

[Osborne and Baldridge2004] Miles Osborne and Jason Baldridge. 2004. Ensemble-based active learning for parse selection. In *HLT-NAACL*, pages 89–96.

[Øvrelid and Nivre2007] Lilja Øvrelid and Joakim Nivre. 2007. When word order and part-of-speech tags are not enough–Swedish dependency parsing with rich linguistic features. In *Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP)*, pages 447–451.

[Owczarzak2009] Karolina Owczarzak. 2009. Depeval (summ): dependency-based evaluation for automatic summaries. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1-Volume 1*, pages 190–198. Association for Computational Linguistics.

[Palmer et al.2009] M. Palmer, R. Bhatt, B. Narasimhan, O. Rambow, D.M. Sharma, and F. Xia. 2009. Hindi Syntax: Annotating Dependency, Lexical Predicate-Argument Structure, and Phrase Structure. In *The 7th International Conference on Natural Language Processing*, pages 14–17.

[Petrov et al.2010] Slav Petrov, Pi-Chuan Chang, Michael Ringgaard, and Hiyan Alshawi. 2010. Uptraining for accurate deterministic question parsing. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 705–713. Association for Computational Linguistics.

[Popel et al.2011] Martin Popel, David Mareček, Nathan Green, and Zdeněk Žabokrtskỳ. 2011. Influence of parser choice on dependency-based MT. In *Proceedings of the Sixth Workshop on Statistical Machine Translation*, pages 433–439. Association for Computational Linguistics.

[Rajendran et al.2010] S Rajendran, G Shivapratap, V Dhanlakshmi, and KP Soman. 2010. Building a wordnet for dravidian languages. In *Proceedings of the Global WordNet Conference (GWC 10)*.

[Ramasamy and Žabokrtskỳ2011] Loganathan Ramasamy and Zdeněk Žabokrtskỳ. 2011. Tamil dependency parsing: results using rule based and corpus based approaches. In *Computational Linguistics and Intelligent Text Processing*, pages 82–95. Springer.

[Rao et al.1998] Durgesh Rao, Pushpak Bhattacharya, and Radhika Mamidi. 1998. Natural language generation for english to hindi human-aided machine translation. *VIVEK-BOMBAY-*, 11:32–39.

[Reichart and Rappoport2007] Roi Reichart and Ari Rappoport. 2007. An ensemble method for selection of high quality parses. In *ACL*, volume 7, pages 408–415.

[Sagae and Lavie2006] Kenji Sagae and Alon Lavie. 2006. Parser combination by reparsing. In *Proceedings of the Human Language Technology Conference of the NAACL, Companion Volume: Short Papers*, pages 129–132. Association for Computational Linguistics.

[Sagae and Tsujii2007] Kenji Sagae and Jun'ichi Tsujii. 2007. Dependency parsing and domain adaptation with lr models and parser ensembles. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, pages 1044–1050.

[Sagae2010] Kenji Sagae. 2010. Self-training without reranking for parser domain adaptation and its impact on semantic role labeling. In *Proceedings of the 2010 Workshop on Domain Adaptation for Natural Language Processing*, pages 37–44. Association for Computational Linguistics.

[Sassano and Kurohashi2010] Manabu Sassano and Sadao Kurohashi. 2010. Using smaller constituents rather than sentences in active learning for japanese dependency parsing. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 356–365.

[Sharma et al.2012] Dipti Misra Sharma, Prashanth Mannem, Joseph vanGenabith, Sobha Lalitha Devi, Radhika Mamidi, and Ranjani Parthasarathi, editors. 2012. *Proceedings of the Workshop on Machine Translation and Parsing in Indian Languages*. The COLING 2012 Organizing Committee, Mumbai, India, December.

[Shieber1987] Stuart M Shieber. 1987. *Evidence against the context-freeness of natural language*. Springer.

[Singla et al.2012] Karan Singla, Aniruddha Tammewar, Naman Jain, and Sambhav Jain. 2012. Two-stage Approach for Hindi Dependency Parsing Using MaltParser. *Training*, 12041(268,093):22–27.

[Smeeton1985] Nigel C Smeeton. 1985. Early history of the kappa statistic.

[Smith and Eisner2009] David A Smith and Jason Eisner. 2009. Parser adaptation and projection with quasi-synchronous grammar features. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 2-Volume 2*, pages 822–831. Association for Computational Linguistics.

[Søgaard2011] Anders Søgaard. 2011. Data point selection for cross-language adaptation of dependency parsers. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers-Volume 2*, pages 682–686. Association for Computational Linguistics.

[Srivastava and Bhat2013] Rishabh Srivastava and Riyaz Ahmad Bhat. 2013. Transliteration systems across indian languages using parallel corpora. *Sponsors: National Science Council, Executive Yuan, ROC Institute of Linguistics, Academia Sinica NCCU Office of Research and Development*, page 390.

[Steedman et al.2003] Mark Steedman, Miles Osborne, Anoop Sarkar, Stephen Clark, Rebecca Hwa, Julia Hockenmaier, Paul Ruhlen, Steven Baker, and Jeremiah Crim. 2003. Bootstrapping statistical parsers from small datasets. In *Proceedings of the tenth conference on European chapter of the Association for Computational Linguistics-Volume 1*, pages 331–338. Association for Computational Linguistics.

[Surdeanu and Manning2010] Mihai Surdeanu and Christopher D Manning. 2010. Ensemble models for dependency parsing: cheap and good? In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 649–652. Association for Computational Linguistics.

[Sureka et al.2014] K Sureka, KG Srinivasagan, and S Suganthi. 2014. An efficiency dependency parser using hybrid approach for tamil language. *arXiv preprint arXiv:1403.6381*.

[Tang et al.2002] Min Tang, Xiaoqiang Luo, and Salim Roukos. 2002. Active learning for statistical natural language parsing. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 120–127. Association for Computational Linguistics.

[Tesnière1959] Lucien Tesnière. 1959. *Eléments de syntaxe structurale*. Librairie C. Klincksieck.

[Tsarfaty et al.2013] Reut Tsarfaty, Djamé Seddah, Sandra Kübler, and Joakim Nivre. 2013. Parsing Morphologically Rich Languages: Introduction to the Special Issue. *Computational Linguistics*, 39(1):15–22.

[Wang et al.2007] Mengqiu Wang, Noah A Smith, and Teruko Mitamura. 2007. What is the jeopardy model? a quasi-synchronous grammar for qa. *EMNLP-CoNLL*, 7:22–32.

[Wann et al.2009] Stephen Wann, Mark Dras, Robert Dale, and Cécile Paris. 2009. Improving grammaticality in statistical sentence generation: Introducing a dependency spanning tree algorithm with an argument satisfaction model. In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*, pages 852–860. Association for Computational Linguistics.

[Wu et al.2004] Ting-Fan Wu, Chih-Jen Lin, and Ruby C Weng. 2004. Probability estimates for multi-class classification by pairwise coupling. *The Journal of Machine Learning Research*, 5:975–1005.

[Xia et al.2008] Fei Xia, Owen Rambow, Rajesh Bhatt, Martha Palmer, and Dipti Misra Sharma. 2008. Towards a multi-representational treebank. *LOT Occasional Series*, 12:159–170.

[Xiong et al.2005] Deyi Xiong, Shuanglong Li, Qun Liu, Shouxun Lin, and Yueliang Qian. 2005. Parsing the penn chinese treebank with semantic knowledge. In *Natural Language Processing–IJCNLP 2005*, pages 70–81. Springer.

[Xu et al.2009] Peng Xu, Jaeho Kang, Michael Ringgaard, and Franz Och. 2009. Using a dependency parser to improve smt for subject-object-verb languages. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 245–253. Association for Computational Linguistics.

[Xue et al.2002] Nianwen Xue, Fu-Dong Chiou, and Martha Palmer. 2002. Building a large-scale annotated Chinese corpus. In *Proceedings of the 19th international conference on Computational linguistics-Volume 1*, pages 1–8. Association for Computational Linguistics.

[Zeman and Resnik2008] Daniel Zeman and Philip Resnik. 2008. Cross-language parser adaptation between related languages. In *IJCNLP*, pages 35–42.

[Zeman and Žabokrtskỳ2005] Daniel Zeman and Zdeněk Žabokrtskỳ. 2005. Improving parsing accuracy by combining diverse dependency parsers. In *Proceedings of the Ninth International Workshop on Parsing Technology*, pages 171–178. Association for Computational Linguistics.

[Zeman2009] Daniel Zeman. 2009. Maximum spanning malt: Hiring worlds leading dependency parsers to plant indian trees. *ICON09 NLP Tools Contest: Indian Language Dependency Parsing. Hyderabad, India*.

[Zhang and Clark2008] Yue Zhang and Stephen Clark. 2008. A tale of two parsers: investigating and combining graph-based and transition-based dependency parsing using beam-search. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 562–571. Association for Computational Linguistics.

[Zhang and Wang2009] Yi Zhang and Rui Wang. 2009. Cross-domain dependency parsing using a deep linguistic grammar. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1-Volume 1*, pages 378–386. Association for Computational Linguistics.

[Zhang et al.2009] Hui Zhang, Min Zhang, Chew Lim Tan, and Haizhou Li. 2009. K-best combination of syntactic parsers. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 3-Volume 3*, pages 1552–1560. Association for Computational Linguistics.