Error Detection and Dependency Parsing

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

Master of Science by Research
in
Computer Science and Engineering

by

Bhasha Agrawal
201207691
bhasha.agrawal@research.iiit.ac.in

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

International Institute of Information Technology
Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled "Error Detection and Dependency Parsing" by Bhasha Agrawal, has been carried out under my supervision and is not submitted elsewhere for a degree.

Date

Adviser: Prof. Dipti Misra Sharma

To my Family..

Acknowledgements

First of all, I would like to thank my adviser Prof. Dipti Misra Sharma for having confidence in me and supporting me to explore the field and work according to my liking. Her suggestions boosted me to work hard and complete my work. A special mention to Dr. Manish Shrivastava who in spite of not being an official guide, introduced me to the field of linguistics and guided me through my difficulties by his valuable suggestions. This work would not have been possible without him. I'm highly grateful to Dr. Samar Husain, who helped me in building my basics, clarifying the doubts and inspired me to pursue my MS.

A special thanks to my seniors, Sambhav Jain and Riyaz Ahmad Bhat for their valuable feedback and discussions. They patiently answered all my queries when I was a novice. I would also like to thank Naman Jain and Aniruddha Tammewar, who have been a great set of people to work with. I am grateful to the vibrant and stimulating LTRC lab environment, which instigated several active discussions to sprang up new ideas and scrutinizing the existing ones. Some deserve special mention : Maaz Anwar Nomani, Sruti Rallapalli, Ankush Soni, Rishabh Srivastava, Kunal Sachdeva, Arpita Batra, Rahul Sharma, Himanshu Sharma, Praveen Dakwale, Urmi Ghosh, Himani Choudhry and Karan Singla. I also thank Rambabu, Srinivas Rao, Satish, Kumarswamy and Lakshmi Narayan for making administrative matters run smoothly.

A special acknowledgment to my family and all my friends for their unconditional love and support. They constantly motivated me towards my goal.

Abstract

Syntactic parsing, a major component of natural language processing, involves understanding the structure of a sentence as per given rules. Despite of enormous research in this field, syntactic parsing for Indian languages is still not at par with other languages like English. The reason being, most of the Indian languages are morphologically rich, free word-order languages (Mor-FWO) and parsing Mor-FWO languages is a challenging task. Apart from this, parsing requires sizable annotated resource called a treebank for machine learning algorithms. Treebank is a text corpus with manually annotated grammatical analysis of each sentence. Unavailability of resources for these languages is a major reason why we do not have good parsers for these languages. In this work, we have proposed methods which aid in semi-automatic development of good quality treebanks for Indian languages by proposing a method to automatically detect potential errors in treebank. With the help of this approach, manual validators can validate treebanks in less time and effort.

Since the availability of Penn Treebank [44], treebanks have played a crucial role in building automatic natural language processing tools for various languages. In particular, treebanks have helped in building robust and efficient syntactic parsers. The availability of syntactic parsers is critical for the further processing of a sentence, e.g. semantic analysis [31]. We, therefore, explore methods to develop treebanks for resource poor languages because we might have huge unannotated data available with us but labeled data is very less for many languages. Annotation being an expensive task, if we obtain a tool which helps in building quality treebanks, development of statistical parsers would be affected a lot.

Statistical parsers are generally trained on data from a singe domain and evaluation is also generally done on the same domain. While when put into natural language applications, we expect input data from any random domain. In such cases, parsers behave clumsily and their performance degrades drastically from what was evaluated. This might be because, while training parsers on data from one domain, parser might learn domain specific features which might not be valid for other domains. So, while working on these two aspects of grammatical analysis of sentences in Indian languages, I also ventured into exploring whether existing parsers can be made robust for new domains.

Contents

Chapter Page

1 Introduction 1
1.1 Key Contributions of the Thesis . . . 1
1.2 Outline . . . 2

2 Dependency Parsing 4
2.1 Introduction . . . 4
2.2 Approaches . . . 5
2.2.1 MaltParser (A Transition-based Dependency Parser) . . . 7
2.2.2 MSTParser (A Graph-based Dependency Parser) . . . 10
2.3 Comparison of MaltParser and MSTPraser . . . 11
2.4 Description of Treebank . . . 14

3 An Automatic Approach to Treebank Error Detection Using a Dependency Parser 15
3.1 Introduction . . . 15
3.2 Related Work . . . 16
3.3 Previous Approach . . . 17
3.4 Our Approach . . . 18
3.4.1 Using Parser Output to Detect Potential Errors . . . 19
3.5 Data Used in Our Approach . . . 23
3.6 Comparison with Other Techniques . . . 24
3.7 Effect of correcting flagged errors . . . 24
3.8 Discussion . . . 26
3.9 Error detection in parser output . . . 26

4 Active Learning employed in building treebanks for resource poor languages 28
4.1 Introduction . . . 28
4.2 Related Work . . . 29
4.3 Data Used . . . 29
4.4 Methodology . . . 30
4.5 Results and Discussion . . . 30

5 Domain Adaptation **33**
5.1 Introduction 33
5.2 Related Work 33
5.3 Our Approach 34
5.4 Data Used 35
5.5 Results and Discussion 35

6 Conclusion and Future Directions **40**
6.1 Future Work 41

List of Figures

Figure Page

2.1 Dependency Structure and Phrase Structure for the English sentence "Mohan reads a book" . . . 4
2.2 Dependency Tree for an Hindi Sentence - "Nitish Kumar Bihar ke agale mukhya-mantri honge." . . . 5
2.3 Sample Hindi Sentence - "Nitish Kumar Bihar ke agale mukhyamantri honge." . 6
2.4 Sample Hindi Sentence with chunk boundaries - "Nitish Kumar Bihar ke agale mukhyamantri honge." . . . 7
2.5 Inter chunk dependency tree for sentence in figure 2.4 . . . 8
2.6 Dependency graph for an English sentence. Adapted from [42]. . . . 9
2.7 Arc-eager transition sequence for the English sentence in Figure 2.6. Adapted from [42]. . . . 10

3.1 Approach proposed by Ambati et al., 2011 ([3]). (Adapted from ([3]).) . . . 17
3.2 Approach proposed by Agarwal et al., 2012 ([2]). (Adapted from ([2]).) . . . 18
3.3 Algorithm used for error detection using our approach . . . 19

4.1 Active learning results for Urdu . . . 30
4.2 Active learning results for Hindi . . . 31
4.3 Active learning results for Telugu . . . 31

5.1 Gold tree(a), Auto tree(b), Auto+cluster Id tree(c) [65] . . . 37
5.2 Gold sentence from Box-office data . . . 38
5.3 Parsed sentence from Box-office data . . . 38

List of Tables

Table Page

2.1 Comparison of MaltParser and MSTParser - 1 11
2.2 Comparison of MaltParser and MSTParser - 2 12
2.3 Comparison of MaltParser and MSTParser - 3 13
2.4 Comparison of MaltParser and MSTParser - 4 13

3.1 Results of validation using parser output 23
3.2 Comparison of present approach with Agarwal et al., 2012([2]) 25
3.3 Comparison of overall (attachment+labeling) results of our approach with previous one Agarwal et al., 2012([2]) 25
3.4 Comparison of parser trained on annotated data with that trained on validated data 25

5.1 Domain adaptation results 36
5.2 Structural differences among various domains 39

Chapter 1

Introduction

Syntactic parsing has always been core component of many natural language applications. Therefore, performance of parsers is a major concern for syntactic parsing research community. In spite of substantial research in the field of syntactic parsing, state-of-the-art parsers are not at par with the industrial standards. As mentioned in abstract, some of the major parsing challenges are:

- **Deficiency of labeled resources:** We have limited quantity of annotated data for many Indian languages which is insufficient for proper learning of statistical tools.
- **Unavailability of good quality treebanks for resource poor languages:** Even though the treeebanks are available, they sometimes contain errors. If the treebank used for preparing the parser is erroneous, those errors will be learned by the parser and it will always make same errors.
- **Training data bias:** Another well known complication which hinders the performance of parsers is domain confinement of parsers. The training data used for data driven or statistical parsers is generally chosen from finite domains but real time natural language applications can expect data from any domain. Parsers in such cases sometimes learn domain specific features and their performance varies immensely across domains.

We in this work, therefore, concentrate to address these problems. We explore methods for automatic error detection in an existing treebank and propose an approach to assist treebank development.

1.1 Key Contributions of the Thesis

Following are the key contributions of this thesis-

- **Automatic error detection in Treebank:**

  Treebanks play an important role in the development of various natural language processing tools. Among other things, they provide crucial language-specific patterns that are exploited by various machine learning techniques. Quality control in any treebanking project is therefore extremely important. Manual validation of the treebank is one of the steps that is generally necessary to ensure good annotation quality. Needless to say, manual validation requires a lot of human time and effort. In this work, we present an automatic approach which helps in detecting potential errors in a treebank. We use a dependency parser to detect such errors. By using this tool, validators can validate a treebank in less time and with reduced human effort.

- **Active learning for natural language parsing:**

  Active learning is a special case of semi-supervised machine learning where a learning algorithm tries to explores methods to collect training data on its own and does not depend on random samples or manual collection of samples for training. In this work, we have used active learning for statistical dependency parsing where new sentences are ranked based on the highest uncertainty an existing parser exerts during parsing while selecting samples. It helps in developing a high quality treebank for resource poor languages with reduced effort, time and cost which in turn results in development of natural language processing tools for the same.

Domain confinement of parsers is another well known concern. Statistical parsers are trained on data compiled from finite domains, but NLP applications are often domain unbound and do freely accept data from any common domain. When we have a parser trained on data of certain domain and we want to use it for parsing data from another domain, biggest problem faced is the difference in vocabulary which makes parsing difficult. Here, we try to explore the problem of domain difference between training and testing data by bridging the gap between diverse vocabulary of different domains. The results are not so promising, so nothing can be said in concrete.

1.2 Outline

This thesis is grouped into following chapters with their brief outlines:

- **Chapter 2**
  In this chapter, we give a brief outline of dependency parsing and phrase structure parsing

and describe two data driven dependency parsers, MaltParser and MSTParser. We then compare performances of these parsers for various features for Hindi.

- **Chapter 3**
  This chapter introduces a method for automatically detecting errors in an existing dependency treebank with the help of a dependency parser. This kind of error detection might aid validators in quick correction and validation of treebanks.

- **Chapter 4**
  In this chapter, we have applied active learning for statistical dependency parsing. Active learning tries to explore methods to interactively collect training data on its own and does not depend on manual collection of training samples. This approach helps in building good quality treebanks for resource poor languages.

- **Chapter 5**
  In this chapter, we attempt the problem of domain adaptation in the area of dependency parsing. We explored whether parsers can be made domain independent by decreasing the vocabulary gap between various domains. We also point structural differences among sentences from various domains.

- **Chapter 6**
  This chapter concludes this dissertation with a highlight of the work done. It also mentions the areas which might be explored and extended for advancements in the existing state of dependency parsing.

Chapter 2

Dependency Parsing

2.1 Introduction

Syntactic parsing, a fundamental task in natural language processing, is the procedure of analyzing a text, made of a succession of tokens or words, to decide its syntactic structure as per a given formal grammar. Formalisms or grammars contain the structural constraints of the language. Dependency grammar and phrase structure grammar are two such formalisms. Phrase structure grammar breaks up the sentence into constituents(phrases), which are then broken into smaller constituents (Figure 2.1 (a)). Dependency grammar draws links connecting individual words (Figure 2.1 (b)).

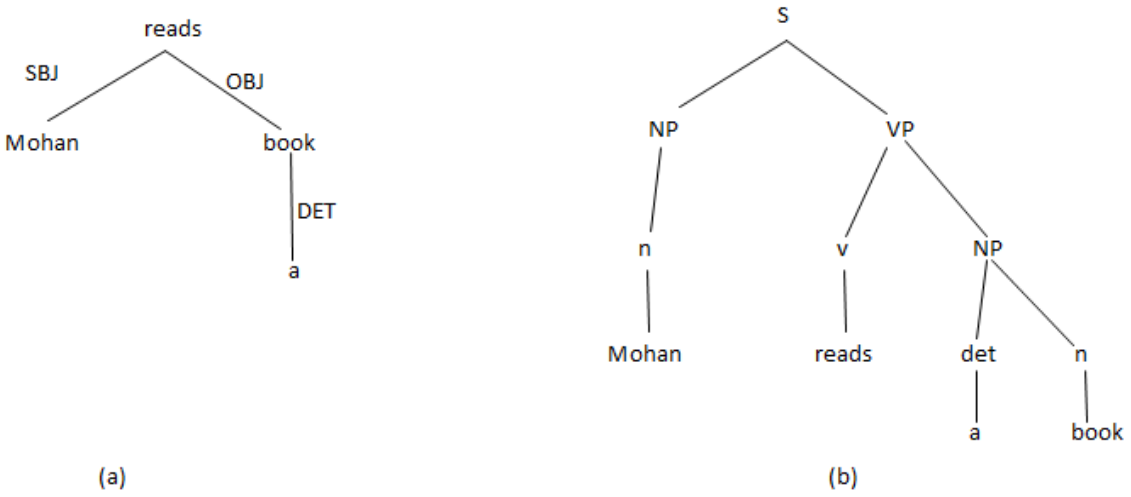

Figure 2.1: Dependency Structure and Phrase Structure for the English sentence “Mohan reads a book”

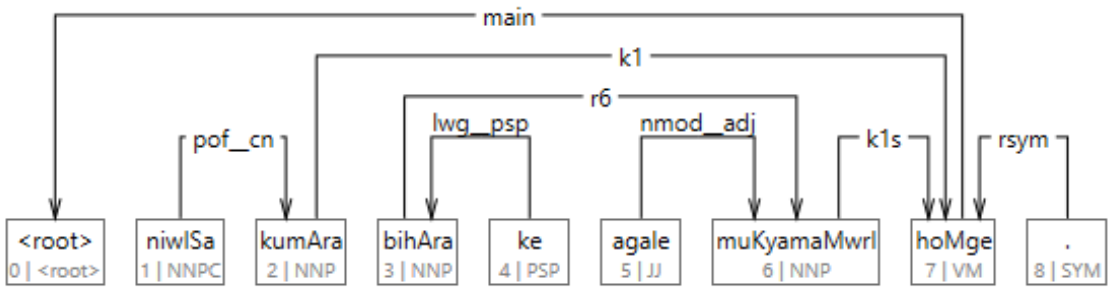

Figure 2.2: Dependency Tree for an Hindi Sentence - "Nitish Kumar Bihar ke agale mukhya-mantri honge."

It has been suggested that morphologically rich free word order languages (MoR-FWO) can be handled better using the dependency based framework than the constituency (phrase structure) based one ([9], [47], [33] and [62]). Hence, we have followed dependency grammar for our entire work.

Figure 2.2 shows dependency tree for a Hindi Sentence and 2.3 shows the corresponding sentence with all features marked. In full parsed tree, dependency relations are marked between all the words in the sentence.

Previous efforts on parsing MoR-FWO languages include [5] who explored two-stage approach of parsing Hindi. It divided the data into two parts namely, interChunks and intraChunks[1]. The inter chunk part of the data contains only dependency relations between chunk heads of the sentences while the intra chunk data has the dependency relations between the tokens of a chunk. Since, the dependency relation labels for interChunk and intraChunk are disjoint, this approach not only reduces search space for parsing but also reduces effort of annotators.

So, we will be working on inter-chunk dependency parsing for our experiments. Figure 2.4 shows the sentence in figure 2.2 with chunk boundaries marked and figure 2.5 shows inter-chunk dependency parsed tree for the same.

2.2 Approaches

Dependency parsing can be typically divided into grammar driven and data driven(statistical) dependency parsing. Grammar driven dependency parsing eliminates parses which violate

[1]A typical chunk consists of a single content word surrounded by a constellation of function words [1]. Chunks are normally taken to be a 'correlated group of words'.

```
1	niwISa	NNPC	<fs af='niwISa,n,m,sg,3,d,0,0' name='niwISa' posn='10'
drel='pof__cn:kumAra' chunkType='child:NP'>

2	kumAra	NNP	<fs af='kumAra,n,m,sg,3,d,0,0' name='kumAra' posn='20' chunkId='NP'
drel='k1:hoMge' chunkType='head:NP'>

3	bihAra	NNP	<fs af='bihAra,n,m,sg,3,o,0_kA,0' name='bihAra' posn='30' chunkId='NP2'
drel='r6:muKyamaMwrI' vpos='vib_2' chunkType='head:NP2'>

4	ke	PSP	<fs af='kA,psp,m,sg,,o,,' name='ke' posn='40' drel='lwg__psp:bihAra'
chunkType='child:NP2'>

5	agale	JJ	<fs af='agalA,adj,m,sg,,o,,' name='agale' posn='50'
drel='nmod__adj:muKyamaMwrI' chunkType='child:NP3'>

6	muKyamaMwrI	NNP	<fs af='muKyamaMwrI,n,m,sg,3,d,0,0' name='muKyamaMwrI'
posn='60' chunkId='NP3' drel='k1s:hoMge' chunkType='head:NP3'>

7	hoMge	VM	<fs af='ho, v,m,sg,3h,,gA,gA' name='hoMge' posn='70' chunkId='VGF'
chunkType='head:VGF' voicetype='active' stype='declarative'>

8	.	SYM	<fs af='.,punc,,,,,,' name='.' posn='80' chunkId='BLK' drel='rsym:hoMge'
chunkType='head:BLK'>
```

Figure 2.3: Sample Hindi Sentence - “Nitish Kumar Bihar ke agale mukhyamantri honge.”

grammatical constraints until only valid parses are left. Parsing, in this approach, is a kind of constraint satisfaction problem([11], [12]). Data driven dependency parsers differ from grammar driven dependency parsers in the sense that they use machine learning to learn a probabilistic model from training data for parsing ([53]).

For natural languages, data driven dependency parsing is preferred over grammar driven dependency parsing because grammar driven parsing might put constraints which are not satisfied by any of the parses proposed for a sentence or there might be more than one parses satisfying the constraints. Data driven parsers generally give single best output for all the inputs thus fixing these problems faced by grammar driven parsers.

Following section introduces two data driven dependency parsers.

```
1   ((      NP          <fs af='kumAra,n,m,sg,3,d,0,0'  head=kumAra posn=20 name='NP' drel='k1:VGF'>
1.1 niwISa              NNPC    <fs af='niwISa,n,m,sg,3,d,0,0'  name=niwISa posn=10>
1.2 kumAra              NNP     <fs af='kumAra,n,m,sg,3,d,0,0'  name=kumAra posn=20>
))
2   ((      NP          <fs af='bihAra,n,m,sg,3,o,0_kA,0'  head=bihAra posn=30 vpos="vib1_2"
    name='NP2' drel='r6:NP3'>
2.1 bihAra NNP          <fs af='bihAra,n,m,sg,3,o,0,0'  name=bihAra posn=30>
))
3   ((      NP          <fs af='muKyamaMwrI,n,m,sg,3,d,0,0'  head=muKyamaMwrI posn=60 name='NP3'
    drel='k1s:VGF'>
3.1 agale   JJ          <fs af='agalA,adj,m,sg,,o,,'  name=agale posn=50>
3.2 muKyamaMwrI NNP     <fs af='muKyamaMwrI,n,m,sg,3,d,0,0'  name=muKyamaMwrI
    posn=60>
))
4   ((      VGF         <fs af='ho,v,m,sg,3h,,gA,gA'  head=hoMge stype=declarative posn=70
    voicetype=active name='VGF'>
4.1 hoMge               VM      <fs af='ho,v,m,sg,3h,,gA,gA'  name=hoMge posn=70>
))
5   ((      BLK         <fs af='.,punc,,,,,,'  head=. posn=80 name='BLK' drel='rsym:VGF'>
5.1 .       SYM         <fs af='.,punc,,,,,,'  name=. posn=80>
))
```

Figure 2.4: Sample Hindi Sentence with chunk boundaries - "Nitish Kumar Bihar ke agale mukhyamantri honge."

2.2.1 MaltParser (A Transition-based Dependency Parser)

[2] MaltParser implements the transition-based approach to dependency parsing, which has two essential components:

- A transition system for mapping sentences to dependency trees
- A classifier for predicting the next transition for every possible system configuration

Given these two components, dependency parsing can be realized as deterministic search through the transition system, guided by the classifier. With this technique, parsing can be performed in linear time for projective dependency trees and quadratic time for arbitrary (possibly non-projective) trees.

MaltParser comes with a number of built-in transition systems [52]. We studied the Nivre's arc-eager [32] and arc-standard. For Morphologically Rich Languages (MRL), arc-eager out-

[2] This section is based on [42]

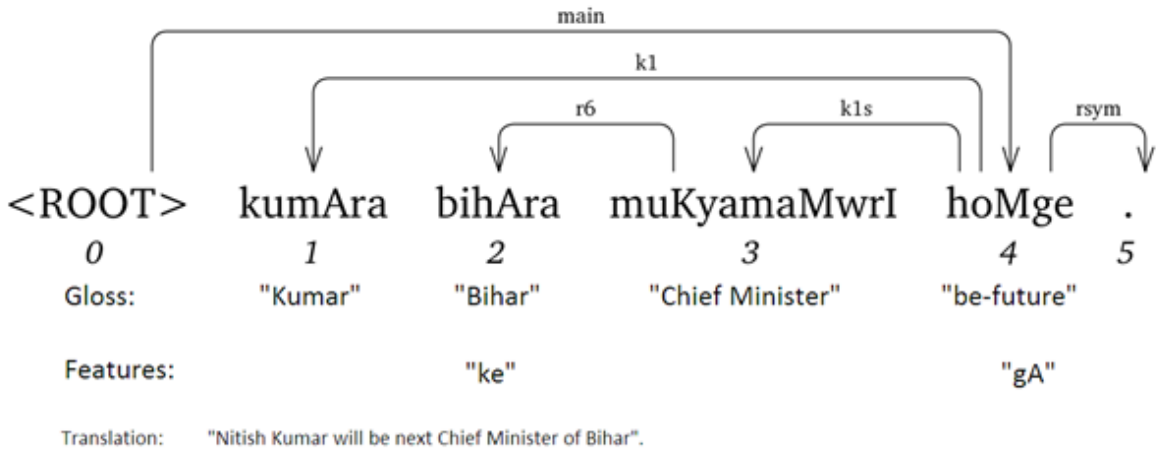

Figure 2.5: Inter chunk dependency tree for sentence in figure 2.4

performs arc-standard [10]. So, we will be using Nivre's arc-eager algorithm for our work. A brief description of arc-eager and arc-standard algorithm is as follows:

The arc-eager algorithm builds a labeled dependency tree over the input sentence in one left-to-right pass. A configuration in the arc-eager projective system contains a stack holding partially processed tokens, an input buffer containing the remaining tokens, and a set of arcs representing the partially built dependency tree. There are four possible transitions (where top is the token on top of the stack and next is the next token in the input buffer):

- **LEFT-ARC (r):** Add an arc labeled r from next to top; pop the stack.
- **RIGHT-ARC (r):** Add an arc labeled r from top to next; push next onto the stack.
- **REDUCE:** Pop the stack.
- **SHIFT:** Push next onto the stack.

Arc-standard has three possible transitions:

- **LEFT-ARC (r):** Add an arc labeled r from next to top; pop the stack.
- **RIGHT-ARC (r):** Add an arc labeled r from top to next; pop the stack and replace next by top.
- **SHIFT:** Push next onto the stack.

Arc-standard is different from arc-eager in the fact that it does not contain "Reduce" transition and "Right Arc" of arc-standard is different from that of arc-eager. Rest two transitions:

"Left Arc" and "Shift" are same in both. Arc-standard does not require "Reduce" transition because it reduces in the transition "Right Arc" only. It pops top of the stack and places it in the place of next input token, i.e., it indirectly removes the next input token simultaneously while making the arc where as arc-eager pushes next input token on the stack and reduces it later. The disadvantage of arc-standard of simultaneously reduction can be seen in the following example:

If the dependency graph is : $\mathbf{x} \rightarrow \mathbf{y} \rightarrow \mathbf{z}$

1. **Transitions made by arc-eager:**

Shift	(x, y|z, $)
Right Arc	(y|x, z, [x → y)]
Right Arc	(z|y|x, $, [x → y ,y → z])

2. **Transitions made by arc-standard:**

Shift	(x, y|z, $)
Right Arc	($, x|z, [x → y)]
Right Arc	(x, z, [x → y])

Now if we use arc-standard algorithm for this example, the parser will stuck in this condition as arc between x and z is not possible while y has been popped. So, it can't make any further transitions and cannot reach termination configuration and given sentence cannot be parsed.

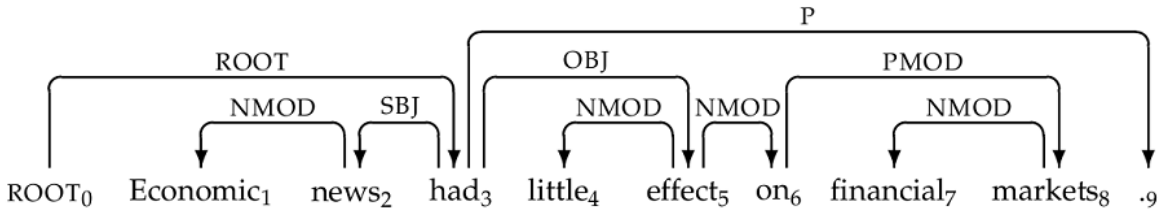

Figure 2.6: Dependency graph for an English sentence. Adapted from [42].

Consider an example English sentence "Economic news had little effect on financial markets." taken from the Penn Treebank. Figure 2.6 shows the dependency tree for this sentence.

Figure 2.7 shows the the sequence of steps for parsing the example sentence described in Figure 2.6 using arc-eager algorithm.

Albeit this system can only draw projective dependency trees, the fact that the trees are labeled allows non-projective dependencies to be captured using the pseudo-projective parsing technique [54].

Transition	Configuration			
	([0],	[1, . . . , 9],	∅	)
SHIFT ⟹	([0, 1],	[2, . . . , 9],	∅	)
LEFT-ARC$_{\text{NMOD}}$ ⟹	([0],	[2, . . . , 9],	$A_1 = \{(2, \text{NMOD}, 1)\}$	)
SHIFT ⟹	([0, 2],	[3, . . . , 9],	A_1	)
LEFT-ARC$_{\text{SBJ}}$ ⟹	([0],	[3, . . . , 9],	$A_2 = A_1 \cup \{(3, \text{SBJ}, 2)\}$	)
RIGHT-ARC$^e_{\text{ROOT}}$ ⟹	([0, 3],	[4, . . . , 9],	$A_3 = A_2 \cup \{(0, \text{ROOT}, 3)\}$	)
SHIFT ⟹	([0, 3, 4],	[5, . . . , 9],	A_3	)
LEFT-ARC$_{\text{NMOD}}$ ⟹	([0, 3],	[5, . . . , 9],	$A_4 = A_3 \cup \{(5, \text{NMOD}, 4)\}$	)
RIGHT-ARC$^e_{\text{OBJ}}$ ⟹	([0, 3, 5],	[6, . . . , 9],	$A_5 = A_4 \cup \{(3, \text{OBJ}, 5)\}$	)
RIGHT-ARC$^e_{\text{NMOD}}$ ⟹	([0, . . . , 6],	[7, 8, 9],	$A_6 = A_5 \cup \{(5, \text{NMOD}, 6)\}$	)
SHIFT ⟹	([0, . . . , 7],	[8, 9],	A_6	)
LEFT-ARC$_{\text{NMOD}}$ ⟹	([0, . . . 6],	[8, 9],	$A_7 = A_6 \cup \{(8, \text{NMOD}, 7)\}$	)
RIGHT-ARC$^e_{\text{PMOD}}$ ⟹	([0, . . . , 8],	[9],	$A_8 = A_7 \cup \{(6, \text{PMOD}, 8)\}$	)
REDUCE ⟹	([0, . . . , 6],	[9],	A_8	)
REDUCE ⟹	([0, 3, 5],	[9],	A_8	)
REDUCE ⟹	([0, 3],	[9],	A_8	)
RIGHT-ARC$^e_{\text{P}}$ ⟹	([0, 3, 9],	[ ],	$A_9 = A_8 \cup \{(3, \text{P}, 9)\}$	)

Figure 2.7: Arc-eager transition sequence for the English sentence in Figure 2.6. Adapted from [42].

2.2.2 MSTParser (A Graph-based Dependency Parser)

[3] MST Parser [46] builds a complete graph including all the words in the input sentence with weights $w(i, j)$ assigned to each edge based on prior contexts. Each word is treated as a node in this graph. Then, it finds out the maximum spanning tree from the built graph and renders it as the output parse.

MSTParser uses the Chu-Liu-Edmonds algorithm ([19], [24]). The algorithm has each vertex in the graph and greedily selects an incoming edge with highest weight. If a tree results, it must be the maximum spanning tree. If not, there must be a cycle. The procedure identifies a cycle and contracts it into a single vertex and recalculates edge weights going into and out of the cycle. It can be shown that a maximum spanning tree on the contracted graph is equivalent to a maximum spanning tree in the original graph [29]. Hence, the algorithm can recursively call itself on the new graph. Using Chu-Liu-Edmonds algorithm, non-projective dependency trees can be generated. However, many languages that allow non-projectivity are still primarily projective. MSTParser uses Eisner algorithm [25] for projective dependency parsing.

[3] This section is based on [46]

	MALT Parser				**MST Parser**			
	Interchunk		Full Parsed Tree		Interchunk		Full Parsed Tree	
	With basic features	TAM/ vibhakti added	With basic features	TAM/ vibhakti added	With basic features	TAM/ vibhakti added	With basic features	TAM/ vibhakti added
LAS(%)	72.70	83.43	88.85	90.53	69.87	81.61	83.29	90.05
UAS(%)	86.02	92.28	95.20	95.33	85.75	91.84	93.15	95.39
LS(%)	74.93	86.40	90.69	92.54	72.18	84.59	84.81	91.84

Table 2.1: Comparison of MaltParser and MSTParser - 1

2.3 Comparison of MaltParser and MSTPraser

We compared performance of these two parsers for Hindi treebank (described in Section 2.4). For comparison, LAS[4] , UAS[5] and LA[6] were calculated. These scores have been used everywhere in this dissertation for performance computation. We used standard '*eval*07.*pl*′ script[7] to for this evaluation.

Table 2.1 shows performance of MaltParser and MSTParser for interchunk dependency parsing and dependency parsing for full parsed tree. For both the types of parsing, we explored affect of TAM (Tense, Aspect and Modality) and vibhakti over basic features(Part Of Speech, Lemma, Word, Chunk Tag, etc.). Hindi being a free word order language, it's very difficult to find the correct dependency relation from basic features. For ex:

(Ram ne) (Mohan ko) (pustak) (dii).

This sentence can be re-written as:

(Mohan ko) (Ram ne) (pustak) (dii).

In the above examples, if we ignore vibhakti and only consider chunk heads, we get following sentences:

(Ram) (Mohan) (pustak) (dii).

(Mohan) (Ram) (pustak) (dii).

In this case, it becomes almost impossible to decide the dependency relations because both 'Ram' and 'Mohan' have same basic features. But the vibhaktis 'ne' and 'ko' tell us that 'Ram'

[4]LAS - Labeled Attachment score (Percentage of words that are assigned both correct head and correct label

[5]Unlabeled Attachment Score (Percentage of words that are assigned correct head)

[6]Labeled Accuracy (Percentage of words that are assigned correct label)

[7]eval07.pl : The official evaluation script for CoNLL 2007 shared task on dependency parsing. Available for download at : `http://pauillac.inria.fr/~seddah/eval07.pl`

	MALT Parser					
	Interchunk			Full Parsed Tree		
	With basic features	TAM/ vibhakti added	GNP and case added	With basic features	TAM/ vibhakti added	GNP and case added
LAS(%)	72.70	83.43	82.52	88.85	90.53	90.46
UAS(%)	86.02	92.28	92.14	95.20	95.33	95.29
LS(%)	74.93	86.40	85.38	90.69	92.54	92.48

Table 2.2: Comparison of MaltParser and MSTParser - 2

is doer while 'Mohan' is beneficiary. This example clearly shows that adding TAM and vibhakti features help in disambiguation of attachments and labels and hence benefit parsing.

We then explored affect of adding GNP (gender, number and person) and case features. These features could only be explored for MaltParser as the model size was too big in case of MSTParser and could not be completed. The results are shown in Table 2.2. These features don't help in parsing and they reduced parsing accuracy.

In table 2.3, we computed head to head accuracy (dependency relations between chunk heads) directly from chunk heads, from local word grouping and full parsed tree. This was again explored for both MaltParser and MSTParser.

Finally we computed full sentence parsing accuracy obtained after expanding inter-chunk data from expander and learning inter & intra chunk dependencies separately. The dependency relation labels for interChunk and intraChunk are disjoint. This approach helps in avoiding intraChunk relations to be marked as interChunk relations and vice-versa. These results are shown in Table 2.4.

In all of these experiments, MaltParser outperformed MSTParser. Apart from that, MaltParser models are comparatively smaller and faster than that of MSTParser. Hence, we have used MaltParser for all of our work in this thesis. From the comparisons shown in these tables, we can see that best results are obtained when parser is trained for inter chunk dependencies and intra chunk relations are marked using expander. We will be working on inter chunk dependencies in all of our experiments as intra chunk are simple to be marked and can be learned separately.

	MALT Parser			**MST Parser**		
	Head to Head accuracy from			Head to Head accuracy from		
	Inter-chunk data (TAM/vibhakti as feature)	Local word grouping(TAM/vibhakti as feature)	Full Parsed Tree	Inter-chunk data (TAM/vibhakti as feature)	Local word grouping(TAM/vibhakti as feature)	Full Parsed Tree
LAS(%)	83.43	83.46	79.40	81.61	80.90	72.86
UAS(%)	92.28	92.16	91.23	91.84	91.10	87.26
LS(%)	86.40	86.48	82.66	84.59	84.21	75.70

Table 2.3: Comparison of MaltParser and MSTParser - 3

	MALT Parser		**MST Parser**	
	Total accuracy after running expander on inter chunk data (with TAM/vibhakti)	Total accuracy from learning inter and intra-chunk dependencies (with TAM/vibhakti)	Total accuracy after running expander on inter chunk data (with TAM/vibhakti)	Total accuracy from learning inter and intra-chunk dependencies (with TAM/vibhakti)
LAS(%)	91.43	91.04	89.51	88.83
UAS(%)	96.62	95.78	95.40	95.35
LS(%)	93.53	92.65	91.26	90.63

Table 2.4: Comparison of MaltParser and MSTParser - 4

2.4 Description of Treebank

Hindi dependency treebank (HDTB) is being developed under the Hindi-Urdu treebank(HUTB) project ([15], [55], [71]). A part of Hindi Treebank (HDTB ver-0.51) has been released for Hindi Dependency Parsing shared task, MTPIL, COLING 2012 ([61]).

It is a multi-representational and multi-layered dependency treebank with morphological, part-of-speech(POS) and dependency labels annotated according to computational paninian framework ([9]) on sentences from news and heritage domain corpus. Annotation begins with the tokenisation of plain text. The tokens thus obtained are then annotated with morphological and POS(part of speech) tag information. After word level annotations, the next step is the grouping of correlated words into chunks. The tagging guidelines followed for POS-tagging and chunking are proposed by Bharati et al., 2006([13]). The final step in the pipeline is the manual dependency annotation. Only the inter-chunk dependencies are marked leaving the dependencies between the words in the chunk unspecified because the intra-chunk dependencies are observed to be highly predictive given the head of a chunk. Thus, in a dependency tree in HDTB, each node is a chunk and the edge represents relations between the connected nodes marked with dependency labels. The dependency annotations are followed by the annotation scheme put forth by Begum et al., 2008([8]). The scheme is based on a grammatical formalism known as Computational Paninian Grammar (CPG) first proposed by Bharati et al., 1995([9]). The dependency tag-set based on the scheme consists of about 43 labels, which can be broadly grouped into two categories: karaka labels and non-karaka labels [14]. The karaka labels are based on the notion of 'karaka', defined as the role played by a participant in an action. The karaka labels, k1-5,7, are centered around the core meaning of a verb. Non-karaka labels like r6, nmod, nmod relc etc. are, however, used to mark relations between nouns (genitives), nouns and their modifiers (adjectival modification, relativization) etc. In addition to these labels there are some special labels like pof and ccof, they do not mark dependency relations, but are used to handle special constructions like conjunct verbs, coordinating conjunctions etc.

Since the dependency annotation is manual, it might contain errors which need validation. We therefore introduce an approach in the next chapter to detect potential errors in an annotated treebank. This approach will assist manual validation and lead towards building high quality treebanks.

Chapter 3

An Automatic Approach to Treebank Error Detection Using a Dependency Parser

3.1 Introduction

Treebanks are an essential resource for developing solutions to various NLP related problems. As a treebank provides important linguistic knowledge, its quality is of extreme importance. The treebank used for experiments in this work is a part of the new multi-layered and multi-representational Hindi Treebanking project [15], [26] (described in previous chapter) whose target is 450k words. As is generally the case, most of the annotation process is either completely manual or semi-automatic, and the validation is completely manual.

The process of developing a treebank involves manual or semi-automatic annotations of linguistic information. A semi-automatic procedure involves annotating the grammatical information using relevant NLP tools (eg. POS taggers, chunker, parsers, etc.). The output of these tools is then manually checked and corrected. Both these procedures may leave errors in the treebank on the first attempt. Therefore, there is usually another step called validation in which these errors are manually identified and corrected. But the validation process is as time-consuming as annotation process. As the data has already been annotated carefully, we need tools that can supplement the validators' task with a view of making the overall task fast, without compromising on reliability. Using such a tool, a validator can directly go to error instances and correct them. So, the tool must have high recall. A human validator can reject unintuitive errors without much effort, so one can compromise a little bit on precision.

In this chapter, we propose such a strategy. The identified errors are classified under two categories (attachment error and labeling error) for the benefit of the validators, who may choose to correct a specific type of error at one time. Our method to identify such errors involves using a dependency parser.

If we can demonstrate considerable benefit/relevance of such a strategy, it might then be a good idea to incorporate it in a treebank development pipeline.

3.2 Related Work

Error detection has become an active topic of research over the last decade due to increase in demand for high quality annotated corpora. Some earlier methods employed for error detection in syntactic annotation (mainly POS and chunk), are by Eskin et al., 2000([26]) and Van et al., 2000([68]). Volokh et al., 2011([70]) employed a method similar to Van et al., 2000([68]), to automatically correct errors at the dependency level. Their main idea was to reproduce the gold standard data using MSTParser, MALTParser and MDParser. They use the outputs of any two parsers to detect error nodes, and mark a node as an error if tags predicted by the two parsers differ from the one in the gold data. In case the predicted tags are different, they use the output of the third parser to check if the tag predicted by the third parser matches with any of the tags predicted by the other two parsers. If the tag matches with any of the other two tags, changes are made in the gold data. Using large corpora, Van et al., 2004([69]) and De et al., 2009([23]) employed error mining techniques. Other efforts towards detection of annotation errors in treebanks include Kaljurand et al., 2004([40]) and Kordoni et al, 2003([41]). Most of the aforementioned techniques work well with large corpora where the word-type frequencies are high. Thus, none of them account for data sparse conditions except for De et al., 2009([23]). Moreover, the techniques employed by Van et al., 2004([69]), De et al., 2009([23]) and Volokh et al., 2011([70]) rely on the output of a reliable state-of-the-art parser.

Work by Ambati et al., 2010([4]), Ambati et al., 2011([3]) and Agarwal et al., 2012([2]) detect errors in a Hindi dependency treebank. They used a combination of both rule-based and hybrid systems to detect annotation errors. A rule-based system contributes towards increasing the precision of the system; it uses robust rules formed using annotation guidelines and the CPG framework, whereas a hybrid system is a combination of a statistical module with rule-based post-processing. The statistical module detects potential errors from the treebank and the rule-based post-processing module prunes out false positives, with the help of robust and efficient rules to increase the precision of the overall system. Next section describes their approach in detail.

3.3 Previous Approach

Ambati et al., 2011 ([3]) used a combination of a rule-based system(uses rules formed using annotation guidelines and framework) and a hybrid system(combination of a statistical system with rule based post processing) to detect the errors. The statistical module detects potential errors from the treebank and the rule-based post-processing module prunes the false positives using rules formed which increases precision of the overall system. "Rule-Based Approach" and "Rule-based post-processing" modules have separate goals. Goal of "Rule-Based Approach" is to detect errors using high precision rules, Whereas goal for the later, is to prune the false positives given by the statistical approach. The statistical system of the hybrid system first extracts the contextual features which help in identifying the correct tag. For example, at the dependency level, apart from node and its parent features, sibling and child features with their respective dependency labels are very useful in predicting the correct dependency label. It then creates a model using maximum entropy classification algorithm and gold training data, tests the system on the testing data and obtains the probabilities for all the possible dependency tags. It then feeds the 1st best and 2nd best tag probabilities into its algorithm to detect errors.

The kind of features used in a statistical system are very important. Thus, one should carefully choose features which help statistical systems to learn effectively. Ambati et al., 2011 ([3]) used parent (POS and CHUNK tag), sibling and child (POS, CHUNK and dependency label) features apart from the node's feature (POS, CHUNK). Figure 3.1 explains their approach.

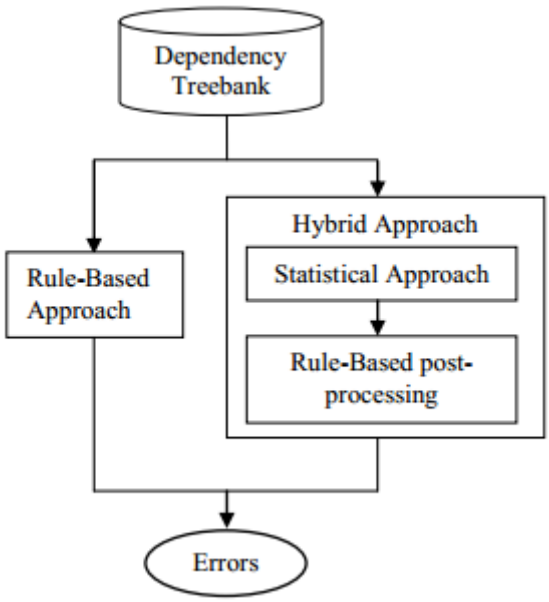

Figure 3.1: Approach proposed by Ambati et al., 2011 ([3]). (Adapted from ([3]).)

Agarwal et al., 2012 ([2]) improved over the PBSM (Probability Based Statistical Module) used by Ambati et al., 2011([3]). They extended the PBSM by adding more linguistically motivated features like dependency labels of the parent, grandparent and count of its children. They also improved the hybrid system by placing a new module 'Rule based correction' before the statistical module. So, they had rule based systems on both sides of statistical system of hybrid module. Figure 3.2 explains their approach.

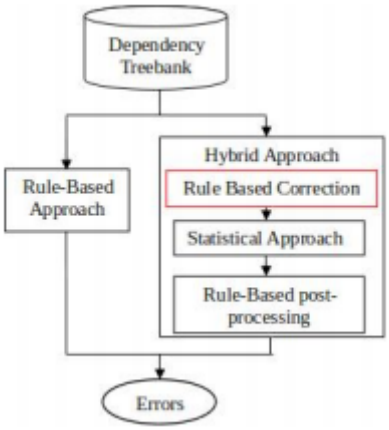

Figure 3.2: Approach proposed by Agarwal et al., 2012 ([2]). (Adapted from ([2]).)

Previous works show that parser outputs may help in detecting errors in Treebank [69], [23] and sometimes, automatic error correction as well [70]. Hence, in our approach, we use MaltParser[1] [53] to detect potential errors in Hindi Dependency Treebank. Baseline accuracy with MaltParser for Hindi are 77.58%, 88.97% and 80.48% for LAS[2] , UAS[3] and LA[4] respectively [6]. Using the parser output alone does not help much because the state-of-the-art parsers for free word order languages like Hindi perform lower than those for fixed word order language like English. So, we also use an algorithm which uses the output produced by MaltParser to detect potential errors in the treebank. Our contribution is a purely statistical system which bypasses manually crafted rules and improves error detection process over previous approaches.

3.4 Our Approach

In this section we describe our approach to error detection using a dependency parser. We detect errors at dependency level annotation. We use inter-chunk dependency trees rather than expanded trees. Figure 3.3 gives a comprehensive description of our approach to detect errors.

[1]MaltParser (version 1.4.1)

[2]LAS - Labeled Attachment Score (Percentage of words that are assigned both correct head and correct label

[3]Unlabeled Attachment Score (Percentage of words that are assigned correct head)

[4]Labeled Accuracy (Percentage of words that are assigned correct label)

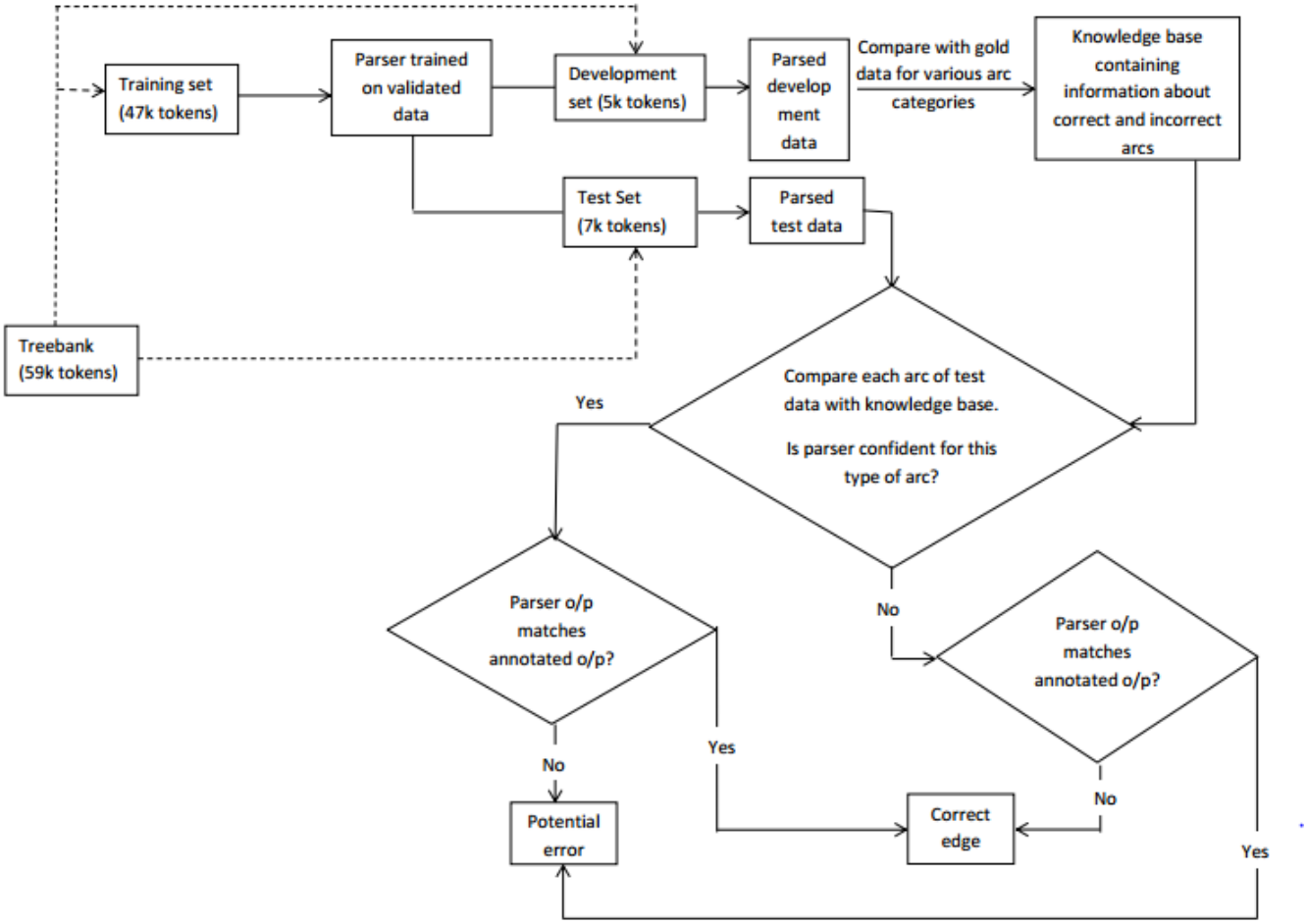

Figure 3.3: Algorithm used for error detection using our approach

3.4.1 Using Parser Output to Detect Potential Errors

In this strategy, we use the output of the parser to detect potential errors in the annotated data. We make an assumption that most of the decisions taken by the parser and the humans are similar, i.e., similar types of errors are made by both. Both, the parser and the annotators find it difficult to parse certain types of structures and both are good at other constructions.

The question may arise why we are using a parser to detect errors made by humans when human performance is better, as compared to a parser's in all respects? The answer is simple. Decisions taken by human beings may sometimes vary but a statistical parser (e.g. MaltParser) gives consistent decisions. Even if the parser gives a wrong output, it will always give the same wrong result for a specific case/instance.

Reasons of human error can be:

1. **Random errors:** Decisions of annotators may change according to their state of mind. They may make random errors if they are not able to concentrate on their work. Though they know the correct decision, they may unknowingly make some errors because of some psychological conditions. Everything being right, an annotator can also by mistake make wrong decisions.

2. **Errors due to unawareness of a concept/Misinterpretation of guideline decisions:** Annotators may make the same mistake again and again if they do not know the concept properly or do not know the correct decision to be taken. Misinterpretation of certain guidelines decision may also cause consistent errors.

3. **Errors in the guidelines:** Some sections of the guidelines are a lways evolving. Some other sections might have errors due to oversight. Annotators will make consistent mistakes in such a case unknowingly.

The types of arcs for which the parser (MaltParser) consistently gives wrong decisions are[5]:

1. **Long distance arcs:** MaltParser is greedy in marking dependencies and searches for the head of a word in the vicinity of the word. It adds a node as the parent of another node as soon as it finds a suitable candidate for it. Hence, it generally marks long distance dependencies also as short distance dependencies.

2. **Non-projective arcs:** The algorithm used in MaltParser always produces projective graphs and cannot handle non-projective dependencies. [51]

The idea here is to use consistent decisions of the parser to mark the human errors. If the parser performs well in some constructions, it will, most of the time, give a correct decision and we can compare annotator's decision with that of the parser and if the decisions made by them are different, we can infer that the decision taken by the annotator may be incorrect (as the parser's decision was right and the annotator's decision does not match). Even if the parser takes wrong decisions in other types of constructions, it will consistently mark them wrong. If we know these cases, we can again compare such decisions with annotator's decision and if they match, we can say that decision taken by the annotator is incorrect (as the parser was wrong and annotator's decision is similar to the parser's). Here, we are only marking the potential errors, so even if we mark a correct decision as an incorrect one, it will not create much problem as it can be checked further during the process of validation. But we must not leave any error unmarked because this would lead to an erroneous treebank.

Our classification of errors is taken from Husain and Agrawal, 2012 [34]. Here parser errors were classified based on:

[5]These errors are because of the algorithm used (Arc-Eager[51]) for parsing. We have used this algorithm because it outperformed other algorithms for hindi. [6]

1. **Edge[6] Type and Non-Projective Edge:** The edges were categorized based on the distinct dependency labels on the edges.

2. **Edge Depth:** Depth of a tree is the total number of levels of links that it has. Consequently, the depth of an edge is basically the level at which it exists in the tree.

Examples of edge types from Husain and Agrawal, 2012 [34] are:

```
1. Main
2. Intra Clausal
    a) Verb Argument Structure
      i)  Complement
      ii) Adjunct
    b) Non-verbal
      i)   Noun-modifier
      ii)  Adjective-modifier
      iii) Apposition
      iv)  Genitive
    c) Others
      i)   Co-ordination
      ii)  Complex Predicate
      iii) Others
3. Inter Clausal
    a) Co-ordination
    b) Sub-ordination
      i)   Conjunction
      ii)  Relative Clause
      iii) Clausal Complement
      iv)  Apposition
      v)   Verb Modifier
```

[6] An edge in a dependency tree is the arc that relates two nodes(which are basically words). These are binary asymmetric relations with labels that specify the relation type.

First of all, Maltparser was trained using manually validated training data and development data was parsed using the trained model. Then using the above mentioned classification, the data was classified according to different categories (edge type and edge depth). After that, the parsed output was analyzed by comparing it with gold data to find out the classes of arcs for which the parser works well and for which it doesn't. Using this analysis, a knowledge base was prepared, which contains the information about the arcs being correctly or incorrectly parsed for various edge type and edge depth.

With the help of this knowledge base, confidence level for different categories of arcs (different edge-types and different edge-depths) was set as 1, if the parser was confident on giving correct decisions for that type of arcs and 0 if it was less confident of giving correct decisions for that type of arcs. For example:

- The parser gives wrong attachments and labels for most of the inter-clausal arcs. They are parsed correctly only if they are at depth 1 or 2. e.g. 'Inter Clausal Relative Clause' is only parsed correctly if it occurs at depth 1, so confidence level for this class of arcs is set to 1 if it occurs at depth 1 and is set to 0 if the arc occurs at any other depth.

- Most of the Intra Clausal arcs are parsed correctly but only if the sentence is not very long. Intra Clausal arcs in general occur at the leaves of a dependency tree. For Intra Clausal arcs, confidence level is 1 for short depths and 0 when depth is high. e.g. Confidence level is 1 for 'Intra Clausal, Verb Argument Structure (Complement)' upto depth 10 and confidence level is 0 when this class of arcs occur at a depth greater than 10.

For boundary conditions, confidence level was set using the results obtained from development data. e.g. for arcs of category 'Intra Clausal, Verb Argument Structure (Complement)', confidence level was set 1 for depth 9, 1 for depth 10, 0 for depth 11 and 0 for depth 12 because confidence level 1 for depth 9 gave better results for development data than confidence level 0. Confidence level 0 for depth 11 was better than 1 and so on. In this manner, database for confidence level (knowledge base) for each category of arcs was prepared.

After preparing the confidence level database (knowledge base) using the development data, testing data was also parsed using the trained model and was flagged with potential errors with the help of confidence level mentioned in the knowledge base. Testing was performed as follows:

For each arc, its category was checked. For that category, confidence level was taken from the knowledge base. If confidence level was 1 and parser output was different from the annotated output, the arc was marked to be a potential error. Again, if confidence level is 0 and parser output matches annotated output, the arc was marked as a potential error. This can be explained in detail as follows:

1. We extract cases for which the parser is highly confident (confidence level 1) on giving correct labels and attachments.

Table 3.1: Results of validation using parser output

Data Precision/Recall	Attachment	Labeling
Development	56.72/79.72	60.38/82.41
Test	44.56/78.44	63.45/89.71

Here, the parser gives correct decisions and if parser output doesn't match the annotated one, there is a possibility that the annotator annotated it wrong. So, we mark that node as a potential error node.

For example, let there be a category of arcs containing the labels 'k1', 'k2', 'k4', etc. and the parser is confident enough on giving correct attachment and label for this class of arcs. Then, if an arc x $\rightarrow$ y is labeled as 'k1' by parser and the annotators marked it 'k2', it is flagged as a potential error.

2. Next, we extract cases for which the parser is less confident (confidence level 0) on giving correct labels.

   In this, the decision taken by the parser might be incorrect and if the parser output matches the annotated output, there is a possibility that the annotator also made the same mistake. Hence, we flag that node as a potential error node.

   For example, again let there be a category of arcs containing the labels 'k1', 'k2', 'k4', etc. and the parser is less confident here, of giving correct attachment and label for this class of arcs. Then, if an arc x $\rightarrow$ y is labeled as 'k1' by the parser and the annotators also mark it 'k1', it is flagged as a potential error.

After this precision and recall are calculated which are shown in Table 3.1.

3.5 Data Used in Our Approach

The data used for the experiments is part of a larger Hindi Dependency Treebank. The size of training, development and testing data is 47k, 5k and 7k respectively. The training data used was manually validated and hence we assume it to be free of any errors. Hence the patterns learned by the dependency parser will generally be consistent, thereby improving parser accuracy.

3.6 Comparison with Other Techniques

In this section, we present a comparison of the results of our strategy with one of the earlier tools [2]. They detected errors in Hindi Dependency Treebank using a hybrid approach, as described earlier in Section 3.2. As showed in Table 3.3, our approach significantly outperforms Agarwal et al., 2012([2]) both in terms of precision and recall. Also, a detailed comparison between their approach and our approach is shown in Table 3.2. There is an overlap of 73.46% of the total errors between our approach and Agarwal et al., 2012([2]).

As is visible from Table 3.2, the earlier approach [2] flagged a large number of nodes as error nodes (1724 nodes were flagged as error nodes while actual errors were 490) out of which 395 errors were correct. So, the precision of this approach was little low. Our approach flagged 672 errors, out of which 434 were correct. This leads to high precision and recall. Correct errors common to both the approaches were 360. Further, both these approaches when used together were able to detect a total of 483 out of 490 errors. It might be a good idea to consider errors flagged by both approaches to cover almost all the errors in the Treebank. Also, Agarwal et al., 2012([2]) tried to capture only labeling errors while we capture attachment errors too, which are difficult to capture as compared to labeling errors.

Our approach outperformed Agarwal et al., 2012 ([2]) though it was a hybrid approach, because they mostly concentrated on capturing as much errors as possible while we have tried to maintain balance between both precision and recall. However, it is very likely that the approach by Agarwal et al., 2012([2]) might not have performed well because of lack of a good size of training data and their rule based approach to increase precision could not cover all concepts of the language. But at the same time, our approach is purely statistical which makes it very much dependent on the performance of the parser used which in turn depends on the availability of good quality training data. So, if the training data is biased, our approach will not perform well while approach by Agarwal et al., 2012 ([2]) takes into consideration language specific rules which ensure considerable precision of their tool irrespective of the data used. Also, Agarwal et al., 2012 ([2]) proposed an interface with the help of which detected errors can be easily corrected in less time while we don't have any such GUI.

3.7 Effect of correcting flagged errors

The errors flagged by our approach were corrected and the parser was trained on validated data. The performance of this newly trained parser was compared with the parser trained on annotated data (without correcting errors). The performance of both the parsers is shown in Table 3.4. The comparison shows that parser trained on error free data performs better than that trained on non-validated data. Hence, the proposed error detection approach indirectly helps in improving the quality of parsers too.

Table 3.2: Comparison of present approach with Agarwal et al., 2012([2])

Total Original Errors	490
Total Errors Flagged by Agarwal et al., 2012([2])	1724
Total Errors Flagged by our approach	672
Total Common Errors Flagged by our approach and Agarwal et al., 2012([2])	522
Agarwal et al., 2012([2]) Errors Correct	395
Our approach Errors Correct	434
Overlap between Agarwal et al., 2012([2]) and our approach	360
Total(Both methods) Flagged Errors	1874
Total(Both methods) Correct Errors	469

Approach	**Precision(%)**	**Recall(%)**
Our approach	**64.58**	**88.57**
Agarwal et al., 2012([2])	23.24	80.61

Table 3.3: Comparison of overall (attachment+labeling) results of our approach with previous one Agarwal et al., 2012([2])

	Parser trained on annotated data	**Parser trained on validated data**
LAS(%)	58.60	**59.83**
UAS(%)	78.77	**82.03**
LS(%)	61.57	**62.16**

Table 3.4: Comparison of parser trained on annotated data with that trained on validated data

3.8 Discussion

In this chapter, we presented an automatic approach to detect potential errors in an annotated treebank to provide aid in its validation task. Precision and recall of the approach are *64.58% and 88.57%* respectively. Unlike the previous approach reported in Agarwal et al., 2012([2]), our approach is language independent. Given a manually validated treebank, all one needs is to prepare a knowledge-base using parser error patterns. We expect, it will be relatively easy to adopt our approach for similar dependency treebanks.

Certain types of errors cannot be detected by our method. For categories where the parser is less confident on taking correct decision, a node was flagged as a potential error only when the annotator's decision matched the parser's decision. There are chances that for a class of arcs, both, the parser and the annotators take different decisions but both the decisions are incorrect. In such a case, since the annotator's output doesn't match the parser's output, we do not flag it as an error even though it is an erroneous node. For example, in the cases of 'Inter Clausal, Clausal Complement' and 'Inter Clausal, Relative Clause' arcs, we came across examples where both parser's decision and annotator's decision were incorrect but different from each other and since parsers are not confident in such decisions, we could not mark them as errors.

The power of our validation approach is bounded by the performance of the parser used. For example, in cases of 'Intra Clausal, Verb Argument Structure(Adjunct)' and 'Intra Clausal, Noun-Modifier', parsers are generally confident, so we flagged annotators' decisions as potential errors if they don't match parsers' decision. But if parser's decision was itself incorrect, annotators' decision was flagged as error although it was correct. If we have better parsers, we can flag potential errors with more confidence and accuracy.

Also, to decide the confidence level for various classes of arcs, recall was used as a metric but there was a trade-off between precision and recall. Hence, if we use F-score as a metric in place of recall, we may achieve better results.

3.9 Error detection in parser output

[7] This work was done in collaboration with Sambhav Jain [38]. It aims at computing an entropy based per edge confusion score, dynamically computed and assigned while parsing. This score tries to give a more informed picture of the parsed output. The measure can also be utilized to flag potential incorrectly parsed edges which later, can either be manually corrected or altogether discarded (to fall back on lower level but more accurate features). The edges exhibiting high confusion score are also highly probable to be incorrect, as the oracle is uncertain in its decision. Using this insight, an edge-label is flagged as potential error if its confusion score is above a pre-calculated threshold(θ).

[7]This section gives a brief description of this work. Complete details are available in [38].

Threshold(θ) is a crucial parameter in the experimental setup. An optimum θ is chosen by making use of the development set. We iteratively increase candidate values for θ, from minimum to maximum possible value of confusion score, with an adequate interval. Corresponding to each of these values, the incorrect edges are flagged and precision, recall & F $-$ score are calculated. The value asserting the maximum F $-$ score is chosen as the final θ.

We conducted experiments on 20 languages, using data from CoNLL-X [17], CoNLL 2007 [50] and MTPIL COLING 2012 [61] shared tasks on dependency parsing. An average F1 $-$ score of 48.96%, precision of 44.19% and recall of 57.23% is obtained over 20 languages in the task. To efficiently capture the efficacy of our approach, another metric was used which corresponds to the percentage of errors detected by inspecting 1%, 5% and 10% of total edges. The metric gives a more precise picture of the effort required to correct errors. Our experiments indicate that on average 42.44% errors can be detected by just inspecting 10% of total edges. This portrays that the effort required here is one fourth as compared to that in conventional sequential correction. On inspecting 5% and 1% of all edges, 24.00% and 5.51% errors can be detected. Best results are obtained for Portuguese where 57.59% of errors are detected by merely inspecting 10% of total edges, while 35.23% and 10.43% are detected on inspecting 5% and 1% edges respectively.

This section presented our effort towards computing a confusion score that can estimate, upfront, the correctness of the dependency parsed tree. The confusion score, accredited with each edge of the output, is targeted to give an informed picture of the parsed tree quality. We supported our hypothesis by experimentally illustrating that the edges with relatively higher confusion score are the predominant parsing errors.

Not only parsed output, manual treebank validation too can benefit from such a score. An n-fold cross validation scheme can be adopted, in this case, to compute and assign confusion scores and detect annotation errors. Also, this score has scope in active learning and self training where unannotated instances exhibiting high confusion can be prioritized for manual annotation. Next chapter proposes use of this score in active learning for treebank development.

Chapter 4

Active Learning employed in building treebanks for resource poor languages

4.1 Introduction

Performance of statistical tools depends on availability of good quality huge annotated treebanks. As annotation of treebanks is a manual process followed by multiple levels of validation to make it error free, acquisition of such a treebank becomes not only an expensive process but also very time consuming and tedious. Treebanks being unavailable for resource poor languages makes development of tools for such languages a very tough job.

This question is at the heart of active learning which assumes that *"a machine learning algorithm can achieve greater accuracy with fewer training labels if it is allowed to choose the data from which it learns. An active learner may pose queries, usually in the form of unlabeled data instances to be labeled by an oracle (e.g. human annotator). Active learning is well-motivated in many modern machine learning problems, where unlabeled data may be abundant or easily obtained, but labels are difficult, time-consuming, or expensive to obtain."* [60]

Active learning is a special case of semi-supervised machine learning where a learning algorithm tries to explores methods to collect training data on its own and does not depend on random samples or manual collection of samples for training. The goal of active learning is to reduce number of training samples required to obtain the same level of performance which is obtained by a huge treebank. This is achieved by informative gathering of samples needed for effective training. The training samples thus selected can be given for annotation and a high

quality treebank can be developed for resource poor languages with reduced effort, time and cost which in turn results in development of natural language processing tools for the same.

Picking samples for annotation from large number of samples depends on the usefulness of the sample. Many selection strategies have been explored for the same. Uncertainty sampling, which is the strategy that has been used in this work, is the simplest and the most commonly used one. In this strategy, usefulness might be calculated by predicting the uncertainty of the tool in processing the sample.

In this work, we have used active learning for statistical dependency parsing where new sentences are ranked based on the highest uncertainty an existing parser exerts during parsing while selecting samples. The reason being, these sentences parsed if annotated and added in training samples, contribute the most in reducing the uncertainty, among other unannotated sentences. The performances of these methods heavily rely on the definition of a good confidence measure, which measures the confidence the model, in our case the parser, has in the solution it proposes. We used the confusion score proposed by Jain et al., 2015([39]) to compute uncertainty in parsing.

4.2 Related Work

Till now, active learning has been explored in the areas of machine learning [20], text classification [45], part of speech tagging [21], automatic speech recognition, information extraction [60], sampling [43] but it has not been studied much in natural language parsing. Active learning for natural language parsing has been explored by [67, 36, 49, 63, 66, 64, 37]. In each of these works, uncertainty was calculated for every individual decision taken by the parser and these uncertainties were used to calculate the uncertainty for a sentence. Thompson et al.,1999([67]) defined uncertainty of an non-parseable sentence as the number of operators applied successfully divided by the number of words. On similar lines, we used confusion score proposed by Jain et al., 2015([39]) for uncertainty calculation of a sentence.

4.3 Data Used

We have performed our experiments for Urdu, Hindi and Telugu. Urdu data was taken from Urdu dependency treebank developed under the Hindi-Urdu treebank project [15], [55]. Hindi data used for experiments was released as part of MTPIL COLING 2012 shared task [61]. This data is a part of Hindi Treebank developed under the same project. Telugu data is being developed in IIIT Hyderabad only. Telugu treebank contained a total of 1634 sentences out of which 978 sentences were used as training data, 330 sentences as development data and 326 sentences as testing data.

4.4 Methodology

Jain et al., 2015([39]) proposed a method for dynamic computation of entropy based confusion score for joint prediction of directed arcs and their dependency labels in a typed dependency parsing framework. This score is calculated on the basis of confusion encountered by the oracle of a transition based data-driven dependency parser. For each sentence in the training set, we computed this confusion score and calculated average of top 5 most confusing arcs(arcs with highest confusion score) and took it as a representation of the confusion score of the entire sentence.

First of all, 25 bootstrap sentences were randomly selected from the training set, the parser was trained using this training set. Then in each step of active learning, 25 most confusing sentences were chosen from the remaining training samples and added to the training set.The result of learning was observed after each round of selection. This performance was compared with random selection where 25 sentences in each round were picked randomly. The comparison of performances of active learning and random picking are shown in figure 4.1, 4.2 and 4.3.

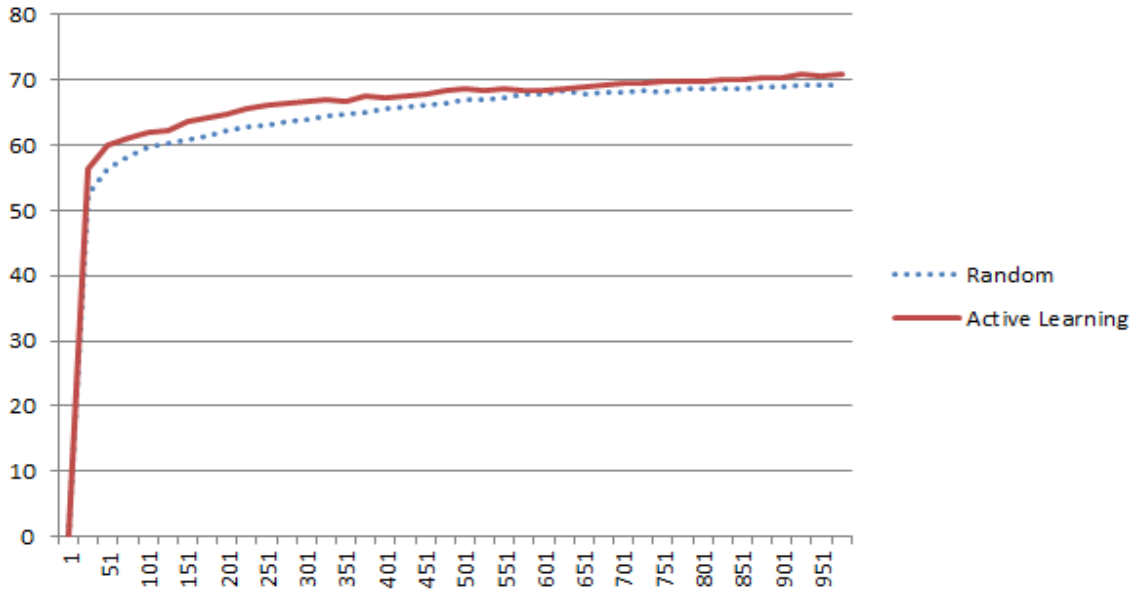

Figure 4.1: Active learning results for Urdu

4.5 Results and Discussion

Figure 4.1, 4.2 and 4.3 show results of active learning for Urdu, Hindi and Telugu respectively. LAS (Labeled Attachment Score) was considered for comparison. These figures show that if sentences are picked through active learning, the performance is better than random picking.

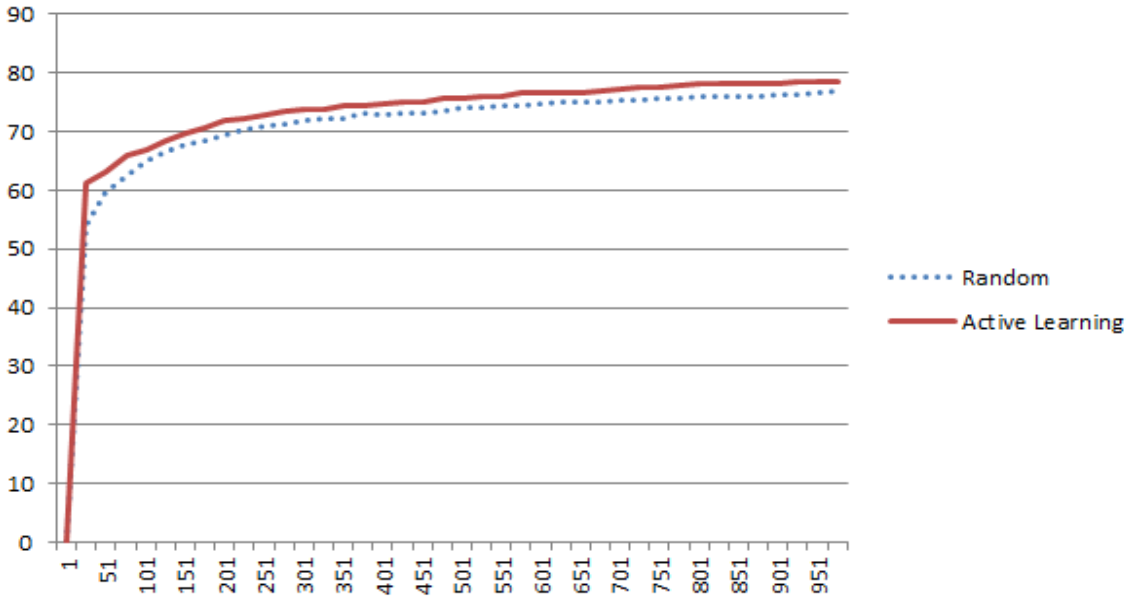

Figure 4.2: Active learning results for Hindi

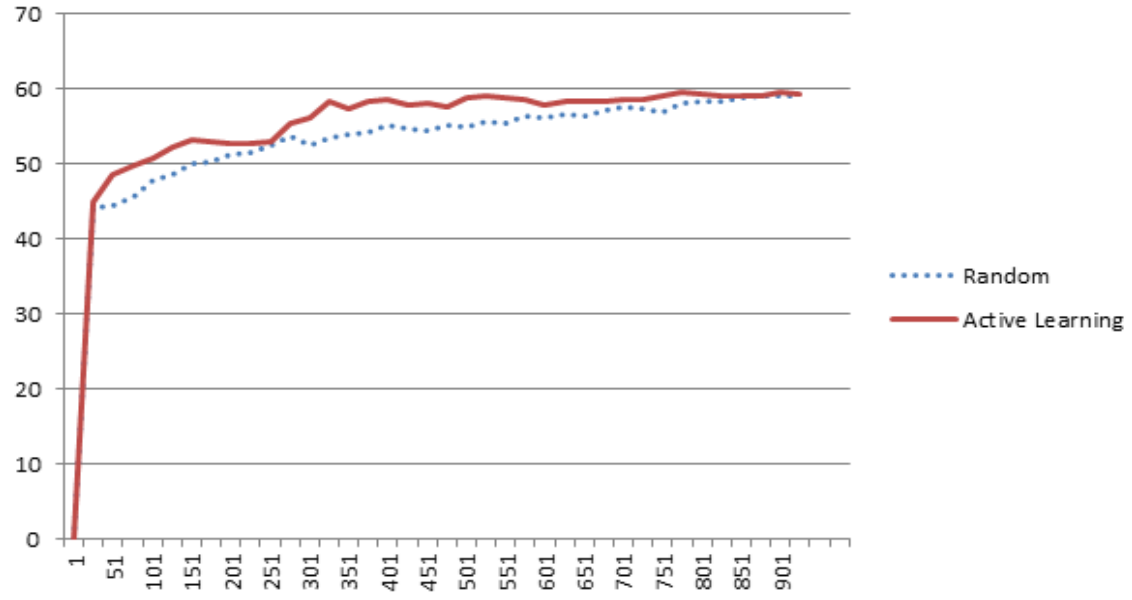

Figure 4.3: Active learning results for Telugu

For each of the languages, the level of parsing accuracy obtained through random picking was obtained in almost half of training data through active learning (Accuracy obtained by using 700 sentences from random picking was approximately same as what was achieved from 350 sentences picked through active learning). This is because, random picking might suggest sentences for annotation which might not be useful for annotation as those types of sentences

might already be present in the existing training set. So, they don't contribute anything in parser performance. While active learning suggests samples which parser finds difficult to parse, so they might be of new construction. Thus it adds value to existing training set and only useful sentences are annotated. It helps in treebank development, as less annotation is required. Thus, we can use active learning to pick sentences which are more informative for the parser and annotate & include only those sentences for constructing the treebank. Parsing accuracy for both random picking and active learning becomes almost equal when approximately 700 sentences are added because at this stage all types of construction are added by both the steps and parser becomes stable.

Active learning is advanced stage of machine learning that has been broadly applied to classification tasks. We have succeeded in showing its usage in one more complex and vital natural process tasks, syntactic parsing. The availability of unannotated data and scarcity of annotated data with complexity of annotation, puts forward selective sampling as a potential helpful technique for natural language learning. Improvised sample selection techniques and their implementation in other important problems might help in progress of using machine learning for building natural language processing tools.

Chapter 5

Domain Adaptation

5.1 Introduction

[1] The problem of domain adaptation emerges in a variety of applications when the learning model is trained on a source data distribution and is expected to perform on a different target data distribution. This problem is of major concern in Natural Language Applications where we generally have a huge collection of labeled data in one domain but need a model which performs equally well in some target domain, because we might have little or no data from the target domain. Even if we can construct data for target domain, it doesn't seem reasonable to retrain the model everytime we get data from various new domains. Domain adaptation aims to build classifiers that are robust to varied distribution. It offers a flexible model that allows us to transfer category independent information between domains.

Current state-of-the-art hindi dependency parsers have achieved reasonably good accuracy but adapting parsers to new domains without availability of target domain labeled training data still remains a problem. In this work, we present an approach which tries to make state-of-the-art hindi parsers (trained on one domain) robust for data from new domains.

5.2 Related Work

Domain adaptation has been done for many natural language processing tasks, e.g., classification [22, 27, 28, 18]. Daume et al., 2009([22]) show that an extremely simple method delivers very good performance on domain adaptation classification tasks. They augmented features of source domain with the "general" pseudo-domain to capture domain independent features. In

[1] Work done in this chapter is reported in [65] which is a joint effort of Aniruddha Tammewar, Karan Singla and me. This work focuses on using word embeddings as an alternative to linguistic features for parsing. The work presented in this thesis is my contribution towards [65] and has not been submitted by anyone else.

the parsing domain, [35] and [30] have shown that simple techniques based on using carefully chosen subsets of the data and parameter pruning can improve the performance of an adapted parser. [59] used a Dirichlet prior on the multi-nomial parameters of a generative parsing model to combine a large amount of training data from a source corpus (WSJ), and small amount of training data from a target corpus (Brown). [48] augmented annotated training data with hierarchical word clusters that were automatically derived from a large unannotated corpus. They evaluated the technique for named-entity tagging.

On similar lines with [22] and [48], we tried to use domain adaptation for parsing by capturing domain independent features with the help of grouping words into clusters from source domain and general domain.

5.3 Our Approach

In this work, we tried to tackle problem of domain difference between training and testing data by tackling the different vocabularies in specific domains. When we have a parser trained on data of certain domain and we want to use it for parsing data from another domain, we need to bridge the gap between diverse vocabulary of different domains, which makes parsing difficult. For this task, we group the words with similar semantics into clusters. These clusters help in handling new words by treating them similar to other words from the same cluster.

Initially, we took all the words from source domain and a general domain[16] and clustered semantically similar words into various groups. For clustering, we stemmed all the words using a stemmer for hindi[57]. These stemmed words were then converted to word vectors of size 100 using Word2Vec[2]. These word vectors were classified into 100 clusters using K-means clustering[7] by scikit-learn.[3]

The cluster ids thus obtained for each word were augmented in the feature column of the training data (CoNLL format). The parser[4] was then trained using this training data. For each of the new domains, clusters were predicted for every stemmed word using the model obtained from Word2Vec and scikit-learn. These cluster ids were again augmented in the test data similar to training data. These cluster-ids help new words to be treated similar to other words of the same cluster. So, if in the testing domain, there is a word which was not found in training data, the parser might make mistake in parsing it. But, if we add a cluster id, it suggests parser to handle it in a way in which other words of the same cluster were handled.

[2]https://radimrehurek.com/gensim/models/word2vec.html [58]

[3]http://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html#sklearn.cluster.KMeans[56]

[4]MaltParser version 1.7 from *http://www.maltparser.org[53]*[53]

We experiment our approach on the data of four domains: box-office[5], cricket, gadget and recipe. Results showed that clustering semantically similar words helped parser in making correct decisions.

5.4 Data Used

The source domain data is a collection of news and heritage domain articles (Hindi Dependency Treebank - HDTB)[6]. General domain used for making clusters was taken from [16]. Target domain data has been developed in IIIT Hyderabad only. Size of each domain data is as follows:

- Box-office : 509 sentences
- Cricket : 546 sentences
- Gadget : 529 sentences
- Recipe : 543 sentences

5.5 Results and Discussion

Table 5.1 shows that adding cluster information for lemmas definitely helps parser in making correct attachments and labels. Maximum improvement is in recipe domain, this might be because vocabulary of recipe domain was most different from news articles and clustering words helped maximum for this domain as vocabulary divergence was reduced drastically.

Figure 5.1 demonstrates the benefit of adding cluster id as feature. The full sentence with chunk boundaries is as follows:

(jab [when]) (1911 mein [in 1911]) (queen Mary [queen Mary]) (yahan [here]) (aai thi, [came aux ,]) (NULL [then]) (British shashko ne [British Rulers -Erg]) (yaha ke [here's]) (pani ke [of water]) (hauj ke upar [above reservoir]) (ek chhat [a roof]) (banwai thi [built aux]) (. [.])

Figure 5.1(a) shows the gold tree, 5.1(b) the parsed tree without adding cluster ids and 5.1(C) the parsed tree after adding cluster ids as a feature. 'Mary' should be marked as 'k1' (doer) of 'came' but it is wrongly marked as 'k7p'(place of action) as the information provided by the default features is not sufficient to identify that the word 'Mary' is name of a person (the POS (part of speech) tag 'NNP' (proper noun) is assigned). The problem is resolved when we add cluster id as a feature.

We found that most of the words in the cluster containing word 'Mary' are British names as the treebank data is from news and heritage domain. Data from Indian heritage domain

[5]Bollywood movies

[6]http://ltrc.iiit.ac.in/treebank_H2014

	Baseline	**After adding cluster-id**
In domain:(News & heritage)		
LAS(%)	84.75	84.9
UAS(%)	93.01	92.95
LS(%)	87.15	87.41
Box-office		
LAS(%)	75.89	76.97
UAS(%)	89.64	90.64
LS(%)	77.67	78.63
Cricket		
LAS(%)	69.81	70.59
UAS(%)	87.7	87.7
LS(%)	72.38	73.39
Gadget		
LAS(%)	71.62	73.1
UAS(%)	85.7	86.41
LS(%)	73.77	75.1
Recipe		
LAS(%)	65.87	68.18
UAS(%)	83.02	84.15
LS(%)	70.34	72.62

Table 5.1: Domain adaptation results

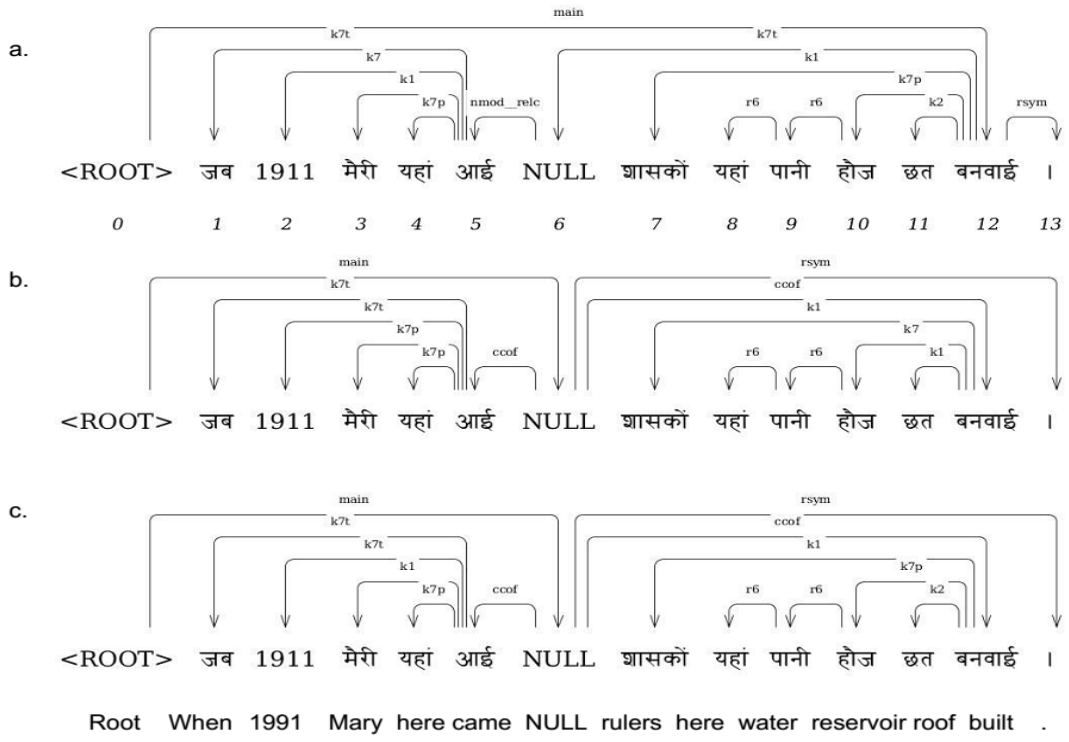

Figure 5.1: Gold tree(a), Auto tree(b), Auto+cluster Id tree(c) [65]

contains description of British people at various places. The information that 'Mary' is name of person and not a place is captured by the cluster id. In similar way, the case of 'k7p' for 'reservoir' and 'k2' for 'roof' is handled.

From above observation, we can say that adding cluster information as a feature resolves ambiguity for parser. This feature helps in parsing data from same domain also. But if clusters are not formed properly, they might even have negative impact on parser's performance. For example, we took a sentence from box-office domain data:

(diwali [diwali]) (din [day]) (release [release]) (hui [aux]) (film [film]) (dino [days]) (weekend [weekend]) (mila [got]).

The correct parse of this sentence is shown in Figure 5.2 and the parse obtained from MaltParser after adding cluster information is available in Figure 5.3. Here, the word 'release' was wrongly kept in a cluster containing names of various actors and actresses which are generally the 'doer' of an action and marked 'k1'. Hence, while parsing 'release' was incorrectly marked as 'k1' instead of 'pof'.

Although, results show that adding cluster information helps the parser in reducing the ambiguity, nothing can be claimed in particular as the data size for target domains was very less and the analysis done is in primitive stage. Efficacy of the approach needs to be researched further with statistical significance test.

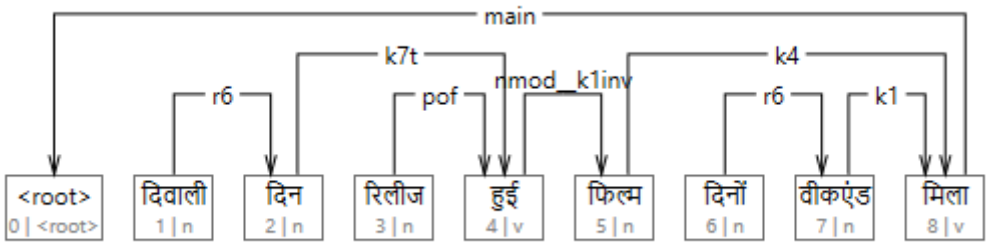

Figure 5.2: Gold sentence from Box-office data

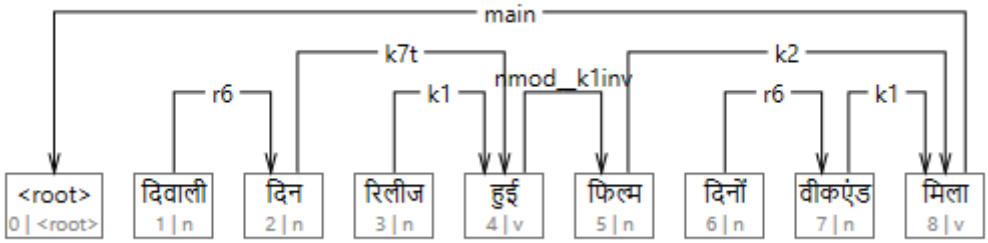

Figure 5.3: Parsed sentence from Box-office data

Domain adaptation does not only involve handling OOV (out of vocabulary) words. Along with vocabulary differences, sentences from various domains vary in their structures too. We inspected sentences from each domain against their structure with the differences shown in Table 5.2. We can see that sentences from News & Heritage domain (source domain) contradict the most with sentences from Recipe domain (one of target domains) in terms of average length of sentences, average depth of sentences and number of non-projective arcs per sentence. On the other hand, sentences from box office domain differ the most with source domain when talked about number of clauses per sentence. These factors might also be responsible towards degraded performance of parsers for new domains. Efforts can be taken towards diminishing these differences and exploring the effect.

	Avg. length of sentences	**Avg. depth of sentences**	**No. of non-projective arcs/sentence**	**No. of clauses/ sentence**
New Articles	11.51	3.85	0.47	1.66
Box Office	7.83	3.06	0.32	1.27
Cricket	9.11	3.08	0.39	1.58
Gadget	8.25	3.08	0.47	1.38
Recipe	7.51	2.6	0.08	1.62

Table 5.2: Structural differences among various domains

Chapter 6

Conclusion and Future Directions

In this dissertation, we have highlighted the importance of syntactic parsing in Natural Language Processing and described dependency parsing. We compared performance of two dependency parsers (MaltParser and MSTParser) for Hindi and explored the role of morphological features in dependency parsing.

Despite extensive advancements in the field of syntactic parsing, parsing for Indian languages is still a challenging task. One of the major reasons is lack of resources. Even if the treebanks are available, their quality is of utmost importance to us as treebanks play a very pivotal role in building natural languages processing tools. Therefore, we proposed methods to assist treebank development and treebank validation. We used active learning techniques to intelligently pick and suggest examples to annotate and include in treebank. For treebank validation, we worked for automatic detection of potential errors in the treebank. These errors if given to the validators, help and speed up the validation process.

We also proposed a method to compute a confusion score which beforehand assesses the correctness of the parsed dependency tree. The confusion score, accredited with each edge of the output, is targeted to give an informed picture of the parsed tree quality. We supported our hypothesis by experimentally illustrating, for 20 languages, that the edges with relatively higher confusion score are the predominant parsing errors.

Another well known complication in this area is the domain confinement of parsers. Traditional statistical learning algorithms consider training data and testing data from the same domain. Unfortunately, this assumption is too strong in practical because NLP applications are often domain unbound and freely accept data from any common domain. These adversities question the utility of existing parsers in NLP applications. We, therefore, presented an approach which tries to make parser robust for data from any domain. We achieve this by decreasing the vocabulary divergence among various domains.

6.1 Future Work

We demonstrated various approaches to improve state-of-the-art dependency parsers albeit there is much scope for improvement. Few areas which need research for further improvement in dependency parsing are as follows:

- While we worked on diminishing vocabulary difference among domains, various domains may vary in structure of sentences only, e.g., Conversational texts. Structural differences can be studied and worked upon to adapt parsers to new domains.
- PQE functionality corresponding to attachments and joint prediction can also be extended to other algorithms & parsers and later be used for ensembling.
- While we explored active learning using above PQE scores, self learning can also be examined and used for treebank development.

Related Publications

1. **Title** : An Automatic Approach to Treebank Error Detection Using a Dependency Parser.
   Authors : **Bhasha Agrawal**, Rahul Agarwal, Samar Husain and Dipti M. Sharma
   Published at : In Proceedings of the 14th CICLing (Conference on Intelligent Text Processing and Computational Linguistics), Samos, Greece. 2013

2. **Title** : Can Distributed Word Embeddings be an alternative to costly linguistic features: A Study on Parsing Hindi.
   Authors : Aniruddha Tammewar, Karan Singla, **Bhasha Agrawal**, Riyaz Bhat and Dipti Misra Sharma
   Published at : In Proceedings of the 6th Workshop on Statistical Parsing of Morphologically Rich Languages (SPMRL 2015), pages 21–30, Bilbao, Spain, July 23rd 2015

Other Publications

1. **TItle** : A Dynamic Confusion Score for Dependency Arc Labels.
   Authors : Sambhav Jain and **Bhasha Agrawal**
   Published at : In 6th International Joint Conference on Natural Language Processing (IJCNLP2013), Nagoya, Japan, 2013

2. **Title** : Employing Oracle Confusion for Parse Quality Estimation.
   Authors : Sambhav Jain, Naman Jain, **Bhasha Agrawal** and Rajeev Sangal
   Published at : In Proceedings of the 16th CICLing (Conference on Intelligent Text Processing and Computational Linguistics), Cairo, Egypt. April 2015

3. **Title** : Analyzing parser errors to improve parsing accuracy and to inform treebanking decisions. (Journal Article)
   Authors : Samar Husain and **Bhasha Agarwal**
   Published at : In Linguistic Issues in Language Technology, 7(11), 2012
   Presented at : The 10th International Workshop on Treebanks and Linguistic Theories (TLT10). Heidelberg, Germany. 2012

Bibliography

[1] S. P. Abney. *Parsing by chunks*. Springer, 1992.

[2] R. Agarwal, B. R. Ambati, and D. M. Sharma. A hybrid approach to error detection in a treebank. *Linguistic Issues in Language Technology*, 7(1), 2012.

[3] B. R. Ambati, R. Agarwal, M. Gupta, S. Husain, and D. M. Sharma. Error detection for treebank validation. *Asian Language Resources collocated with IJCNLP 2011*, page 23, 2011.

[4] B. R. Ambati, M. Gupta, S. Husain, and D. M. Sharma. A high recall error identification tool for hindi treebank validation. In *LREC*, 2010.

[5] B. R. Ambati, S. Husain, S. Jain, D. M. Sharma, and R. Sangal. 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, 2010.

[6] B. R. Ambati, S. Husain, J. Nivre, and R. Sangal. 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, 2010.

[7] D. Arthur and S. Vassilvitskii. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics, 2007.

[8] R. Begum, S. Husain, A. Dhwaj, D. M. Sharma, L. Bai, and R. Sangal. Dependency annotation scheme for indian languages. In *IJCNLP*, pages 721–726. Citeseer, 2008.

[9] A. Bharati, V. Chaitanya, R. Sangal, and K. Ramakrishnamacharyulu. *Natural language processing: a Paninian perspective*. Prentice-Hall of India New Delhi, 1995.

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

[11] A. Bharati and R. Sangal. 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, 1993.

[12] A. Bharati, R. Sangal, and T. P. Reddy. A constraint based parser using integer programming. *Proc. of ICON*, 2002.

[13] A. Bharati, R. Sangal, D. M. Sharma, and L. Bai. Anncorra: Annotating corpora guidelines for pos and chunk annotation for indian languages. *LTRC-TR31*, 2006.

[14] A. Bharati, D. M. Sharma, S. Husain, L. Bai, R. Begam, and R. Sangal. Anncorra: Treebanks for indian languages, guidelines for annotating hindi treebank, 2009.

[15] R. Bhatt, B. Narasimhan, M. Palmer, O. Rambow, D. M. Sharma, and F. Xia. 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, 2009.

[16] O. Bojar, V. Diatka, P. Rychlý, P. Straňák, V. Suchomel, A. Tamchyna, and D. Zeman. HindMonoCorp 0.5. LINDAT/CLARIN digital library at Institute of Formal and Applied Linguistics, Charles University in Prague.

[17] S. Buchholz and E. Marsi. 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, 2006.

[18] C. Chelba and A. Acero. Adaptation of maximum entropy capitalizer: Little data can help a lot. *Computer Speech & Language*, 20(4):382–399, 2006.

[19] Y.-J. Chu and T.-H. Liu. On shortest arborescence of a directed graph. *Scientia Sinica*, 14(10):1396, 1965.

[20] D. A. Cohn, Z. Ghahramani, and M. I. Jordan. Active learning with statistical models. *Journal of artificial intelligence research*, 1996.

[21] I. Dagan and S. P. Engelson. Committee-based sampling for training probabilistic classifiers. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 150–157, 1995.

[22] H. Daumé III. Frustratingly easy domain adaptation. *arXiv preprint arXiv:0907.1815*, 2009.

[23] D. De Kok, J. Ma, and G. Van Noord. A generalized method for iterative error mining in parsing results. In *Proceedings of the 2009 Workshop on Grammar Engineering Across Frameworks*, pages 71–79. Association for Computational Linguistics, 2009.

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

[25] J. M. Eisner. Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of the 16th conference on Computational linguistics-Volume 1*, pages 340–345, 1996.

[26] E. Eskin. Automatic corpus correction with anomaly detection. In *Proceedings of NAACL*, pages 148–153, 2000.

[27] J. R. Finkel and C. D. Manning. Hierarchical bayesian domain adaptation. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter*

of the Association for Computational Linguistics, pages 602–610. Association for Computational Linguistics, 2009.

[28] R. Florian, H. Hassan, A. Ittycheriah, H. Jing, N. Kambhatla, X. Luo, H. Nicolov, and S. Roukos. A statistical model for multilingual entity detection and tracking. Technical report, DTIC Document, 2004.

[29] L. Georgiadis. Arborescence optimization problems solvable by edmonds' algorithm. *Theoretical Computer Science*, 301(1):427–437, 2003.

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

[31] D. Gildea and M. Palmer. The necessity of parsing for predicate argument recognition. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 239–246. Association for Computational Linguistics, 2002.

[32] Y. Goldberg and J. Nivre. A dynamic oracle for arc-eager dependency parsing. In *COLING*, pages 959–976, 2012.

[33] R. A. Hudson. *Word grammar*. Blackwell Oxford, 1984.

[34] S. Husain and B. Agrawal. Analyzing parser errors to improve parsing accuracy and to inform tree banking decisions. *Linguistic Issues in Language Technology*, 7(1), 2012.

[35] R. Hwa. Supervised grammar induction using training data with limited constituent information. In *Proceedings of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistics*, pages 73–79. Association for Computational Linguistics, 1999.

[36] R. Hwa. Sample selection for statistical grammar induction. In *Proceedings of the 2000 Joint SIGDAT conference on Empirical methods in natural language processing and very large corpora: held in conjunction with the 38th Annual Meeting of the Association for Computational Linguistics-Volume 13*, pages 45–52. Association for Computational Linguistics, 2000.

[37] R. Hwa. On minimizing training corpus for parser acquisition. In *Proceedings of the 2001 workshop on Computational Natural Language Learning-Volume 7*, page 10. Association for Computational Linguistics, 2001.

[38] S. Jain. Employing oracle confusion for parse quality estimation : Ms thesis, 2013.

[39] S. Jain, N. Jain, B. Agrawal, and R. Sangal. Employing oracle confusion for parse quality estimation. In *Computational Linguistics and Intelligent Text Processing*, pages 213–226. Springer, 2015.

[40] K. Kaljurand. Checking treebank consistency to find annotation errors, 2004.

[41] V. Kordoni. Strategies for annotation of large corpora of multilingual spontaneous speech data. In *The workshop on Multilingual Corpora: Linguistic Requirements and Technical Perspectives held at Corpus Linguistics*. Citeseer, 2003.

[42] S. Kübler, R. McDonald, and J. Nivre. Dependency parsing. *Synthesis Lectures on Human Language Technologies*, 1(1):1–127, 2009.

[43] D. D. Lewis and J. Catlett. Heterogeneous uncertainty sampling for supervised learning. In *Proceedings of the eleventh international conference on machine learning*, pages 148–156, 1994.

[44] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini. Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330, 1993.

[45] A. K. McCallumzy and K. Nigamy. Employing em and pool-based active learning for text classification. In *Proc. International Conference on Machine Learning (ICML)*, pages 359–367. Citeseer, 1998.

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

[47] I. A. Mel čuk. *Dependency syntax: theory and practice*. SUNY Press, 1988.

[48] S. Miller, J. Guinness, and A. Zamanian. Name tagging with word clusters and discriminative training. In *HLT-NAACL*, volume 4, pages 337–342, 2004.

[49] S. A. Mirroshandel and A. Nasr. Active learning for dependency parsing using partially annotated sentences. In *Proceedings of the 12th International Conference on Parsing Technologies*, pages 140–149. Association for Computational Linguistics, 2011.

[50] J. Nilsson, S. Riedel, and D. Yuret. The conll 2007 shared task on dependency parsing. In *Proceedings of the CoNLL shared task session of EMNLP-CoNLL*, pages 915–932. sn, 2007.

[51] J. Nivre. 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, 2004.

[52] J. Nivre. Algorithms for deterministic incremental dependency parsing. *Computational Linguistics*, 34(4):513–553, 2008.

[53] J. Nivre, J. Hall, J. Nilsson, A. Chanev, G. Eryigit, S. Kübler, S. Marinov, and E. Marsi. Maltparser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13(02):95–135, 2007.

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

[55] M. Palmer, R. Bhatt, B. Narasimhan, O. Rambow, D. M. Sharma, and F. Xia. Hindi syntax: Annotating dependency, lexical predicate-argument structure, and phrase structure. In *The 7th International Conference on Natural Language Processing*, pages 14–17, 2009.

[56] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and

E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[57] A. Ramanathan and D. D. Rao. A lightweight stemmer for hindi. In *the Proceedings of EACL*, 2003.

[58] R. Řehůřek and P. Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA. `http://is.muni.cz/publication/884893/en`.

[59] B. Roark and M. Bacchiani. Supervised and unsupervised pcfg adaptation to novel domains. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 126–133. Association for Computational Linguistics, 2003.

[60] B. Settles. Active learning literature survey. *University of Wisconsin, Madison*, 52(55-66):11, 2010.

[61] D. M. Sharma, P. Mannem, J. vanGenabith, S. L. Devi, R. Mamidi, and R. Parthasarathi, editors. *Proceedings of the Workshop on Machine Translation and Parsing in Indian Languages*. The COLING 2012 Organizing Committee, Mumbai, India, December 2012.

[62] S. M. Shieber. *Evidence against the context-freeness of natural language*. Springer, 1987.

[63] M. Steedman, R. Hwa, S. Clark, M. Osborne, A. Sarkar, J. Hockenmaier, P. Ruhlen, S. Baker, and J. Crim. Example selection for bootstrapping statistical parsers. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 157–164. Association for Computational Linguistics, 2003.

[64] M. Steedman, M. Osborne, A. Sarkar, S. Clark, R. Hwa, J. Hockenmaier, P. Ruhlen, S. Baker, and J. Crim. 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, 2003.

[65] A. Tammewar, K. Singla, B. Agrawal, R. Bhat, and D. M. Sharma. Can distributed word embeddings be an alternative to costly linguistic features: A study on parsing hindi. In *Proceedings of the 6th Workshop on Statistical Parsing of Morphologically Rich Languages (SPMRL 2015)*, pages 21–30, 2015.

[66] M. Tang, X. Luo, and S. Roukos. 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, 2002.

[67] C. A. Thompson, M. E. Califf, and R. J. Mooney. Active learning for natural language parsing and information extraction. In *ICML*, pages 406–414. Citeseer, 1999.

[68] H. Van Halteren. *The detection of inconsistency in manually tagged text*. na, 2000.

[69] G. Van Noord. Error mining for wide-coverage grammar engineering. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, page 446. Association for Computational Linguistics, 2004.

[70] A. Volokh and G. Neumann. Automatic detection and correction of errors in dependency tree-banks. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers-Volume 2*, pages 346–350. Association for Computational Linguistics, 2011.

[71] F. Xia, O. Rambow, R. Bhatt, M. Palmer, and D. Misra Sharma. Towards a multi-representational treebank. *LOT Occasional Series*, 12:159–170, 2008.