

**To Design Chhattisgarhi and Hindi Part of Speech Tag Set And
implementing Chhattisgarhi and Hindi parser.**

A Thesis submitted

to

CHATTISGARH SWAMI VIVEKANAND TECHNICAL UNIVERSITY

Bhilai (C.G) , India

for

*the Award of Degree
of*

Bachelor of Engineering

in

Information Technology

By

Shubham kumar sahu

Enrollment No. : AP9174

Under the Guidance of
Prof. (Mr) Vikas Pandey
Assistant Professor



Department of Information Technology
Bhilai Institute of Technology
Bhilai-490020, CHHATTISGARH , INDIA
(An ISO 9002 Certified Institute)

session : 2014 – 2018

DECLARATION

I the undersigned solemnly declare that the report of the thesis work entitled **To Design Chhattisgarhi and Hindi Part of Speech Tag Set And implementing Chhattisgarhi and Hindi parser.**, is based on my own work carried out during the course of my study under the supervision of **Mr. Vikas Pandey**

I assert that the statements made and conclusions drawn are an outcome of the project work . I further declare that to the best of my knowledge and belief that the report does not contain any work which has been submitted for the award of any other degree/diploma/certificate in this university /deemed university of India or any other country.

(Signature of the Candidate)

Name of the candidate : Shubham kumar sahu
Roll No. : 3013314058
Enrollment No. : AP9174

Signature of Supervisor
Prof. (Mr.) Vikas Pandey
Reader
Information Technology
B..I.T, Bhilai, (C.G)

CERTIFICATE BY THE SUPERVISOR

This is to certify that the report of the thesis entitled **To Design Chhattisgarhi and Hindi Part of Speech Tag Set And implementing Chhattisgarhi and Hindi parser** is a record of research work carried out by Shubham kumar sahu bearing Roll No: 3013314058 & Enrollment No. AP9174 under my guidance and supervision for the award of Degree of Bachelor Of Engineering in Information Technology of Chhattisgarh Swami Vivekanand Technical University , Bhilai (C.G) , India .

To the best of my knowledge and belief the thesis

- i) Embodies the work of the candidate him/herself ,
- ii) Has duly been completed,
- iii) Fulfils the requirement of the Ordinance relating to the B.E. degree of the University and
- iv) Is up to the desired standard both in respect of contents and language for being referred to the examiners .

(Signature of the Supervisor)

Name : Prof. (Mr.) Vikas Pandey
 Designation : Asstt. Professor
 Department : Information Technology
 Name and Address of the Institute : B.I.T, Bhilai, (C.G)

Forwarded to Chhattisgarh Swami Vivekanand Technical University
 Bhilai

(Signature of Director/Principal)
 Bhilai Institute of Technology
 Durg -490020, CHHATTISGARH , INDIA

CERTIFICATE BY THE EXAMINERS

The thesis entitled **To Design Chhattisgarhi and Hindi Part of Speech Tag Set And implementing Chhattisgarhi and Hindi parser.** Submitted by Shubham kumar sahu (301334058 , AP9174.) has been examined by the undersigned as a part of the examination and is hereby recommended for the award of the degree of Bachelor of Engineering in Information Technology of Chhattisgarh Swami Vivekanand University , Bhilai (C.G) .

Internal Examiner
Date :

External Examiner
Date :

ACKNOWLEDGEMENT

We wish to acknowledge with a deep sense of hearty gratitude and indebtedness to Dr. Ani Thomas, (HOD Information Technology), who gave us this opportunity to experience project work & his valuable suggestion during this project have been invaluable.

We take this opportunity to voice & record our sincerest gratefulness towards our esteem Supervisor **Mr. Vikas Pandey** under whose able guidance the project work has been brought to completion.

Our heart leaps up in thankfulness for his benevolence & time to time help, valuable suggestions, constructive criticism & active interest in the successful completion of this project work.

We are also thankful to all our honorable teachers of the Information Technology Department and our parents whose valuable support helped us and kept us motivated all through.

(Signature of Student)
Name : Shubham kumar sahu
B.I.T, Bhilai, (C.G)

LIST OF SYMBOLS

1. O - Asymptotic Notation
2. θ - Asymptotic Notation
3. Σ - alphabet (finite set).

LIST OF ABBREVIATIONS

1. DNA – Deoxybric Nucleic Acid

LIST OF TABLES

1. Table 3.2 :

LIST OF FIGURES

Fig 2.2 (a):

Table of Contents

CHAPTER	TITLE	PAGE NO.
I	Introduction	7-9
II	Literature Review	10
	2.1	11
	2.2	12-13
III	Problem Identification	14
	3.1 Hardware & Software Requirement	15
	3.2	16
IV	Methodology	17
	4.1	18-23
	4.2	24
V	Result & Discussion	25
	5.1 Output Screen	26-28
	5.2 Coding	29-42
VI	Conclusion & scope of further work	43
	6.1 Advantage Of Project	44
	6.2 Future Scope Of Project	45
	Bibliography	46-47
	Appendix	

CHAPTER – I

INTRODUCTION

Introduction

Part of speech (POS) tagging is the process of assigning the part of speech tag or other lexical class marker to each and every word in a sentence. In many Natural Language Processing applications such as word sense disambiguation, information retrieval, information processing, parsing, question answering, and machine translation, POS tagging is considered as the one of the basic necessary tool. Identifying the ambiguities in language lexical items is the challenging objective in the process of developing an efficient and accurate POS Tagger. Literature survey shows that, for Indian languages, POS taggers were developed only in Hindi, Bengali, Punjabi and Dravidian languages. Some POS taggers were also developed generic to the Hindi, Bengali and Telugu languages. All proposed POS taggers were based on different Tagset, developed by different organization and individuals. This paper addresses the various developments in POS-taggers and POS-tagset for Indian language, which is very essential computational linguistic tool needed for many natural language processing (NLP) applications.

Part of speech tagging (POS tagging) has a crucial role in different fields of natural language processing (NLP) including machine translation. Parts-of-speech-tagging is defined as the process of assigning to each word in a sentence, a label which indicates the status of that word within some system of categorizing the words of that language according to their morphological and/or syntactic properties. A part-of-speech is a grammatical category, commonly including verbs, nouns, adjectives, adverbs, determiner, and so on. Some classic examples for POS Taggers available in English are Brill tagger, Treectagger, CLAWS tagger, online tagger ENG TWOL. Most of these methods used rule based, stochastic or morphological inputs. The Fig. 1 shows the development of various corpus and POS taggers using different approaches. The analysis of languages is a complex procedure in India. In Indian languages, most of natural language processing work has been done in Hindi, Tamil, Malayalam and Marathi. These languages have several part-of-speech taggers that use different mechanisms. Research on part-of-speech tagging has been closely tied to corpus linguistics. Earlier work in POS tagging for Indian languages was mainly based on rule based approaches. But the fact that rule-based method requires expert linguistic knowledge and handwritten rule. Due to the morphological richness of Indian languages, researchers faced a great difficulty to write complex linguistic rules and the rule based approach did not result well in many cases. Later, researchers shifted to stochastic and other approaches and developed some better POS taggers in various Indian languages. Even though stochastic methods need very large corpora to be effective, many successful POS were developed and used in various natural language processing tasks for Indian language. In most of the Indian languages, the ambiguity is the key issue that must be addressed and solved while designing a pos tagger. For different context words behave differently and hence the challenge is to correctly identify the POS tag of a token appearing in a particular context. This paper gives a survey on developments of various POS taggers in Indian languages. The following sections of this chapter are organized as follow. The first section gives a brief description about various attempts in POS taggers in Indian languages. The second section is about the different Tagset developed for Indian languages.

CHAPTER -II

Literature Review

Literature Review

As per records and my knowledge no previous work has been done for chhattisgarhi language has officially been made language recently only and no work has been done on this and for Hindi tag set are already decided by the government and for hindi language already previous works had been done

2.1 Hindi Parser-based on CKY algorithm

by nitin and ambrish

Parsing of Hindi in particular is not a very widely explored territory. There have been many attempts at developing a good Parser for Hindi. Parsing is the process of analyzing an input sequence in order to determine its grammatical structure with respect to a given formal grammar. Hindi parser is Efficient parser that parses Hindi language sentence. It parses Hindi language with respect to a formal grammar which I have been defining. It parses Hindi language at sentence level. It uses Cocke-Kasami-Younger (CKY) Parsing algorithm to parse sentence. CKY Parsing algorithm is a dynamic programming approach to parse context free grammar (CFG). Its worst case time complexity is- $\Theta(n^3)$ where n is the length of the string to be parsed. The standard version of CKY can only recognize languages defined by context-free grammars in Chomsky Normal Form (CNF)[1]. A CFG is a 4-tuple $G = (V, \Sigma, S, P)$, where V is a finite set of non-terminals and Σ is a finite set of terminal symbols, P is a finite set of production rules of the form $A \Rightarrow \alpha$ with $A \in V$, $\alpha \in (V \cup \Sigma)^*$ and $S \in V$ is a starting symbol. CNF is defined as a form having productions of the type either $A \Rightarrow BC$ or $A \Rightarrow a$, where A, B and C are non-terminal symbols and a is a terminal symbol.

2.2 HMM BASED POS TAGGER FOR HINDI

by hemant and iti

Part of Speech (POS) Tagging is the first step in the development of any NLP Application. It is a task which assigns POS labels to words supplied in the text. This is the reason why researchers consider this as a sequence labeling task where words are considered as sequences which needs to be labeled. Each word's tag is identified within a context using the previous word/tag combination. POS tagging is used in various applications like parsing where word and their tags are transformed into chunks which can be combined to generate the complete parse of a text. Taggers are used in Machine Translation (MT) while developing a transfer based MT Engine. Here, we require the text in the source language to be POS tagged and then parsed which can then be transferred to the target side using transfer grammar. Taggers can also be used in Name Entity Recognition (NER) where a word tagged as a noun (either proper or common noun) is further classified as a name of a person, organization, location, time, date etc. Tagging of text is a complex task as many times we get words which have different tag categories as they are used in different context. This phenomenon is termed as lexical ambiguity. For example, let us consider text in Table 1. The same word 'सोने' is given a different label in the two sentences. In the first case it is termed as a common noun as it is referring to an object (Gold Ornament). In the second case it is termed as a verb as it is referring to an experience (feelings) of the speaker. This problem can be resolved by looking at the word/tag combinations of the surrounding words with respect to the ambiguous word (the word which has multiple tags). Over the years, a lot of research has been done on POS tagging. Broadly, all the efforts can be categorized in three directions. They are: rule based approach where a human annotator is required to develop rules for tagging words or statistical approach where we use mathematical Computer Science & Information Technology (CS & IT) formulations and tag words or hybrid approach which is partially rule based and partially statistical. In the context of European languages POS taggers are generally

developed using machine learning approach, but in the Indian context, we still do not have a clear good approach. In this paper we discuss the development of a POS tagger for Hindi using Hidden Markov Model (HMM). सोने के आभूषण महंगे हो गए हैं NN PSP NN JJ VM VAUX VAUX उसका दल सोने का है PRP NN VM PSP VM Table 1. Example of Lexical Ambiguity The paper is organized with literature survey in Section 2 which is succeeded by section 3 which describes the HMM approach for POS tagging Hindi text and section 4 which shows evaluation results followed by section 5 which concludes the paper.

2.3 Shallow Parsing with Conditional Random Fields

by Fei Sha and Fernando Pereira

Sequence analysis tasks in language and biology are often described as mappings from input sequences to sequences of labels encoding the analysis. In language processing, examples of such tasks include part-of-speech tagging, named-entity recognition, and the task we shall focus on here, shallow parsing. Shallow parsing identifies the non-recursive cores of various phrase types in text, possibly as a precursor to full parsing or information extraction (Abney, 1991). The paradigmatic shallow parsing problem is NP chunking, which finds the nonrecursive cores of noun phrases called base NPs. The pioneering work of Ramshaw and Marcus (1995) introduced NP chunking as a machine-learning problem, with standard datasets and evaluation metrics. The task was extended to additional phrase types for the CoNLL- 2000 shared task (Tjong Kim Sang and Buchholz, 2000), which is now the standard evaluation task for shallow parsing. Most previous work used two main machine-learning approaches to sequence labeling. The first approach relies on k-order generative probabilistic models of paired input sequences and label sequences, for instance hidden Markov models (HMMs) (Freitag and McCallum, 2000; Kupiec, 1992) or multilevel Markov models (Bikel et al., 1999). The second approach views the sequence labeling problem as a sequence of classification problems, one for each of the labels in the sequence. The classification result at each position may depend on the whole input and on the previous k classifications. 1 The generative approach provides well-understood training and decoding algorithms for HMMs and more general graphical models. However, effective generative models require stringent conditional independence assumptions. For instance, it is not practical to make the label at a given position depend on a window on the input sequence as well as the surrounding labels, since the inference problem for the corresponding graphical model would be intractable. Non-independent features of the inputs, such as capitalization, suffixes, and surrounding words, are important in dealing with words unseen in training, but they are difficult to represent in generative models.

The sequential classification approach can handle many correlated features, as demonstrated in work on maximum-entropy (McCallum et al., 2000; Ratnaparkhi, 1996) and a variety of other linear classifiers, including winnow (Punyakanok and Roth, 2001), AdaBoost (Abney et al., 1999), and support-vector machines (Kudo and Matsumoto, 2001). Furthermore, they are trained to minimize some function related to labeling error, leading to smaller error in practice if enough training data are available. In contrast, generative models are trained to maximize the joint probability of the training data, which is not as closely tied to the accuracy metrics of interest if the actual data was not generated by the model, as is always the case in practice. However, since sequential classifiers are trained to make the best local decision, unlike generative models they cannot trade off decisions at different positions against each other. In other words, sequential classifiers are myopic about the impact of their current decision on later decisions (Bottou, 1991; Lafferty et al., 2001). This forced the best sequential classifier systems to resort to heuristic combinations of forward-moving and backward-moving sequential classifiers (Kudo and Matsumoto, 2001). Conditional random fields (CRFs) bring together the best of generative and classification models. Like classification models, they can accommodate many statistically correlated features of the inputs, and they are trained discriminatively. But like generative models, they can trade off decisions at different

sequence positions to obtain a globally optimal labeling. Lafferty et al. (2001) showed that CRFs beat related classification models as well as HMMs on synthetic data and on a part-of-speech tagging task. In the present work, we show that CRFs beat all reported single-model NP chunking results on the standard evaluation dataset, and are statistically indistinguishable from the previous best performer, a voting arrangement of 24 forward- and backward-looking support-vector classifiers (Kudo and Matsumoto, 2001). To obtain these results, we had to abandon the original iterative scaling CRF training algorithm for convex optimization algorithms with better convergence properties. We provide detailed comparisons between training methods. The generalized perceptron proposed by Collins (2002) is closely related to CRFs, but the best CRF training methods seem to have a slight edge over the generalized perceptron.

2.4 Bengali Part of Speech Tagging using Conditional Random Field

by asif and sivaji

Part of Speech (POS) tagging is the task of labeling each word in a sentence with its appropriate syntactic category called part of speech. POS tagging is a very important preprocessing task for language processing activities. This helps in doing deep parsing of text and in developing Information extraction systems, semantic processing etc. Part of Speech (POS) tagging for natural language texts are developed using linguistic rule, stochastic models and a combination of both. Stochastic models (Cutting, 1992; Merialdo, 1994; Brants, 2000) have been widely used in POS tagging task for simplicity and language independence of the models. Among stochastic models, Hidden 1

http://shiva.iiit.ac.in/SPSAL2007/iiit_tagset_guidelines.pdf Markov Models (HMMs) are quite popular. Development of a stochastic tagger requires large amount of annotated corpus. Stochastic taggers with more than 95% word-level accuracy have been developed for English, German and other European languages, for which large labeled data is available. The problem is difficult for Indian languages (ILs) due to the lack of such annotated large corpus. Simple HMMs do not work well when small amount of labeled data are used to estimate the model parameters. Incorporating diverse features in an HMM-based tagger is difficult and complicates the smoothing typically used in such taggers. In contrast, a Maximum Entropy (ME) based method (Ratnaparkhi, 1996) or a Conditional Random Field (CRF) based method (Lafferty et al., 2001) can deal with diverse, overlapping features. (Smriti et al., 2006) proposed a POS tagger for Hindi that has an accuracy of 93.45% with the exhaustive morphological analysis backed by high coverage lexicon and a decision tree based learning algorithm (CN2). International Institute of Information Technology (IIIT), Hyderabad, India initiated a POS tagging contest, NLPAL Contest06 2 for the Indian languages in 2006. Several teams came up with various approaches and the highest accuracies were 82.22% for Hindi, 84.34% for Bengali and 81.59% for Telugu. As part of the SPSAL20073 workshop in IJCAI-07, a competition on POS tagging and chunking for south Asian languages was conducted by IIIT, Hyderabad. The best accuracies reported were 78.66% for Hindi (Avinesh and Karthick, 2007), 77.37% for Telugu (Avinesh and Karthick, 2007) and 77.61% for Bengali (Sandipan, 2007). 2

http://lrc.iiitnet/nlpai_contest06 3 <http://shiva.iiit.ac.in/SPSAL2007> 131 In this paper, we have developed a POS tagger based on Conditional Random Field (CRF) that has shown an accuracy of 87.3% with the contextual window of size four, prefix and suffix of length upto three, NE information of the current and the previous words, POS information of the previous word, digit features, symbol features and the various gazetteer lists. It has been experimentally shown that the accuracy of the POS tagger can be improved significantly by introducing lexicon (Ekbal and Bandyopadhyay, 2007a), named entity recognizer (Ekbal et al., 2007b) and word suffix features for handling the unknown words. Experimental results show the effectiveness of the proposed model with an accuracy of 90.3%.

CHAPTER -III

PROBLEM IDENTIFICATION

Problem Identification

The identification of research problem is the first and foremost step that every researcher has to undertake. At times, it becomes rather difficult for an inexperienced researcher or a novice/beginner in research to conceptualize a research problem. In general, a research problem should be understood as some difficulty, unclear situation which a researcher experiences in practical or theoretical context and wants to obtain a tangible explanation, clarification or offer solution to it.

As the state Chhattisgarh was formed in the year 2000 which was earlier the part of the state Madhya Pradesh so the language of Chhattisgarh has now been made so as per our concern no work has been done for this language so we are putting our efforts towards this so as to make proper tag sets and parser for this language so this language can be more developed and will be used in digital formats too.

Chhattisgarhi is an official language in the Indian state of Chhattisgarh. Spoken by 17.5 million people. In this paper we will see the work that has been done in the field of natural language processing (NLP) using Chhattisgarhi language and other state languages. The main goal of NLP is to create machine learning, create translator, create dictionary and create POS tagger. POS tagger is one of the important tools that are used to develop language translator and information extraction so that computer-based systems be compatible for natural language processing. Part-of-speech tagging is the process of assigning a part-of-speech like noun, verb, pronoun, preposition, adverb, adjective or other lexical class marker to each word in a sentence. There are different types of POS taggers that exist, are based on probabilistic approach and some based on morphological approaches. So in this paper we will see various developments of POS tagger and the major work has been done using Chhattisgarhi and other Indian state languages.

So basically with the help of our technical and programming knowledge we are going to make a tag set and a language parser for Chhattisgarhi language and tag set for Hindi language has already been decided by the Indian government so for Hindi language we are only going to make a language parser. So the problem here is to find out the corpus of Chhattisgarhi language and we need to learn with the experts about the language, and need to learn the tools for tagging and creating softwares.

Hardware & Software Requirement :

Processor – Intel(R) Core(TM) i5 processor CPU @ 2.5 GHz.

Memory – RAM size 4GB

Disk Space Required – 20 GB

Software Requirements – Java, Sanchay.

Operating System – Windows/ Ubuntu(64- bit OS)

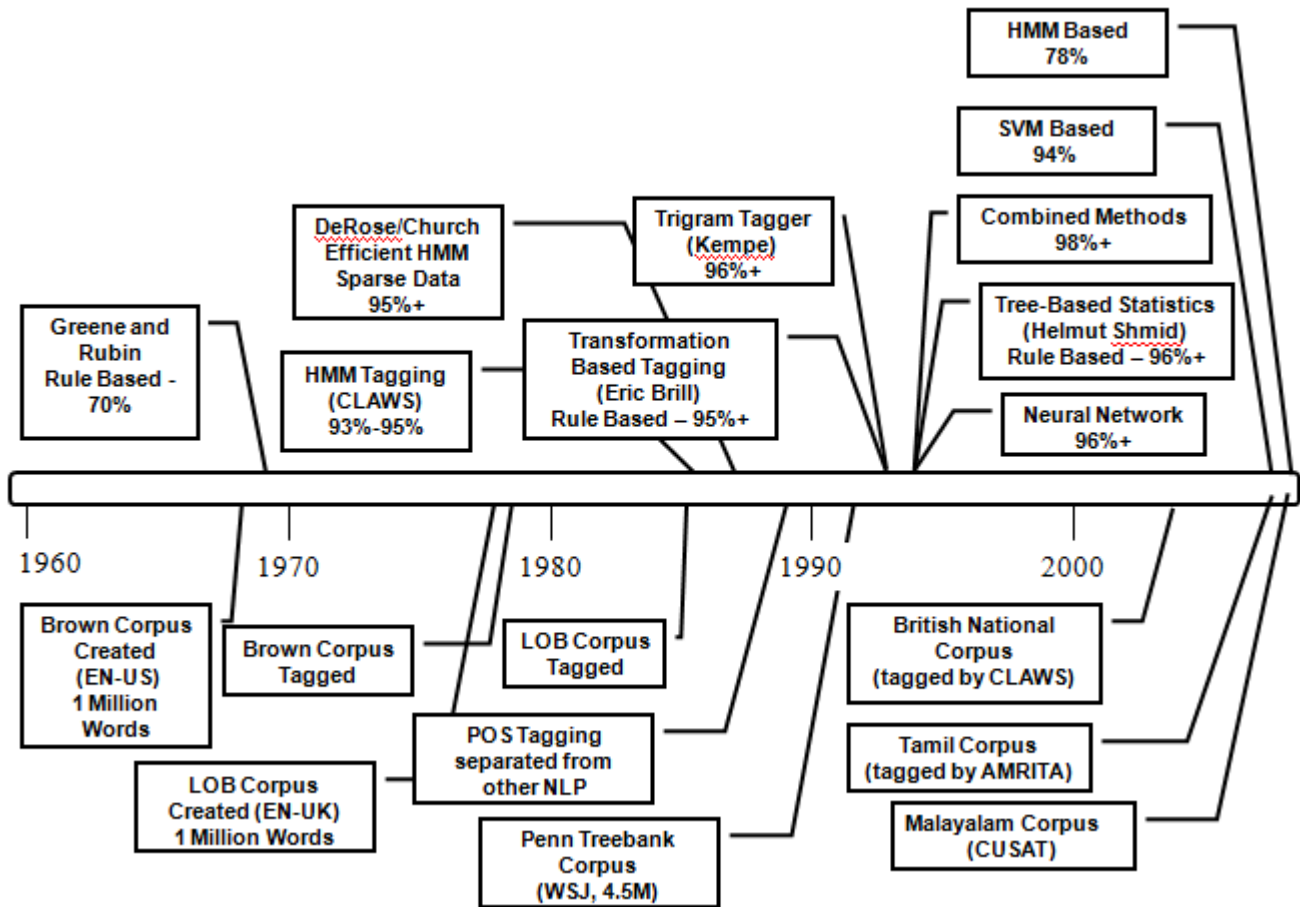
CHAPTER -IV

METHODOLOGY

Methodology

PARTS OF SPEECH TAGGING FOR INDIAN LANGUAGES

Some classic examples for POS Taggers available in English are Brill tagger, Tree tagger, CLAWS tagger and online tagger ENGTWOL. In Indian languages, most of the natural language processing work has been done in Hindi, Tamil, Malayalam and Marathi. These languages have several part-of-speech taggers based on different mechanisms. Research on part-of-speech tagging has been closely tied to corpus linguistics. The Fig. 2.3 shows the development of various corpus and POS taggers using different approaches.

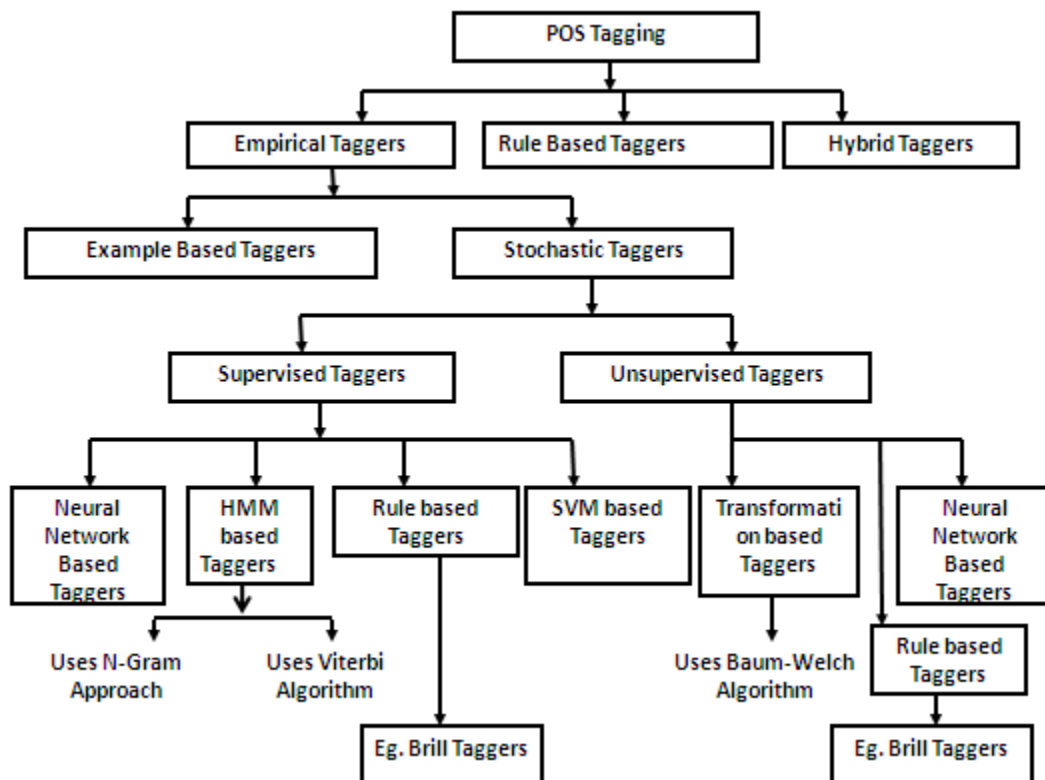


Earlier work in POS tagging for Indian languages was mainly based on rule based approaches. But the fact that rule-based method requires expert linguistic knowledge and hand written rule. Due to the morphological richness of Indian languages, researchers faced a great difficulty to write complex linguistic rules and the rule based approach did not result well in many cases. Later, researchers shifted to stochastic and other approaches and developed some better POS taggers in various Indian languages. Even though stochastic methods need very large corpora to be effective, many successful POS taggers were developed and used in various NLP tasks for Indian language. In most of the Indian languages, the ambiguity is the key issue that must be addressed and solved while designing a POS tagger. For different context, words behave differently and hence the challenge is to correctly identify the POS tag of a token appearing in a particular context. Literature survey shows that, for Indian languages, POS taggers were developed only in Hindi, Bengali, Panjabi and Dravidian languages. Some noticeable attempts were done in Dravidian languages like Tamil, Telugu and Malayalam except Kannada language. Some POS taggers were also

developed generic to the Hindi, Bengali and Telugu languages. All proposed POS taggers were based on various Tagset, developed by different organization and individuals. This section gives a survey on developments of various POS taggers in Indian languages. The following sub sections are organized as follows: The first sub section gives a brief description about various attempts in POS taggers in Indian languages. The second sub section is about the different Tagset developed for Indian languages.

Parts of Speech Tagging Approaches

POS taggers are broadly classified into three categories called Rule based, Empirical based and Hybrid based. In case of rule based approach, hand-written rules are used to distinguish the tag ambiguity. The empirical POS taggers are further classified into Example based and Stochastic based taggers. Stochastic taggers are either HMM based, choosing the tag sequence which maximizes the product of word likelihood and tag sequence probability, or cue-based, using decision trees or maximum entropy models to combine probabilistic features. The stochastic taggers are further classified into supervised and unsupervised taggers. Each of these supervised and unsupervised taggers are categorized into different groups based on the particular algorithm used. The Fig. 2.4 below shows the classification of parts of speech approaches. In the recent literature, several approaches to POS tagging based on statistical and machine learning techniques are applied, including: HMMs, Maximum Entropy taggers, Transformation-based learning, Memory-based learning, Decision Trees, AdaBoost, and Support Vector Machines. Most of the taggers have been evaluated on the English WSJ corpus, using the Penn Treebank set of POS categories and a lexicon constructed directly from the annotated corpus. Although the evaluations were performed with slight variations, there was a wide consensus in the late 90's that the state-of-the-art accuracy for English POS tagging was between 96.4% and 96.7%. In the recent years, the most successful and popular taggers in the NLP community have been the HMM-based TnT tagger, the Transformation Based Learning (TBL) tagger and several variants of the Maximum Entropy (ME) approach.



POS Taggers for Hindi Language

A number of POS taggers were developed in Hindi language using different approaches. In the year 2006, three different POS tagger systems were proposed based on Morphology driven, ME and CRF approaches respectively. There are two attempts for POS tagger developments in 2008, both are based on HMM approaches and proposed by Manish Shrivastava and Pushpak Bhattacharyya. Nidhi Mishra and Amit Mishra proposed a Part of Speech Tagging for Hindi Corpus in 2011. In an another attempt, a POS tagger algorithm for Hindi was proposed by Pradipta Ranjan Ray, Harish V., Sudeshna Sarkar and Anupam Basu.

i) In the first attempt, Smriti Singh proposed a POS tagging methodology which can be used by languages having lack of resources [39]. The POS tagger is built based on hand-crafted morphology rules and does not involve any sort of learning or disambiguation process. The system makes use of locally annotated modestly-sized corpora of 15,562 words, exhaustive morphological analysis backed by high-coverage lexicon and a decision tree based learning algorithm called CN2. The tagger also uses the affix information stored in a word and assigns a POS tag by taking in consideration the previous and the next word in the Verb Group (VG) to correctly identify the main verb and the auxiliaries. The system uses Lexicon lookup for identifying the other POS categories. The performance of the system was evaluated by a 4-fold cross validation over the corpora and found 93.45% accuracy.

ii) Aniket Dalal, Kumar Nagaraj, Uma Sawant and Sandeep Shelke, proposed a POS tagger based on ME based approach [39]. To develop a POS tagger based on ME approach requires feature functions extracted from a training corpus. Normally a feature function is a boolean function which captures some aspect of the language which is relevant to the sequence labelling task. The experiment showed that the performance of the system depend on size of the training corpus. There is an increase in performance till it reaches 75% of the training corpus after which there is a reduction in accuracy due to over fitting of the trained model to training corpus. The least and best POS tagging accuracy of the system was found to be 87.04% and 89.34% and the average accuracy over 10 runs was 88.4%.

iii) The third POS tagger is based Conditional Random Fields developed by Agarwal Himashu and Amni Anirudh in 2006 [39]. This system makes use of Hindi morph analyzer for training purpose and to get the root-word and possible POS tag for every word in the corpus. The training is performed with CRF++ and the training data also contains other information like suffixes, word length indicator and special characters. A corpus size of 1, 50,000 words were used for training and testing purposes and accuracy of the system was 82.67%.

iv) The HMM based approach was intended to utilize the morphological richness of the languages without resorting to complex and expensive analysis [39]. The core idea of this approach was to explode the input in order to increase the length of the input and to reduce the number of unique types encountered during learning. This idea increases the probability score of the correct choice and at the same time decreasing the ambiguity of the choices at each stage. Data sparsity also decreases by new morphological forms for known base words. Training and testing were performed with an exploded corpus size of 81751 tokens, which were divided into 80% and 20% parts respectively.

v) An improved Hindi POS tagger was developed by employing a naive (longest suffix matching) stemmer as a pre-processor to the HMM based tagger [40]. Apart from a list of possible suffixes, which can be

easily created using existing machine learning techniques for the language, this method does not require any linguistic resources. The reported performance of the system was 93.12%.

vi) Nidhi Mishra and Amit Mishra proposed a Part of Speech Tagging for Hindi Corpus in 2011 [41]. In the proposed method, the system scans the Hindi corpus and then extracts the sentences and words from the given corpus. Also the system search the tag pattern from database and display the tag of each Hindi word like noun tag, adjective tag, number tag, verb tag etc.

vii) Based on lexical sequence constraints, a POS tagger algorithm for Hindi was proposed by Pradipta Ranjan Ray, Harish V., Sudeshna Sarkar and Anupam Basu [42]. The proposed algorithm acts as the first level of part of speech tagger, using constraint propagation, based on ontological information, morphological analysis information and lexical rules. Even though the performance of the POS tagger has not been statistically tested due to lack of lexical resources, it covers a wide range of language phenomenon and accurately captures the four major local dependencies in Hindi.

POS for Hindi

tag set decided by

Department of Information Technology
Ministry of Communications & Information Technology
Govt. of India

Sl. No	Category			Label	Annotation Convention**	Examples	Remarks
	Top level	Subtype (level 1)	Subtype (level 2)				
1	Noun			N	N	ladakaa, raajaa, kitaaba	
1.1		Common		NN	N_NN	kitaaba, kalama, cashmaa	
1.2		Proper		NNP	N_NNP	Mohan, ravi, rashmi	
1.4		Nloc		NST	N_NST	Uupara, nñice, aage, pñiche	
2	Pronoun			PR	PR	Yaha, vaha, jo	
2.1		Personal		PRP	PR_PRP	Vaha, main, tuma, ve	
2.2		Reflexive		PRF	PR_PRF	Apanaa, swayam, khuda	
2.3		Relative		PRL	PR_PRL	Jo, jis, jab, jahaaM,	
2.4		Reciprocal		PRC	PR_PRC	Paraspara, aapasa	
2.5		Wh-word		PRQ	PR_PRQ	Kauna, kab, kahaaM	
		Indefinite		PRI	PR_PRI	Koii, kis	

3	Demonstrative			DM	DM	Vaha, jo, yaha,	
3.1		Deictic		DMD	DM_DMD	Vaha, yaha	
3.2		Relative		DMR	DM_DMR	jo, jis	
3.3		Wh-word		DMQ	DM_DMQ	kis, kaun	
		Indefinite		DMI	DM_DMI	KoI, kis	
4	Verb			V	V	giraa, gayaa, sonaa, haMstaa, hai, rahaa	
4.1		Main		VM	V_VM	giraa, gayaa, sonaa, haMstaa,	
4.2		Auxiliary		VAUX	V_VAUX	hai, rahaa, huaa,	
5	Adjective			JJ	JJ	sundara, acchaa, baRaa	
6	Adverb			RB	RB	jaldii, teza	
7	Postposition			PSP	PSP	ne, ko, se, mein	
8	Conjunction			CC	CC	aur, agar, tathaa, kyonki	
8.1		Co-ordinator		CCD	CC_CCD	aur, balki, parantu	
8.2		Subordinator		CCS	CC_CCS	Agar, kyonki, to, ki	
9	Particles			RP	RP	to, bhii, hii	
9.1		Default		RPD	RP_RPD	to, bhii, hii	
9.3		Interjection		INJ	RP_INJ	are, he, o	
9.4		Intensifier		INTF	RP_INTF	bahuta, behada	
9.5		Negation		NEG	RP_NEG	nahiin, mata, binaa	
10	Quantifiers			QT	QT	thoRaa, bahuta, kucha, eka, pahalaa	
10.2		Cardinals		QTC	QT_QTC	eka, do, tiina,	
10.3		Ordinals		QTO	QT_QTO	pahalaa, duusaraa	
11	Residuals			RD	RD		
11.1		Foreign word		RDF	RD_RDF		A word written in script other than the script of the original text
11.2		Symbol		SYM	RD_SYM	\$, &, *, (,)	For symbols such as \$, & etc
11.3		Punctuation		PUNC	RD_PUNC	., : ;	Only for punctuations
11.4		Unknown		UNK	RD_UNK		
11.5		Echowords		ECH	RD_ECH	(Paanii-) vaanii, (khaanaa-) vaanaa	

POS tagset for chhattisgarhi

tag set for any indian languages decided by

Department of Information Technology
Ministry of Communications & Information Technology
Govt. of India

Sl. No	Category			Label	Annotation Convention**	Remarks
	Top level	Subtype (level 1)	Subtype (level 2)			
1	Noun			N	N	
1.1		Common		NN	N_NN	
1.2		Proper		NNP	N_NNP	The verbal noun sub type is only for languages such as Tamil and Malayalam)
1.3		Verbal		NNV	N_NNV	
1.4		Nloc		NST	N_NST	
2	Pronoun			PR	PR	
2.1		Personal		PRP	PR_PRP	
2.2		Reflexive		PRF	PR_PRF	
2.3		Relative		PRL	PR_PRL	
2.4		Reciprocal		PRC	PR_PRC	
2.5		Wh-word		PRQ	PR_PRQ	
2.6		INDEFINITE		PRI	PR_PRI	
3	Demonstrative			DM	DM	
3.1		Deictic		DMD	DM_DMD	
3.2		Relative		DMR	DM_DMR	
3.3		Wh-word		DMQ	DM_DMQ	
3.4		Indefinite		DMI	DM_DMI	
4	Verb			V	V	
4.1		Main		VM	V_VM	
4.1.1			Finite	VF	V_VM_VF	
4.1.2			Non-finite	VNF	V_VM_VNF	
4.1.3			Infinitive	VINF	V_VM_VINF	
4.1.4			Gerund	VNG	V_VM_VNG	
4.2		Verbal		VN	V_VN	<i>paTitam,</i>

						<i>naTattam, naTanam</i>
4.2		Auxiliary		VAUX	V_VAUX	
4.2.1			Finite	VAUX	V_VAUX_VF	
4.2.2			Non-finite	VNF	V_VAUX_VNF	
4.2.3			Infinitive	VINF	V_VAUX_VINF	
4.2.4			Gerund	VNG	V_VAUX_VNG	
4.2.5			PARTICIP LE NOUN	VNP	V_VAUX_VNP	
5	Adjective			JJ		
6	Adverb			RB		Only manner adverbs
7	Postposition			PSP		
8	Conjunction			CC	CC	
8.1		Co-ordinator		CCD	CC_CCD	
8.2		Subordinator		CCS	CC_CCS	
8.2.1			Quotative	UT	CC_CCS_UT	
9	Particles			RP	RP	
9.1		Default		RPD	RP_RPD	
9.2		Classifier		CL	RP_CL	
9.3		Interjection		INJ	RP_INJ	
9.4		Intensifier		INTF	RP_INTF	
9.5		Negation		NEG	RP_NEG	
10	Quantifiers			QT	QT	
10.1		General		QTF	QT_QTF	
10.2		Cardinals		QTC	QT_QTC	
10.3		Ordinals		QTO	QT_QTO	
11	Residuals			RD	RD	
11.1		Foreign word		RDF	RD_RDF	A word written in script other than the script of the original text
11.2		Symbol		SYM	RD_SYM	For symbols such as \$, & etc
11.3		Punctuation		PUNC	RD_PUNC	Only for punctuations
11.4		Unknown		UNK	RD_UNK	
11.5		Echowords		ECH	RD_ECH	

MORPHOLOGICAL APPROACHES AND SURVEY FOR INDIAN LANGUAGES

Morphology plays an essential role in MT and many other NLP applications. Developing a well fledged MAG tools for highly agglutinative languages is a challenging task. The function of morphological analyzer is to return all the morphemes and their grammatical categories associated with a particular word form. For a given root word and grammatical information, morphological generator will generate the particular word form of that word.

The morphological structure of an agglutinative language is unique and capturing its complexity in a machine analyzable and generatable form is a challenging job. Analyzing the internal structure of a particular word is an important intermediate stage in many NLP applications especially in bilingual and multilingual MT system. The role of morphology is very significant in the field of NLP, as seen in applications like MT, QA system, IE, IR, spell checker, lexicography etc. So from a serious computational perspective the creation and availability of a morphological analyzer for a language is important. To build a MAG for a language one has to take care of the morphological peculiarities of that language, specifically in case of MT. Some peculiarities of language such as, the usage of classifiers, excessive presence of vowel harmony etc. make it morphologically complex and thus, a challenge in NLG.

The literature shows that development of morphological analysis and generation work has been successfully done for languages like English, Chinese, Arabic and European languages using various approaches from last few years. A few attempts have been made in developing morphological analysis and generation tool for Indian languages. Even though the morphological analyzer and generator play an important role in MT, literature shows that it is still an ongoing process for various Indian languages. This paper addresses the various approaches and developments in morphological analyzer and generator for Indian language.

The first sub section of this section discusses various approaches that are used for building morphological analyzer and generator tool for Indian languages. The second subsection gives a brief explanation about different morphological analyzer and generator developments for Indian languages.

Morphological Analyzer and Generator Approaches

There are many language dependent and independent approaches used for developing morphological analyzer and generator [63]. These approaches can also be classified generally into corpus based, rule based and algorithmic based. The corpus based approach, where a large sized well generated corpus is used for training with a machine learning algorithm. The performance of the system depends on the feature and size of the corpus. The disadvantage is that corpus creation is a time consuming process. On the other hand, rule based approaches are based on a set of rules and dictionary that contains root and morphemes. In rule based approaches every rule depends on the previous rule. So if one rule fails, it will affect the entire rules that follow it. When a word is given as an input to the morphological analyzer and if the corresponding morphemes are missing in the dictionary then the rule based system fails. Literature shows that there are number of successful morphological analyzer and generator development for languages like English, Chinese, Arabic and European languages using these approaches [24]. Recent developments in Indian language NLP shows that many morphological analyzer and generators are created successfully using these approaches. Brief descriptions of most commonly used approaches are as follows:

Corpus Based Approach

In case of corpus based approach, a large sized well generated corpus is required for training. Any machine learning algorithm is used to train the corpus and collect the statistical information and other necessary features from the corpus. The collected information is used as a MAG model. The

performance of the system will depend on the feature and size of the corpus. The disadvantage is that corpus creation is a time consuming process. This approach is suitable for languages having well organized corpus.

Paradigm Based Approach

For a particular language, each word category like nouns, verbs, adjectives, adverbs and postpositions will be classified into certain types of paradigms. Based on their morphophonemic behavior, a paradigm based morphological compiler program is used to develop MAG model. In the paradigm approach a linguist or the language expert is asked to provide different tables of word forms covering the words in a language. Based on this information and the feature structure with every word form, a MAG can be built. The paradigm based approach is also well suited for highly agglutinative language nature. So paradigm based approach or the variant of this scheme has been used widely in NLP. Literature shows that morphological analyzers are developed for almost all Indian languages using paradigm based approach.

Finite State Automata Based Approach

Finite state machine or Finite State Automation (FSA) or finite automation uses regular expressions to accept or reject a string in a given language [64]. In general, an FSA is used to study the behavior of a system composing of state, transitions and actions. When FSA starts working, it will be in the initial stage and if the automation is in any one of the final state it accepts the input and stops working.

Two- Level Morphology Based Approach

In 1983, Kimmo Koskenniemi, a Finnish computer scientist developed a general computational model for word-form recognition and generation called Two- level morphology [64]. This development was one of the major breakthroughs in the field of morphological parsing, which is based on morphotactics and morphophonemics concepts. The advantage of two- level morphology is that the model does not depend on a rule compiler, composition or any other finite-state algorithm. The "two-level" morphological approach consists of two levels called lexical and surface form and a word is represented as a direct, letter-for-letter correspondence between these forms. The Two-level morphology approach is based on the following three ideas:

- Rules are symbol-to-symbol constraints that are applied in parallel, not sequentially like rewrite rules.
- The constraints can refer to the lexical context, to the surface context, or to both contexts at the same time.

• Lexical lookup and morphological analysis are performed in tandem.

2.4.1.5 FST Based Approach
FST is a modified version of FSA by accepting the principles of a two level morphology. A FST, essentially is a finite state automaton that works on two (or more) tapes. The most common way to think about transducers is as a kind of “translating machine” which works by reading from one tape and writing onto the other. FST’s can be used for both analysis and generation (they are bidirectional) and it acts as a two level morphology. By combining the lexicon, orthographic rules and spelling variations in the FST, we can build a morphological analyzer and generator at once.

Stemmer Based Approach

Stemmer uses a set of rules containing list of stems and replacement rules to stripping of affixes. It is a program oriented approach where the developer has to specify all possible affixes with replacement rules. Potter algorithm is one of the most widely used stemmer algorithm and it is freely available. The

advantage of stemmer algorithm is that it is very suitable to highly agglutinative languages like Dravidian languages for creating MAG.

Suffix Stripping Based Approach

Highly agglutinative languages such as Dravidian languages, a MAG can be successfully built using suffix stripping approach. The advantage of the Dravidian language is that no prefixes and circumfixes exist for words. Words are usually formed by adding suffixes to the root word serially. This property can be well suited for suffix stripping based MAG. Once the suffix is identified, the stem of the whole word can be obtained by removing that suffix and applying proper orthographic (sandhi) rules. A set of dictionaries like stem dictionary, suffix dictionary and also using morphotactics and sandhi rules, a suffix stripping algorithm successfully implements MAG.

Directed Acrylic Word Graph Based Approach

Directed Acrylic Word Graph (DAWG) is a very efficient data structure that can be used for developing both morphological analyzer and generator. DAWG is language independent and it does not utilize any morphological rules or any other special linguistic information. But DAWG is very suitable for lexicon representation and fast stringmatching, with a great variety of application. Using this approach, the University of Partas Greece developed MAG for Greek language for the first time. There after the method was applied for other languages including Indian languages.

PARSING APPROACHES AND SURVEY FOR INDIAN LANGUAGES

Syntactic parsing is the task of recognizing a sentence and assigning a syntactic structure to it. A syntactic parser is an essential tool used for various NLP applications and natural language understanding. Literature survey on natural language parsing reveals that majority of the parsing techniques were developed specifically for English, Arabic and European languages using different approaches. Literature shows that the rule based grammar refinement process is extremely time consuming and difficult and also failed to analyze accurately a large corpus of unrestricted text. Hence, most modern parsers are based on statistical or at least partly statistical, which allows the system to gather information about the frequency with which various constructions occur in specific contexts. Every statistical approach requires the availability of aligned corpora which are: large, of good-quality and representative.

Parsing of sentences is considered to be an important intermediate stage for semantic analysis in NLP application such as IR, IE and QA. The study of structure of sentence is called syntax. It attempts to describe the grammatical order in a particular language in term of rules which in detail explain the underlying structure and a transformational process. Syntax provides rules to put together words to form components of sentences and to put together these components to form meaningful sentences. In NLP, syntactic parsing or more formally syntactic analysis is the process of analyzing and determining the structure of a text which is made up of sequence of tokens with respect to a given formal grammar. Because of the substantial ambiguity present in the human language, whose usage is to convey different semantics, it is much difficult to design the features for NLP tasks. The main challenge is the inherent complexity of linguistic phenomena that makes it difficult to represent the effective features for the target learning models.

Indian languages are highly inflectional, relatively free word order and agglutinative. Because of these features most of these techniques cannot be applied to Indian language context directly. Most of the Indian language parsers were developed recently and still it is an ongoing process. This section addresses the various approaches and parsing developments for Indian languages.

Parsing Approaches

A well known parsing approach known as Nivre's parser was successfully implemented in a variety of languages like relatively free-word order language like Turkish, inflectionally rich language like Hindi, fixed word order language like English, and relatively case-less and less inflectional language like Swedish. Another simple approach called CFG formalism was used in languages like Dutch, Turkish and English to develop parsers. In order to suit the context of Indian languages, a formalism called „Paninian Grammar Framework“ was developed. Collin's and Mc-Donald's parser are the other well known parsing techniques. Generally, natural language parsing can be broadly classified into three categories: (i) rule based (ii) statistical based and (iii) generalized parsers [77]. All the developed parsers belong to any one of these categories and follow either „top-down“ or „bottom-up“ direction. Statistical and rule based parsing techniques are called „data-driven“ and „grammar-driven“ approaches respectively.

Rule Based Parsers

A rule- based parser uses the hard – coded rules to identify the best parse tree for a given grammar. In a rule based parsing, production rules are applied recursively and as a result overlapping problem may arise. Dynamic programming (DP) technique can be used to solve the overlapping problem efficiently. The cache for sub parse trees in the DP-based parsers is called the „chart“ and consequently the DP-based parsers are called „chart parsers“. The CYK algorithm developed by Cocke (1970), Kasami (1965), Younger (1967) and Early algorithm developed by Jurafsky and Martin, in 2000 belong to rule based parsers.

Statistical Based Parsers

The main effort in parsing of a sentence is to resolve the ambiguities. It is very hard to write complex rules to resolve such ambiguities. In contrast to the rule based approach, statistical parsing algorithms collect statistical data from correctly parsed sentences, and resolves ambiguity by experience. The advantage of statistical approach is that it covers the whole grammar usage of the language. The performance of the statistical parsers depends on training corpus used to gather statistical information about the grammar of the language. Instead of using rules to find the correct parse tree, statistical parsers select the best parse tree from possible candidates based on the statistical information. Sampson proposed the first statistical based parsing in 1986, using a manually designed language model based on a set of transition networks, and a stimulated annealing decoding search PCFG algorithm. CFG and based parsers are the examples for statistical parsers.

Generalized parsing

The framework behind both rule based and statistical parsing are similar. Using this advantage, Goodman proposed a general parsing algorithm based on semiring idea. In the year 2005, Melamed suggested another generalized parsing algorithm which was based on semiring parsing. Melamed generalized algorithm consists of five components such as: grammar, logic, semiring, search strategy and termination condition. As the name suggests, grammar defines terminal and non-terminal symbols, as well as a set of production rules. Logic defines the mechanism of how the parser runs by generating new partial parse trees. The semiring defines how partial parse trees are scored. The search strategy defines the order in which partial parse trees are processed and the termination condition defines when to stop the logic necessarily

Parser Developments in Indian Languages: A Literature Survey

A series of statistically based parsers for English are developed by various researchers namely: Charniak-1997, Collins-2003, Bod et al. - 2003 and Charniak and Johnson- 2005 [86, 87]. All these parsers are trained and tested on the standard benchmark corpora called Wall Street Journal (WSJ). A probability model for a lexicalized PCFG was developed by Charniak in 1997. In the same time Collins described three generative parsing models, where each model is a refinement on the previous one, and achieved improved performance. In 1999 Charniak introduced a much better parser called maximum-entropy parsing approach. This parsing model is based on a probabilistic generative model and uses a maximum-entropy inspired technique for conditioning and smoothing purposes. In the same period Collins also presented a statistical parser for Czech using the Prague Dependency Treebank. The first statistical parsing model based on a Chinese Treebank was developed in 2000 by Bikel and Chiang. A probabilistic Treebank based parser for German was developed by Dubey and Keller in 2003 using a syntactically annotated corpus called „Negra“. The latest addition to the list of available Treebank is the „French Le Monde“ corpus and it was made available for research purposes in May 2004. Ayesha Binte Mosaddeque & Nafid Haque wrote CFG for 12 Bangla sentences that were taken from a newspaper [76]. They used a recursive descent parser for parsing the CFG.

Even though natural language parsers play an important role in MT and other natural language applications, it is still an ongoing process for Indian languages. Comparing with foreign languages, a very little work has been done in the area of NLP for Indian languages. This section will give a brief description about various developments contributed towards natural language parsing in Indian languages.

- i) In the year 2009, B.M. Sagar, Shobha G and Ramakanth Kumar P proposed a way of producing context free grammar for the Noun Phrase and Verb Phrase agreement in Kannada Sentences [21]. In this approach, a recursive descent parser is used to parse the CFG. The system works in two stages: First, it generates the context free grammar of the sentence. In the second stage, a recursive descent parser called Recursive Descent Parser of Natural Language Tool Kit (NLTK) was used to test the grammar. As a summary, it is a grammar checking system such that for a given sentence parser says whether the sentence is syntactically correct or wrong depending upon the Noun and Verb agreement. They have tested the system using around 200 sample sentences and obtained encouraging results.
- ii) Natural Language constructs for Venpa class of Tamil Poetry using Context Free Grammar was implemented by Bala Sundara Raman L, Ishwar S, and Sanjeeth Kumar Ravindranath in 2003 [79]. They used Push Down Automata parser to parse the CFG in the proposed system.
- iii) A rule based grammar checking mechanism for Hindi sentences was developed by Singh and D.K. Lobiyal in 1993 [80]. The system is suitable for all types of sentences with compound, conjunct or complex verb phrases. In the proposed model, verb and suffixes have been divided into finer subcategories to simplify the process of associating semantics with syntax. Using Lexical Functional Grammar formalism, the base grammar rules are being augmented with functional equations. This technique is used to bridge the gap between syntax and semantics of a sentence. They have developed a parser for the grammar. The grammar rules are assigned priority in a manner that the most frequently applicable rule gets higher priority than the less frequently applicable rule. The system works in such a way that, the grammar rules get fired on priority basis to make the parsing efficient.
- iv) Selvam M, Natarajan A M, and Thangarajan R proposed a statistical parsing of Tamil sentences using phrase structure hybrid language model in the year 2008 [81]. They have built a statistical language model based on Trigram for Tamil language with medium of 5000 words. In the experiment they showed that statistical parsing gives better performance through tri-gram probabilities and large

vocabulary size. In order to overcome some disadvantages like focus on semantics rather than syntax, lack of support in free ordering of words and long term relationship of the system, a structural component is to be incorporated. The developed hybrid language model is based on a part of speech tagset for Tamil language with more than 500 tags. The developed phrase structured Treebank was based on 326 Tamil sentences which covers more than 5000 words. The phrase structured Treebank was trained using immediate head parsing technique. Two test cases with 120 and 40 sentences have been selected from trained set and test set respectively. They reported that, the performance of the system is better than the grammar model.

v) Akshar Bharati and Rajeev Sangal described a grammar formalism called the „Paninian Grammar Framework“ that has been successfully applied to all free word Indian languages [82]. They have described a constraint based parser for the framework. Paninian framework uses the notion of karaka relations between verbs and nouns in a sentence. They showed that the Paninian framework applied to modern Indian languages will give an elegant account of the relation between vibhakti and karaka roles and that the mapping is elegant and compact.

vi) B.M. Sagar, Shobha G and Ramakanth Kumar P proposed a CFG Analysis for simple Kannada sentences in 2010 [21]. They have explained the writing of CFG for a simple Kannada sentence with two sets of examples. In the proposed system, a grammar is parsed with Top-Down and Bottom-Up parsers and they found that a Top-Down parser is more suitable to parse the given grammatical production.

vii) A dependency parser system for Bengali language was developed by Aniruddha Ghosh, Pinaki Bhaskar, Amitava Das and Sivaji Bandyopadhyay in 2009 [83]. They have performed two separate runs for Bengali. A statistical CRF based model followed by a rule-based post-processing technique has been used. They have used ICON 2009 datasets for training the system. They have trained the probabilistic sequence model with the morphological features like root word, POS-tag, chunk tag, vibhakti and dependency relation from the training set data. The output of the baseline CRF based system is filtered by a rule-based post-processing module by using the output obtained through the rule based dependency parser. The system demonstrated an Unlabelled Attachment Score (UAS) of 74.09%, labelled attachment score (LAS) of 53.90% and labelled accuracy score (LS) of 61.71% respectively.

viii) Sengupta, P and B.Chaudhuri proposed a delayed syntactic-encoding-based LFG parsing strategy for an Indian Language- Bangla, in 1997 [84]. This was just an attempt in parsing of Bengali language based on the detection and formation of the proper rule set to identify characteristics of inter-chunk relations. They have tested the system on a sample of about 250 simple and complex sentences picked from newspaper clippings. Results show that, even though phrasal orderings were quite random, almost all simple sentences in active voice were correctly parsed.

ix) Parsing of Indian Languages using the freely available Malt- Parser system was developed by Joakim Nivre in 2009 [85]. He developed transition-based dependency parsers using Malt- Parser system for three Indian languages like Bangla, Hindi and Telugu. With the Malt- Parsing technique he showed that, parsing can be performed in linear time for projective dependency trees and quadratic time for arbitrary trees. A small test set of 150 sentences was used to analyse the performance of the system. The performance of the system was slightly better for Bangla and Hindi languages but for Telugu it was lower than the baseline results.

x) Hiring world's leading dependency parsers to plant Indian trees, a voting parser was proposed by Daniel Zeman in 2009 [86] called Maximum Spanning Malt (MST). The system consists of three existing, freely available dependency parsers, two of which (MST and Malt) have been known to produce state-of-the-art structures on data sets for other languages. Various settings of the parsers were

explored in order to adjust them for the three Indian languages like Hindi, Bengali and Telugu, and a voting approach was used to combine them into a super parser. He showed that „case“ and „vibhakti“ are important features for parsing Hindi while their usability in Bangla and Telugu is limited by data sparseness. Based on these features, he developed best combined parsers for these languages.

xi) A constraint based Dependency parsing has been attempted and applied to a free-word order language Bangla by Sankar, Arnab Dhar and Utpal Garain in 2009 [87]. They have used a structure simplification and demand satisfaction approach to dependency parsing in Bangla language. A well known and very effective grammar formalism for free word order language called Paninian Grammatical model was used for this purpose. The main idea behind this approach was to simplify complex and compound sentential structures first, then to parse the simple structures so obtained by satisfying the „Karaka“ demands of the VGs and to rejoin such parsed structures with appropriate links and Karaka labels. A Treebank of 1000 annotated sentences was used for training the system. The performance of the system was evaluated with 150 sentences and achieves accuracies of 90.32%, 79.81%, and 81.27% for unlabelled attachments, labelled attachments and label scores, respectively.

xii) Bharat Ram Ambati, Phani Gadde and Karan Jindal explored two data-driven parsers called Malt and MST on three Indian languages namely Hindi, Telugu and Bangla in 2009 [88]. In their experiment, they merged both the training and development data and did 5-fold cross-validation for tuning the parsers. They also extracted best settings from the cross validation experiments and these settings are applied on the test data of the contest. Finally they evaluated the individual and average results on both coarse-grained and fine-grained tagset for all the three languages. They observed that for all the languages Malt performed better over MST+maxent. They also modified the implementation of MST to handle vibhakti and TAM markers for labelling. They reported that, the average of best unlabelled attachment, labelled attachment and labelled accuracies are 88.43%, 71.71% and 73.81% respectively.

xiii) A hybrid approach for parsing Bengali sentences was proposed by Sanjay Chatterji, Praveen Sonare, Sudeshna Sarkar and Devshri Roy in 2009 [89]. The system was based on data driven dependency parser. In order to improve the performance of the system, some handcrafted rules are identified based on the error patterns on the output of the baseline system.

xiv) A constraint based Hindi dependency parser was developed by Meher Vijay Yeleti and Kalyan Deepak in 2009 [90]. In the proposed system a grammar driven approach was complemented by a controlled statistical strategy to achieve high performance and robustness. The proposed system uses two stage constraint based hybrid approach to dependency parsing. They defined two stages and this division leads to selective identification and resolution of specific dependency relations at the two stages. They also used hard constraints and soft constraints to build an efficient and robust hybrid parser. From the experiment they found out that the best labelled and unlabelled attachment accuracies for Hindi are 62.20% and 85.55% respectively.

xv) Prashanth Mannem proposed a bidirectional dependency parser for Hindi, Telugu and Bangla languages in 2009 [91]. The developed parser uses a bidirectional parsing algorithm with two operations projection and non-projection to build the dependency tree. The performance of the proposed parser was evaluated based on the test data sentences. He reported that the system achieves a labelled attachment score of 71.63%, 59.86% and 67.74% for Hindi, Telugu and Bangla respectively on the treebank with fine-grained dependency labels. Based on the coarse-grained labels the dependency parser achieved 76.90%, 70.34% and 65.01% accuracies respectively.

xvi) Pradeep Kumar Das proposed a generative approach to the computation of basic verbal-strings in Hindi in 2008 [92]. He described a way to examine the possibility of developing a computational parser

for verb morphology in Hindi that would generate correct verbal stems for different kinds of tense and aspects.

xvii) A parsing criteria for Assamese text was described by Navanath Saharia, Utpal Sharma and Jugal Kalita in 2011 [93]. They described the practical analysis of Assamese sentences from a computational perspective rather than from linguistics perspective. This approach can be used to parse the simple sentences with multiple noun, adjective and adverb clauses.

xviii) An attempt to study the semantic relation of Causality or Cause-Effect was proposed by Sobha Lalitha Devi and Menaka S in 2011 [94]. They also described how semantic relation of Causality is marked in Tamil, how the causal markers in Tamil manifest in texts, their syntactic and semantic properties and how this information can be represented so as to handle causal information and reasoning.

xix) Akshar Bharati, Mridul Gupta, Vineet Yadav, Karthik Gali and Dipti Misra Sharma proposed a simple parser for Indian Languages in a dependency framework in 2009 [95]. They described a parser which uses a grammar driven approach to annotate dependency relations in both inter and intra chunk at an intermediary level. They described a grammar oriented model that makes use of linguistic features to identify relations. The proposed parser was modelled based on Paninian grammatical approach which provides a dependency grammar framework. They also compared the proposed parser performance against the previous similar attempts and reported its efficiency.

xx) An approach to expedite the process of manual annotation of a Hindi dependency Treebank was described by Gupta, Vineet Yadav, Samar Husain and Dipti Misra Sharma [96]. They proposed a way by which consistency among a set of manual annotators could be improved. They have also showed that their setup can be useful for evaluating, when an inexperienced annotator is ready to start participating in the production of the treebank. The performance of the system was evaluated on sample sets of data.

xxi) An unlabelled dependency parsing on graph based method for building multilingual weakly supervised dependency parsers for Hindi language was proposed by Jagadeesh Gorla, Anil Kumar Singh, Rajeev Sangal, Karthik Gali, Samar Husain, and Sriram Venkatapathy [78]. The system consists of two steps where the first step involves marking the chunks and the chunk heads of a given sentence and then identifying the intra-chunk dependency relations. The second step involves learning to identify the inter-chunk dependency relations. They reported that the system achieved a precision of 80.83% for sentences of size less than 10 words and 66.71% overall. They concluded that the result obtained was significantly better than the baseline in which random initialization is used.

MACHINE LEARNING

Machine learning deals with techniques that allow computers automatically learn to make accurate predictions based on past observations. The major focus of machine learning is to extract information from data automatically, by computational and statistical methods. Machine learning techniques are being used for solving many tasks of Natural Language processing. They include speech recognition, document categorization, document segmentation, part-of-speech tagging, and word-sense disambiguation, named entity recognition, parsing, MT and transliteration.

There are two main tasks involved in machine learning; learning/training and prediction. The system is given with a set of examples called training data. The primary goal is to automatically acquire effective and accurate model from the training data. The training data provides the domain knowledge i.e., characteristics of the domain from which the examples are drawn. This is a typical task for inductive learning and is usually called concept learning or learning from examples. The larger the amount of training data, usually the better the model will be. The second phase of machine learning is the prediction, wherein a set of inputs are mapped into the corresponding target values. The main challenge of machine learning is to create a model, with good prediction performance on the test data i.e., model with good generalization on unknown data.

Machine learning algorithms are categorized based on the desired outcome of the algorithm into Supervised learning, Unsupervised learning, Semi-supervised learning, Reinforcement learning and Transduction. In the proposed work all the statistical based computational linguistic tools are developed using supervised machine learning approach. In supervised learning the target function is completely specified by the training data. There is a label associated with each example. If the label is discrete, then the task is called classification. Otherwise, for real valued labels, the task becomes a regression problem. Based on the examples in the training data, the label for new case is predicted. Hence, learning is not only a question of remembering but also of generalization to unseen cases. Any change in the learning system can be seen as acquiring some kind of knowledge. So, depending on what the system learns, the learning is categorized as

- **Model Learning:** The system predicts values of unknown function. This is called as prediction and is a task well known in statistics. If the function is discrete, the task is called classification. For continuous-valued functions it is called regression.
- **Concept learning:** The systems acquire descriptions of concepts or classes of objects.
- **Explanation-based learning:** Using traces (explanations) of correct (or incorrect) performances, the system learns rules for more efficient performance of unseen tasks.
- **Case-based (exemplar-based) learning:** The system memorizes cases (exemplars) of correctly classified data or correct performances and learns how to use them (e.g. by making analogies) to process unseen data.

CHAPTER -V

Result & Discussion

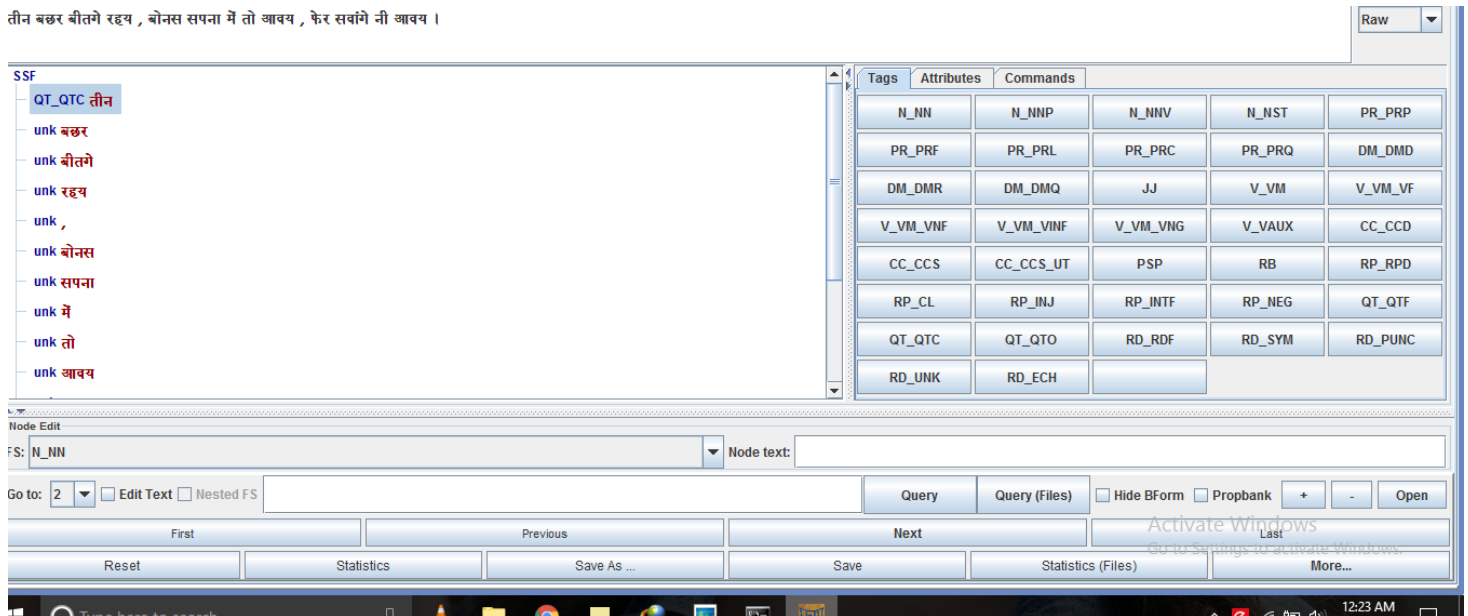
Result& Discussion

Finally we have done tagging of 69000 words manually and formed a automatic tagger using CRF++ tool and a shallow parser is made using open source code provided by the IIIT Hyderabad for shallow parser

Output screens and results are :-

Manual Tagging

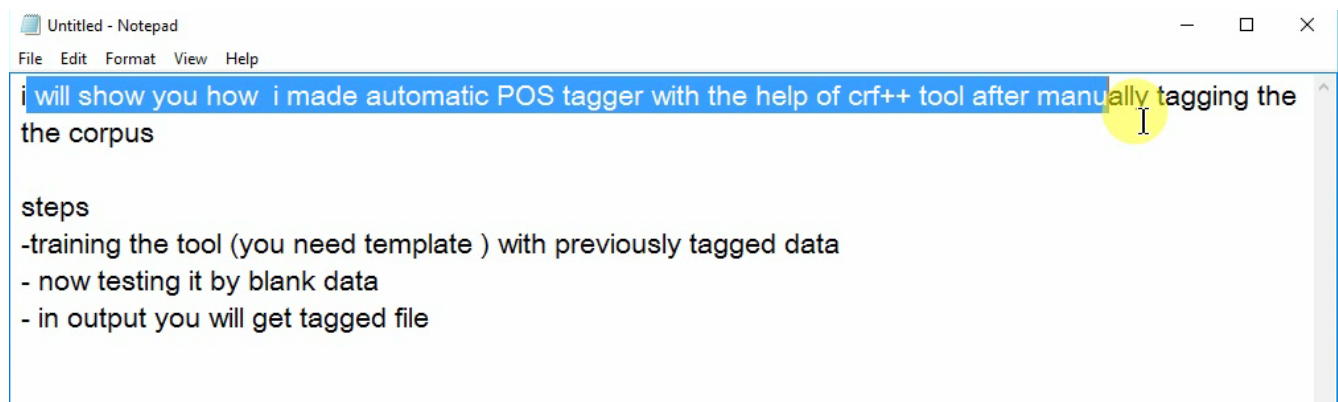
Manual tagging is done by the Sanchay software provided by the IIIT hyderabad



Automatic Tagging

Automatic tagger we have made using the CRF++ tool we have modified the tool according to our use for tagging the hindi language words

method for making pos tagger



learning of tagger

```

C:\Windows\system32\cmd.exe

C:\Users\shubham\Desktop\Minor project\Softwares\CRF++-0.58>crf_learn template train.txt model
CRF++: Yet Another CRF Tool Kit
Copyright (C) 2005-2013 Taku Kudo, All rights reserved.

reading training data:
Done!0.01 s

Number of sentences: 1
Number of features: 495
Number of thread(s): 4
Freq:
eta:      0.00010
C:        1.00000
shrinking size: 20
iter=0 terr=0.90476 serr=1.00000 act=495 obj=46.14172 diff=1.00000
iter=1 terr=0.00000 serr=0.00000 act=495 obj=38.97204 diff=0.15538
iter=2 terr=0.00000 serr=0.00000 act=495 obj=27.42572 diff=0.29627
iter=3 terr=0.00000 serr=0.00000 act=495 obj=26.80543 diff=0.02262
iter=4 terr=0.00000 serr=0.00000 act=495 obj=26.80215 diff=0.00012
iter=5 terr=0.00000 serr=0.00000 act=495 obj=26.80214 diff=0.00000
iter=6 terr=0.00000 serr=0.00000 act=495 obj=26.80214 diff=0.00000
iter=7 terr=0.00000 serr=0.00000 act=495 obj=26.80214 diff=0.00000

Done!0.09 s

C:\Users\shubham\Desktop\Minor project\Softwares\CRF++-0.58>crf_test -m model test.txt >out.txt
C:\Users\shubham\Desktop\Minor project\Softwares\CRF++-0.58>

```

testing or output of tagger

```

C:\Windows\system32\cmd.exe

C:\Users\shubham\Desktop\Minor project\Softwares\CRF++-0.58>crf_learn template train.txt model
CRF++: Yet Another CRF Tool Kit
Copyright (C) 2005-2013 Taku Kudo, All rights reserved.

reading training data:
Done!0.00 s

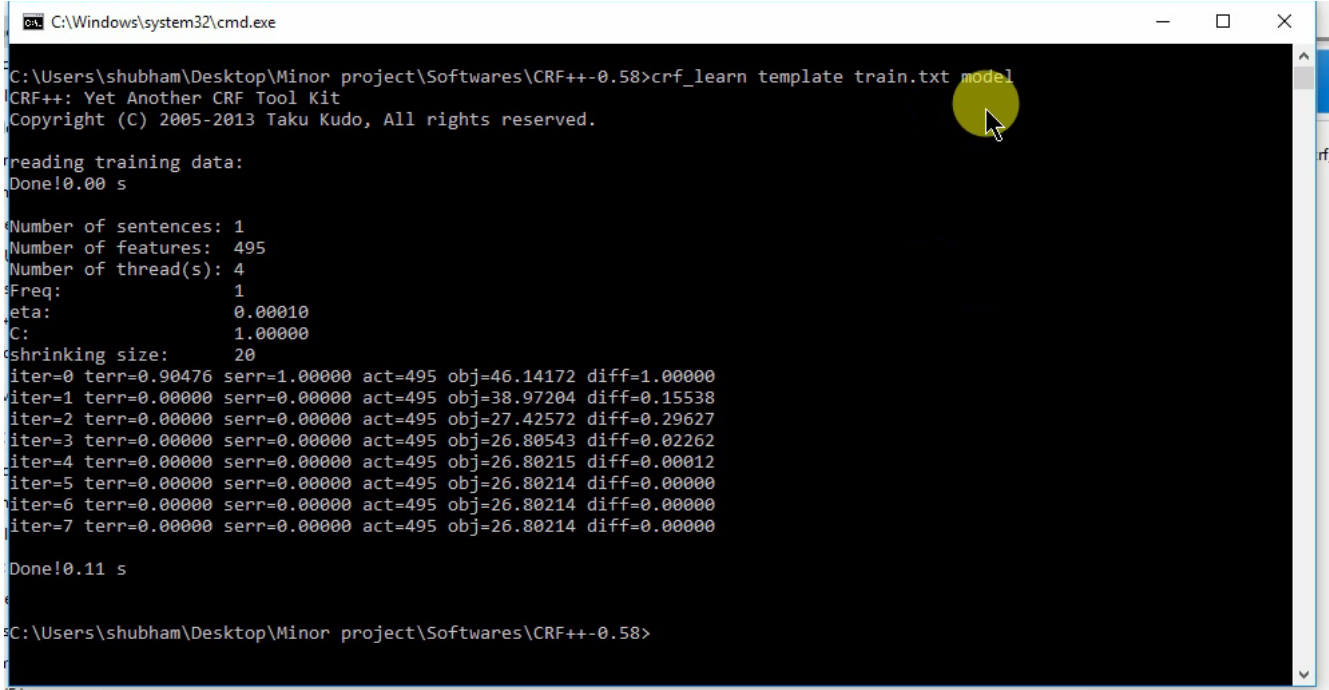
Number of sentences: 1
Number of features: 495
Number of thread(s): 4
Freq:
eta:      0.00010
C:        1.00000
shrinking size: 20
iter=0 terr=0.90476 serr=1.00000 act=495 obj=46.14172 diff=1.00000
iter=1 terr=0.00000 serr=0.00000 act=495 obj=38.97204 diff=0.15538
iter=2 terr=0.00000 serr=0.00000 act=495 obj=27.42572 diff=0.29627
iter=3 terr=0.00000 serr=0.00000 act=495 obj=26.80543 diff=0.02262
iter=4 terr=0.00000 serr=0.00000 act=495 obj=26.80215 diff=0.00012
iter=5 terr=0.00000 serr=0.00000 act=495 obj=26.80214 diff=0.00000
iter=6 terr=0.00000 serr=0.00000 act=495 obj=26.80214 diff=0.00000
iter=7 terr=0.00000 serr=0.00000 act=495 obj=26.80214 diff=0.00000

Done!0.11 s

C:\Users\shubham\Desktop\Minor project\Softwares\CRF++-0.58>crf_test -m model test.txt > tagged.txt
C:\Users\shubham\Desktop\Minor project\Softwares\CRF++-0.58>

```

template for tagger and using a model for tagger



```
C:\Windows\system32\cmd.exe

C:\Users\shubham\Desktop\Minor project\Softwares\CRF++-0.58>crf_learn template train.txt model
CRF++: Yet Another CRF Tool Kit
Copyright (C) 2005-2013 Taku Kudo, All rights reserved.

reading training data:
Done!0.00 s

Number of sentences: 1
Number of features: 495
Number of thread(s): 4
Freq: 1
eta: 0.00010
C: 1.00000
shrinking size: 20
iter=0 terr=0.90476 serr=1.00000 act=495 obj=46.14172 diff=1.00000
iter=1 terr=0.00000 serr=0.00000 act=495 obj=38.97204 diff=0.15538
iter=2 terr=0.00000 serr=0.00000 act=495 obj=27.42572 diff=0.29627
iter=3 terr=0.00000 serr=0.00000 act=495 obj=26.80543 diff=0.02262
iter=4 terr=0.00000 serr=0.00000 act=495 obj=26.80215 diff=0.00012
iter=5 terr=0.00000 serr=0.00000 act=495 obj=26.80214 diff=0.00000
iter=6 terr=0.00000 serr=0.00000 act=495 obj=26.80214 diff=0.00000
iter=7 terr=0.00000 serr=0.00000 act=495 obj=26.80214 diff=0.00000

Done!0.11 s

C:\Users\shubham\Desktop\Minor project\Softwares\CRF++-0.58>
```


Parser for Hindi

Shallow Parser Hindi (Version 4.0)

Shallow Parser Hindi :

The Hindi shallow parser gives the analysis of a Hindi sentence at various levels. The analysis begins at the morphological level and accumulates at results of POS tagger and chunker*. The final output combines the results of all these levels and shows them in a single representation (called Shakti Standard Format)**.

1- Requirements:

Operating System : LINUX (tested on Fedora-8 lower versions)

Compiler/Interpreter/Librarie(s): g++, Perl and SSF API's

2- Directory Structure:

shallow-parser-hin-4.0

```

---bin
  |--sys
  |  |--convertor
  |  |  |--hin
  |  |--sl
  |  |  |--tokenizer
  |  |--morph
  |--postagger
  |  |--chunker
  |--pruning
  |  |--pickonemorph
  |  |--headcomputation
  |--vibhakticomputation
---data_bin
  |--sl
  |  |--morph
  |--postagger
  |  |--chunker
  |--pruning

```

```
|---doc
```

```
|--tests
```

```
---README-developer
```

```
---README-user
```

---README-INSTALL-FAQ

To see the Intermediate output of the modules check
OUTPUT.tmp/filename.tmp

2- Documents

[i] doc/chunk_pos_ann_guidelines.pdf : Chunking and POS annotation guidelines

[ii] doc/ssf-guide-4oct07.pdf : Shakti Standard Format Guide

- [iii] doc/wx-chart.jpg : WX notation

Running method

Shallow Parser Hindi (Version 4.0)

Shallow Parser Hindi :

The Hindi shallow parser gives the analysis of a Hindi sentence at various levels. The analysis begins at the morphological level and accumulates at results of POS tagger and chunker*. The final output combines the results of all these levels and shows them in a single representation (called Shakti Standard Format)**.

0 - To Install:

Please check INSTALL file.

1 - How To Run??

To run shallow parser hindi from command prompt with the following command:

```
shallow_parser_hin < INPFILE > OUTFILE
(or)
shallow_parser_hin INPFILE OUTFILE
```

The above will run the parser on an input file called INPFILE containing a single sentence of Hindi in UTF-8 (Unicode UTF-8), and produce the output shallow parse in OUTFILE in UTF-8.

Note: The shallow parser hindi runs with a single sentence in text form in the INPFILE.

1.1 Handling other encodings (in case your input file is not in UTF-8 but --- in the popular wx encoding):

```
shallow_parser_hin --in_encoding=wx --out_encoding=utf \
< INPFILE > OUTFILE
```

*sample test file is in tests/shallowparser_hin_wx.rin

1.2 Getting output of different stages of shallow parser (such as morph --- analyzer, POS tagger, chunker, etc.):

```
shallow_parser_hin --mode=debug < INPFILE > OUTFILE
```

Note: To check the intermediate output check OUTPUT.tmp in current directory.

1.3 For help regarding options:

```
shallow_parser_hin --help
```

1.4 General form with all the options:

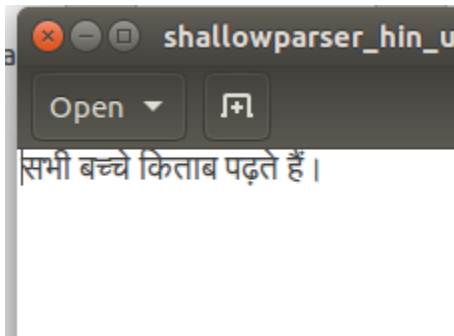
```
shallow_parser_hin --mode=debug --in_encoding=wx \
--out_encoding=utf --input=tests/shallowparser_hin_wx.rin
```

Note: To check the intermediate output check
 OUTPUT.tmp/shallowparser_hin_wx.rin.tmp

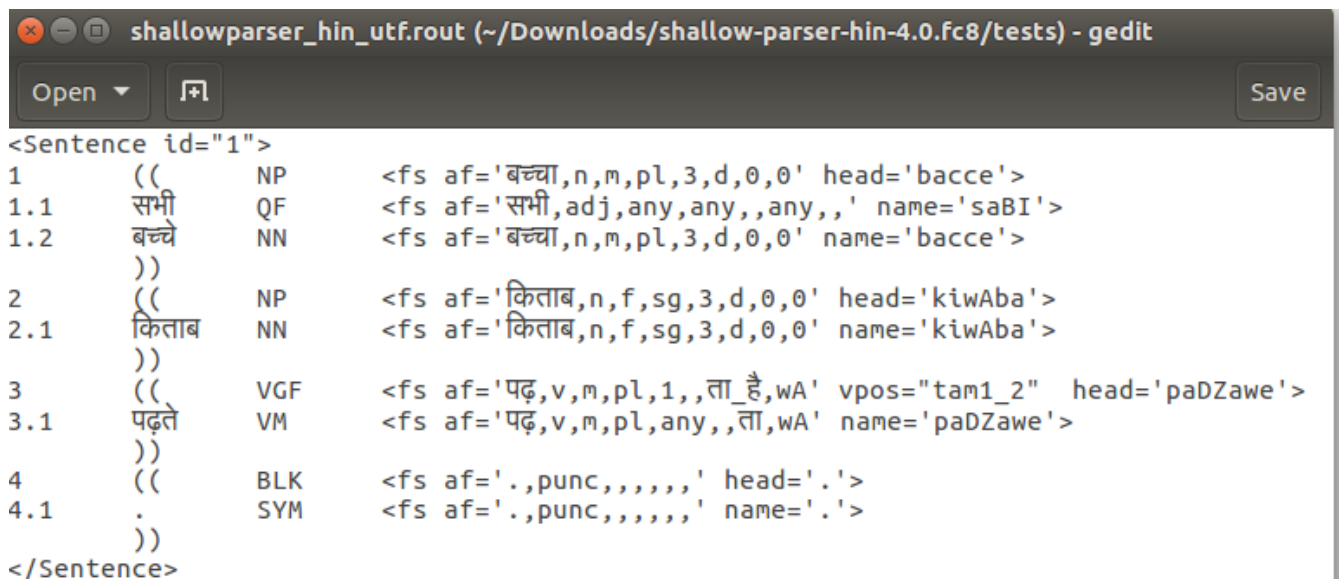
shallow_parser_hin [options]

```
--mode=fast|debug
    Creates intermediate files in "debug" mode in
    OUTPUT.tmp/filename.tmp
--in_encoding=wx|utf
    encoding of the input text .
--out_encoding=wx|utf
    encoding of the output text.
--input
    input file
--output
    output file (default stdout)
-h, --help
    display help and exit
-v, --version
    output version information and exit
```

input sample



output of the sample



Coding

Tagger header file code

```

/*
  CRF++ -- Yet Another CRF toolkit

  $Id: crfpp.h 1592 2007-02-12 09:40:53Z taku $;

  Copyright(C) 2005-2007 Taku Kudo <taku@chasen.org>
*/
#ifndef CRFPP_CRFPP_H_
#define CRFPP_CRFPP_H_

/* C interface */
#ifdef __cplusplus
#include <cstdio>
#else
#include <stdio.h>
#endif

#ifdef __cplusplus
extern "C" {
#endif

#ifdef _WIN32
#include <windows.h>
# ifdef DLL_EXPORT
#   define CRFPP_DLL_EXTERN __declspec(dllexport)
#   define CRFPP_DLL_CLASS_EXTERN __declspec(dllexport)
# else
#   define CRFPP_DLL_EXTERN __declspec(dllimport)
# endif
#endif

#ifndef CRFPP_DLL_EXTERN
# define CRFPP_DLL_EXTERN extern
#endif

#ifndef CRFPP_DLL_CLASS_EXTERN
# define CRFPP_DLL_CLASS_EXTERN
#endif

#ifndef SWIG
typedef struct crfpp_t crfpp_t;
typedef struct crfpp_model_t crfpp_model_t;

/* C interface */
CRFPP_DLL_EXTERN crfpp_model_t* crfpp_model_new(int, char**);
CRFPP_DLL_EXTERN crfpp_model_t* crfpp_model_new2(const char*);
CRFPP_DLL_EXTERN crfpp_model_t* crfpp_model_from_array_new(int, char**, const
char *, size_t);
CRFPP_DLL_EXTERN crfpp_model_t* crfpp_model_from_array_new2(const char*, const
char *, size_t);
CRFPP_DLL_EXTERN const char * crfpp_model_get_template(crfpp_model_t*);
CRFPP_DLL_EXTERN void crfpp_model_destroy(crfpp_model_t*);
CRFPP_DLL_EXTERN const char * crfpp_model_strerror(crfpp_model_t *);

```

```

CRFPP_DLL_EXTERN crfpp_t*    crfpp_model_new_tagger(crfpp_model_t *);

CRFPP_DLL_EXTERN crfpp_t* crfpp_new(int, char**);
CRFPP_DLL_EXTERN crfpp_t* crfpp_new2(const char*);
CRFPP_DLL_EXTERN void    crfpp_destroy(crfpp_t*);
CRFPP_DLL_EXTERN int     crfpp_set_model(crfpp_t *, crfpp_model_t *);
CRFPP_DLL_EXTERN int     crfpp_add2(crfpp_t*, size_t, const char **);
CRFPP_DLL_EXTERN int     crfpp_add(crfpp_t*, const char*);
CRFPP_DLL_EXTERN size_t  crfpp_size(crfpp_t*);
CRFPP_DLL_EXTERN size_t  crfpp_xsize(crfpp_t*);
CRFPP_DLL_EXTERN size_t  crfpp_dsize(crfpp_t*);
CRFPP_DLL_EXTERN const float* crfpp_weight_vector(crfpp_t*);
CRFPP_DLL_EXTERN size_t  crfpp_result(crfpp_t*, size_t);
CRFPP_DLL_EXTERN size_t  crfpp_answer(crfpp_t*, size_t);
CRFPP_DLL_EXTERN size_t  crfpp_y(crfpp_t*, size_t);
CRFPP_DLL_EXTERN size_t  crfpp_ysize(crfpp_t*);
CRFPP_DLL_EXTERN double  crfpp_prob(crfpp_t*, size_t, size_t);
CRFPP_DLL_EXTERN double  crfpp_prob2(crfpp_t*, size_t);
CRFPP_DLL_EXTERN double  crfpp_prob3(crfpp_t*);
CRFPP_DLL_EXTERN void    crfpp_set_penalty(crfpp_t *, size_t i, size_t j, double
penalty);
CRFPP_DLL_EXTERN double  crfpp_penalty(crfpp_t *, size_t i, size_t j);
CRFPP_DLL_EXTERN double  crfpp_alpha(crfpp_t*, size_t, size_t);
CRFPP_DLL_EXTERN double  crfpp_beta(crfpp_t*, size_t, size_t);
CRFPP_DLL_EXTERN double  crfpp_emission_cost(crfpp_t*, size_t, size_t);
CRFPP_DLL_EXTERN double  crfpp_next_transition_cost(crfpp_t*, size_t,
size_t, size_t);
CRFPP_DLL_EXTERN double  crfpp_prev_transition_cost(crfpp_t*, size_t,
size_t, size_t);
CRFPP_DLL_EXTERN double  crfpp_best_cost(crfpp_t*, size_t, size_t);
CRFPP_DLL_EXTERN const int* crfpp_emission_vector(crfpp_t*, size_t, size_t);
CRFPP_DLL_EXTERN const int* crfpp_next_transition_vector(crfpp_t*, size_t,
size_t, size_t);
CRFPP_DLL_EXTERN const int* crfpp_prev_transition_vector(crfpp_t*, size_t,
size_t, size_t);
CRFPP_DLL_EXTERN double  crfpp_Z(crfpp_t*);
CRFPP_DLL_EXTERN int     crfpp_parse(crfpp_t*);
CRFPP_DLL_EXTERN int     crfpp_empty(crfpp_t*);
CRFPP_DLL_EXTERN int     crfpp_clear(crfpp_t*);
CRFPP_DLL_EXTERN int     crfpp_next(crfpp_t*);
CRFPP_DLL_EXTERN int     crfpp_test(int, char **);
CRFPP_DLL_EXTERN int     crfpp_test2(const char *);
CRFPP_DLL_EXTERN int     crfpp_learn(int, char **);
CRFPP_DLL_EXTERN int     crfpp_learn2(const char *);
CRFPP_DLL_EXTERN const char* crfpp_strerror(crfpp_t*);
CRFPP_DLL_EXTERN const char* crfpp_ynname(crfpp_t*, size_t);
CRFPP_DLL_EXTERN const char* crfpp_y2(crfpp_t*, size_t);
CRFPP_DLL_EXTERN const char* crfpp_x(crfpp_t*, size_t, size_t);
CRFPP_DLL_EXTERN const char** crfpp_x2(crfpp_t*, size_t);
CRFPP_DLL_EXTERN const char* crfpp_parse_tostr(crfpp_t*, const char*);
CRFPP_DLL_EXTERN const char* crfpp_parse_tostr2(crfpp_t*,
const char*, size_t);
CRFPP_DLL_EXTERN const char* crfpp_parse_tostr3(crfpp_t*, const char*,
size_t, char *, size_t);
CRFPP_DLL_EXTERN const char* crfpp_tostr(crfpp_t*);
CRFPP_DLL_EXTERN const char* crfpp_tostr2(crfpp_t*, char *, size_t);

CRFPP_DLL_EXTERN void crfpp_set_vlevel(crfpp_t *, unsigned int);
CRFPP_DLL_EXTERN unsigned int crfpp_vlevel(crfpp_t *);

```

```

CRFPP_DLL_EXTERN void crfpp_set_cost_factor(crfpp_t *, float);
CRFPP_DLL_EXTERN float crfpp_cost_factor(crfpp_t *);
CRFPP_DLL_EXTERN void crfpp_set_nbest(crfpp_t *, size_t);
#endif

#ifdef __cplusplus
}
#endif

/* C++ interface */
#ifdef __cplusplus
namespace CRFPP {

class Tagger;

class CRFPP_DLL_CLASS_EXTERN Model {
public:
#ifdef SWIG
// open model with parameters in argv[]
// e.g, argv[] = {"CRF++", "-m", "model", "-v3"};
virtual bool open(int argc, char** argv) = 0;

// open model with parameter arg, e.g. arg = "-m model -v3";
virtual bool open(const char* arg) = 0;

// open model with parameters in argv[].
// e.g, argv[] = {"CRF++", "-v3"};
virtual bool openFromArray(int argc, char** argv,
                           const char *model_buf,
                           size_t model_size) = 0;

// open model with parameter arg, e.g. arg = "-m model -v3";
virtual bool openFromArray(const char* arg,
                           const char *model_buf,
                           size_t model_size) = 0;
#endif
// return template string embedded in this model file.
virtual const char *getTemplate() const = 0;

// create Tagger object. Returned object shared the same
// model object
virtual Tagger *createTagger() const = 0;

virtual const char* what() = 0;

virtual ~Model() {}
};

class CRFPP_DLL_CLASS_EXTERN Tagger {
public:
#ifdef SWIG
// open model with parameters in argv[]
// e.g, argv[] = {"CRF++", "-m", "model", "-v3"};
virtual bool open(int argc, char** argv) = 0;

// open model with parameter arg, e.g. arg = "-m model -v3";
virtual bool open(const char* arg) = 0;

```

```

// add str[] as tokens to the current context
virtual bool add(size_t size, const char **str) = 0;

// close the current model
virtual void close() = 0;

// return parameter vector. the size should be dsize();
virtual const float *weight_vector() const = 0;
#endif

// set Model
virtual bool set_model(const Model &model) = 0;

// set vlevel
virtual void set_vlevel(unsigned int vlevel) = 0;

// get vlevel
virtual unsigned int vlevel() const = 0;

// set cost factor
virtual void set_cost_factor(float cost_factor) = 0;

// get cost factor
virtual float cost_factor() const = 0;

// set nbest
virtual void set_nbest(size_t nbest) = 0;

// get nbest
virtual size_t nbest() const = 0;

// add one line to the current context
virtual bool add(const char* str) = 0;

// return size of tokens(lines)
virtual size_t size() const = 0;

// return size of column
virtual size_t xsize() const = 0;

// return size of features
virtual size_t dsize() const = 0;

// return output tag-id of i-th token
virtual size_t result(size_t i) const = 0;

// return answer tag-id of i-th token if it is available
virtual size_t answer(size_t i) const = 0;

// alias of result(i)
virtual size_t y(size_t i) const = 0;

// return output tag of i-th token as string
virtual const char* y2(size_t i) const = 0;

// return i-th tag-id as string
virtual const char* yname(size_t i) const = 0;

// return token at [i,j] as string(i:token j:column)

```



```

// return transition feature vector of [j-th tag at i-th token] to
// [k-th tag at(i-1)-th token]
virtual const int* prev_transition_vector(size_t i,
                                         size_t j, size_t k) const = 0;
#endif

// normalizing factor(log-prob)
virtual double Z() const = 0;

// do parse and change the internal status, if failed, returns false
virtual bool parse() = 0;

// return true if the context is empty
virtual bool empty() const = 0;

// clear all context
virtual bool clear() = 0;

// change the internal state to output next-optimal output.
// calling it n-th times, can get n-best results,
// Need to specify -nN option to use this function, where
// N>=2
virtual bool next() = 0;

// parse 'str' and return result as string
// 'str' must be written in CRF++'s input format
virtual const char* parse(const char* str) = 0;

#ifdef SWIG
// return parsed result as string
virtual const char* toString() = 0;

// return parsed result as string.
// Result is saved in the buffer 'result', 'size' is the
// size of the buffer. if failed, return NULL
virtual const char* toString(char* result, size_t size) = 0;

// parse 'str' and return parsed result.
// You don't need to delete return value, but the buffer
// is rewritten whenever you call parse method.
// if failed, return NULL
virtual const char* parse(const char *str, size_t size) = 0;

// parse 'str' and return parsed result.
// The result is stored in the buffer 'result'.
// 'size2' is the size of the buffer. if failed, return NULL
virtual const char* parse(const char *str, size_t size1,
                        char *result, size_t size2) = 0;
#endif

// return internal error code as string
virtual const char* what() = 0;

virtual ~Tagger() {}
};

/* factory method */

// create CRFPP::Tagger instance with parameters in argv[]

```

```

// e.g, argv[] = {"CRF++", "-m", "model", "-v3"};
CRFPP_DLL_EXTERN Tagger *createTagger(int argc, char **argv);

// create CRFPP::Tagger instance with parameter in arg
// e.g. arg = "-m model -v3";
CRFPP_DLL_EXTERN Tagger *createTagger(const char *arg);

// create CRFPP::Model instance with parameters in argv[]
// e.g, argv[] = {"CRF++", "-m", "model", "-v3"};
CRFPP_DLL_EXTERN Model *createModel(int argc, char **argv);

// load model from [buf, buf+size].
CRFPP_DLL_EXTERN Model *createModelFromArray(int argc, char **argv,
                                             const char *model_buf,
                                             size_t model_size);

// create CRFPP::Model instance with parameter in arg
// e.g. arg = "-m model -v3";
CRFPP_DLL_EXTERN Model *createModel(const char *arg);

// load model from [buf, buf+size].
CRFPP_DLL_EXTERN Model *createModelFromArray(const char *arg,
                                             const char *model_buf,
                                             size_t model_size);

// return error code of createTagger();
CRFPP_DLL_EXTERN const char *getTaggerError();

// alias of getTaggerError();
CRFPP_DLL_EXTERN const char *getLastError();
}

#endif
#endif

```

CFF++ Program code

```

#include <iostream>
#include "crfpp.h"

// c++ -O3 example.cpp -lcrfpp

int main(int argc, char **argv) {

    // -v 3: access deep information like alpha,beta,prob
    // -nN: enable nbest output. N should be >= 2
    CRFPP::Tagger *tagger =
        CRFPP::createTagger("-m model -v 3 -n2");

    if (!tagger) {
        std::cerr << CRFPP::getTaggerError() << std::endl;
        return -1;
    }

    // clear internal context
    tagger->clear();

    // add context
    tagger->add("Confidence NN");
    tagger->add("in IN");
    tagger->add("the DT");
    tagger->add("pound NN");
    tagger->add("is VBZ");
    tagger->add("widely RB");
    tagger->add("expected VBN");
    tagger->add("to TO");
    tagger->add("take VB");
    tagger->add("another DT");
    tagger->add("sharp JJ");
    tagger->add("dive NN");
    tagger->add("if IN");
    tagger->add("trade NN");
    tagger->add("figures NNS");
    tagger->add("for IN");
    tagger->add("September NNP");

    std::cout << "column size: " << tagger->xsize() << std::endl;
    std::cout << "token size: " << tagger->size() << std::endl;
    std::cout << "tag size: " << tagger->ysize() << std::endl;

    std::cout << "tagset information:" << std::endl;
    for (size_t i = 0; i < tagger->ysize(); ++i) {
        std::cout << "tag " << i << " " << tagger->yname(i) << std::endl;
    }

    // parse and change internal stated as 'parsed'
    if (!tagger->parse()) return -1;

    std::cout << "conditional prob=" << tagger->prob()
        << " log(Z)=" << tagger->Z() << std::endl;
}

```

```

for (size_t i = 0; i < tagger->size(); ++i) {
    for (size_t j = 0; j < tagger->xsize(); ++j) {
        std::cout << tagger->x(i, j) << '\t';
    }
    std::cout << tagger->y2(i) << '\t';
    std::cout << std::endl;

    std::cout << "Details";
    for (size_t j = 0; j < tagger->ysize(); ++j) {
        std::cout << '\t' << tagger->yname(j) << "/prob=" << tagger->prob(i,j)
            << "/alpha=" << tagger->alpha(i, j)
            << "/beta=" << tagger->beta(i, j);
    }
    std::cout << std::endl;
}

// when -n20 is specified, you can access nbest outputs
std::cout << "nbest outputs:" << std::endl;
for (size_t n = 0; n < 10; ++n) {
    if (!tagger->next()) break;
    std::cout << "nbest n=" << n << "\tconditional prob=" << tagger->prob() << std::endl;
    // you can access any information using tagger->y()...
}
std::cout << "Done" << std::endl;

delete tagger;

return 0;
}

```

CRF++ Template for Hindi Tagger

Unigram

Context Features:

U01:%x[-1,0] # previous_word

U02:%x[0,0] # current_word

U03:%x[1,0] # next_word

About Sanchay

A Brief Introduction to Sanchay

Sanchay is an open source platform for working on languages, especially South Asian languages, using computers and also for developing Natural Language Processing (NLP) or other text processing applications. It consists of various tools and APIs for this purpose. It is still in the development stage and the design has not yet stabilized, but components like a text editor with customizable support for languages and encodings, annotation interfaces, etc. have been released as an experimental version (0.1) on Sourceforge.net.

Sanchay is already being used for annotation etc. at LTRC, IIIT, Hyderabad. It was also made available for the NLPAIL ML Contest and will be used for the IJCAI 2007 Workshop on Shallow Parsing. Some of the components in the released version are: Syntactic annotation interface, generalised table and tree components, SSF (Shakti Standard Format) API, feature structure API, parallel corpus markup interface, customizable language and encoding support, Sanchay text editor, language and encoding identification, file splitter and format converter, task setup generator (only for syntactic annotation), a simple but powerful data structure called Properties Manager along with a GUI for common purposes like customization of applications, a find/replace/extract tool, a naive preprocessor for annotation, and a tree visualizer for phrase structure and dependency relations. User documentation has been provided for some of these components. More will be added soon. Some API documentation for programmers will also be provided later.

Many other components are in the pipeline. Hopefully other people will get involved with the development so that Sanchay can provide much needed support for South Asian languages for as many purposes as possible.

Sanchay has an object oriented architecture where the emphasis is on good design based on things like modularity, reusability, extensibility and maintainability. The implementation is purely in Java, which means it is platform independent and can be used on Windows as well as Linux without needing any extra setup except installing JDK or JRE.

CHAPTER -VI

Conclusion & scope of further work

Conclusion & scope of further work

Finally we can conclude that our work is fully possible for Automatically tagging the chhattisgarhi as well as the Hindi words if you train the software for hindi then it will work for hindi and if you train for chhattisgarhi then it will work for the chhattisgarhi and it can work for any language you want and the shallow parser is fully capable for parsing the hindi sentences

Advantage Of Project

The project has various advantages like :-

- ➔ The tagger could be used for various languages
- ➔ we have a tagger database for 69,000 chhattisgarhi words
- ➔ Sanchay could be used for all languages based on dev naagri lipi
- ➔ tagger could also be used for sanskrit , and other types of languages
- ➔ Accuracy of tagger and parser is high

Future Scope Of Project

The project has successful in tagging the chhattisgarhi words and parsing the hindi sentences but here we have not made and full parser for chhattisgarhi so things to be done in future

- create a chhattisgarhi parser
- chhattisgarhi morphological analyzer
- full hindi parser with graphical representation
- language translator for hindi to chhattisgarhi
- language translator for chhattisgarhi to hindi
- language translator for English to chhattisgarhi and vice-versa
- creating a web based translator

References :

1. Antony.P.J. 2013. *Machine translation approaches and survey for Indian languages*. Computational linguistics and Chinese language processing.18 (1). 47-48.
2. Fei Sha and Fernando Pereira, Shallow Parsing with Conditional Random Fields, Proceedings of HLT-NAACL 2003, Main Papers , pp. 134-141, Edmonton, May-June 2003
3. Nitin Hambir, Hindi Parser-based on CKY algorithm ,Int.J.Computer Technology & Applications, Vol 3 (2), 851-853
4. Rijuka Pathak, Natural Language Chhattisgarhi: A Literature Survey, International Journal of Engineering Trends and Technology (IJETT) – Volume 12 Number 2 - Jun 2014
5. G.M. Ravi Sastry ,Sourish Chaudhuri ,P. Nagender Reddy” .An HMM based Part-Of-Speech tagger and statistical chunker for 3 Indian languages”
6. Dhanalakshmi V., Anand Kumar M., Rekha R.U., Arun Kumar C., Soman K.P., Rajendran S, “Morphological Analyzer for Agglutinative Languages Using Machine Learning Approaches”, Advances in Recent Technologies in Communication and Computing, International Conference on Advances in Recent Technologies in Communication and Computing, 2009.
7. Dinesh Kumar and Gurpreet Singh Josan, “Part of Speech Taggers for Morphologically Rich Indian Languages: A Survey”, International Journal of Computer Applications (0975 – 8887), Volume6–No.5, www.ijcaonline.org/volume6/number5/pxc3871409.pdf, September, 2010
8. Manish Shrivastava and Pushpak Bhattacharyya, “Hindi POS Tagger Using Naive Stemming: Harnessing Morphological Information Without Extensive Linguistic Knowledge”, Proceeding of the ICON 2008.
9. “A Part of Speech Tagger for Indian Languages (POS tagger)”, shiva.iiit.ac.in/SPSAL2007/iiit_tagset_guidelines.pdf, 2007.
10. Sinha R.M.K, Sivaraman, K., Agrawal, A., Jain, R., Srivastava, R. and Jain, A., “ANGLABHARTI: a multilingual machine aided translation project on translation from English to Indian languages”, IEEE International Conference on: Systems, Man and Cybernetics, Intelligent Systems for the 21st Century, 1995.
11. Sudip Naskar and Sivaji Bandyopadhyay, “Use of Machine Translation in India: Current Status”, In the Proceedings of MT SUMMIT X; Phuket, Thailand, September 13-15, 2005.
12. Vamshi Ambati and Rohini U, “A Hybrid Approach to Example based Machine Translation for Indian Languages”, ICON 2007.