A

Part-of-Speech Tagging of Bhojpuri Data

Mayank, Indian Institute of Technology, Banaras Hindu University, Varanasi
Janvijay Singh, Indian Institute of Technology, Banaras Hindu University, Varanasi
Anil Kumar Singh, Indian Institute of Technology, Banaras Hindu University, Varanasi
Shubham Saini, Indian Institute of Technology, Banaras Hindu University, Varanasi
Rakesh Kumar, Indian Institute of Technology, Banaras Hindu University, Varanasi

This paper shows the part-of-speech tagging of Bhojpuri data using four state-ofthe-art POS taggers namely TnT(Trigrams and Tags), CRF++(Conditional Random Fields), SVMStruct(Structured Support Vector Machine) and MaxEnt(Maximum Entropy). We attempt to use these Partof-Speech Taggers on a Bhojpuri trained corpus consisting of one-fifth of a million words. We hence move to the conclusion of selecting one of these taggers to be the best for our training corpus and show some of the problems associated with POS tagging of Bhojpuri language.

CCS Concepts:•**Computing methodologies** → *Machine translation; Language resources;*

General Terms: Natural Language Processing, Algorithms

Additional Key Words and Phrases: NLP, SVM, Markov Models, CRF, Maximum Entropy

ACM Reference Format:
Gang Zhou, Yafeng Wu, Ting Yan, Tian He, Chengdu Huang, John A. Stankovic, and Tarek F. Abdelzaher, 2010. A multifrequency MAC specially designed for wireless sensor network applications. *ACM Trans. Asian Low-Resour. Lang. Inf. Process.* V, N, Article A (January YYYY), 8 pages.
DOI: http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

POS-Tagging or Part-of-Speech Tagging also called word category disambiguation, is the procedure of using the word definition and its context to find its corresponding part-of-speech tag. Giving the definition more simply, It labels words as being Nouns, Adjectives, Verbs, etc.

1.1. Initial Work

The training corpus that we used was initially in a raw format. There was a lot of noise in the data and the data format did not match the specified default formats for these taggers. So, we pruned the noise from the data. We already knew that there were free implementations of CRF++ and SVMStruct. For the remaining two taggers we used edu.stanford.nlp.tagger.maxent.MaxentTagger class of Stanford POS tagger for MaxEnt and TnT class of the NLTK library to implement our own program to find the correct POS tags for the test data.

2. PART-OF-SPEECH TAGGING

2.1. POS tagging using MaxEnt

We used the maxent class of the Stanford POS tagger to implement our tagger. MaxEnt is a stateof-the-art POS tagger which is easy and fast at training a model as well as tagging the test data. We trained our model using this API. We first generated a propsFile for use in model building. This file is a bank for all the flags that are provided by the API for specifying the algorithm to run in a fashion more specific to your needs. We used a generic architecture for building our model. A generic architecture is a feature extractor that is language independent and works well for the UTF-8 encoded data. We had a large corpus consisting of around 200 thousand words. So we ran a random number generator in python3.4 to extract random 25% data from the corpus to be used for testing and the rest 75% data to be used for training. We repeated this

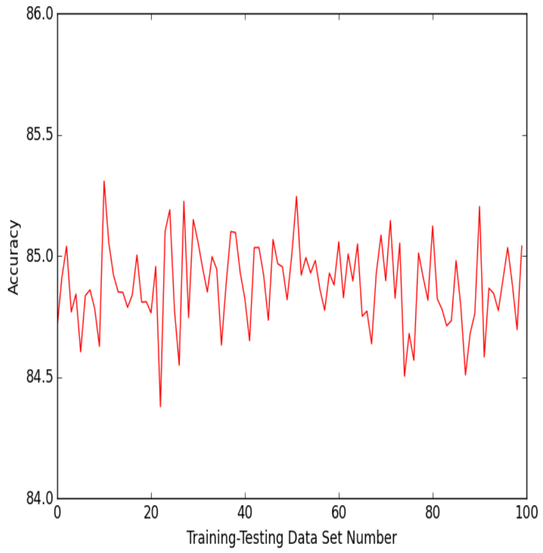


Fig. 1. Accuracy of MaxEnt for 100 different test data sets

execution for a 100 times to get 100 different training and testing files. Once, the model were made for all the 100 training data files testing was started and results were recorded. A python script was then run to check the accuracy of the tagged data and the results were quite impressive.

2.2. POS tagging using TnT Tagger

TnT is a statistical Part-of-Speech tagger which stands for Trigrams and Tags. It can be trained on a variety of languages and virtually any tagset [Brants 2000]. Its algorithm already has many smoothing techniques built in.
NLTK has a module nltk.tag.tnt which implements the TnT() class. We used this class to construct our own Part-of-Speech Tagger based on TnT. There comes often a difficulty in storing the model generated by TnT().train(). The API assumes that the model generation and subsequent tagging on the test data will be done on-the-fly. This posed a big problem to what we were thinking of building a model and then storing it and testing it on the test data. We came across a library in python3.4 Pickle. This library stores any python object in form of binary bit strings onto a file using pickle.dump() which can then be loaded on the memory using pickle.load(). Although it affects the speed of the tagger, but provides it with greater convenience. We built 12 training testing data sets from the tagged corpus available to us by using a random number generator to find which sentences out pf the corpus remain in test data of the test data and which remain in training data. We did not have good computational resources as we ran the whole experiment on a laptop with 5th gen Intel i7 processor. This constrained us from making 100 randomly generated test and training files as we did in MaxEnt because the speed of using MaxEnt for tagging and training proved out to be at least 30- 40 times better than any other tagger that we are considering in this report. Once, the models were built for all the 10 training data files, we tagged the corresponding 12 test data files and recorded the results. There was very little deviation of accuracies for each test data from the mean accuracy in this case.

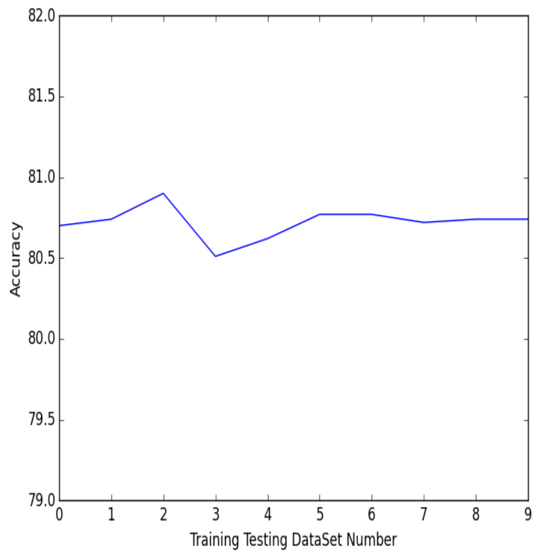


Fig. 2. Accuracy of TnT for different test data sets

Table I. Template Used for CRF++

n-gram Feature	Feature Name	Components
1.	U00	%x[-2,0]
2.	U01	%x[-1,0]
3.	U02	%x[0,0]
4.	U03	%x[1,0]
5.	U04	%x[2,0]
6.	U05	%x[-1,0]/%x[0,0]
7.	U06	%x[0,0]/%x[1,0]
8.	B	Bigram

2.3. POS tagging using CRF++ Tagger

CRF stands for Conditional Random Fields. CRF++ is an open source implementation of CRF algorithm for sequence labeling. CRF has an advantage over other taggers that it can be used for a variety of NLP operations like Named Entity Recognition, Text Chunking and Information Extract i on along with POS-tagging. The tagged corpus was divided into test and training data sets using a random number generator in python3.4 and 10 test and training files were created. We conducted tests on the same training and testing data increasing the complexity of the template file the file that defines the features to be used by the tagger, with every iteration. The results of this test are shown below:

The results show the best tagging accuracy for the following template file (File no. 10) and hence, we used this template file to conduct further experiments. We see a dip in accuracy of the graph from feature file no. 6 to 7. It is so because the feature no 7 considered uni-gram features up to the 6th word ahead of the current word. This complexity will not work well for the training corpus size that we used. If we however increased the corpus size, we can fit more complex features with gaining accuracy.

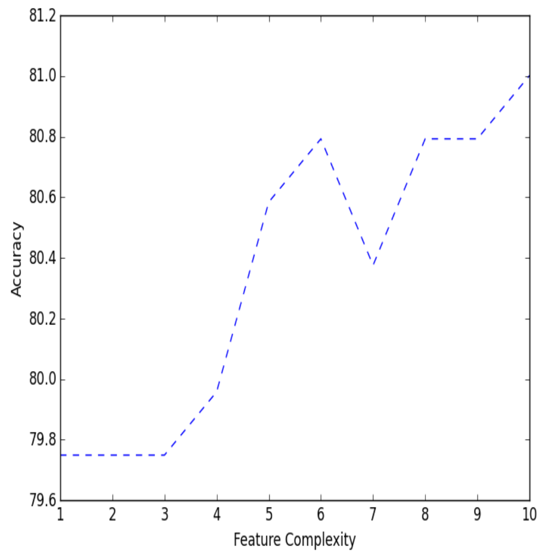


Fig. 3. Template Complexity and Accuracy for CRF++

Here %x[a,b] denotes a-th word from current word and b-th tag from current tag. B denotes a default bi-gram feature provided by CRF++ toolkit. The test data was then tagged using the respective generated models and results were noted.

2.4. POS tagging using SVMStruct Tagger

SVM stands for Support Vector Machine, which is a widely used algorithm in Machine Learning. But, SVM can not be used for sequence labeling [Jesus Gimenez 2004]. This requires for the use of Structured Support Vector Machine(SVMStruct). It belongs to the category of supervised learning algorithms. It approximates the output values using an approximation hypothesis trained by labeled training data. It can predict trees , sets and sequences unlike the SVM, which can only predict regression and classification. It is used widely in parsing and Markov models in POS-tagging. The tagged corpus was run through a random number generator to find 10 different sets of testing and training data in 1:3 ratio. The following feature table was used for building models for all these training data files.

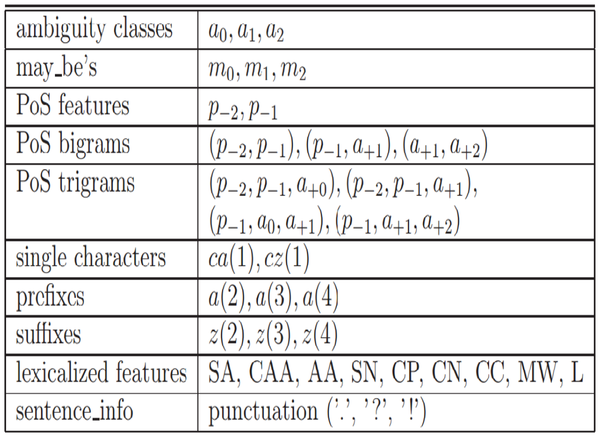
The test files were then put into the tagger using a bash script and the results were recorded.

3. RESULTS

All the four taggers were run and the recorded results are shown below. For the taggers where it was possible to take up to 100 different test and training data sets, average accuracy is taken.

4. ERROR ANALYSIS

The error scan on the tagging results shows us some interesting facts about our tagged corpus. Average unseen words that were not seen in the training data but appeared for the first time when tagging was done on the test data were 12% of total words in the test data. The average unseen component tag percentage is shown in the table below:



ambiguity classes	a_0, a_1, a_2
may_be's	m_0, m_1, m_2
PoS features	p_{-2}, p_{-1}
PoS bigrams	$(p_{-2}, p_{-1}), (p_{-1}, a_{+1}), (a_{+1}, a_{+2})$
PoS trigrams	$(p_{-2}, p_{-1}, a_{+0}), (p_{-2}, p_{-1}, a_{+1}),$ $(p_{-1}, a_0, a_{+1}), (p_{-1}, a_{+1}, a_{+2})$
single characters	$ca(1), cz(1)$
prcfixcs	$a(2), a(3), a(4)$
suffixes	$z(2), z(3), z(4)$
lexicalized features	SA, CAA, AA, SN, CP, CN, CC, MW, L
sentence_info	punctuation ('.', '?', '!')

Fig. 4. Feature Table Used for SVMStruct

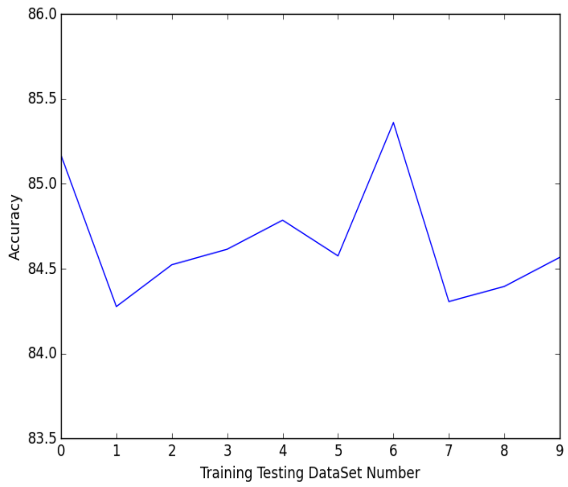


Fig. 5. Accuracy of SVMStruct for different test data sets

Table II. Results of Tagging

Tagger	**No. of models generated**	**Average Accuracy**
CRF++	10	83.396
SVMStruct	10	84.657
MaxEnt	10	84.798
TnT	10	80.721
MaxEnt	100	84.882

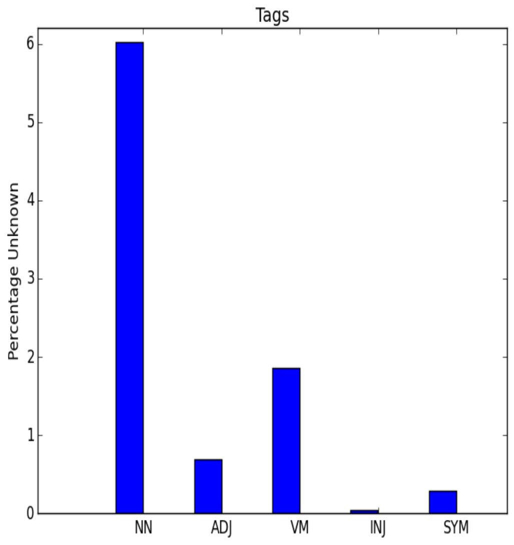


Fig. 6. Percentage of Unknown tags in test data

Table III. Unseen Tag Percentage

Tag Symbol	**Tag Name**	**Unseen % in test data**
NN	Noun	6.02
JJ	Adjective	0.69
VM	Verb	1.86
INJ	Preposition	0.036
SYM	Symbol	0.286

It can be seen that most of the unseen word count comes from nouns and verbs. If we assume that most of the unseen words in the test data will be nouns, as having all nouns in the train data is difficult because of the diverse nature of this tag, then giving every unseen word the NN tag improves our total accuracy by about 6This experiment was conducted on our implementation of TnT, as we were free to introduce any modifications easily in our own implementation. The results obtained suggested a flat increase in the tagging accuracy of TnT tagger from an average accuracy of 80 to 86. This experiment shows that this small change can also increase the accuracy to a good percentage. We hence did an experiment to determine whether on increasing the test data size, how much does accuracy change. We selected 50%, 75%, 85% and 95% of total corpus as our training data. The experiment was conducted using SVMStruct tagger running on default options set. The results of this experiment showed that accuracy keeps on improving and reach to 85.47% with 95% corpus as train data and remaining 5% as test data.

5. CONCLUSION

Having described the experiments we conducted on the tagged corpus of Bhojpuri, we conclude that MaxEnt tagger performs best for this language and corpus giving the maximum accuracy of 84.8821%. SVMStruct also performs well on the corpus giving a comparable accuracy of 84.651We also conclude that 12% average unseen word count

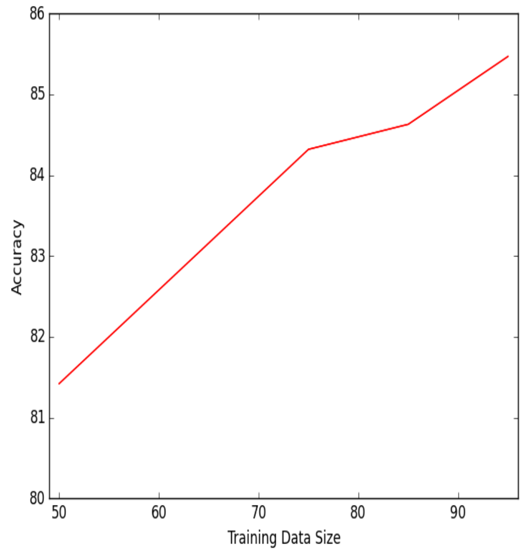


Fig. 7. Accuracy Vs Training Data Size

Table IV. Training Data Size Vs Accuracy

Training Data Size(%)	**Test Data Size(%)**	**Accuracy % using SVM**
50	50	81.42
75	25	84.32
85	15	84.63
95	5	85.47

was obtained on a scan of the test data which is very bad. This counts for the bad accuracy that we achieve with these state-of-the-art taggers. Given below is a plot showing the tagging accuracies for some common tags taken from a random SVMStruct tagging iteration. This shows us some of the tags that are more prone to errors.

Here CC: Coordinating Conjunction, PRP: Personal Pronoun and RB: Adverb. We can increase the accuracy to 90+% by taking a bigger training corpus which will count for lesser unseen words in the test data. As the bar chart shows adjectives (JJ), adverbs (RB) and prepositions (INJ) perform really badly compared other tags, a bigger and more accurately labelled corpus can improve accuracies for these tags. We can also use an external dynamic dictionary to check for unseen words from and assign them their most likely tag. We can also use a morph analyzer to give suitable tags based on word morphology in case on unseen words.

6. FUTURE RESEARCH

We would like to propose the need for building more accurately tagged Bhojpuri corpus which is larger than at least a million words. This would require linguists showing interest in working in the development of better NLP tools for Bhojpuri language. Special focus needs to be given to the tags more prone to mistakes like JJ, INJ and RB as we have discussed above.

We will also work on taking more specific algorithmic parameters than the general parameters we took in writing this paper. As shown in the paper taking different tem-

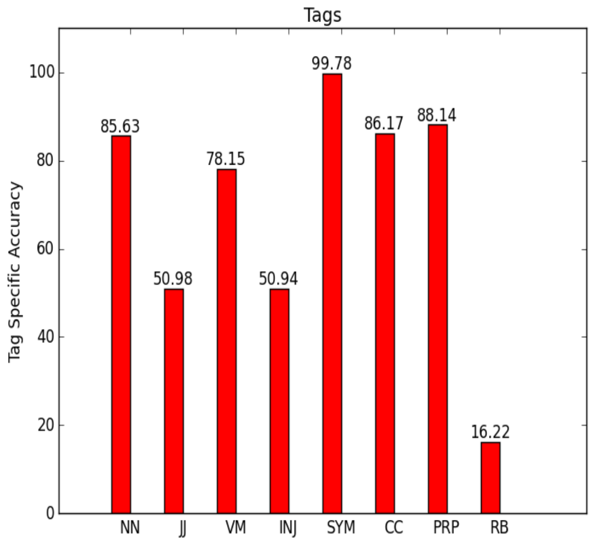


Fig. 8. Accuracy of some common tags

plates for CRF++ results in decent accuracy changes.
As is is evident from our work that Nouns have the largest percentage among the Unseen tags, so we propose that a good Hindi corpus can also be used to check if an Unknown tag comes. This idea comes from the fact that Proper Nouns like names, are similar in Hindi and Bhojpuri.

REFERENCES

Thorsten Brants. 2000. TnT – A Statistical Part-of-Speech Tagger. In *Proceedings of the 6th Applied Natural Language Processing Conference and the 1st North American Chapter of Association for Computational Linguistics*. 224–231.

Lluis Marquez Jesus Gimenez. 2004. SVMTool: A general POS tagger generator based on Support Vector Machines.. In *Proceedings of the 4th. Language Resources and Evaluation Conference (LREC)*. http://www.lrec-conf.org/proceedings/lrec2004/pdf/597.pdf