Development of POS tagger for English-Bengali Code-Mixed data

Tathagata Raha[1], Sainik Kumar Mahata[2], Dipankar Das[3], Sivaji Bandyopadhyay[4]
[1]IIIT, Hyderabad
[234]Jadavpur University, Kolkata
[1]tathagata.raha@research.iiit.ac.in, [2]sainik.mahata@gmail.com,
[3]dipankar.dipnil2005@gmail.com, [4]sivaji_cse_ju@yahoo.com

Abstract

Code-mixed texts are widespread nowadays due to the advent of social media. Since these texts combine two languages to formulate a sentence, it gives rise to various research problems related to Natural Language Processing. In this paper, we try to excavate one such problem, namely, Parts of Speech tagging of code-mixed texts. We have built a system that can POS tag English-Bengali code-mixed data where the Bengali words were written in Roman script. Our approach initially involves the collection and cleaning of English-Bengali code-mixed tweets. These tweets were used as a development dataset for building our system. The proposed system is a modular approach that starts by tagging individual tokens with their respective languages and then passes them to different POS taggers, designed for different languages (English and Bengali, in our case). Tags given by the two systems are later joined together and the final result is then mapped to a universal POS tag set. Our system was checked using 100 manually POS tagged code-mixed sentences and it returned an accuracy of 75.29%.

1 Introduction

A **P**arts-**o**f-**S**peech (POS) Tagger is a piece of software that reads the text in some language and assigns parts of speech tags, such as noun, verb, adjective, etc., to each word/token. POS Tags are useful for building parse trees, which may be used to build textbfNamed **E**ntity **R**ecognizers (NER) or Dependency Parsers. POS Tagging is also useful for building lemmatizers, which are used to reduce a word to its root form. POS taggers for widely spoken languages have been developed in abundance. But such resources are very scarce for low resourced languages.

On the other hand, code-mixing is simply a mix of two or more languages in communication. Due to the emergence of social media, a lavish amount of digital code-mixed data is generated. This is because people nowadays are very comfortable with multilingualism. This phenomenon has produced a section of researchers, who contemplate code-mixed texts as being a new language.

As mentioned earlier, since POS tagging systems for low resourced languages are hard to come by, developing one that will cater to code-mixed text is trivial. POS tagging systems, if developed for Code-Mixed data, can lead to deciphering many complex **N**atural **L**anguage **P**rocessing (NLP) tasks and hence, we attempt to develop the same in this reported work. We try to focus on creating a POS tagger for English-Bengali code-mixed data, as languages such as Bengali are morphologically rich in nature.

Our method includes scraping of code-mixed English-Bengali tweets on Twitter and cleaning them. The Bengali words in these tweets were in Roman script. These cleaned tweets were used as a development dataset for building our system. Our system starts with tagging individual tokens of a tweet with their respective languages, either English, Bengali or Unknown. This step will give rise to segments/sub-sequences of the tweet, written in the same language. It is to be noted that tokens tagged as Unknown were discarded. The segments will then be passed to two POS taggers, one designed for English and the other designed for Bengali. The output from the POS taggers will then be joined together to get the final POS tagged, code-mixed tweet. Since the POS tagging modules of English and Bengali use different tag sets, we further map the tags to a manually defined universal POS tag set. This step produces a final POS tagged tweet with uniform tags. The architecture of the proposed model is shown in Figure 1.

The remainder of the paper is organized as follows. Section 2 documents a brief state-of-art on

D M Sharma, P Bhattacharyya and R Sangal. Proc. of the 16th Intl. Conference on Natural Language Processing, pages 143–149
Hyderabad, India, December 2019. ©2019 NLP Association of India (NLPAI)

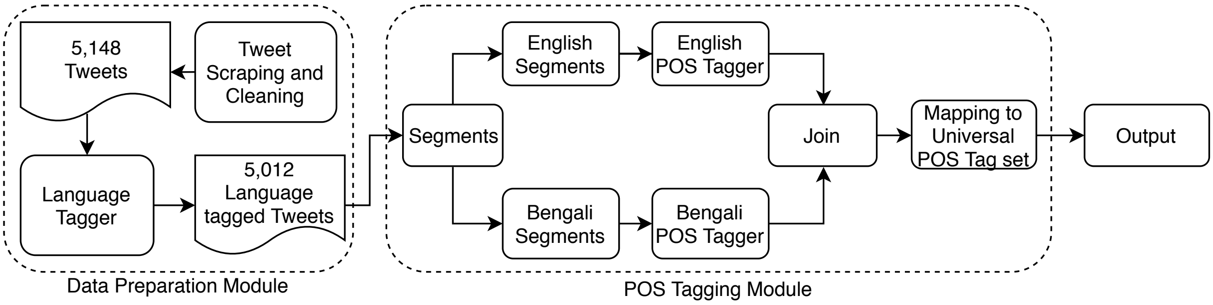

Figure 1: Architecture of the proposed model.

this domain. Section 3 defines the data preparation steps. Section 4 defines the pipeline which helps us in POS tagging the code-mixed tweets. This will be followed by the results in Section 5 and concluding remarks in Section 6.

2 Related Work

In the past few years, a lot of significant work has been done in the field of Parts of Speech tagging. The first significant POS tagger came in the early Nineties which was a rule-based tagger (Karlsson et al., 2011). One of the English rule-based taggers had an accuracy of 99.5% (Samuelsson and Voutilainen, 1997). POS taggers based on statistical approaches were also used during this time, which was based on statistical models like bi-gram,tri-gram and Markov Models (DeRose, 1988; Cutting et al., 1992; Dermatas and Kokkinakis, 1995; Meteer et al., 1991; Merialdo, 1994). Subsequently, POS tagger based on both statistical methods and a rule-based approach was proposed by Brill (1992).

Use of Conditional Random Fields for the development of POS taggers was proposed by Lafferty et al. (2001), Shrivastav et al. (2006) and Sha and Pereira (2003). Nakamura et al. (1990) used neural networks for POS tagging for the first time.

POS taggers for the Bengali language was also built by Seddiqui et al. (2003). This POStagger was built on the analysis of the Bengali morphemes. Other works have been done in Bengali POS tagging by Hasan et al. (2007) and Dandapat et al. (2007) which were rule-based and semi-supervised.

Pimpale and Patel (2016) attempted to tag code-mixed data using Stanford POS tagger. He trained the POS tagger on constrained data of Hindi, Bengali, and Telugu, mixed with English. They garnered accuracy figures of 71%. Similarly, Sarkar (2016) used the HMM model on constrained code-mixed data and achieved an accuracy figure of 75.60%.

Pipeline architecture for POS tagging of code-mixed data was first used by Barman et al. (2016). The training data was very low in their case and the LID (language identification) and transliteration models used were based on Support Vector Machines (SVM) and manual transliteration. Our approach also used pipeline architecture similar to theirs, but our model does not require any annotated data to train the system. Also, the LID and transliteration modules, in our case, have been fully trained with much larger data, using Deep Learning architecture.

3 Data Preparation

We decided to use a development dataset for building our system. It is to be noted that this data was used to build the proposed system and not to train it. Since code-mixed data consisting of English and Bengali language are difficult to find, we decided to scrape such data from Twitter. The collected tweets contained multiple degrees of noise and hence, it needed to be cleaned before using it to develop our future systems. After cleaning the tweets, they were subjected to a Language Tagger module that tagged every token of the tweet with their corresponding language (English, Bengali, and Unknown, in this case).

3.1 Tweet Scraping and Cleaning

Initially, we had to assemble the development data, consisting of English-Bengali code-mixed data, that will be used to build the POS tagger model. For this, we scraped tweets from Twitter, as it is a social media handle with a huge repository of such data. Our tweet scraper module used the Twint module[1], a python package that helps to scrape tweets. The program was fed with a list of Bengali (Romanized) keywords that will be used to scrape the tweets. Later, the Twint object iterates the keywords and recovers tweets corresponding to the same keywords.

Using this method, 5,148 code-mixed tweets containing English and Bengali (Romanized) words were collected. The collected tweets were noisy and hence we needed to clean it beforehand to proceed. The cleaning module was a manifold approach that involved cleaning links, smileys, Emojis, Hashtags, and Mentions (Usernames).

3.2 Language Tagging and Segmentation

We observed that there is no end-to-end POS tagger available that can jointly tag English and Bengali tokens. Thus we decided to segment the cleaned tweets, into Bengali and English. This was done so that tokens in different language segments can be tagged with their respective POS tags, separately.

For segmenting the tweets, the words needed to be tagged with their corresponding language. To develop such a **L**anguage **T**agging (LT) model, we collected 11,060 Romanized words of Bengali and 7,223 words of English. We developed a binary classification model that takes as input, the tokens of a tweet (in character embedding) and outputs the language of the word to either English or Bengali. Tokens (in character embedding) were fed to a stacked LSTM of size 2. The output vectors from the LSTM cells were then fed to a fully connected layer, which then mapped the words to its specific language. For the given model, Activation was kept as *Sigmoid*, Optimizer used was *Adam* and Loss used was *Binary Crossentropy*. Batch Size was kept at 30. The program was executed for 30 epochs and the model was validated using a validation split of 0.2.

The architecture of the language tagging module is shown in Figure 2. The model returned a validation accuracy of 91%. It is to be noted that, characters apart from alphabets and numbers were tagged as Unknown'. Tweets with no language tag and only unknown tags were discarded. An example of language tagging is shown in Table 1. Statistics of the tweets after cleaning and language tagging are shown in Table 2.

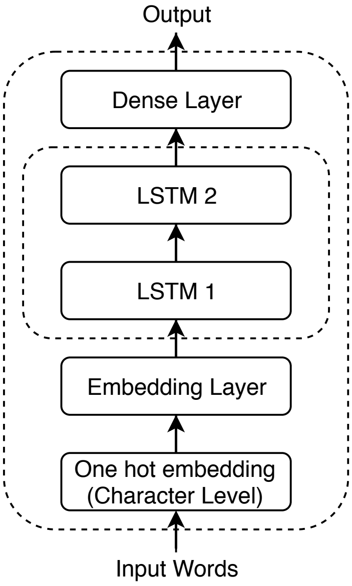

Figure 2: Language Tagging Module.

I loved the golpo and khabar ta khub nice chilo .
I\en loved\en the\en golpo\bn and\en khabar\bn ta\bn khub\bn nice\en chilo\bn .\un

Table 1: Example of Language Tagging.

After the language tagging is done, a segmentation module partitions the code-mixed input into segments concerning its language tags. In our case, segments are sub-sequences of the instance, written in the same language. An example of segmentation is shown below, where strings in brackets denote segments;

1. (Movie)$_{\text{En}}$ (ta bhalo chilo)$_{\text{Bn}}$ (but mid point)$_{\text{En}}$ (e amar khub)$_{\text{Bn}}$ (boring)$_{\text{En}}$ (lagte shuru korlo)$_{\text{Bn}}$.
2. (I had to go)$_{\text{En}}$ (karon o khub)$_{\text{Bn}}$ (urgently)$_{\text{En}}$ (daklo amaye)$_{\text{Bn}}$.

3.3 Language Switch Analysis

Language tagged tweets were then analyzed to examine switching patterns. For this, the tweets were tokenized and a list of bigrams was extracted. Since the tokens of the tweets are tagged with their specific language, we could find out the count of bigrams with respect to EN-EN (both tokens of biagram are in English), BN-BN (both tokens of biagram are in Bengali), EN-BN (fist token of biagram is in English and second in Bengali) and BN-EN (fist token of biagram is in English and second in Bengali).

4 Parts of Speech Tagging

After the data preparation step, the language tagged segments are passed to the corresponding

[1] https://pypi.org/project/twint/

Particulars	Number
No. of tweets before LT	5,148
No. of tweets after LT	5,012
No. of tokens before LT	1,44,17
No. of tokens after LT	1,41,47
No. of tweets with no language tag	136

Table 2: Statistics of tweets after cleaning and Language Tagging.

Switch	Count	Freq >500	Freq >1000
EN-BN	17,758	199	88
BN-EN	17,562	166	53
EN-EN	43,859	539	203
BN-BN	16,535	98	39

Table 3: Language Switch Analysis.

language POS tagger for the final tagging. Two different POS tagging systems were used for English and Bengali. For POS tagging the English Segments we used the Stanford POS tagger[2] and the output was recorded.

For the Bengali segments, we used a tagger developed by Das et al. (2014). They trained the tagger on 10,000 Bengali (Devanagari) POS tagged sentences and tested it on 2,000 Bengali (Devanagari) sentences. Their model returned 92% accuracy. To use their model, we had to transliterate the Bengali segments into its corresponding Devanagari script. The model developed to do the same is described in Section 4.1.

4.1 Bengali Transliteration

To develop the transliteration system, we initially collected 22,781 Romanized Bengali words and manually transliterated them to its Devanagari counterpart. We developed a Sequence-to-Sequence model that takes as input the Romanized Bengali words and outputs the Bengali words in the Devanagari script. The embedding used in this model was at the character level.

The model consists of two parts: an Encoder and Decoder. The encoder takes as input, Romanized Bengali characters, creates one-hot vectors of the same and passes this to the Embedding layer. The output of the embedding layer is given to a stacked LSTM cell, which produces a context vector of the input word. The Decoder module takes as input the Bengali characters in Devanagari script, creates a one-hot vector of the same and passes it to an embedding layer. The output of the embedding layer is given to a stacked LSTM cell which is initialized with the state of the encoder module. The stacked LSTM cell then produces Bengali characters (in Devanagari script) as output, with an offset of a one-time step. The activation of the model was selected as *Softmax*, Optimizer used was *Adam* and Loss used was *Sparse Categorical Crossentropy*. Batch Size was kept at 1024. The program was executed for 50 epochs and the model was validated using a validation split of 0.1.

The validation accuracy of the model was recorded as 87%. The architecture of the model is shown in Figure 3.

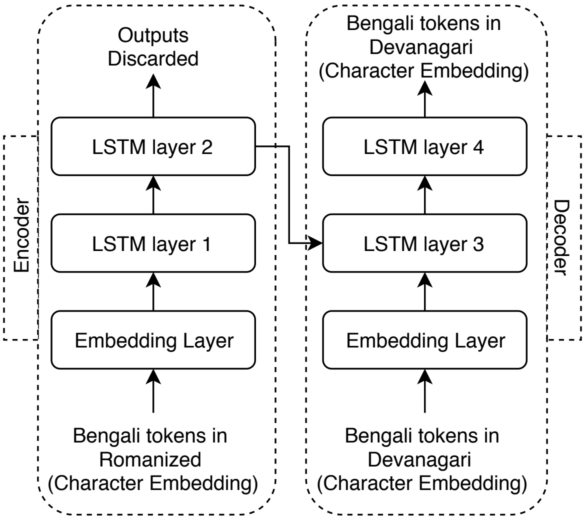
Outputs Discarded
LSTM layer 2
LSTM layer 1
Embedding Layer
Bengali tokens in Romanized (Character Embedding)
Encoder
Bengali tokens in Devanagari (Character Embedding)
LSTM layer 4
LSTM layer 3
Embedding Layer
Bengali tokens in Devanagari (Character Embedding)
Decoder

Figure 3: Back transliteration model

The transliterated segments are then fed to the Bengali POS tagger and the corresponding outputs are recorded.

After POS tagging both the English and Bengali segments, the results are joined together to get a POS tagged code-mixed tweet.

4.2 Mapping to Universal POS Tag Set

The final POS tagged code-mixed tweets need to be generalized to a universal system because the POS tags of the Bengali and English POS taggers are different. This is because English and Bengali POS taggers have different grammar and thus use different POS tag sets. To simplify this situation, we use a universal POS tag set that comprises the tags as showed in Table 4. The table shows the universal tags in bold and italics while the other texts define the universal tag.

For mapping the English POS tags to this universal POS tag set we use `map_tag` which is an inbuilt tool of NLTK. It maps the English tags to these tags based on some pre-defined rules.

The mapping of the Bengali POS tags (Stanford

[2] https://nlp.stanford.edu/software/tagger.shtml

POS	Univ.Tag	POS	Univ. Tag
Adjective	***ADJ***	Adposition	***ADP***
Determiner	***DET***	Noun	***NOUN***
Pronoun	***PRON***	Verb	***VERB***
Adverb	***ADV***	Conjunction	***CONJ***
Numeral	***NUM***	Particle	***PRT***
Punctuation	***SYM***	Other	***X***
Demonstrative	***DEM***	Intensifier	***INTF***
Reduplicative		RDP	

Table 4: Universal tag set, where text in bold and italics denote the tag and the text above define the tags

POS tags) to the universal POS tag set is shown in Table 5. Here, text in bold and italics denotes the universal tag, while the other defines the Stanford POS tags.

Syst. Tag	Univ. Tag	Syst. Tag	Univ. Tag
NN	***NOUN***	VM	***VERB***
NNP		VAUX	
INTJ		JJ	***ADJ***
PRP	***PRON***	QF	
WQ		RB	***ADV***
DEM	***DEM***	NEG	
PSP	***ADP***	RP	***PRT***
CC	***CONJ***	INTF	***INTF***
QC	***NUM***	RDP	***RDP***
SYM	***SYM***	UN	***UN***
DET	***DET***	Other	***X***

Table 5: Mapping of Bengali POS tags to the universal tagset. Text in bold and italics denotes the universal tag, while the other defines the Stanford POS tags

Finally, the POS tagged segments (mapped to the universal POS tagset) are recorded as the final output.

5 Results

Since there is no automated evaluation metric present to assess the quality of POS tagging a code-mixed sentence, we hired a linguist who was proficient in both Bengali and English. The linguist was asked to prepare a test data comprising of 100 English-Bengali code-mixed sentences. Further, the linguist was asked to POS tag the tokens, based on the universal POS tagset, separately. The linguist was told to look into the context of the sentence while tagging the tokens. This approach was used to properly

- tag ambiguous words, such as ’to’, which occurs in both English and Bengali.
- tag words in the switching point.

The same test data was tagged using our system as well. To calculate the agreement between the manual annotation and system annotation, we used Krippendorff’s Alpha (Krippendorff, 2011), and the metrics and the confusion are shown in Table 6

POS Tag	**Man. Tag**	**Syst. Tag**	**Diff. & Conf.**
NOUN	522	538	16 ADJ VERB
VERB	286	259	27 NOUN PRON
ADJ	169	141	28 NOUN VERB
PRON	104	118	14 ADJ ADV
ADV	93	63	30 VERB ADV
SYM	59	60	1 NUM
CONJ	58	49	9 NOUN VERB
DET	54	53	1 VERB
ADP	54	49	5 PRT
PRT	21	18	3 ADJ
DEM	10	11	1 NOUN
NUM	9	9	0
INTF	3	6	3 VERB
RDP	1	1	0
UN	0	606	606
K’s \alpha (Interval)	0.7522		

Table 6: Agreement Analysis between manual tagged and system tagged POS tags

Inter-system annotation agreement scores described in Table 6 evaluates the overall system. To dive deeper, we evaluated every sentence of the test data. This was done using two methods.

Method 1:
For a code-mixed sentence, the POS tag of every token in the same manually annotated sentence as compared to the POS tag of every token in the same system annotated sentence. score_{A} was calculated as

$$\text{score}_{\text{A}} = \frac{\text{\# matched POS tags with manual tagged sentence}}{\text{\# tokens in the manually annotated sentence}}$$

Method 2: POS tagging of tokens that lie in the language switching point,i.e., $word_{\text{English}} \leftrightarrow word_{\text{Bengali}}$, is of utmost importance as the context of the two words may change. As a result, POS tags may also differ. In this context, score_{B} was calculated by multiplying 0.25 to score_{A} and taking the absolute value of its log value, if POS tags (for the language switching point) in the manually annotated sentence and the system annotated sentence, match. The multiplying factor was kept at 0.25 as there can be four bigrams, i.e., EN-EN, BN-BN, EN-BN, and BN-EN.

If there is more than one switching point and the POS tags match, the multiplying factor was repeated for the number of switching. So, if there are two switching points, and the POS tags match, score_A will be multiplied by 0.25 and 0.25 to get score_B.

$$\text{score}_\text{B} = |\log(\text{score}_\text{A} * (0.25)^\text{n})|$$

, where n denotes the number of language switching points present and the trailing * denote that the formula holds true if certain conditions are met.

With the help of the above methods, Score_A and Score_B were calculated for every sentence and finally, the average for the whole test data was calculated. With method 1, our algorithm garnered accuracy of **72.72%** and with method 2, the accuracy increased to **75.29%**.

6 Conclusion

In this work, we have devised a modular system that can POS tag English-Bengali code-mixed sentences. The system uses sub-modules to perform the same. Owing to the fact, that the sub-modules can be trained for any given language, the proposed approach can be used to tag a variety of code-mixed data involving any two language pairs.

The system can be enhanced further if the sub-modules can be trained using more annotated data. E.g., if the POS tagger for the Bengali language could have been trained using more data, the problem of tagging untrained tokens with 'UN' tags could have been solved. Also, the problem of wrongly tagging tokens, e.g., tagging NOUN as ADJ, VERB and tagging PRON as NOUN, VERB, etc., could have been solved. This would have made the Bengali POS tagging module more robust. The same applies to the transliteration module as well.

In the future, we would like to develop an end-to-end system, so that the errors of one sub-module do not propagate to the other sub-modules.

References

Utsab Barman, Joachim Wagner, and Jennifer Foster. 2016. Part-of-speech tagging of code-mixed social media content: Pipeline, stacking and joint modelling. In *Proceedings of the Second Workshop on Computational Approaches to Code Switching*, pages 30–39.

Eric Brill. 1992. A simple rule-based part of speech tagger. In *Proceedings of the third conference on Applied natural language processing*, pages 152–155. Association for Computational Linguistics.

Doug Cutting, Julian Kupiec, Jan Pedersen, and Penelope Sibun. 1992. A practical part-of-speech tagger. In *Third Conference on Applied Natural Language Processing*, pages 133–140.

Sandipan Dandapat, Sudeshna Sarkar, and Anupam Basu. 2007. Automatic part-of-speech tagging for bengali: An approach for morphologically rich languages in a poor resource scenario. In *Proceedings of the 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions*, pages 221–224. Association for Computational Linguistics.

A. Das, U. Garain, and A. Senapati. 2014. Automatic detection of subject/object drops in bengali. In *2014 International Conference on Asian Language Processing (IALP)*, pages 91–94.

Evangelos Dermatas and George Kokkinakis. 1995. Automatic stochastic tagging of natural language texts. *Computational Linguistics*, 21(2):137–163.

Steven J DeRose. 1988. Grammatical category disambiguation by statistical optimization. *Computational linguistics*, 14(1):31–39.

Fahim Muhammad Hasan, Naushad UzZaman, and Mumit Khan. 2007. Comparison of different pos tagging techniques (n-gram, hmm and brills tagger) for bangla. In *Advances and innovations in systems, computing sciences and software engineering*, pages 121–126. Springer.

Fred Karlsson, Atro Voutilainen, Juha Heikkilae, and Arto Anttila. 2011. *Constraint Grammar: a language-independent system for parsing unrestricted text*, volume 4. Walter de Gruyter.

Klaus Krippendorff. 2011. Computing krippendorff's alpha-reliability.

John Lafferty, Andrew McCallum, and Fernando CN Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data.

Bernard Merialdo. 1994. Tagging english text with a probabilistic model. *Computational linguistics*, 20(2):155–171.

Marie Meteer, Richard M Schwartz, and Ralph M Weischedel. 1991. Post: Using probabilities in language processing. In *IJCAI*, pages 960–965.

Masami Nakamura, Katsuteru Maruyama, Takeshi Kawabata, and Kiyohiro Shikano. 1990. Neural network approach to word category prediction for english texts. In *Proceedings of the 13th conference on Computational linguistics-Volume 3*, pages 213–218. Association for Computational Linguistics.

Prakash B Pimpale and Raj Nath Patel. 2016. Experiments with pos tagging code-mixed indian social media text. *arXiv preprint arXiv:1610.09799*.

Christer Samuelsson and Atro Voutilainen. 1997. Comparing a linguistic and a stochastic tagger. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics and Eighth Conference of the European Chapter of the Association for Computational Linguistics*, pages 246–253. Association for Computational Linguistics.

Kamal Sarkar. 2016. Part-of-speech tagging for code-mixed indian social media text at icon 2015. *arXiv preprint arXiv:1601.01195*.

Md Hanif Seddiqui, AKMS Rana, Abdullah Al Mahmud, and Taufique Sayeed. 2003. Parts of speech tagging using morphological analysis in bangla. In *Proceeding of the 6th International Conference on Computer and Information Technology (ICCIT)*.

Fei Sha and Fernando Pereira. 2003. Shallow parsing with conditional random fields. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 134–141. Association for Computational Linguistics.

M Shrivastav, R Melz, Smriti Singh, K Gupta, and P Bhattacharyya. 2006. Conditional random field based pos tagger for hindi. *Proceedings of the MSPIL*, pages 63–68.