Part-of-Speech Tagging for Code-Switched, Transliterated Texts without Explicit Language Identification

Kelsey Ball
kelseytaylorball@gmail.com

Dan Garrette
Google Research
dhgarrette@google.com

Abstract

Code-switching, the use of more than one language within a single utterance, is ubiquitous in much of the world, but remains a challenge for NLP largely due to the lack of representative data for training models. In this paper, we present a novel model architecture that is trained exclusively on monolingual resources, but can be applied to unseen code-switched text at inference time. The model accomplishes this by jointly maintaining separate word representations for each of the possible languages—or scripts in the case of transliteration—allowing each to contribute to inferences without forcing the model to commit to a language. Experiments on Hindi-English part-of-speech tagging demonstrate that our approach outperforms standard models when training on monolingual text without transliteration, and testing on code-switched text with alternate scripts.

1 Introduction

Code-switching,[1] the linguistic phenomenon of switching between more than one language within a single utterance, is ubiquitous in multilingual societies worldwide, but presents particular challenges for even the most standard NLP tasks. For example, state-of-the-art neural part-of-speech (POS) taggers are trained on POS-annotated data to learn the relationships between words and tags, and rely on word embeddings trained from large (usually unannotated) corpora to capture information about each word's typical contexts (Ling et al., 2015; Wang et al., 2015). But while monolingual corpora, both annotated and unannotated, are relatively easy to acquire, code-switched text is much more difficult to collect in large quantities. Annotated code-switched texts in particular are not only expensive to create, requiring particularly skilled bilingual linguists as annotators, but they will never exist for the complete set of pairs (or triples, etc.) of languages between which an individual may switch. Further complicating the problem is that texts featuring code-switching also tend to feature *transliteration*—writing in a non-standard script—to avoid having to switch keyboard settings mid-sentence. When Hindi is transliterated from Devanagari to Latin script, tokens are no longer easy to distinguish by language.[2] Previous work addresses this problem by first attempting to identify the correct language and form of a word before feeding it into an appropriate monolingual tagger (Vyas et al., 2014; Solorio and Liu, 2008; Sharma et al., 2016). As is typically the case in NLP, such pipelines suffer from the problem of cascading errors; e.g., failures of the language identification will cause problems in the tag prediction (Barman et al., 2016). Other approaches have trained supervised models on POS-annotated, code-switched data (Jamatia et al., 2015; Ghosh et al., 2016; Gupta et al., 2017; Barman et al., 2016; Sequiera et al., 2015, *inter alia*), resources which are expensive to create and unavailable for most language pairs.

In this work, we present a novel POS-tagging model architecture that can be trained exclusively on available monolingual standard-orthography resources, but that can be applied to texts that contain code-switching and transliteration. Because our model learns from monolingual training data, we are able to learn a code-switched POS-tagger for any combination of languages with POS-annotated data, regardless of whether any existing

[1] The terms *code-mixing* and *-switching* are used for similar phenomena; for simplicity, we use only *code-switching*.

[2] We will use the term *transliterated* to imply conversion from a standard script to a non-standard script, e.g., Devanagari to Latin for Hindi, and *detransliterated* to imply the reverse, and, for simplicity, when we say *written in Devanagari* or *Latin*, we mean Devanagari or Latin *script*.

(annotated or unannotated) code-switched corpus exists. By allowing multiple word embeddings (one for each of the possible languages) to jointly represent each token, we avoid explicit language identification and the resulting error propagation.

2 Task Setup

Training data We train our POS tagger using the Universal Dependencies (UD) treebanks for Hindi and English, which are annotated for part-of-speech (Nivre et al., 2016).[3] The English treebank contains roughly 22K sentences; the Hindi contains roughly 16K sentences. Sentences in both languages are taken from news text and similarly formal sources, and the Hindi is written entirely in Devanagari.

Test data We evaluated our POS tagging models on the code-switched Twitter data released by Bhat et al. (2018), which is annotated using the UD POS tag set.[4] All tweets are written entirely in Latin script and have at least one token of both Hindi and English, with 45% of the total tokens being Hindi, 38% English, 17% labeled as universal, mixed, named entity, or acronym.

Word embeddings For monolingual word embeddings, we use *indic-word2vec*,[5] for which the English embeddings are trained from 280M sentences and Hindi from 40M. The word vectors are learned using a skip-gram model with negative sampling, implemented in the word2vec toolkit (Mikolov et al., 2013). We use the 50,000 most frequent words in each vocabulary.

Transliteration To convert words between scripts, we make use of the *Indictrans* open-source transliterator,[6] which is able to convert, bidirectionally, between Latin and the scripts of Indic languages (Bhat et al., 2015). Because Hindi does not have a formal system for transliteration, those writing Hindi in Latin—or English in Devanagari—tend to write phonetically according to their own conventions. Thus, we make use of the ability of Indictrans to generate multiple transliteration alternatives, simulating the variation in Hindi transliterated writing.

[3] universaldependencies.org/

[4] github.com/CodeMixed UniversalDependencies/UD_Hindi_English

[5] bitbucket.org/irshadbhat/indic-word2vec-embeddings

[6] github.com/libindic/indic-trans

3 Model

The central challenge of our task is to be able to train on data that differs drastically from the text seen at inference time. Because of the inherent limitations on what kinds of data are available—annotations and word embeddings are generally only available on monolingual data written in standard scripts—we are interested in designing a model that can be trained on monolingual sentences in which English is written in Latin script and Hindi in Devanagari, but is able to predict POS tags on code-mixed text in which both English and Hindi are written in Latin script.

We accomplish this by representing input words not as a single vector (or as a word embedding affixed to sub-word representations), but as a joint representation that maintains embeddings for each language such that they can be learned and updated during training from the monolingual data, and mixed appropriately at inference time without having to commit to one or the other language. As seen in Figure 1, our model represents an input as the concatenation of four parts: a Hindi word embedding, an English word embedding, the output of a bi-LSTM over Hindi character embeddings, and a bi-LSTM over English characters.

Since our model is trained on monolingual Hindi and English sentences, when a token is encountered during training, we know what language it is in. If we encounter an English sentence, then for each token, we look up its word embedding in the appropriate monolingual vector space to use in the English side of the full representation, and we use the word's characters as input to the English character LSTM. For Hindi sentences, the procedure is similar but for the other side of the full representation. In either case, when we are training on a word in one language, we use zero vectors as the inputs for the other language's side of the full input representation.

At inference time, we will encounter only Latin script words, but we will not know the language of any token. Therefore, we use our joint-representation model to represent the token as both languages simultaneously. The Latin word is looked up in the English word embeddings to use as input to the English side, and the Latin characters are used for the English character LSTM. For the Hindi side we detransliterate the input word to Devanagari and perform the same actions.

The design of our model has a few important

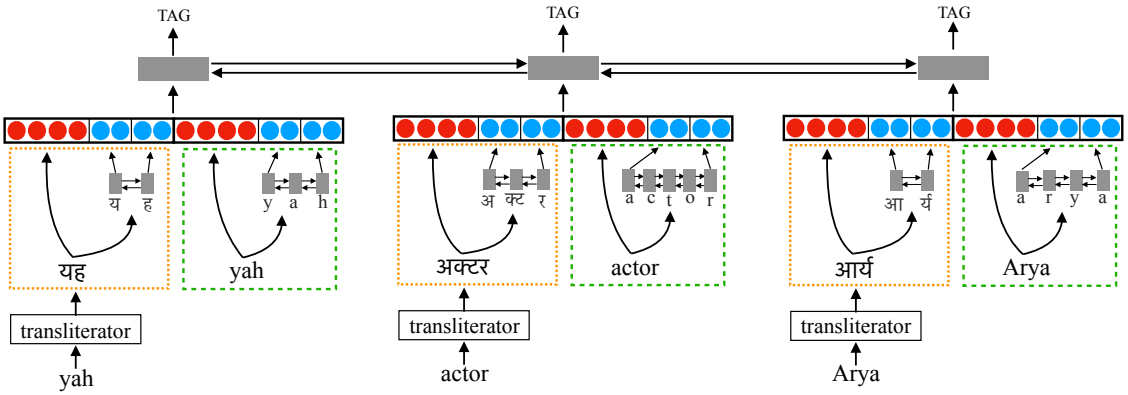

Figure 1: Our model. All inputs are treated simultaneously as both languages. At training time, for a (monolingual) Hindi input, only the contents of the orange/dotted boxes apply, and the English half of the vector is all zeros; for (monolingual) English, only the green/dashed boxes apply. At inference time, where the language of each token is unknown, the full figure applies.

advantages. First, because of Hindi's lack of a formalized transliteration system, Latin-script Hindi tends to have a very high amount of spelling variation. In our model (as opposed to the baseline model described below), the script conversion always goes from Latin to Devanagari, which naturally allows us to take unpredictable variant spellings and collapse them into standardized spellings, making it more likely for us to find the word in the embeddings listing. Second, there are some types of words for which it does not make sense to say that they are in one language or another. For example, a person's name might be equally valid as Devanagari Hindi or Latin English. In these cases, our model would allow both to influence a prediction as well as the rest of the bi-LSTM chain.

Finally, our model allows for the incorporation of external language prediction information, including soft predictions. To test this, we ran experiments in which we used a fully-unsupervised token-level language identification approach based on Rijhwani et al. (2017) to estimate a probability distribution over language labels for each token in the test data. Our approach was to run Forward-Backward on an HMM that was initialized to prefer same-language transitions and whose emissions were initialized using Laplace-smoothed maximum-likelihood-estimated character n-gram probabilities from each language's monolingual corpus (Dempster et al., 1977; Kupiec, 1992). We used the induced language probabilities to weight each side of the input representation at inference time: if the unsupervised model's output said that a particular test token's probability of being Hindi was 70%, then we would multiply the Hindi side of the input vector by 0.7 and the English side by 0.3, thus instructing the model to let the Hindi word embedding (and character-LSTM output) exert more influence during POS prediction.

4 Experiments

In our evaluation, we used the framework of a basic bi-LSTM tagger to compare our model for generating representations of tokens that maintain language ambiguity against a standard model that uses a single word embedding. In this section, we explain the experimental setup, baseline approach, and our experimental variants.[7]

4.1 POS tagger framework

For all of our experiments, we follow the basic structure of the bi-LSTM architecture of Bhat et al. (2018): word and sub-word embeddings are concatenated to form the input representations of each token, and connected together by a bi-LSTM that outputs POS predictions via multilayer perceptron and softmax layers (Hochreiter and Schmidhuber, 1997; Graves and Schmidhuber, 2005; Ling et al., 2015; Wang et al., 2015). Sub-word embeddings are the outputs of a bi-LSTM over characters (20-dim); word embeddings are initialized with *indic-word2vec* (64-dim) embeddings. Both character and word embeddings are updated during training. The bi-LSTM has 100

[7]Our code, implemented in DyNet (Neubig et al., 2017), can be found at https://github.com/kelseyball/cs-transliterated-pos-tagging.

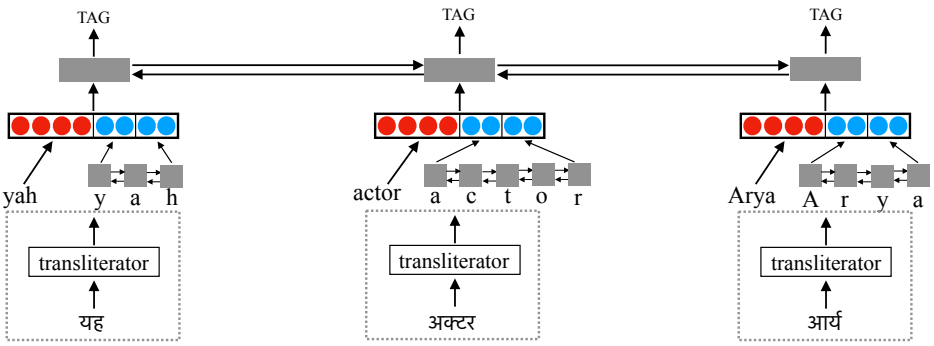

Figure 2: Baseline model. All inputs are treated as if they are one language. The contents of the dotted boxes only apply at training time and only to (monolingual) Hindi inputs since annotated Devanagari Hindi data must be converted to Latin script; at inference time, inputs are always in Latin script.

hidden states. We use an SGD trainer with learning rate=0.1 and dropout of 0.3 across 20 epochs. We train the model it by pooling POS-annotated English and Hindi UD sentences; the English is entirely Latin script, and Hindi is entirely Devanagari. Our baseline and experimental models differ in how a token is represented.

4.2 Baseline

Previous work on code-switched POS tagging has achieved accuracies as high as 90.20% (Bhat et al., 2018), but because those kinds of accuracies require supervised, in-domain training data, they are not directly comparable with the scenario that we are concerned with. Therefore, we compare against a natural baseline for our task: use a monolingual tagger and treat everything as if it were a single language. To deal with the problem of separately trained word embeddings, we use the approach and implementation of Artetxe et al. (2018)[8] to transform both sets of monolingual embeddings so that translationally equivalent words have similar embeddings. We also ran the baseline without this transformation, but found that using the transformed embeddings resulted in a 0.5% absolute improvement in accuracy over just using the separately trained embeddings directly.

Training For an English sentence, each token concatenates the English-lookup embedding to the character LSTM output, and similarly for Hindi, with the exception that the word is transliterated before inputting to the character LSTM. Because of the high spelling variation, we selected Latin spellings by sampling uniformly from the top-5 transliteration results. Always taking the 1-best resulted in a 4.3% absolute loss.

[8] github.com/artetxem/vecmap

Model		Acc.
Baseline	(monolingual representation)	70.92
Ours	(multi-language representation)	75.29
Ours	w/ forced language choice	76.04
Ours	w/ languages weighted by HMM	**77.40**
Ours	w/ oracle language choice	80.53
Bhat et al. (2018)	w/ supervision	90.20

Table 1: Results showing POS prediction accuracy. The baseline model treats all inputs as effectively monolingual (§4.2); the middle three lines are our model: giving equal weight to each language's side of the token's representation, using the unsupervised HMM 1-best output to zero-out one language's side of the token's representation, and using the unsupervised HMM probability estimates to weight each side of the token's representation. The oracle line is our model using gold language labels to zero-out one language's side.

Inference Each word is looked up in the English embeddings *and* its detransliteration is looked up in the in the Hindi embeddings. If the word is found in only one, that vector is used, and if found in neither, we use an *unknown word* vector. For a fair comparison to our model, if both lookups return results, we use the 1-best prediction from the unsupervised language identification HMM as discussed in §3 to decide which to use.

4.3 Unsupervised soft language prediction

One feature of our model is its ability to incorporate externally-derived, per-token soft language

predictions as weights at inference time without having to make hard language decisions. To evaluate whether this improves accuracy, we used the fully-unsupervised HMM language prediction (§3) and tested our model using: no weighting, leaving the embeddings untouched; using the HMM's 1-best prediction to zero out the other language's inputs; and multiplying each language's embeddings by its probability estimated by the HMM.

The results of our evaluation can be seen in Table 1. While the best scenario for the baseline tagger gives 70.92% accuracy, the most basic setup for our model yields 75.29%. Informing our model with some fully-unsupervised HMM-derived language information, but forcing a hard language decision, is somewhat beneficial, yielding 76.04%, though it trades off the benefits of flexibility and generality in representation for some extra information. And finally, using that unsupervised information for *soft* weighting performs even better, reaching 77.40%, a total absolute improvement over the baseline of ∼6.5%, clearly demonstrating the value of maintaining ambiguity about a word's language.

5 Conclusion

In this paper, we presented a novel POS-tagging model for code-mixed, transliterated text that avoids the need for expensive, supervised corpora or language-identification pipelines. We demonstrated that avoiding explicit language identification in code-switched contexts is beneficial. Though we emphasize the paucity of annotated code-switched corpora, previous work has demonstrated that a small amount of supervision can go a long way (Garrette and Baldridge, 2013); future work might explore how much supervision is needed to close the gap between our model and Bhat et al. (2018).

While we address some of the most prominent issues with code-switching, our model does not deal with style, formality, or domain mismatches between the formal training data and informal evaluation data. This problem is treated in recent work on POS-tagging for social media text (Owoputi et al., 2013; Gimpel et al., 2011), but these issues have yet to be fully explored in code-switched contexts. We leave these for future work.

Acknowledgements

This work was supported by a Fulbright-Nehru grant from the United States-India Educational Foundation (USIEF) for the first author. We would like to thank Irshad Bhat, Dipti Misra Sharma, Manish Shrivastava, Karl Pichotta, Emily Pitler, Luke Zettlemoyer, Phoebe Mulcaire, Gaurav Singh Tomar, Michael Collins, and the anonymous reviewers for their assistance and feedback.

References

Mikel Artetxe, Gorka Labaka, and Eneko Agirre. 2018. Generalizing and improving bilingual word embedding mappings with a multi-step framework of linear transformations. In *Proceedings of AAAI*.

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*.

Irshad Ahmad Bhat, Riyaz Ahmad Bhat, Manish Shrivastava, and Dipti Misra Sharma. 2018. Universal dependency parsing for Hindi-English code-switching. In *Proceedings of NAACL*.

Irshad Ahmad Bhat, Vandan Mujadia, Aniruddha Tammewar, Riyaz Ahmad Bhat, and Manish Shrivastava. 2015. IIIT-H system submission for FIRE2014 shared task on transliterated search. In *Proceedings of the Forum for Information Retrieval Evaluation*, FIRE '14.

A. P. Dempster, N. M. Laird, and D. B. Rubin. 1977. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1).

Dan Garrette and Jason Baldridge. 2013. Learning a part-of-speech tagger from two hours of annotation. In *Proceedings of NAACL*.

Souvick Ghosh, Satanu Ghosh, and Dipankar Das. 2016. Part-of-speech tagging of code-mixed social media text. In *Proceedings of the Second Workshop on Computational Approaches to Code Switching*.

Kevin Gimpel, Nathan Schneider, Brendan O'Connor, Dipanjan Das, Daniel Mills, Jacob Eisenstein, Michael Heilman, Dani Yogatama, Jeffrey Flanigan, and Noah A Smith. 2011. Part-of-speech tagging for Twitter: Annotation, features, and experiments. In *Proceedings of ACL*.

Alex Graves and Jurgen Schmidhuber. 2005. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks*, 18(5).

Deepak Kumar Gupta, Shubham Tripathi, Asif Ekbal, and Pushpak Bhattacharyya. 2017. SMPOST: parts of speech tagger for code-mixed Indic social media text. *http://arxiv.org/abs/1702.00167*.

Sepp Hochreiter and Jurgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8).

Anupam Jamatia, Björn Gambäck, and Amitava Das. 2015. Part-of-speech tagging for code-mixed English-Hindi Twitter and Facebook chat messages. In *Proceedings of RANLP*.

Julian Kupiec. 1992. Robust part-of-speech tagging using a hidden Markov model. *Speech and Language*, 6.

Wang Ling, Chris Dyer, Alan W Black, Isabel Trancoso, Ramon Fermandez, Silvio Amir, Luis Marujo, and Tiago Luis. 2015. Finding function in form: Compositional character models for open vocabulary word representation. In *Proceedings of EMNLP*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Proceedings of NIPS*.

Graham Neubig, Chris Dyer, Yoav Goldberg, Austin Matthews, et al. 2017. Dynet: The dynamic neural network toolkit. *ArXiv e-prints*.

Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Yoav Goldberg, Jan Hajic, Christopher D Manning, Ryan T McDonald, Slav Petrov, Sampo Pyysalo, Natalia Silveira, et al. 2016. Universal dependencies v1: A multilingual treebank collection. In *Proceedings of LREC*.

Olutobi Owoputi, Brendan O'Connor, Chris Dyer, Kevin Gimpel, Nathan Schneider, and Noah A Smith. 2013. Improved part-of-speech tagging for online conversational text with word clusters. In *Proceedings of NAACL*.

Shruti Rijhwani, Royal Sequiera, Monojit Choudhury, Kalika Bali, and Chandra Sekhar Maddila. 2017. Estimating code-switching on Twitter with a novel generalized word-level language detection technique. In *Proceedings of ACL*.

Royal Sequiera, Monojit Choudhury, and Kalika Bali. 2015. POS tagging of Hindi-English code mixed text from social media: Some machine learning experiments. In *Proceedings of International Conference on NLP*.

Arnav Sharma, Sakshi Gupta, Raveesh Motlani, Piyush Bansal, Manish Shrivastava, Radhika Mamidi, and Dipti Misra Sharma. 2016. Shallow parsing pipeline for Hindi-English code-mixed social media text. In *Proceedings of NAACL*.

Thamar Solorio and Yang Liu. 2008. Part-of-speech tagging for English-Spanish code-switched text. In *Proceedings of EMNLP*.

Yogarshi Vyas, Spandana Gella, Jatin Sharma, Kalika Bali, and Monojit Choudhury. 2014. POS tagging of English-Hindi code-mixed social media content. In *Proceedings of EMNLP*.

Peilu Wang, Yao Qian, Frank K. Soong, Lei He, and Hai Zhao. 2015. Part-of-speech tagging with bidirectional long short-term memory recurrent neural network. *arXiv preprint abs/1510.06168*.