Automatic Processing of Code-mixed Social Media Content

Utsab Barman

B.Tech., M.Tech.

A dissertation submitted in fulfilment of the requirements for the award of

Doctor of Philosophy (Ph.D.)

to the

Dublin City University
School of Computing

Supervisors:
Dr. Jennifer Foster
Dr. Joachim Wagner

July 2018

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of Ph.D. is entirely my own work, that I have exercised reasonable care to ensure that the work is original, and does not to the best of my knowledge breach any law of copyright, and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

Signed:

(Candidate) ID No.:

Date:

Contents

List of Figures

List of Tables

Acknowledgments

My deepest regards and thanks to my supervisors, Dr. Jennifer Foster and Dr. Joachim Wagner, probably, the best supervisors one can ever have. Without them finishing this thesis was not possible by any means. I also like to thank Dr. Andy Way and Dr. Qun Liu for their motivation and support. My sincere thanks to Dr. Monojit Choudhury for his valuable comments, suggestions and inspirations. I would like to thank ADAPT centre for providing me the opportunity to pursue this PhD. My thanks and regards to the colleagues and friends of ADAPT. Finally, I would like to thank my parents for their support and love, without them this journey was impossible.

Automatic Processing of Code-mixed Social Media Content

Utsab Barman

Abstract

Code-mixing or language-mixing is a linguistic phenomenon where multiple language mix together during conversation. Standard natural language processing (NLP) tools such as part-of-speech (POS) tagger and parsers perform poorly because such tools are generally trained with monolingual content. Thus there is a need for code-mixed NLP. This research focuses on creating a code-mixed corpus in English-Hindi-Bengali and using it to develop a world-level language identifier and a POS tagger for such code-mixed content. The first target of this research is word-level language identification. A data set of romanised and code-mixed content written in English, Hindi and Bengali was created and annotated. Word-level language identification (LID) was performed on this data using dictionaries and machine learning techniques. We find that among a dictionary-based system, a character-n-gram based linear model, a character-n-gram based first order Conditional Random Fields (CRF) and a recurrent neural network in the form of a Long Short Term Memory (LSTM) that consider words as well as characters, LSTM outperformed the other methods. We also took part in the First Workshop of Computational Approaches to Code-Switching, EMNLP, 2014 where we achieved the highest token-level accuracy in the word-level language identification task of Nepali-English. The second target of this research is part-of-speech (POS) tagging. POS tagging methods for code-mixed data (e.g. pipeline and stacked systems and LSTM-based neural models) have been implemented, among them, neural approach outperformed the other approach. Further, we investigate building a joint model to perform language identification and POS tagging jointly. We compare between a factorial CRF (FCRF) based joint model and three LSTM-based multi-task models for word-level language identification and POS tagging. The neural models achieve good accuracy in language

identification and POS tagging by outperforming the FCRF approach. Furthermore, we found that it is better to go for a multi-task learning approach than to perform individual task (e.g. language identification and POS tagging) using neural approach. Comparison between the three neural approaches revealed that without using task-specific recurrent layers, it is possible to achieve good accuracy by careful handling of output layers for these two tasks e.g. LID and POS tagging.

Chapter 1

Introduction

Code-mixing is a linguistic phenomenon where language switching occurs at a sentence boundary (inter-sentential), or within a sentence (intra-sentential) or within a word (word-level). This phenomenon can be observed among multilingual speakers in many languages. While communicating, writers/speakers mix their native and a foreign language and/or they embed foreign language words in native language sentences. It is a well known practice among social media users from language-dense areas and from bilingual or multilingual societies (Cárdenas-Claros and Isharyanti, 2009; Shafie and Nayan, 2013). It can occur not only between languages but also dialects of the same language (Gardner-Chloros, 1991). Further, languages that have non-Roman script are often written using Roman script in social media and SMS communication (Sowmya et al., 2010). This is known as Romanisation. Figure 1.1 is an example of Romanisation, where Bengali has been written using Roman scripts. The following is an example, taken from a Facebook group of Indian students which exhibits trilingual code-mixing and Romanisation:

Bengali Script: একটা পোস্ট লাইক করেছে বলেই এই !!!

Phonetically Typed: ekta post like koreche bolei ei !!!!

Figure 1.1: Example of Romanisation, Bengali has been written using Roman scripts.

Example 1.0.1. *Original: Yaar tu to, GOD hain.* ***tui*** *JU* ***te ki korchis****? Hail u man!*

Translation: Buddy you are GOD. What are you doing in JU? Hail u man!

Three languages are present in this comment: English, Hindi (italics) and Bengali (bold). Bengali and Hindi words are written in romanised forms. These phenomena (code-mixing and Romanisation) can occur simultaneously and increase the ambiguity of words. For example, in the previous comment, 'to' could be mistaken as an English word but it is a romanised Hindi word. Moreover, the romanised form of a native word may vary according to the user's preference. When speakers mix languages within a conversation, a sentence or even within a word and Romanisation is present, standard NLP tasks, for example, Language Identification (LID), Part-of-Speech (POS) Tagging and Name Entity Recognition (NER) are challenging.

Code-mixed NLP in Indian languages is an interesting problem and it is being investigated by a number of researchers (Das and Gambäck, 2014; Sequiera et al., 2015b; Dey and Fung, 2014; Bhat et al., 2018) in recent years. Indian languages that belong to the same language family (e.g. Bengali and Hindi) also share a large number of common words. The romanised versions of such words may follow the same orthographic structure which will increase the ambiguity for automatic processing. Besides, there are also other challenges associated with code-mixed NLP, discussed in the next subsection (Section 1.1).

In this thesis, we present our studies of automatic processing (LID and POS Tagging) of code-mixed social media content in the following Indian languages: English, Bengali and Hindi[1]. The rest of the chapter is organized as follows: Section 1.1 describes the challenges associated with code-mixed NLP in Indian language. We formulate our research questions in Section 1.2. Section 1.3 enlists the contributions of this thesis and in 1.4 we describe the structure of this thesis.

[1]We have also carried out SVM based LID for Nepali-English data in the shared task of LID at the First Workshop on Computational Approaches to Code Switching, EMNLP, 2014 Solorio et al. (2014b). These results are included in the Appendix.

1.1 Challenges of Code-Mixed NLP

Following are the challenges associated with automatic NLP when dealing with code-mixed and romanised Indian social media content.

- **Ambiguous Words:** Let us consider the Example 1.0.1, the word 'to' in the example could be mistaken as an English word but it is a romanised Hindi word. Again, 'man' is a English word but the similar lexical form can be viewed as a variable representation of a romanised Bengali word 'mon'. Furthermore, Bengali and Hindi belong to the same language family and *share a set of common vocabulary*. For example, 'baba', 'ma', 'ghar', 'desh' belongs to the shared vocabulary of Bengali, Hindi and many other Indian languages. During automatic processing (e.g. LID, POS tagging), these words must be tagged properly given the proper context.

- **Variable Lexical Representations:** The romanised form of a native word may vary according to the user's preference. There is no standard lexical form for most romanised Bengali and Hindi words. For example, 'korchis' in Example 1.0.1 is a romanised Bengali word and it can be written as 'krchis', 'krchs', 'korchs' and 'karchis'.

- **Word-level Code-mixing:** Language mixing also occur at word-level. For example, 'chapless' is mixed word of Bengali and English. The first part 'chap' is a Bengali word and the latter part 'less' is an English suffix. To detect and to identify the involved languages for such words is probably the deepest level of language identification which needs to be addressed for automatic code-mixed NLP.

- **Reduplication:** Another challenge is proper tagging of reduplicated words. This phenomenon is quite common in Indian languages. For example, 'deka teka', the first word 'deka' means 'to see' in Bengali, however the second word does not have a meaning on its own but together with the first word it

becomes a multi word expression (MWE) which means 'to see'. Reduplication also takes part in code-mixing in an interesting fashion. For example 'post fost', 'boss toss'. The first words (i.e. 'post' and 'boss') in these example are English which are followed by Bengali reduplicated words (i.e. 'fost' and 'toss'). Moreover in the second example, 'toss' also bears lexical ambiguity (i.e. English word 'Toss'). Word-level LID and POS tagging of such expressions are difficult when multiple languages and Romanisation is involved.

- **Word Orders:** Indian languages and English follow different word orders. English follows Subject-Verb-Object format. On the other hand, Indian languages (e.g. Bengali and Hindi) follow Subject-Object-Verb format. When code-mixing occurs between English and Indian languages, these two types of word-order get mixed up. This phenomenon presents challenges to POS tagging. POS taggers trained with monolingual content on a particular language might get biased to a specific word-order and perform poorly for the other languages.

1.2 Research Questions

Code-mixing is a prominent characteristic of multilingual social media content. The nature of social media content is noisy and exhibits several challenges for automatic NLP, for example, short-length, spelling variation, non-grammatical constructs. Code-mixing increases the challenges by incorporating lexical ambiguity (i.e. same spelling but different meanings in different languages). For this reason, code-mixing is attracting serious attention within the NLP community worldwide. In recent research various challenges associated with code-mixed NLP (e.g. language modelling, POS tagging, parsing, machine translation) have been investigated and discussed in depth (Çetinoğlu et al., 2016). A number of workshops are being organized to re-investigate standard NLP tasks for code-mixed social media data, for example, Language Identification (LID) (Solorio et al., 2014a), Part-of-Speech

(POS) tagging (Jamatia and Das, 2016), Sentiment Analysis (Patra et al., 2018).

The aim of this thesis is to develop NLP capabilities for automatic processing of code-mixed and romanised social media content. We focus on two important tasks: (1) word-level LID and (2) POS tagging. The first one is essential for a number of NLP tasks (e.g. POS tagging, Machine Translation) and the second one also plays a vital role for automatic understanding (e.g. sentiment analysis) of code-mixed social media content. We also introduce a novel multi-tasking framework to perform LID and POS tagging jointly. Though we use standard machine learning (ML) methods like Support Vector Machines (SVM) and Conditional Random Fields (CRF), we also explore the use of Recurrent Neural Networks (RNN). Standard machine learning methods relies on hand-crafted features. The generation of such features demands human effort and domain knowledge. On the other hand, neural approaches learn the representations or features needed for the task automatically. In recent NLP literature, it has been seen that neural approaches outperform standard feature based ML methods significantly (Collobert and Weston, 2008). With this in mind, we formulate our first research question:

> *1. Is it possible to achieve better performance than feature based ML (SVM and CRF) by using a neural approach for LID and POS tagging of code-mixed social media content?*

It has been seen in recent literature of code-mixed NLP that monolingual taggers are used to achieve multilingual objectives (Solorio and Liu, 2008b; Vyas et al., 2014). Considering these facts we aim to investigate the following question:

> *2. How well can we exploit monolingual taggers to perform POS tagging for code-mixed data? Further, can a neural approach be effective?*

Recent studies have shown that performing a joint labelling or multi-task learning (MTL) results in better performance than performing individual tasks (Chen et al., 2016). Multi-task learning with code-mixed data is a relatively under explored area.

In that context we want to investigate a multi-task model for code-mixed data that performs LID and POS tagging jointly. Considering the previous studies of MTL in NLP, it is interesting to see:

> *3. How well does a neural MTL approach perform for code-mixed content? Does MTL achieve better accuracy than individual taggers for code-mixed data?*

Based on these (1, 2 and 3), we present task specific research questions for LID in Section 1.3 of Chapter 4, for POS tagging we present task specific research questions in Section 1.2 of Chapter 5 and for MTL these questions are presented in Section 1.3 of Chapter 6.

1.3 Contributions

In this thesis, we present our study 'Automatic Processing of Code-mixed Social Media Content' with a trilingual (English-Bengali-Hindi) code-mixed corpus. We investigate word-level LID, POS tagging and a muti-tasking framework to perform LID and POS tagging simultaneously. Following are the contributions of this research:

- **Corpus:** Language and POS annotated English-Bengali-Hindi code-mixed corpus, collected from Facebook (Barman et al., 2014a, 2016). This corpus has been used by other researchers (Das and Gambäck, 2014; Çetinoğlu et al., 2016).

- **Experiments on Language Identification:** We perform word-level language identification using dictionaries, SVM, CRF and LSTM. We investigate the use of hand-crafted features (e.g. character-n-grams, presence in dictionaries, length of a word and capitalisation information) with SVM and CRF. We investigate three different word embeddings (e.g. word2vec, CWE and fasttext) with LSTM. We propose a novel LSTM-based neural architecture that

combines word and character embeddings to learn a common representation of words, that can be useful for word-level LID with code-mixed data. We find that the proposed model can outperform a word-level LSTM based system significantly. Moreover, it can also achieve good word-level performance for less frequent instances (e.g. name entity, acronym and word-level code-mixing) in our code-mixed corpus.

- **Experiments on POS Tagging:** We investigate POS tagging with our code-mixed corpus using SVM and CRF with hand-crafted features, using a pipeline of a language identifier and three monolingual taggers (e.g. English, Hindi and Bengali), stacking (Solorio and Liu, 2008b) approaches that combine monolingual taggers, language identifiers and transliterations of romanised code-mixed data and a word + character-level LSTM-based system (same architecture as LID) that incorporates language information in word representation. We find that, among all of these approaches, the LSTM-based approach performs best. We also find that a stacking approach is better than the monolingual tagger combination, and features like LID labels and transliterations of tokens increase the performance of a stacking approach.

- **Experiments on MTL approaches:** We perform joint labelling of LID and POS tagging using a Factorial CRF and three neural MTL approaches. To the best of our knowledge this is the first attempt to employ neural MTL to perform word-level tasks (e.g. LID and POS tagging) with code-mixed social media content. We find that a neural approach that uses a common representation of words to predict multiple labels, LID and POS, is outperformed by individual neural LID and POS taggers. On the other hand, the two other neural approaches that use the common representation of words and share the output of one task to the input of another task, can perform better than the previously mentioned neural MTL approach and individual neural LID and POS taggers. Finally, we show that it is not always necessary to introduce

multiple RNN layers to model multiple tasks, using only a shared RNN layer and output propagation it is possible to achieve similar performance. Further, a shared RNN layer combined with output propagation can reduce the number of model parameters and training time significantly compared to a MTL model that uses multiple RNN to model multiple tasks.

The following papers were published during the course of the project.

- Barman, U., Das, A., Wagner, J., and Foster, J. (2014a). Code mixing: A challenge for language identification in the language of social media. In Proceedings of the First Workshop on Computational Approaches to Code Switching. EMNLP 2014, Conference on Empirical Methods in Natural Language Processing, pages 13–23, Doha, Qatar. Association for Computational Linguistics.

  This paper describes our data collection (English-Bengali-Hindi code-mixed corpus), annotation (language) and initial experiments on word-level LID.

- Barman, U., Wagner, J., Chrupa la, G., and Foster, J. (2014b). DCU-UVT: Word-level language classification with code-mixed data. In Proceedings of the First Workshop on Computational Approaches to Code Switching , pages 127–132.

  This paper is the result of our participation in the shared task of LID at the First Workshop on Computational Approaches to Code Switching, EMNLP, 2014. In this shared task our language identifier for Nepali-English code-mixed social media content achieved highest word-level accuracy among all participants.

- Barman, U., Wagner, J., and Foster, J. (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.

This paper describes the POS annotated subset of our English-Bengali-Hindi code-mixed corpus. In this paper we compare three approaches to POS tagging: (1) pipeline (e.g. linear combination of LID and monolingual taggers), (2) stacking and (3) joint modelling using Factorial CRF (FCRF).

The following paper was published during the course of the project but this is not related to the topic of this thesis.

- Wagner, J., Arora, P., Cortes, S., Barman, U., Bogdanova, D., Foster, J., and Tounsi, L. (2014). DCU: Aspect-based polarity classification for semeval task 4. In Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval2014), pages 223–229.

  This paper describes our work on aspect-based sentiment analysis. Our SVM-based system achieved the best performance among all participants of this shared task.

1.4 Structure of Thesis

The rest of the thesis is structured as follows:

- **Chapter 2:** In this chapter we discuss linguistic aspects of code-mixing including types and motivations for code-mixing and two popular linguistic theories of code-mixing. We also describe some aspects of code-mixing in Indian languages.

- **Chapter 3:** In this chapter we describe our data collection and annotation process. Our corpus is a trilingual romanised code-mixed corpus of English, Bengali and Hindi. We also present the characteristics of the annotated data in this chapter.

- **Chapter 4:** This chapter describes the machine learning methods used in our experiments. These methods are Support Vector Machine (SVM), Condi-

tional Random Fields (CRF) and Recurrent Neural Networks (RNN). We also describe neural word embeddings in this chapter.

- **Chapter 5:** Word-level language identification is a key preprocessing step that needs to be resolved before any other steps, e.g. part-of-speech tagging and parsing. The task is to assign language labels to each word of a sentence. We report our language identification experiments with English-Hindi-Bengali code-mixed data using both dictionary lookup and machine learning methods, e.g. SVM, CRF and RNN in the form of Long Short Term Memory (LSTM).

- **Chapter 6:** POS tagging is the process of assigning each word in a sentence to its particular POS. This is an important step that helps in further understanding (sentiment analysis, named-entity recognition etc.) of natural language. In this chapter we present our experiments on POS tagging with code-mixed data using machine learning methods (e.g. SVM, CRF and LSTM) and using combinations of monolingual taggers.

- **Chapter 7:** Multi-task learning is the approach of modelling several tasks together. In this chapter, we perform multi-task learning, LID and POS tagging of code-mixed data using a Factorial CRF and three neural approaches. These neural approaches are mainly based on LSTM but differ in architectures.

- **Chapter 8:** In this chapter, we discuss our findings based on the research questions of individual chapters and propose future directions for this research.

Chapter 2

Code-mixing: Linguistic Aspects

A number of researchers have investigated different aspects of code-mixing including socio-linguistic (Auer, 1984; Kachru, 1978) and structural analysis (Sankoff and Poplack, 1981; Myers-Scotton and Jake, 1995; Muysken et al., 2000). In this chapter we discuss these aspects of code-mixing. We also describe two widely discussed theories of code-mixing - the grammatical constraint model of Poplack (1980) and the Matrix Language Frame theory of Myers-Scotton and Jake (1995) in this chapter.

2.1 Types of Code-mixing

Scholars distinguish between inter-sentence, intra-sentence and intra-word code mixing. **Inter-Sentential** code-mixing occurs at sentence boundaries. Example 2.1.1 is an instance of Hindi-English inter-sentential code-mixing. English is written in bold and Hindi in italics.

Example 2.1.1.

Itna izzat diye aapne mujhe !!! ***Tears of joy.*** *:'(:'(*

Some researchers, (Poplack, 1980) also refer to **'extra-sentential'** code-mixing, this occurs when certain words or phrases from one language are inserted at the end of a sentence of another language. Some English examples are: "okay", "well" or "you know". **Intra-Sentential** mixing occurs within a sentence or a clause. Exam-

ple 2.1.2 is an example of English-Bengali intra-sentential code-mixing. English is written in bold and Bengali in italics.

Example 2.1.2.

... sei bujhe ***i have 2 choose between*** *dwaitto nie chokano ba* ***move on*** *,*

however difficult it may be ...

On the other hand, **word-level** mixing occurs within a word itself, such as at a morpheme boundary. In Example 2.1.3 the word '**tym***er*' is an example of word-level code-mixing. The word '**tym**' is a lexical variant of the English word 'time' and 'er' is a Bengali genitive inflection. Similarly in Example 2.1.4 the word '*chap***less**' is another example of word-level code-mixing. In the text fragment the word '*chap*' is Bengali. It means 'tension' and 'less' is an English suffix. The meaning of this text fragment is 'tension-less job'.

Example 2.1.3. ***tym****er dam*

English Translation: value of time

Example 2.1.4. *chap****less*** *kaj*

English Translation: tension-less job

2.2 Motivations for Code-mixing

Studies can be found in the literature (Milroy and Muysken, 1995; Muysken et al., 2000; Alex, 2008; Sharma and Motlani, 2015; Auer, 2013) that mainly discuss the sociological and conversational necessities behind code-mixing as well as its linguistic properties. While investigating the types of code-mixing, Muysken et al. (2000) described some linguistic processes that are responsible for the generation of different types and patterns of code-mixed content: these are , 'insertion', 'alternation' and 'congruent lexicalisation'. Muysken et al. (2000) define code-mixing as following:

> "... all cases where lexical items and grammatical features from two languages appear in one sentence".

Some researchers have stated that code-mixing is the representation of the exchange of language models, like morphemes, terms, or sentences from one language into another (Kachru, 1976). Li (2000) and San (2009) indicate that mainly linguistic motivations trigger code-mixing in highly bilingual societies. On the other hand, Dewaele (2010) claims that 'strong emotional arousal' instigates code-mixing. Social media study (Hidayat, 2012) show that social media users mainly use inter-sentential switching, and reports that 45% of the switching is motivated by lexical needs, 40% is used for talking about a particular topic, and 5% for content clarification. Dey and Fung (2014) present a speech corpus of English-Hindi code-mixing in student interviews. They analyse the grammatical contexts and motivations behind code-mixing in their corpus. They find that each sentence in their corpus contains almost 68% of Hindi word and 32% English words. Moreover, intra-sentential code-mixing is more prominent than inter-sentential switching in their corpus. They also find that most Hindi speakers, during the interviews, perform code-switching (from English to Hindi) because of the ease of use.

2.3 Linguistic Theories of Code-mixing

2.3.1 Grammatical Constraints of Poplack (1980)

Poplack (1980); Sankoff and Poplack (1981) investigated the grammatical aspects of code-switching and proposed that during code-switching the constituents preserve their monolingual structural characteristics. According to (Poplack, 1980; Sankoff and Poplack, 1981) code-mixing is governed by two constraints: the free-morpheme constraint and the equivalence constraint. The free morpheme constraint indicates that a language switch can occur between a lexical form and a bound morpheme unless that lexical form is integrated into the language of the bound morpheme. The second constraint states that language switches happen only at points where the surface structures of the languages coincide, i.e. that in a bilingual code-switched sentence, a code switch tends to occur at a point where the juxtaposition of languages

does not 'violate a syntactic rule of either language'. Let us consider an example:

Example 2.3.1. *chap nei* ***I 'll take care of that*** *...*

The above snippet is an example of Bengali (italics) English (bold) bilingual code-mixing and supports the model of Poplack (1980) because the constituents on both sides of the switch are grammatical in Bengali and English. However, scholars often criticise this theory for being insufficiently restrictive, sometimes code-mixed sentences that are valid by the theory does not occur in real life communication. For example:

Example 2.3.2. *Original:*

They had visto la película italiana

Translation: They had seen the Italian movie

The above sentence seems to be valid by free-morpheme constraint but generally does not appear in Spanish English code-mixing (Belazi et al., 1994).

2.3.2 Matrix Language Frame (MLF)

Matrix Language Frame (Joshi, 1982; Myers-Scotton, 1997, 1995) proposes a framework to explain bilingual code-mixing. It assumes, during bilingual intra-sentential code-mixing, the participating languages are related in the following way - (1) one language (L1) provides a morpho-syntactic frame which is also the dominant language, (2) the other language (L2) acts as guest and words from the other language are embedded in the morpho-syntactic frame of L1. Let us consider the following snippet:

Example 2.3.3. *Yaar tu to* ***GOD*** *hain.*

In the above example, two languages are present English (bold) and Hindi. It can be clearly seen that Hindi is the matrix language of the sentence and English word 'GOD' is 'embedded' in the sentence. In MLF model morphemes are distinguished

in two categories - the content morphemes and system morphemes. Content morphemes comes from the embedded language (L2) and are generally - nouns, verbs and adjectives. In the above example the word 'GOD' is an example of a content morpheme. On the other hand, morphemes that constitutes the grammatical frame are system morphemes (e.g function words) . According to this model, during bilingual code-mixing, system morphemes can only come from the matrix language and content morphemes can be taken from both the matrix language and embedded language. In this theory, distinctions are also made between the matrix language hierarchy and embedded language hierarchy. The former can be described as one language taking control of the grammar of the bilingual utterance and latter can be described as when the switched elements are at the periphery of the utterance. Bhat et al. (2016); Winata et al. (2018b) have argued about the limitations of MLF regarding the identification of the matrix language, sometimes it is not straightforward due to the criteria by which matrix language is determined.

It is worth mentioning that in recent years, these two models were implemented computationally. The models of Poplack (1980) and Myers-Scotton and Jake (1995) were implemented in recent studies (Pratapa et al., 2018; Bhat et al., 2016). Pratapa et al. (2018) tried to create synthetic grammatically valid code-mixed data based on the Equivalence Constraint theory of Poplack (1980). They showed that appropriate use of this synthetic data with original data reduce the perplexity of an code-mixed language model. On the other hand Bhat et al. (2016) implemented models for intra-sentential code-switching based on the Equivalence Constraint (Poplack, 1980) and Matrix Language Frame (Myers-Scotton and Jake, 1995) and proposed a new model that combines features of both the theories. They also observed that both of the models are highly constrained and do not allow some commonly seen code-mixed patterns of Hindi-English data of social media.

2.4 Indian Languages and Code-mixing

Some important studies of code-mixing in Indian languages was done by Kachru (Kachru, 1976, 1986, 1983, 1978). He described code-mixing as a 'nativisation' process (Kachru, 1983). He also pointed out that in South Asia as carrying English is a symbol of 'modernization', 'success', and 'mobility' (Kachru, 1986). Code mixing is very frequent in the Indian sub-continent (Gambäck and Das, 2014) because languages change within very short geographical distances and people generally have a basic knowledge of their neighboring languages. India is a country with hundreds of spoken languages, among which 22 are official [1]. English and Hindi are two of the widely spoken languages of India. Almost every state in India has their own language. Further, a number of dialects are also possible for Indian languages. The nature of code-mixing in Indian languages varies from state to state. For example, in a state where Bengali is the main language (e.g. West Bengal), code-mixing can occur between English-Bengali, English-Hindi, Bengali-Hindi, moreover trilingual mixing of English-Hindi-Bengali is also possible. Kachru (1983) called the processes of mixing as Englishization (English is mixed), Nativisation (where Native language is mixed). It is worth mentioning that Indian languages that belong to the same language family (e.g. Bengali and Hindi) also share a large number of common words. The romanised versions of such words may follow the same orthographic structure which will increase the ambiguity for automatic processing of code-mixed content from Indian social media.

2.5 Conclusion

In this chapter, we have discussed different types of code-mixing, motivations behind code-mixing and two popular linguistic theories regarding code-mixing. We have also discussed the nature of Indian language code-mixing. In the next chapter, we will describe the data and the annotation process of our code-mixed corpus.

[1] `https://en.wikipedia.org/wiki/Languages_with_official_status_in_India`

Chapter 3

Data

This chapter talks about the data collection and the annotation procedure for social media text. The collected and processed data set is a mixture of three languages: English, Hindi and Bengali. To perform code-mixed NLP we choose Indian languages because India is a country hundreds of spoken languages, among which 22 are official [1]. Code mixing is very frequent in the Indian sub-continent (Gambäck and Das, 2014) because languages change within very short geographical distances and people generally have a basic knowledge of their neighboring languages.

The data source for this research is a Facebook forum of Jadavpur University, Kolkata, India. The corpus is created using the posts and the comments of this forum. The reason to select such data source is that researchers have found, code mixing is frequent among speakers who are multilingual and younger in age (Cárdenas-Claros and Isharyanti, 2009). The participants of this forum are students between the 20-30 year age group. The university is located at Kolkata, West Bengal, which is one of the metro cities of India, where Bengali is the major spoken language. However, it is the political and the cultural capital of West Bengal so Hindi and English is also spoken widely. The young population of the city is trilingual. Bengali is the mother tongue, Hindi is widely spoken for geo-cultural reasons and English is the medium of higher education. This chapter describes the corpus creation, the

[1] https://en.wikipedia.org/wiki/Languages_with_official_status_in_India

annotation rules and data characteristics. This corpus has been used create the language identifiers, the POS taggers and the multitasking systems.

3.1 English-Hindi-Bengali Facebook Corpus

A Facebook group[2] and 11 Facebook users (known to the authors) were selected to obtain publicly available posts and comments. The Facebook graph API explorer was used for data collection. Since these Facebook users are from West Bengal, the most dominant language is Bengali (Native Language), followed by English and then Hindi (National Language of India). The posts and comments in Bengali and Hindi script were discarded during data collection, resulting in 2335 posts and 9813 comments.

3.1.1 Annotation

Four annotators took part in the annotation task. Three were computer science students and the other was one of the author. The annotators are proficient in all three languages of our corpus. A simple annotation tool was developed which enabled these annotators to identify and distinguish the different languages present in the content by tagging them. Annotators were supplied with 4 basic tags (viz. *sent*ence, *frag*ment, *incl*usion and *wlcm* (word-level code mixing)) to annotate different levels of code mixing. Under each tag, six attributes were provided, viz. *English (en), Bengali (bn), Hindi (hi), Mixed (mixd), Universal (univ) and Undefined (undef).* The attribute *univ* is associated with symbols, numbers, emoticons and universal expressions (e.g. *hahaha*, *lol*). The attribute *undef* is specified for a sentence or a word for which no language tags can be attributed or cannot be categorized as *univ*. In addition, annotators were instructed to annotate named entities separately. What follows are descriptions of each of the annotation tags.

Sentence (sent): This tag refers to a sentence and can be used to mark *inter-*

[2]https://www.facebook.com/jumatrimonial

sentential code mixing. Annotators were instructed to identify a sentence with its base language (e.g. *en*, *bn*, *hi* and *mixd*) or with other types (e.g. *univ*, *undef*) as the first task of annotation. Only the attribute *mixd* is used to refer to a sentence which contains multiple languages in the same proportion. A sentence may contain any number of inclusions, fragments and word-level code mixing. A sentence can be attributed as *univ* if and only if it contains symbols, numbers, emoticons, chat acronyms and no other words (Hindi, English or Bengali). A sentence can be attributed as *undef* if it is not a sentence marked as *univ* and has words/tokens that cannot be categorized as Hindi, English or Bengali. Some examples of sentence-level annotations are the following:

1. **English-Sentence:**

   [sent lang="en"] *what a.....6 hrs long...but really nice tennis....* [/sent]

2. **Bengali-Sentence:**

   [sent lang="bn"] *shubho nabo borsho.. :)* [/sent]

3. **Hindi Sentence:**

   [sent lang="hi"] *karwa sachh ..... :(* [/sent]

4. **Mixed-Sentence:**

   [sent lang="mixd"] [frag lang="hi"] *oye hoye ..... angreji me kahte hai ke* [/frag] [frag lang="en"] *I love u.. !!!* [/frag] [/sent]

5. **Univ-Sentence:**

   [sent-lang="univ"] *hahahahahahah....!!!!!* [/sent]

6. **Undef-Sentence:**

   [sent lang="undef"] *Hablando de una triple amenaza.* [/sent]

Fragment (frag): This refers to a group of foreign words, grammatically related, in a sentence. The presence of this tag in a sentence conveys that *intra-sentential code mixing* has occurred within the sentence boundary. Identification of fragments (if present) in a sentence was the second task of annotation. A *sentence (sent)* with attribute *mixd* must contain multiple *fragments (frag)* with a specific

language attribute. In the fourth example above, the sentence contains a Hindi fragment *oye hoye ..... angreji me kahte hai ke* and an English fragment *I love u.. !!!*, hence it is considered a *mixd* sentence. A fragment can have any number of inclusions and word-level code mixing. In the first example below, *Jio* is a popular Bengali word appearing in the English fragment *Jio.. good joke*, hence tagged as a Bengali inclusion. One can argue that the word *Jio* could be a separate Bengali inclusion (i.e. can be tagged as a Bengali inclusion outside the English fragment). But looking at the syntactic pattern and the sense expressed by the comment, the annotator kept it as a single unit. In the second example below, an instance of word-level code mixing, *typer*, has been found in an English fragment (where the root English word *type* has the Bengali suffix *r*).

1. **Fragment with Inclusion:**

   [sent-lang="mixd"] [frag-lang="en"] [incl-lang="bn"] *Jio..* [/incl] *good joke* [/frag] [frag lang="bn"] *"amar Babin"* [/frag] [/sent]

2. **Fragment with Word-Level code mixing:**

   [sent-lang="mixd"] [frag-lang="en"] *" I will find u and marry you "* [/frag] [frag-lang="bn"] [wlcm-type="en-and-bn-suffix"] *typer* [/wlcm] *hoe glo to! :D* [/frag] [/sent]

Inclusion (incl): An inclusion is a foreign word or phrase in a sentence or in a fragment which is assimilated or used very frequently in native language. Identification of inclusions can be performed after annotating a sentence and fragment (if present in that sentence). An inclusion within a sentence or fragment also denotes *intra-sentential code mixing*. In the example below, *seriously* is an English inclusion which is assimilated in today's colloquial Bengali and Hindi. The only tag that an inclusion may contain is word-level code mixing.

1. **Sentence with Inclusion:**

   [sent-lang="bn"] *Na re* [incl-lang="en"] *seriously* [/incl] *ami khub kharap achi.* [/sent]

Word-Level code mixing (wlcm): This is the smallest unit of code mixing. This tag was introduced to capture *intra-word code mixing* and denotes cases where code mixing has occurred within a single word. Identifying word-level code mixing is the last task of annotation. Annotators were told to mention the type of word-level code mixing in the form of an attribute (Base Language + Second Language) format. Some examples are provided below. In the first example below, the root word *class* is English and *e* is an Bengali suffix that has been added. In the third example below, the opposite can be observed – the root word *Kando* is Bengali, and an English suffix *z* has been added. In the second example below, a named entity *suman* is present with a Bengali suffix *er*.

1. **Word-Level code mixing (EN-BN):**
   [wlcm-type="en-and-bn-suffix"] *classe* [/wlcm]
2. **Word-Level code mixing (NE-BN):**
   [wlcm-type="NE-and-bn-suffix"] *sumaner* [/wlcm]
3. **Word-Level code mixing (BN-EN):**
   [wlcm-type="bn-and-en-suffix"] *kandoz* [/wlcm]

3.1.2 Inter Annotator Agreement

We calculate word-level inter annotator agreement (Cohen's Kappa) on a subset of 100 comments (randomly selected) between two annotators. Two annotators are in agreement about a word if they both annotate the word with the same attribute (*en, bn, hi, univ, undef*), regardless of whether the word is inside an inclusion, fragment or sentence. Our observations that the word-level annotation process is not a very ambiguous task and that annotation instruction is also straightforward are confirmed in a high inter-annotator agreement (IAA) with a Kappa value of 0.884.

3.1.3 Data Characteristics

In our corpus, inter- and intra-sentential code mixing are more prominent than word-level code mixing, which is similar to the findings of Hidayat (2012) . Our corpus contains every type of code mixing in English, Hindi and Bengali viz. inter/intra-sentential and word-level as described in the previous section. Some examples of different types of code mixing in our corpus are presented below.

1. **Inter-Sentential:**

   [sent-lang="hi"] *Itna izzat diye aapne mujhe* !!! [/sent]

   [sent-lang="en"] *Tears of joy. :'(:'(* [/sent]

2. **Intra-Sentential:**

   [sent-lang="bn"] [incl-lang="en"] *by d way* [/incl] *ei* [frag-lang="en"] *my craving arms shall forever remain empty .. never hold u close ..* [/frag] *line ta baddo* [incl-lang="en"] *cheezy* [/incl] :P ;) [/sent]

3. **Word-Level:**

   [sent-lang="bn"] [incl-lang="en"] *1st yr* [/incl] *eo to ei* [wlcm-type="en+bnSuffix"] *tymer* [/wlcm] *modhye sobar jute jay ..* [/sent]

Annotators were instructed to tag an English word as English irrespective of any influence of word borrowing or foreign inclusion but an inspection of the annotations revealed that English words were sometimes annotated as Bengali or Hindi. There are two reasons why this is happening:

- **Same Words Across Languages** Some words are the same (e.g. *baba*, *maa*, *na*, *khali*) in Hindi and Bengali because both of the languages originated from a single language *Sanskrit* and share a good amount of common vocabulary. It also occurred in English-Hindi and English-Bengali as a result of *word borrowing*. Most of these are commonly used inclusions like *clg*, *dept*, *question*, *cigarette*, and *topic*. Sometimes the annotators were careful enough to tag such words as English and sometimes these words were tagged in the annotators'

native languages. During cross checking of the annotated data the same error patterns were observed for multiple annotators, i.e. tagging commonly used foreign words into native language. It demonstrates that these English words are highly assimilated in the conversational vocabulary of Bengali and Hindi.

- **Phonetic Similarity of Spellings** Due to phonetic typing some words share the same surface form across two and sometimes across three languages. As an example, *to* is a word in the three languages: it has occurred 1209 times as English, 715 times as Bengali and 55 times as Hindi in our data. The meaning of these words (e.g. *to*, *bolo*, *die*) is different in different languages. This phenomenon is perhaps exacerbated by the trend towards short and noisy spelling in SMC.

When building word-level annotated corpora for code-mixed NLP, such characteristics (e.g. word-borrowing, annotation errors) can result in poor performance for the classifiers due to label inconsistency. To reduce such errors, manual inspection has been done over the annotated corpora to select and extract correctly labelled sentences. No normalization (e.g. 'toh' for 'to' in Bengali) is done for phonetically similar words across languages. We assume that correct labels for these words can be learned during classification if contextual clues are considered. Further, we assign 'mixed' label to all word-level code-mixing instances in this subset. The curated subset contains 1,239 code-mixed posts and comments from the English-Bengali-Hindi corpus. This corpus contains word-level language annotations. Each word in the corpus is tagged with one of the following labels: (1) English (en), (2) Hindi (hi), (3) Bengali (bn), (4) Mixed (mixed), (5) Universal (univ), (6) Named Entity (ne) and (7) Acronym (acro). The label *Universal* is associated with symbols, punctuation, numbers, emoticons and universal expressions (e.g. *hahaha* and *lol*). This subset contains 25,383 tokens and 6,883 types.

The language label statistics of our data are shown in the Table 3.1. In terms of tokens Bengali is the dominant language (48.40%) of the corpus, followed by English

Label	Count	Percentage
bn	12285.0	48.40
en	5917.0	23.31
hi	1496.0	5.89
mixed	67.0	0.26
ne	657.0	2.59
acro	213.0	0.84
univ	4748.0	18.71

Table 3.1: English-Bengali-Hindi Code-mixed Data Set (Curated): Language label distribution of tokens.

(23.31%) and the least frequent language is Hindi (5.89%) in terms of token. The proportion of universal tokens is 18.71%. The instances of word-level code-mixing is 0.26%, which is the lowest percentage among all labels. The proportion of named entities and acronyms are also very low, 2.59% and 0.84% respectively.

We also measure the label of type-level ambiguity (discussed in Section 3.1.3) in our corpus. We find 95.64% of all types are unambiguous and 4.32% are ambiguous in nature. Among the ambiguous types, the most frequent ambiguity is the Bengali-Hindi ambiguity, which is 33.66% of total ambiguous types. Some examples of these types are 'toh', 'baap', 'dar', 'jiyo' and 'mamu'. These ambiguous types are either from the shared vocabulary of Bengali and Hindi or a result of Romanisation. The proportion of Bengali-English ambiguity is relatively low (19% of total ambiguous types), this ambiguity is caused by phonetic similarity of spellings due to Romanisation. This fact is also valid for English Hindi ambiguous types, which contributes, only 4.33% to the total ambiguous types. The most ambiguous type found in the data is 'a', which is present as a part of a name entity, as an acronym, as an English token, as a Bengali token and as a part of word-level code-mixing.

Given the token-level tags, we also analyze the amount of code-mixed and monolingual sentences in our corpus. We split a post or a comment based on the four types of sentence marker, '.', repeated occurrences of '.', '!' and '?'. The sentence-level language distribution is shown in Table 3.2. The splitting results in 3,048 sentences of which 53.21% are monolingual and the rest are code-mixed sentences.

Label	Count	Percentage
bn	660.0	21.65
en	428.0	14.04
hi	139.0	4.56
univ	395.0	12.96
en bn	1264.0	41.47
en hi	111.0	3.64
hi bn	24.0	0.79
en bn hi	27.0	0.89

Table 3.2: English-Bengali-Hindi Code-mixed Data Set (Curated): Language label distribution of sentences.

We find that 45.90% of the total sentences are bilingual code-mixing. Only 0.89% of the total sentences are trilingual code-mixing. The most frequent type of bilingual code-mixing is Bengali-English code-mixing (41.47% of the total sentences), followed by English-Hindi (3.64% of the total sentences). The least frequent type of bilingual code-mixing is Bengali-Hindi which is only 0.79% of the total sentences. We also find that 12.96% of the total sentences are universal sentences. These sentences consist of language independent tokens, e.g. symbols, numbers, names. We find that the average length of monolingual sentences varies between 6.37 to 7.32 except the universal sentences. The average length of universal sentences is 1.51. The average length of bilingual sentences varies from 8.75 to 11.68 and the trilingual sentences have an average length of 17.85.

Manual annotation for POS for this subset is done using the universal POS tag set [3] of Petrov et al. (2011). These annotations were performed by an annotator who is proficient in all three languages of the corpus. As we had no second annotator proficient in all three languages, we cannot present the inter-annotator agreement for the annotations. The POS label distributions for our data set are shown in Table 3.3. In terms of frequency, the top three POS categories are noun, verb and punctuation, (30% tokens are noun, 16% are verb and 15% are punctuation). The least frequent labels are numbers, 'X' (mostly emoticons) and conjunctions.

[3]An alternative tag set is the one introduced for code-mixed data by Jamatia and Das (2014) However, we prefer the universal tag set because of its simplicity, its applicability to many languages and its popularity within the NLP community.

Label	Count
ADJ	1836
ADP	1255
ADV	701
CONJ	686
DET	834
NOUN	7751
NUM	148
PRON	2306
PRT	1234
PUNCT	3959
VERB	4086
X	587

Table 3.3: English-Bengali-Hindi Code-mixed Data Set (Curated): POS label distribution of tokens.

P\L	en	bn	hi	mixed	ne	acro	univ	Total
ADJ	566	1184	85	0	0	0	1	1836
ADP	340	750	163	0	0	0	2	1255
ADV	323	342	28	1	0	0	7	701
CONJ	164	491	20	0	0	0	11	686
DET	318	495	20	1	0	0	0	834
NOUN	2740	3519	478	63	657	213	81	7751
NUM	33	5	0	0	0	0	110	148
PRON	463	1640	199	0	0	0	4	2306
PRT	87	1061	83	0	0	0	3	1234
PUNCT	0	0	0	0	0	0	3959	3959
VERB	871	2796	414	2	0	0	3	4086
X	12	2	6	0	0	0	567	587

Table 3.4: English-Bengali-Hindi Code-mixed Data Set (Curated): POS distribution across language labels

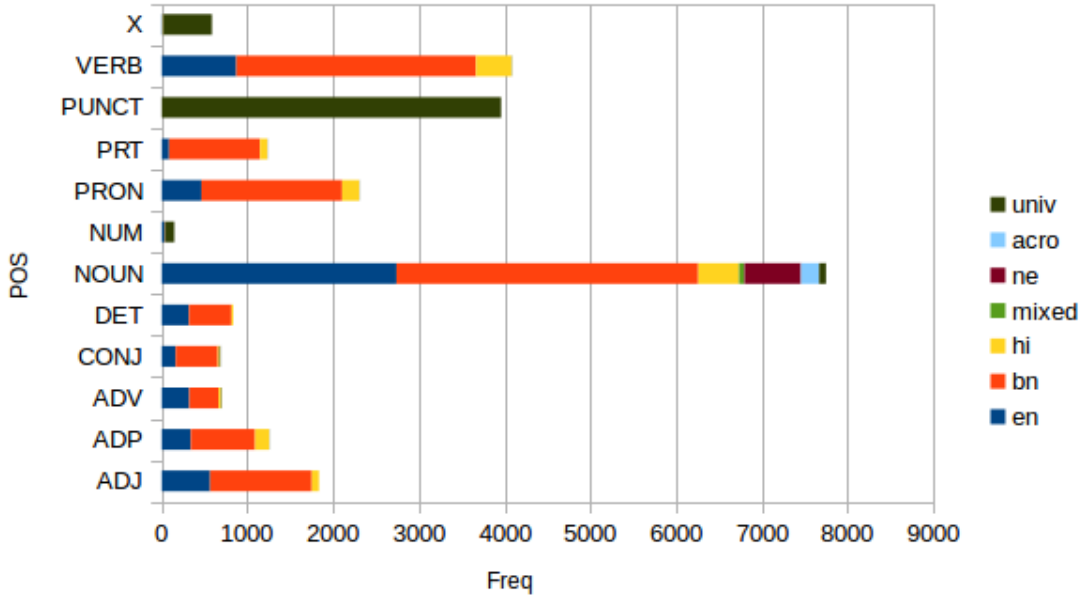

Figure 3.1: English-Bengali-Hindi Code-mixed Data Set (Curated): POS distribution across language labels, column visualization.

The distribution of POS tags across language labels is depicted in Table 3.4 and in Figure 3.1. In terms of language, Bengali is the most dominant across all POS labels except 'NUM'. Only 5 occurrences of Bengali numbers can be seen in the corpus. It can be observed that some universal tokens (0.01%) are tagged as nouns. These tokens are mainly universal sounds (e.g. 'vroooomm vroooomm') and creative spelling variations (e.g. 'dadaaaaaaa', 'a****a', 'kkrrr'). Most of the mixed tokens (97%) are nouns, these are mainly nouns with suffixes from other languages, for example: '*school*_er, *time*_r' (Bengali suffixes - 'er', 'r' are augmented with English nouns).

3.2 Unlabelled Monolingual Data

The following corpora (unlabelled) were collected to create dictionaries and to train word embeddings in our experiments.

- **Bengali Song Lyrics Corpus:** This is a collection of romanised Bengali song lyrics collected from `www.lyricsbangla.com/`. It contains 310,565 tokens and

30,933 types.

- **Hindi Song Lyrics Corpus:** This is a collection of romanised Bengali song lyrics collected from `www.smriti.com/hindi-songs`. It contains 501,212 tokens and 19,547 types.

- **Unlabelled Facebook Corpus:** This is a collection of 77K unlabelled monolingual or code-mixed posts and comments in English, Bengali and Hindi. It contains 1,553,290 tokens and 86,662 types.

3.3 Conclusion

In this chapter, we have describe the data collection and annotation procedure. We also have described our data characteristics. We found intra-sentential code-mixing is prominent in our social media corpus. We also found that word-ambiguity due to Romanisation is also present in our corpus. In the next chapters, we use this corpus to build LID and POS tagging systems.

Chapter 4

Language Identification

Language identification (LID) is the task of identifying the language(s) of a given text or utterance. The necessity of accurate LID of a given document is multifold. For example, delivering web content in user's native language is an important factor to attract visitors (Kralisch and Mandl, 2006). Several NLP and Information Retrieval (IR) methods start with the assumption that the language of an input is known or fixed. In that context, a major target of LID is to make sure that only relevant input (i.e in a particular language) is presented to such NLP and IR systems. For Machine Translation (MT), LID is also a prerequisite factor that describes the source language of a document from which it is to be translated into a target language.

LID has been investigated both linguistically (Johnson, 1993; Grefenstette, 1995) and statistically (Dunning, 1994; Damashek, 1995) and has been categorized as a text categorization task (Cavnar and Trenkle, 1994; Elworthy, 1999). It has been investigated at various granularities: from document-level (Dunning, 1994; Damashek, 1995; Prager, 1999; McNamee, 2005), sub document-level (Yamaguchi and Tanaka-Ishii, 2012; King and Abney, 2013), sentence-level (Zaidan and Callison-Burch, 2011) and word-level (Nguyen and Doğruöz, 2013; Solorio and Liu, 2008a). Document level LID is the task of assigning a unique language to a document. Researchers have achieved good accuracy in document-level LID with formal content (Dunning, 1994;

Damashek, 1995; McNamee, 2005) (e.g. news articles and Wikipedia). It is relatively easier than other types of LID (e.g. sentence-level, word-level) when large monolingual documents and a small number of languages are considered (McNamee, 2005). Sub-document level LID is performed for multilingual documents, is more difficult than the document-level LID and has been investigated for forum (King and Abney, 2013) and blog posts (Yamaguchi and Tanaka-Ishii, 2012). Sentence-level LID is more fine grained than the previous two and has been studied for different domains, for example: short text (Baldwin and Lui, 2010a), queries (Gottron and Lipka, 2010), tweets (Lui and Baldwin, 2014). The last and probably the deepest level LID (Das and Gambäck, 2014) is performed on the word-level with code-mixed content, and in recent years has been investigated by a large number of NLP researchers (Solorio et al., 2014b; Sequiera et al., 2015b).

The majority of research on LID is concentrated on widely spoken and well resourced languages (e.g. English) (Hughes et al., 2006). On the other hand, there is a growing need for research on the low-resourced languages, especially, in the domain of the social media content. The nature of social media content is noisy and exhibits several socio-linguistic phenomena, e.g. short length, informality, Romanisation, spelling variation, multilingualism. Each of these factors presents challenges to the LID systems that are trained with formal content. So researchers (Hughes et al., 2006; Das and Gambäck, 2015) have focused more on obtaining accurate word-level LID for social media content. Further, Hong et al. (2011) show that half of the tweets in Twitter are not English. Hughes et al. (2006) also argue that the approaches which perform well for major languages may not perform well for minority languages. In recent research, supporting low resourced languages (Lui and Baldwin, 2014; Hong et al., 2011), handling code-mixing (Solorio and Liu, 2008a), Romanisation of non-Roman script languages (Barman et al., 2014a; Das and Gambäck, 2014) have become popular topics of LID research.

4.1 Chapter Organization

In this chapter, we present our experiments on word-level LID with a trilingual (English-Bengali-Hindi) code-mixed social media content. We discuss related studies of LID with code-mixing in Section 4.2, present our research questions in Section 4.3, and describe our experiments in Section 4.4. Section 4.5 describes the analysis of the performance of different systems and finally we summarize this chapter in Section 4.6 by answering the research questions and by highlighting other findings.

4.2 Background

Before discussing the related research of word-level LID with code-mixed data, first we discuss some important work on LID. As discussed above, LID has been investigated mainly in four ways: document classification, short text classification, word and sub word-level sequence classification.

One of the most important works on automatic LID was carried out by Cavnar and Trenkle (1994), where LID was treated as document classification. They use n-gram models to create document and language profiles and classified each document by measuring rank-order similarity across language profiles. The use of n-grams has been popularized by Cavnar and Trenkle (1994) and many other researchers have used and updated it in many ways (Dunning, 1994; Sibun and Reynar, 1996). Dunning (1994) used Markov models and a Bayesian classifier that discriminates words and byte n-grams across languages in the training set and finds similar patterns in the test set. They performed their study for news articles. Grefenstette (1995) performs a comparative study of two methods: (1) character trigram-based and (2) short word-based for European languages and found both of these methods perform well for long texts (word count $\geq$ 50), but that a trigram-based method is more robust for shorter texts. Prager (1999) used a *tf-idf* based vector space model which uses the cosine similarity between the training and test language model. In their study words and n-grams were used to classify monolingual documents in 20 dif-

ferent languages and found that for short texts their method did not perform well. In another comparative study (Padró and Padró, 2004), Markov models, trigram frequency vectors, and n-gram text categorization were used to perform LID in six different languages, among these models, the Markov models outperformed the other two methods. Similar studies have been made by other researchers (Grothe et al., 2008), where methods like short word-based model, frequent word-based model and the character n-gram-based model are compared, after tuning different parameters, the performance of all three methods become similar. Supervised machine learning based LID (e.g. k-Nearest Neighbours, SVM, Naive Bayes) with n-grams as features was presented in the study of Baldwin and Lui (2010b). They performed LID for web documents and found that a simple 1-NN model and an SVM with a linear kernel (that uses byte n-grams as features) is best for the their task. They concluded that LID becomes challenging (1) if the number of target languages is large, (2) the size of the training set is small and (3) document length is shorter. Later, Lui and Baldwin (2011) developed a Naive Bayes-based method for cross-domain language identification, which mines the discriminated features across 97 languages and across 5 data sets and outperforms the method of Cavnar and Trenkle (1994). Finally, Lui and Baldwin (2012) developed an off-the-shelf LID tool (langid.py) that uses Naive Bayes algorithm with n-grams as features. It is a well known tool for LID among the NLP community for its ease of use and robustness. Among other document-level LID studies, hidden Markov models (HMM) (Xafopoulos et al., 2004), URL-based LID by keyword extraction (Baykan et al., 2008) and corpus creation for low-resourced languages (Scannell, 2007) can be seen.

Sentence-level LID, on the other hand, is much harder than document level LID because of the short length and possible inclusion of noise (i.e. tweets or social media posts and comments). Tromp and Pechenizkiy (2011) used a graph-based n-gram approach to classify tweets in six languages. Bergsma et al. (2012) investigated LID techniques with tweets of 9 languages. In their work they only considered monolingual tweets. Vogel and Tresner-Kirsch (2012) included linguistic features

on top of the method of Tromp and Pechenizkiy (2011) and achieved near perfect accuracy. The use of Bayesian classifiers was also investigated for tweets (Laboreiro et al., 2013) and short texts (e.g.film subtitles) (Winkelmolen and Mascardi, 2011). Goldszmidt et al. (2013) used character frequencies to build a boot-strapping-based statistical LID to classify tweets of five different languages. They used Wikipedia for training and found that Wikipedia is not suitable for such data because the words of social media are quite different from their training data. Carter et al. (2013) investigated the use of contextual features, hash tags, author, mentions in five major European languages: Dutch, English, French, German, and Spanish. They found the combination of all features performed best. LID for short text in Indian languages (e.g. Hindi, Bengali, Marathi, Punjabi, Oriya, Telugu, Tamil, Malayalam and Kannada) was investigated by Murthy and Kumar (2006) using linear regression. However, their work was based on language specific scripts. They did not consider Romanisation in their work.

The granularity of multilingualism is not limited to the sentence level. Code-mixing (e.g. intra-sentential and word-level) is a prominent example of this fact. When multiple languages are mixed in a sentence it is necessary to use a word-level strategy. Recently, a number of research articles have been published for code-mixed LID. Solorio and Liu (2008a) tried to predict the points inside a set of spoken Spanish-English sentences where the speakers switch between the two languages. To do this, they used a dictionary-based and character n-gram-based LID approach. They also showed that by exploiting language information, monolingual POS taggers can be used to achieve a good accuracy to detect code-switch points. Yamaguchi and Tanaka-Ishii (2012) performed language identification through dynamic programming using artificial multilingual data, created by randomly sampling text segments from monolingual documents. King and Abney (2013) used weakly semi-supervised methods to perform word-level language identification. A dataset of 30 languages has been used in their work. They explored several language identification approaches, including a Naive Bayes classifier for individual word-level classifica-

tion and sequence labelling with CRF trained with Generalized Expectation criteria (Mann and McCallum, 2008, 2010), which achieved the highest scores. Another work on this topic is Nguyen and Doğruöz (2013). They report on language identification experiments performed on Turkish and Dutch forum data. Experiments were been carried out using language models, dictionaries, logistic regression classification and CRF. They found that language models are more robust than dictionaries and that contextual information is helpful for the task. Lignos and Marcus (2013) performed LID for bi-lingual Spanish-English tweets using the ratio of the probability of a word in a language. However, the risk of word ambiguity due to lexical similarity is not included in their work. Voss et al. (2014) present their work on Romanized Moroccan Arabic, English and French code-mixed tweets using a Maximum Entropy classifier. Chittaranjan et al. (2014) performed language identification with CRF with hand-crafted orthographic and contextual features. Rijhwani et al. (2017) on the other hand, attempted to obtain a generalized model to identify code-mixing for a number of language pairs by combining multiple hidden Markov models.

Word-level LID for code-mixed data gained popularity through recent share tasks, for example, the First Workshop of Computational approaches to code-switching, (Solorio et al., 2014a), the FIRE 2015 Shared Task on Mixed Script Information Retrieval (Sequiera et al., 2015b) and the Second Workshop of Computational Approaches to Code-switching (Diab et al., 2016). Word-level LID for multiple language pairs (e.g. Nepali-English, Spanish-English, Hindi-English) was investigated in these workshops by many researchers using a number of machine learning algorithms, including, CRF (Chittaranjan et al., 2014; Jain and Bhat, 2014; Xia, 2016), SVM (Barman et al., 2014b; Bar and Dershowitz, 2014), logistic regression (Shirvani et al., 2016) and LSTM (Jaech et al., 2016). Other than neural methods most of the researchers focused on extracting handcrafted orthographic and contextual information as features in their algorithm. These features are mainly: character n-grams, use of language model and dictionaries, capitalisation information, presence of number and punctuation in a word etc.

Neural methods for code-mixed word-level LID are gaining popularity. One of the benefits of neural approaches is that it allows us to avoid handcrafted feature engineering. Neural word representations or embeddings are often used in the place of hand crafted features like character n-grams. Some researchers have used word representations obtained from shallow neural models as features to standard sequence labelling methods (e.g. CRF). For example, Xia (2016) used word embeddings with handcrafted features for CRF and found that using both types of features results in better word-level accuracy. Chang and Lin (2014), on the other hand, used Jordan-type RNN and Elman-type RNN with character n-grams and word embeddings as features and concluded that adding character n-grams as features improves the performance of their RNN-based models. The use of character n-grams as features along with word embeddings is mainly due to the morphological structure of a word. Some embedding (e.g. word2vec) techniques use the word as the atomic unit to learn representations. Sub-word-level information is not present when words are considered as atoms. To retain the word and character level information, Samih et al. (2016), used an architecture of two parallel LSTMs, one used word embeddings and the other used character embeddings of a word as input, and generated two different word representations, which are concatenated, and finally a softmax/CRF was applied to the concatenated representation to predict the language of a token. Another architecture was proposed by Jaech et al. (2016), who used a CNN-LSTM architecture where CNNs were used to obtain word vectors from character representation, and then fed as a sequence of vectors to a bidirectional LSTM, followed by a softmax to generate the output. LSTM with character and phonetic encoding was used to classify Bengali-English code-mixed data (Mandal et al., 2018).

4.3 Research Questions

In this chapter we focus on word-level LID with code-mixed and romanised social media content collected from Facebook. Three languages are involved, English and

two relatively low resourced Indian languages, Hindi and Bengali. For word level language identification we try to find the answers of the following:

- How well does a simple method like a dictionary-based language identification system behaves for such data?
- How well does the non-neural machine learning method (e.g. Support Vector Machines and Conditional Random Fields) perform with useful features (e.g. character-n-grams) in language identification experiments with such data ? Do these methods perform better than dictionary-based system?
- Neural word embeddings have been successfully used in many NLP tasks in recent years, e.g., language modelling, POS tagging. Can the recurrent neural sequence labelling methods (e.g. Long Short Term Memory) with these embeddings perform better than Support Vector Machines and Conditional Random Fields based systems in language identification?
- Social media code-mixed data with ad-hoc Romanisation exhibits word ambiguity across languages. The contextual information is necessary to resolve this ambiguity. How well can sequential learning methods (e.g. CRF, LSTM) learn from contextual clues to disambiguate such words ?
- When exploring neural methods, the choice of neural embedding is an important factor. There are some embedding techniques that consider individual words as units (e.g. `word2vec`). Also some other embeddings (e.g. `fasttext`) are available which not only consider the words but also considers the compositions features (e.g. character-n-grams) of a word to learn the word representation. In this context we propose to investigate the following question: which one is the better choice for social media's code-mixed unnormalised data `word2vec` or `fasttext` ?

4.4 Experiments

4.4.1 Resource

We use the following corpora in our experiments as dictionaries and as unlabelled training data to obtain neural embeddings for word and characters.

- **text8:** This is an English corpus, a collection of Wikipedia articles and contains 17M tokens and 253,855 types. The corpus is freely available with the word2vec toolkit.

- **Bengali Song Lyrics Corpus:** This is a collection of romanised Bengali song lyrics collected from `www.lyricsbangla.com/`. It contains 310,565 tokens and 30,933 types.

- **Hindi Song Lyrics Corpus:** This is a collection of romanised Bengali song lyrics collected from `www.smriti.com/hindi-songs`. It contains 501,212 tokens and 19,547 types.

- **LexNormList:** Spelling variation is a well known phenomenon in social media content. We use a lexical normalization dictionary created by Han et al. (2012) to handle the different English spelling variations in our data.

- **Unlabelled Facebook Corpus:** This is a collection of 77K unlabelled monolingual or code-mixed posts and comments in English, Bengali and Hindi. It contains 1,553,290 tokens and 86,662 types.

4.4.2 Dictionaries

We start with a dictionary-based system as our baseline. We process English, Bengali and Hindi dictionaries from lower-cased and most-frequent words of text8, Bengali lyrics and Hindi lyrics corpus. We also include LexNormList and relevant 4/5 of the training data as dictionaries in our experiment. The architecture of the

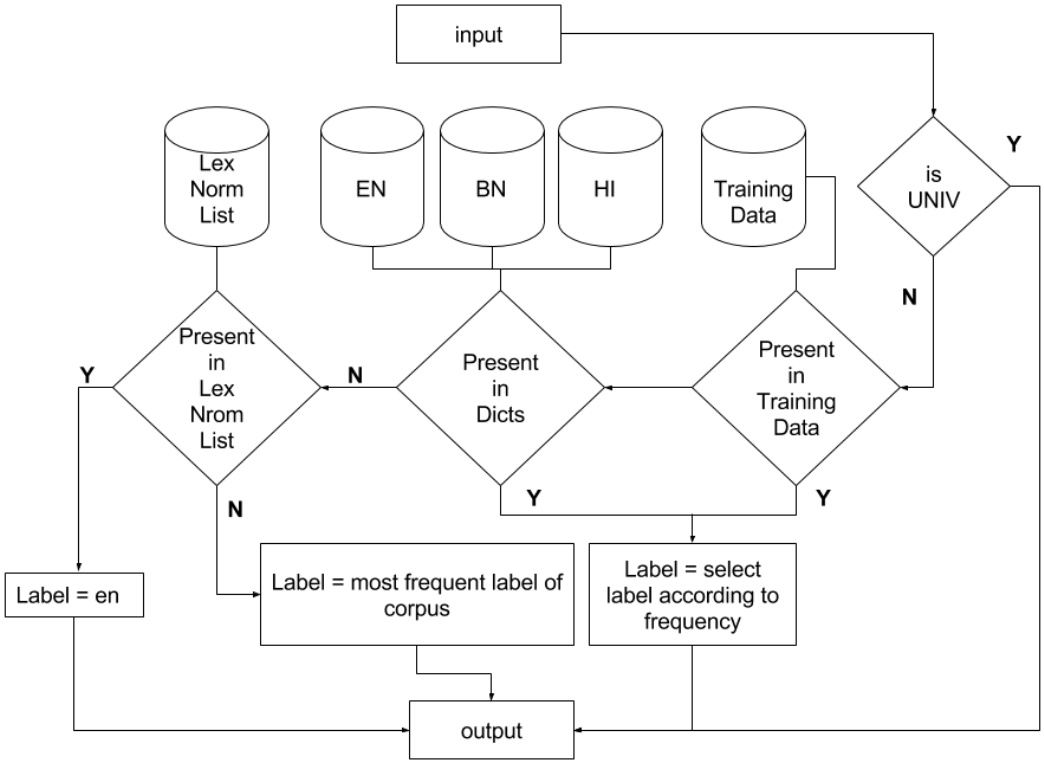

Figure 4.1: Architecture of the dictionary-based system, where EN = text8 corpus, BN = Bengali lyrics corpus, HI = Hindi lyrics corpus, Training Data = relevant 4/5 of training data and LexNorm List = lexical normalisation list of Baldwin

dictionary-based system is shown in Figure 4.1. To predict the label for each token in the relevant test split, we consider the following steps:

- A simple rule-based method is applied to predict universal expressions as the first step. A token is considered as univ if any of the following conditions satisfies:

    1. All characters of the token are symbols or numbers.
    2. The token contains certain repetitions identified by regular expressions (e.g. hahaha).
    3. The token is a hash-tag or URL or mention-tags (e.g. @Sumit).
    4. Tokens (e.g. lol) identified by a word list compiled from the relevant 4/5th of the training data.

- If a token is not 'UNIV', then we consult the relevant 4/5 of the training data. If the test token is present in the training data we predict the most frequent label according to the training split.

- If the test token is absent from the training data we consult the English, Bengali and Hindi dictionaries. Prediction is made based on (i) the presence and (ii) the normalised frequency of the token in three dictionaries.

- If the token is absent from the relevant training split and in the dictionaries, we check the presence of the token in LexNormList. We predict the 'EN' if the token is present in the LexNormList otherwise we choose the most frequent label in the relevant training split (e.g. 'BN').

Dictionaries	**Accuracy(%)**
All-E	52.75
All-B	60.00
All-H	20.83
E	70.18
EB	84.99
EBH	80.70
EBHT	89.35
EBHTL	**90.22**

Table 4.1: Average cross-validation accuracy of dictionary-based detection, where E = text8 corpus, B = Bengali song lyrics, H = Hindi Song Lyrics, T = relevant 4/5 of the training data, and L = LexNormList. All-E, All-B and All-H are those systems where all tokens are predicted as English, Bengali and Hindi respectively. Reported results are average of 5-fold cross-validation accuracy.

We try different combinations and frequency thresholds of the above dictionaries. Table 4.1 shows the average accuracy of 5-fold cross-validation of some dictionary combinations. We find that using all dictionaries (e.g. **EBHTL** in Table 4.1) gives the best cross-validation accuracy (90.22%).

4.4.3 Support Vector Machine (SVM)

We perform experiments with a SVM classifier (linear kernel) for different combinations of the following features:

1. **character n-gram Features (G):** We use the most common n-grams which is used by many language identification researchers. Following the work of King and Abney (2013), we select character n-grams (n=1 to 5) and the word as the features in our experiments.

2. **Presence in Dictionaries (D):** We use presence in a dictionary as a features for all available dictionaries in previous experiments.

3. **Length of the word (L):** We create multiple features for length using a decision tree (J48). We use length as the only feature to train a decision tree for each fold and use the nodes obtained from the tree to create Boolean features.

4. **Capitalisation (C):** We use 3 Boolean features to encode capitalization information: whether any letter in the word is capitalized, whether all letters in the word are capitalized and whether the first letter is capitalized.

5. **Context (P_iN_i):** Contextual clues can play a very important role in word-level language identification. To provide contextual information we consider the previous and next word of token as features.

In our experiments we identified the word boundaries. We use '$' to indicate start of a word and '£' to indicate end of a word. Table 4.2 is an example of our feature set.

Feature Name	**Features Examples (with $value = 1$)**
G (char-n-gram)	$a, m, a, r£, $am, ma, ar£, $ama, mar£, $amar£
D (Dictionary)	<dict-train-bn>
L (Length ranges)	<4-6>
C (Capitalization)	-
P_1N_1 (Context)	<p1-je>,<n1-prothom>

Table 4.2: Features generated for a word '*amar*' which is a part of a text fragment: '*je amar prothom*'.

According to Hsu et al. (2010) the SVM linear kernel with parameter C optimization is good enough when dealing with a large number of features. Though

an RBF kernel can be more effective than a linear one, it is possible only after proper optimization of C and γ parameters, which is computationally expensive for a large feature set. Parameter optimization (C range 2^{-15} to 2^{10}) for SVM are performed for each feature set and best cross-validation accuracy is found for the GDLCP_1N_1-based run (93.06%) at C = 0.0312 (see Table 4.3).

Features	**Accuracy**	**C**
G	91.75	0.0156
GD	92.45	0.0009
GDL	92.61	0.0019
GDLC	92.59	0.0019
GDLCP_1N_1	**93.06**	**0.0312**

Table 4.3: Average cross-validation accuracy for SVM word-level classification, G = char-n-gram, L = binary length features, D = presence in dictionaries and C = capitalization features and P-i = previous i word(s) , N-i = next i word(s).

4.4.4 Conditional random field (CRF)

We employ a first order linear chain CRF with stochastic gradient decent and $L1$ penalty parameters. We use similar features as those in SVM LID experiments. For character n-grams we use prefix and suffix n-grams of length 5. We use the dictionary-based predictions of the baseline system to generate a single dictionary feature for each token and only the raw length value of a token instead of binarised length features. The capitalisation features are kept same as SVM-based experiments. We optimize each feature combination by varying the learning rate from 2^{-5} to 2^0. The best performance of 92.29% average cross-validation accuracy is obtained with GDLCP_1N_1 feature set and using learning rate = 0.0625.

4.4.5 Long Short Term Memory (LSTM)

LSTM RNNs have shown good performance in sequence labelling tasks (e.g. LID, POS tagging, NER). In code-mixed LID, the use of LSTMs has been investigated in recent years (Samih et al., 2016; Mandal et al., 2018). We use the bi-directional

Features	Accuracy	Learning Rate
G	91.74	0.125
GD	92.24	0.031
GDL	91.56	0.001
GDLC	92.25	0.125
GDLCP_1N_1	**92.29**	**0.0625**

Table 4.4: Average cross-validation accuracy for CRF word-level classification, G = char-n-gram, L = binary length features, D = presence in dictionaries and C = capitalization features and P-i = previous i word(s) , N-i = next i word(s).

LSTM to perform word-level language identification. Two different architectures are employed to perform word-level LID. The first one is a simple bi-directional LSTM which only considers word vectors as input, and the second one is a combination of two LSTMs, one for the character level and another for the word level.

4.4.5.1 Embeddings and Common Settings

As features to the LSTM network, we compare the use of `word2vec` (Mikolov et al., 2013), which treats words as atoms, the `CWE` (Chen et al., 2015) which jointly learns both word embeddings and character embeddings and `fasttext` (Joulin et al., 2016; Bojanowski et al., 2016) which considers bags of word n-grams and represents each word by a sum of its character n-grams. `CWE` and `fasttext` embeddings can be expected to be useful for morphological rich languages and also effective with high spelling variation such as in social media content, especially with ad-hoc Romanisation of non-Latin scripts. The detailed information of these embedding can be found in Chapter 3. All embeddings are trained with negative sampling using a corpus that incorporates monolingual, and code-mixed contents in English, Hindi and Bengali (see Section 4.4.1). The hyper parameters used to train the embeddings are described in Table 4.5. It is worth mentioning that, `word2vec` and `CWE` embedding do not generate word vectors for out-of-vocabulary (OOV) words, to cope with this we introduce a special token 'unk' to replace the low frequency words (frequency <= 5) in our corpus. However, `fasttext` can generate word vectors for OOV words as it considers character n-grams while building the vector representation of a word.

Hyper Paramters	Value
Learning rate	0.05
Minimum word count	2
Context window	3
Vector size	100
Iteration	50

Table 4.5: Hyper parameters for embedding training

During training, `fasttext` considers a range of character n-grams as an extra hyper parameter which has been set to the range of 1 to 5. The rest of the hyper parameters of `fasttext` are the same as `word2vec` and `CWE`. We use these embedding models to obtain word and character representations to initialize the embedding layer of our LSTM-based models.

Neural networks tend, to over-fit when operating on small sets of training data. To mitigate this, dropout (Hinton et al., 2012) is used in different parts of our architectures. Dropout with a certain threshold indicates that that percentage of the internal neurons will be switched off during training. Dropout is applied after an embedding layer, after bi-LSTM and before the output layer of the network. As prepossessing steps, maximum word length is set to 30 characters, maximum sentence length (i.e. the temporal dimension of LSTM) is set to 78 words. All words having less than 30 characters are padded with a special token to achieve same length for all words, similar thing is done with sentences to achieve the same sentence length over the data.

4.4.5.2 Word-level Bidirectional LSTM

This model takes an input sequence of words, $(w = w_1, ..., w_T)$. For a given word w_t, a vector representation $x_t \in R^{d^w}$ is generated from a word embedding look-up table which is the first layer of this network. For the sequence of such vectors $x_{1:T}$ the LSTM network maps the input sequence $x_{1:T}$ to an output sequence $y_{1:T}$ by performing calculation over the network unit activations using the following equations

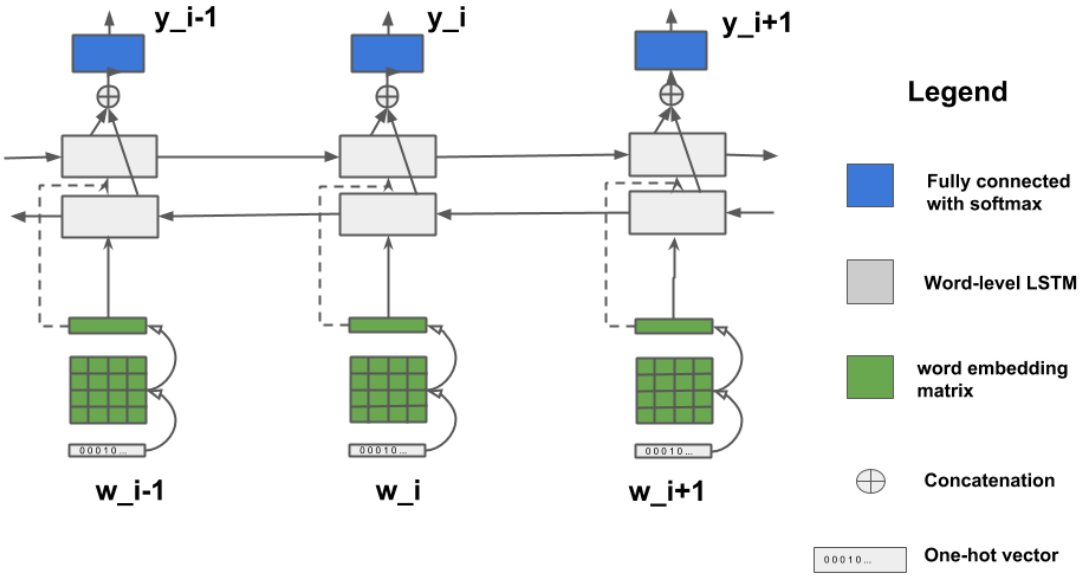

Figure 4.2: Model Architecture: Word-level LSTM Model for LID, unrolled across three time steps, where $w_{i-1:i+1}$ are the words and $y_{i-1:i+1}$ are language labels.

in an iterative way from $t = 1$ to T:

$$i_t = \sigma(W_i^x x_t + W_i^h h_{t-1} + W_i^c c_{t-1} + b_i)$$

$$f_t = \sigma(W_f^x x_t + W_f^h h_{t-1} + W_f^c c_{t-1} + b_f)$$

$$o_t = \sigma(W_o^x x_t + W_o^h h_{t-1} + W_o^c c_{t-1} + b_o)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tanh(W_c^x x_t + W_c^h h_{t-1} + b_c)$$

$$h_t = o_t \odot \tanh(c_t)$$

At time step t, $x_t \in R^{d^w}$ is the input vector to the LSTM, i_t specifies the input gate's activation vector, f_t is the forget gate's activation vector, o_t is the output gate's activation vector, c_t denotes the cell state vector, and h_t denotes the hidden vector of LSTM unit. Ws specifies weight matrices, bs specifies bias vectors, σ is the logistic sigmoid activation function, tanh is the hyperbolic tangent activation function and $\odot$ is the entry-wise product of vectors. For two LSTMs (one in forward

Hyper Paramters	Value
Learning rate	0.05
LSTM hidden unit	256
Dropout rate	0.5
Batch size	64
Epochs	10
word vector dimension (d^w)	100

Table 4.6: Hyper for word-Level LSTM

direction and another in backward direction), we have two output vectors $h_t^{forward}$ and $h_t^{backward}$, the final output vector $\tilde{h_t}$ is a concatenation of these two vectors,

$$\tilde{h_t} = h_t^{forward} \oplus h_t^{backward}$$

Here, $\oplus$ is the concatenation operation. Finally, the concatenated vector passes through a fully connected layer with a $softmax$ activation function to generate y_t:

$$y_t = softmax(W_y \tilde{h_t} + b_y) \tag{4.1}$$

The architecture of this model is illustrated in Figure 4.2. We use categorical cross entropy as the loss function to this network:

$$L_{cross\_entropy}(y, \hat{y}) = -\sum_i y_i log(\hat{y}_i) \tag{4.2}$$

The hyper parameters of our model are described in Table 4.6.

This network is used to monitor the performance of different embeddings: `word2vec`, `CWE` and `fasttext`. Table 5.2 shows that `fasttext` outperforms `word2vec` and `cwe` with both CBOW and skip-gram models. The best performance (96.38%) is achieved by the `fastext` with skip-gram embedding in terms of average 5-fold cross-validation.

Type	Model	Accuracy	Type	Model	Accuracy
word2vec	*CBOW*	*95.20*	*word2vec*	*skip-gram*	*95.16*
CWE	CBOW	95.31	CWE	skip-gram	95.14
	CBOW+P	95.43		skip-gram+P	95.76
	CBOW+L	95.59		skip-gram+L	95.81
fasttext	*CBOW*	**96.27**	*fasttext*	*skip-gram*	**96.38**

Table 4.7: Accuracy of LSTM with `word2vec`, `CWE` and `fasttext` embeddings. Reported results are average five fold cross-validation accuracy.

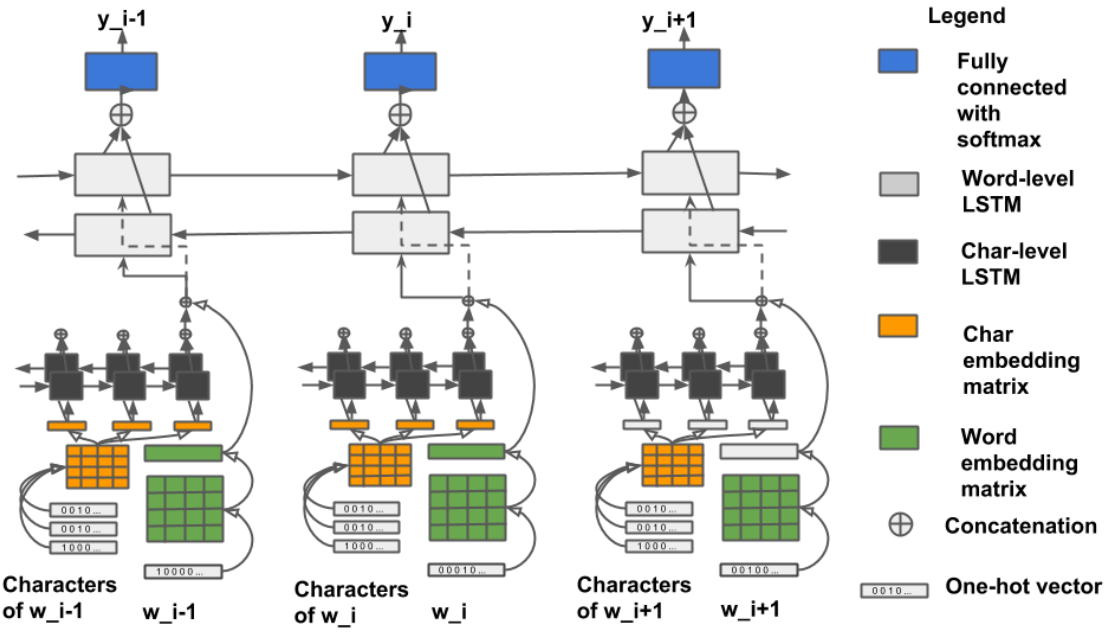

Figure 4.3: Model Architecture: Word + Character-level LSTM model, unrolled across three time steps, where $w_{i-1:i+1}$ are the words and $y_{i-1:i+1}$ are language labels.

4.4.5.3 Word + Character-level Bidirectional LSTM

This model considers words and the characters of the words as input to the network. For a given word w_t, at time step t, let the character sequence of the word be $c^{w_t}_{1:M} = \{c^{w_t}_1, c^{w_t}_2, ..., c^{w_t}_m, ..., c^{w_t}_M\}$. A character's one-hot representation $c^{w_t}_m$ passes through a character embedding look-up table and generates d^c dimensional vector ch_m. These embeddings are obtained using the `fasttext` skip-gram model. Thus for w_t an embedding matrix of size $M \times d^c$ is generated which serves as the input to the character-level bi-directional LSTM (char-LSTM). The char-LSTM encodes its input to a fixed size representation h^c_M, which is the concatenation of the final output vectors (after M steps) of forward and backward char-LSTMs. Further, a vector representation $x_t \in R^{d^w}$ for w_t is generated from a word embedding look-up table. Finally, the two representations (i.e. x_t and h^c_M) of w_t are concatenated to obtain a single representation v_t. For, a sequence of words, $w_1, w_2, ...w_T$ a vector representation $v_1, v_2, ...v_T$ is obtained and is passed to another bidirectional LSTM (word-LSTM) which operates at word-level and generates the output $y_1, y_2, ..., y_T$. The architecture of this model is shown in Figure 4.3. Following is the formal definition of the model (on the next page):

Char-LSTM

$$i_m^{char} = \sigma(W_i^{ch} ch_m + W_i^h h_{m-1}^{char} + W_i^c c_{m-1}^{char} + b_i^{char})$$

$$f_m^{char} = \sigma(W_f^{ch} ch_m + W_f^h h_{m-1}^{char} + W_f^c c_{m-1}^{char} + b_f^{char})$$

$$o_m^{char} = \sigma(W_o^{ch} ch_m + W_o^h h_{m-1}^{char} + W_o^c c_{m-1}^{char} + b_o^{char})$$

$$c_m^{char} = f_m^{char} \odot c_{m-1}^{char} + i_m^{char} \odot \tanh(W_c^{ch} ch_m + W_c^h h_{m-1}^{char} + b_c^{char})$$

$$h_m^{char} = o_m^{char} \odot \tanh(c_m^{char})$$

after final step, i.e $m = M$, considering bi-directional case

$$h_M^c = (h_M^{char})^{forward} \oplus (h_M^{char})^{backward}$$

obtaining single representation v_t for w_t

$$v_t = h_M^c \oplus x_t$$

Word-LSTM

$$i_t^{word} = \sigma(W_i^v v_t + W_i^h h_{t-1}^{word} + W_i^c c_{t-1}^{word} + b_i^{word})$$

$$f_t^{word} = \sigma(W_f^v v_t + W_f^h h_{t-1}^{word} + W_f^c c_{t-1}^{word} + b_f^{word})$$

$$o_t^{word} = \sigma(W_o^v v_t + W_o^h h_{t-1}^{word} + W_o^c c_{t-1}^{word} + b_o^{word})$$

$$c_t^{word} = f_t^{word} \odot c_{t-1}^{word} + i_t^{word} \odot \tanh(W_c^v v_t + W_c^h h_{t-1}^{word} + b_c^{word})$$

$$h_t^{word} = o_t^{word} \odot \tanh(c_t^{word})$$

considering bi-directional case

$$\tilde{h_t^{word}} = (h_t^{word})^{forward} \oplus (h_t^{word})^{backward}$$

$$y_t = softmax(W_y \tilde{h_t^{word}} + b_y)$$

(4.3)

Hyper Paramters	Value
Learning rate	0.05
LSTM hidden unit	256
Dropout rate	0.5
Batch size	64
Epochs	10
word vector dimension (d^w)	100
character vector dimension (d^c)	100

Table 4.8: Hyper parameters for word + character-level LSTM

Model	CBOW	skip-gram
Word-level LSTM	96.27	96.38
Word+Char-level LSTM	97.23	**97.98**

Table 4.9: Comparison of word-level and word+char-level LSTM for *fasttext* embeddings. Reported results are average of five fold cross-validation

We use categorical cross entropy (see Equation 4.2) as the loss function to this network. The hyper parameter of this model is described in Table 5.3. The word + character-level LSTM outperforms the word-level LSTM model skip-gram embedding by 1.04% when using `fasttext` CBOW and 1.60% when using `fasttext` skip-gram embedding (Table 4.9).

4.5 Analysis and Discussion

Manual analysis is performed with the best performing systems of each kind. These systems are: (i) dictionary-based system (e.g. EBHTL), (ii) SVM-based system (e.g. GDLCP_1N_1), (iii) CRF-based system (e.g. GDLCP_1N_1) and (iv) word +

Label	Precision	Recall	F-Score
bn	0.99	0.99	0.99
en	0.96	0.98	0.97
hi	0.95	0.96	0.96
mixed	0.95	0.57	0.71
ne	0.95	0.78	0.86
acro	0.86	0.78	0.82
univ	1.00	0.99	0.99

Table 4.10: Performance of word+char-level LSTM.

Label(s)	Dict	SVM	CRF	LSTM
en	90.99	93.00	92.49	98.47
bn	95.18	94.81	96.91	99.16
hi	58.62	72.05	74.19	96.25
mixed	17.91	50.74	17.91	56.71
ne	25.41	75.49	24.80	77.92
acro	55.86	76.05	53.05	78.40
univ	97.95	99.05	97.99	99.11
overall	90.22	93.06	92.29	**97.98**

Table 4.11: Per label accuracy of different systems, where dict = dictionary-based system (e.g. EBHTL), SVM = SVM-based system (e.g. GDLCP_1N_1), CRF = CRF-based system (e.g. GDLCP_1N_1) and LSTM = LSTM-based system (e.g. word + character level bidirectional LSTM-based system).

character level bidirectional LSTM-based system with `fasttext` skip-gram embeddings. Table 4.11 shows the per label and overall accuracy on our data produced by these systems. In terms of five-fold cross-validation accuracy the word + character level bidirectional LSTM-based system achieves the highest accuracy with a score of 97.98%. We further perform different analysis on these results described in the following subsections.

4.5.1 Statistical Significance Test

For statistical significance testing we use two-sided bootstrap re-sampling Efron (1979) by implementing the pseudo-code of Graham et al. (2014). We use a sample size of 1K and $\alpha = 0.05$. We find that (i) the improvement of SVM, CRF and LSTM over dictionary-based system is statistically significant, (ii) the improvement of SVM over CRF is also statistically significant and (iii) the improvement of LSTM over SVM and CRF is statistically significant.

4.5.2 Ambiguous vs Non-ambiguous Words

The output of these systems are analysed for the ambiguous and non-ambiguous tokens. Figure 4.4 shows the performance of different systems for these two categories. We find that for the ambiguous tokens LSTM achieves highest accuracy (95.02%)

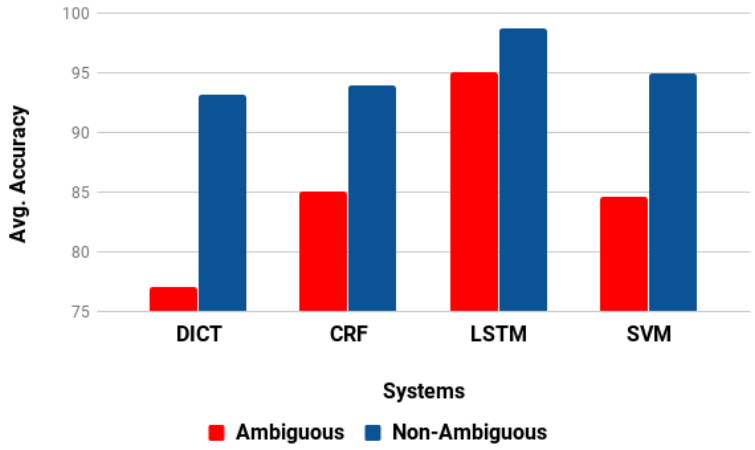

Figure 4.4: Performance of different systems for ambiguous and non-ambiguous tokens where DICT = dictionary-based system (e.g. EBHTL), SVM = SVM-based system (e.g. GDLCP_1N_1), CRF = CRF-based system (e.g. GDLCP_1N_1) and LSTM = LSTM-based system (e.g. word + character level bidirectional LSTM-based system).

which is followed by the performance of CRF ((85.06%), SVM (84.51%) and the dictionary-based system (77.05%). LSTM also achieves highest accuracy for the non-ambiguous tokens (98.63%). The performance of CRF (93.88%), SVM (94.93%) and dictionary-based system (93.10%) are competitive for the non-ambiguous tokens.

4.5.3 Code-Mixing Points

A code-mixed point is the point where language changes happen (e.g. English to Bengali, Hindi to English). We consider a token as a code-mixed point (token-0) if the language of the token has been changed compared to the language of the previous token. Labels such as 'ne', 'acro', 'univ' and 'mixed' are not considered as change of language. It can be observed that the LID accuracy suffers at the code-mixed points. Figure 4.5 shows the result of our analysis, where +1 means one token to right of a code-mixed point and -1 means one token to the left. It can be seen that all tested methods perform poorly at code-mixed points. Performance of these systems increases as the distance to code-mixed points increases. For example, let us consider the following sentence:

aap/**hi**/*hi* toh/**hi**/*hi* [go/**en**/*hi*] went/**en**/*en* gone/**en**/*en*

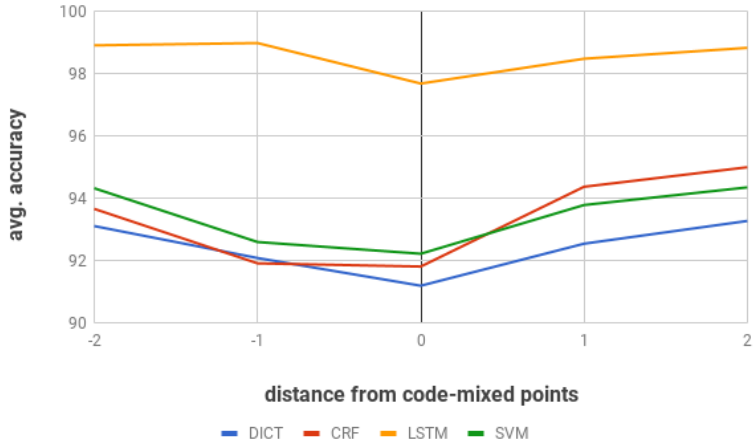

Figure 4.5: Performance at code-mixed points and surroundings: Considering 0 as code-mixing point, +i means i token to right of a code-mixed point and -i means i token to the left where DICT = dictionary-based system (e.g. EBHTL), SVM = SVM-based system (e.g. GDLCP_1N_1), CRF = CRF-based system (e.g. GDLCP_1N_1) and LSTM = LSTM-based system (e.g. word + character level bidirectional LSTM-based system).

In the above sentence 'go' is the code-mixed point. Gold labels are written in bold and predicted labels are written in italics. It can be seen the word in the code-mixed point is actually an English word but all systems have classified it incorrectly. As a reason it can be noted that this particular lexical form 'go' is shared across three languages due to Romanisation. In Bengali and Hindi it is also frequent. Such ambiguous words when preceded by a particular language (in this case Hindi) all of our systems tend to tag it by the language of the previous word. To understand this effect, we also investigate that how an ambiguous token affects the accuracy at a code-mixed point (see Figure 4.6). We find that 20.69% code-mixed points are made of ambiguous tokens. We observe that the performance of all methods suffers if an ambiguous token is present at a code-mixed point. On the other hand for a non-ambiguous token at a code-mixed point, the accuracy of all systems is higher than 93.00%.

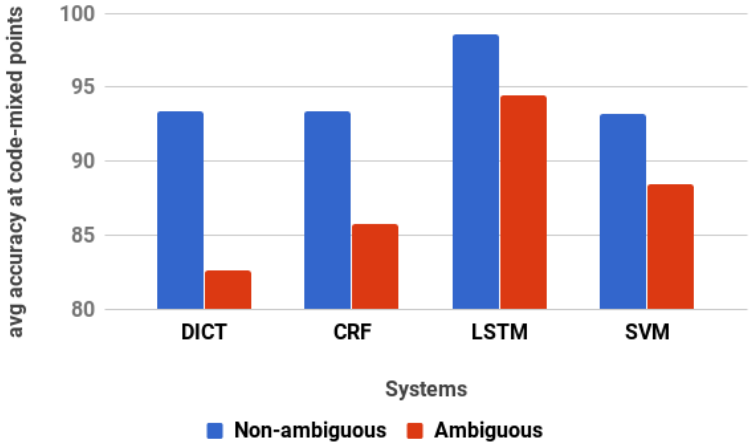

Figure 4.6: Performance at code-mixed point: Ambiguous and non-ambiguous tokens, where DICT = dictionary-based system (e.g. EBHTL), SVM = SVM-based system (e.g. GDLCP_1N_1), CRF = CRF-based system (e.g. GDLCP_1N_1) and LSTM = LSTM-based system (e.g. word + character level bidirectional LSTM-based system).

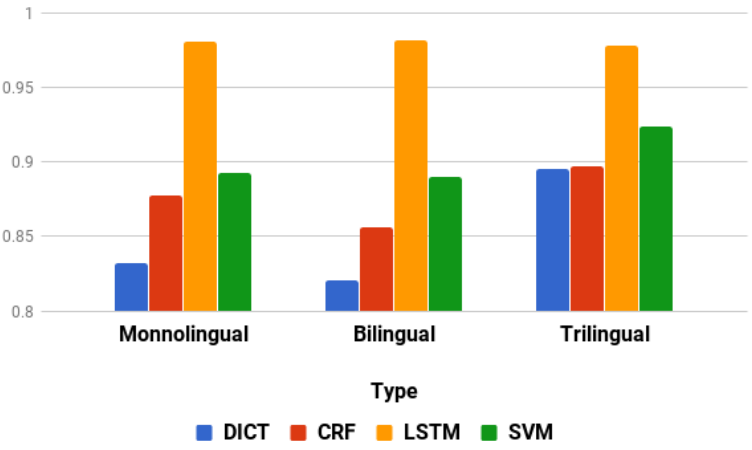

Figure 4.7: Performance of different systems on monolingual and code-mixed sentences where DICT = dictionary-based system (e.g. EBHTL), SVM = SVM-based system (e.g. GDLCP_1N_1), CRF = CRF-based system (e.g. GDLCP_1N_1) and LSTM = LSTM-based system (e.g. word + character level bidirectional LSTM-based system).

4.5.4 Monolingual vs Code-Mixed Sentences

Figure 5.4 shows the performance of different systems for monolingual and for code-mixed sentences. Among monolingual sentences, we observe that LSTM performs best in terms of average word-level accuracy of 98.06%. The dictionary-based system is always outperformed by SVM, CRF and LSTM. For bilingual and trilingual mixing, again LSTM is the best performer achieving 98.16% and 97.75% average word-level accuracy. Other systems, suffers heavily when classifying bilingual and bilingual sentences (less than 90%) but for tringual sentences the performance of all systems is near (dictionary) or over (SVM, CRF) 90%. To understand this effect we analyse the number of code-switching points in bilingual and trilingual sentences, and find that the total number of code-mixing points is 2,891 in bilingual sentences, whereas it is only 43 for the trilingual sentences. It has been seen that the performance of all of the systems suffers at code-mixed points. This can be one of the reasons of the poor performance of all the systems in a bilingual content.

4.5.5 Word-based Vs. Word+Character-based LSTM

Table 4.12 shows the comparison of the two systems word-based LSTM and word + character-based LSTM models. It can be observed that for each label the word + character-based model outperforms the word-based model. The performance for 'mixed', 'ne' and 'acro' achieved significant improvement when character-level LSTM has been used in the network. This indicates that even if the two networks have been presented with the same word representation (`fasttext`) that considers the character n-grams, the word + character level model has learned some extra features that are helpful to classify word-level code-mixing, name-entity and acronym. The word + character level LSTM consistently outperforms the word-level model.

Label	Word-LSTM	Word+Character LSTM
bn	0.98	0.99
en	0.96	0.97
hi	0.93	0.96
mixed	*0.44*	*0.71*
ne	*0.71*	*0.86*
acro	*0.62*	*0.82*
univ	0.99	0.99

Table 4.12: Comparison of Word-based and Word + Character-based LSTM: Reported Results are F1 scores.

Label	en	bn	hi	mixed	ne	acro	univ
en	5827	37	20	1	9	14	9
bn	74	12182	12	1	6	4	6
hi	16	33	1440	0	5	0	2
mixed	21	6	0	38	0	2	0
ne	57	41	39	0	512	7	1
acro	31	8	3	0	4	167	0
univ	26	10	3	0	3	0	4706

Table 4.13: Confusion Matrix of Word + Character-level LSTM Model

4.5.6 Error Categories

Table 4.13 shows the confusion matrix for the best performer (word + character-level LSTM). The most frequent error categories are: 'bn' to 'en' (i.e. Bengali word is miss-classified as English) and 'ne' to 'en, bn and hi'. For the first category, it has been observed that the most of the miss-classified Bengali words are ambiguous in nature, for example: 'day', 'pass' and 'ful'. Some errors ('bn'-'en') also occurred for the code-mixed reduplicated expressions, for example - 'kiss-tiss, boss-toss, post-fost'. For these expressions, the second word is the Bengali word but both of the words are tagged as English. For other instances of this error category, we find that either the Romanisaed versions of these words are very close to some English words, for example 'prten (pertain), chrome (chrame)', or these words are very short words ($length <= 2$), for example: 'a, k, v' with a very small amount of context, for example:

- ... k bol ?

- a ma ...
- or jonne ..

Named entities are miss-classified as Bengali, Hindi or English. In romanised social media content, feature like 'capitalisation' is inconsistent. In the Indian languages (e.g. indo-aryan language family) ambiguity occurs between between common and proper nouns for named entity identification. For example: the word, 'akash' can be a name (e.g. 'akash sinha', proper noun) or it can be a common noun which means 'sky'. Further, adhoc Rromanisation also increases ambiguity. Considering these facts we observe that most of 'ne to other languages' errors follow the ambiguity pattern. For example, 'rimjim', 'nana', 'mamata' for Bengali (each of these word can be a part of named entity or can be other words depending on the context). These words have been incorrectly classified as Bengali words. We also find that that most of these words appeared after a Bengali word in the content. As for Hindi, similar error patterns can be observed where a ambiguous word (e.g. 'ne'/'hi') preceded by a Hindi word has been incorrectly classified. For example: 'day', 'bhole', 'zahir'. We also observe another pattern of error where the first token of a named entity has been incorrectly classified as a language token. For example:

- ... science city ..
- ... the telegraph ...

The words, 'science' and 'the' is included in the named entity but all of our systems have produced wrong level (e.g. 'en') for the two tokens. Name entity identification is benefited if features like POS, chunks are available. However, in our LID experiments we did not include such features whcih can be an explanation for this type of error.

The top three error categories produced by SVM, CRF and dictionary-based systems are shown in Table 4.14. The most common error pattern produced by all three systems (see first row of Table 4.14) is hi-bn, i.e. Hindi words that are

Errors	DICT	CRF	SVM
hi-bn	533	316	559
en-bn	426	333	303
ne-bn	283	278	314

Table 4.14: Top three errors categories for different systems where DICT = dictionary-based system (e.g. EBHTL), SVM = SVM-based system (e.g. GDLCP_1N_1), CRF = CRF-based system (e.g. GDLCP_1N_1).

classified as Bengali words. The following comment is an example where this type of error has been occurred:

er/bn manei/bn holo-vai/bn **sab/hi** ka/hi nahi/hi lagta/hi

The word **'sab'** is a shared word among Bengali and Hindi. It appeared 8 times as Bengali and 5 times as Hindi in the data set. In this sentence the word 'sab' is a code-mixed point and is misclassified as Bengali by all of our systems. Following is an example of a en-bn error, i.e. English word 'age' has been classified as an Bengali word by all of our systems.

.../univ from/en my/en personal/en experience/en **age/en** doesn't/en matter/en

The word 'age' is an ambiguous word and appeared 17 times as Bengali, 2 times as Hindi and 2 times as English in our data set. An example of 'ne-bn' error category is 'Manna/ne **Dey/ne**', where **Dey** has been classified as Bengali by all of our systems. The word 'Dey' is also an ambiguous word which appeared 6 times as Bengali and only once as a part of name entity in our data set.

4.6 Conclusion

We have performed 7-way word-level language identification with trilingual code-mixed content using a dictionary-based baseline, with an SVM, with a CRF and with an LSTM. We have explored character n-grams, presence in dictionaries, length, capitalisation, previous and next word as features with SVM and CRF. We also

explored the use of neural word-embeddings with LSTM. In this section we answer the set of research questions mentioned in this chapter (Section 4.3) which were the motivation for this chapter.

- We find that a carefully designed dictionary-based system that combines monolingual and code-mixed data can perform reasonably well (over 90%) with this trilingual code-mixed data.

- However, an SVM (93.06%) and a CRF (92.29%) when trained with useful features (e.g. character n-grams, presence in dictionaries, length, capitalisation, previous and next word) can outperform the dictionary-based system significantly.

- It is possible to outperform SVM and CRF using LSTMs (word-level and word + character-level) with neural word embeddings (`fasttext`). In fact, our word + character-level LSTM outperforms all other systems in our experiments (98%).

- The word ambiguity due to Romanisation and other reasons (e.g. shared vocabulary across Bengali and Hindi) is a critical point of word-level LID. We observe that all of our systems suffer due to this ambiguity. Among all the systems our word + character-level LSTM performs better than the other, and achieves 95% word-level average accuracy for these words (Figure 4.4).

- We observe that embeddings that use characters as well as words (e.g. `CWE`, `fasttext`) are better than embeddings (`word2vec`) that only consider words as a feature. The results are depicted in Table 5.2.

We also observe that the accuracy of each model suffers at the code-mixed points (Figure 4.5) due to the presence of ambiguous tokens (Figure 4.6). It is also observed in error analysis that code-mixed reduplicated expressions are difficult instances to classify. As for word and word + character-level model, we find that the latter has performed better at word-level. Both of these models are presented with an

embedding that considers character n-grams. However, the word + character-level model learns some extra information from the character-level representation of the word. It can be seen in Table 4.12 and Table 4.9 which indicates that necessity of the character-level representation in LID for code-mixed social media content. In the next chapter, we describe our experiments of POS tagging, we investigate the use of LID (e.g. as a feature, as decision boundary for language specific chunks) in different POS tagging methods. We also use SVMs, CRFs and word + character-based neural model in POS tagging task.

Chapter 5

Part-of-Speech Tagging

Part-of-Speech (POS) tagging is the process of automatically assigning lexical categories (noun, pronoun, verb etc.) to each word in a sentence and is a well-known problem in NLP which facilitates the further processing and understanding of natural language through other modules, e.g. word sense disambiguation, parsing and sentiment analysis. POS tagging is a well explored problem in NLP and researchers have developed highly accurate POS taggers for monolingual and formal content. The classical problem in automatic POS tagging is to handle the ambiguity of words. The following are the examples:

- *A*/DT ***plant***/NN *needs*/VB ***light***/NN *and*/CC *water*/NN *./.*
- *Each*/DT *student*/NN *will*/MD ***plant***/VB *one*/CD *./.*

Each word in the above examples is annotated with the POS tag of the Penn Treebank tag set (Marcus et al., 1993). In the first sentence the word 'plant' is used as a noun (NN), but in the second example it is used as a verb (VB). In the case of social media data, POS tagging is harder because of spelling variations and informal writing style. Moreover, the task is even harder if the social media data is code-mixed and romanised.

- .../SYM ***age***/NN *doesnt*/VBZ
  matter/VB .../SYM ***take***/VB *a*/DT *bow*/NN *:D*/SYM .../SYM

- *manush*/bn/N_NN *to*/bn/RP_RPD ***age***/bn/N_NST *nijeke*/bn/PR_PRF *chene*/bn/V_VM *jane*/bn/V_VM *valo*/bn/JJ *kore*/bn/V_AUX .../univ/SYM *nijer*/bn/PR_PRF *culture*/en/N_NN ***take***/bn/PSP *boje*/bn/V_VM .../univ/*SYM*

The above examples are Facebook posts, taken from our English-Bengali-Hindi code-mixed data. The first example is an English post and the tagging follows the convention of the Penn Tree bank Marcus et al. (1993). The second example is a Bengali-English code-mixed post. Here, we follow the standardised LDC-IL POS tag set to tag this post and we add language tags for convenience.[1] In the first example, 'age' is an English word, whereas in the second example 'age' is a Bengali romanised word. Another instance of vocabulary sharing due to Romanisation can be found for the word 'take' in both of these examples. In the second example the presence of an English word 'culture' indicates that this post is code-mixed. However, to the annotators it is clear that 'take' is a Bengali word in this sentence and that it is a post-position (PSP), not a verb as in English. Considering these two examples, it can be assumed that code-mixing and phonetic typing increase POS ambiguity.

5.1 Chapter Organization

In this chapter, we present our experiments on POS with a trilingual (English-Bengali-Hindi) code-mixed social media content. We discuss related studies of POS tagging in Section 5.2, present our research questions in Section 5.3, and describe our experiments in Section 5.4. Section 5.5 describes the analysis of the performance of different systems and finally we summarize this chapter in Section 5.6 by answering the research questions and by highlighting other findings.

[1] http://www.ldcil.org/standardsTextPOS.aspx

5.2 Background

Different methods have been developed for the POS tagging problem, e.g. hand-written rule-based (Voutilainen, 1995) and stochastic (Cutting et al., 1992; Brants, 2000) methods. Brants (2000) developed a stochastic HMM-based tagger called TnT. The tagger uses a second order Markov model with smoothing techniques (e.g. linear interpolation) and suffix analysis for handling unknown words. Toutanova and Manning (2000) used a Maximum Entropy Markov Model (MEMM) with capitalisation features and a set of disambiguation features (e.g. the tense of verbs) and achieved over 96% accuracy on a Wall Street Journal (WSJ) test set. Nakagawa et al. (2001) proposed a method for POS tagging of unknown English words using SVMs. They used contextual and sub-string information of a token as features in their experiments. A disadvantage of their system is that it takes a long time to train. Giménez and Màrquez (2004) proposed an SVM-based POS tagger (SVM-Tool) which is faster than the previous one. They used a rich feature set to train their system. These features are word n-grams, orthographic features, the length of a word, prefix and suffix information of a word and the POS features from the training data (e.g. POS window and ambiguous classes for a word). They tested their system on two languages, Spanish and English, and they found that their tagger outperformed the TnT tagger. Schmid (1994b) combined a Markov model with estimation of transition probabilities with decision trees. The system is called "Tree-Tagger" and parameters for 19 languages are available.[2] Lafferty et al. (2001) used CRFs in POS tagging, which achieved better performance than HMM and MEMM in their experiments. Other machine learning techniques have also been used in this area, e.g. decision trees (Black et al., 1992) and CRFs (Lafferty et al., 2001).

POS tagging in social media is hard due to the nature of social media content. Foster et al. (2011) evaluated off-the-shelf tools in POS tagging and parsing of Twitter data. They found that propagation of POS tagging errors leads to poor parsing results. Gimpel et al. (2011) performed POS tagging on Twitter data with

[2] `http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/`

CRFs and adapted their features to Twitter characteristics. They also designed and developed a POS tag set for Twitter. Owoputi et al. (2013) used a first order MEMM for POS tagging in Twitter data with large-scale unsupervised word clusters and lexical features. They found that their MEMM-based model outperformed their previous CRF-based model. As an explanation, they pointed out that the quality of lexical features is much more important than the type of sequence model.

POS tagging of Indian languages is a very challenging task. The reasons are the limited availability of annotated corpora and the morphological richness of Indian languages. Indian languages can be categorised into two major linguistic families, namely Indo-Aryan (e.g. Bengali and Hindi) and Dravidian (e.g. Tamil and Telugu). The two groups of languages have different grammatical structure. Moreover the Dravidian family is agglutinative and the Indo-Aryan family is inflective. Many POS taggers for Indian languages use morphological analysers. For example, Ray et al. (2003) performed POS tagging in Hindi using lexical sequence constraints, ontological information and a morphological analyser. Singh et al. (2006) used morphological information and the CN2 decision tree induction algorithm to build a Hindi POS tagger. They tested their system on a BBC news corpus and achieved over 93% accuracy. Palanisamy and Devi (2006) developed a POS tagger for Tamil using an HMM-based model. Shrivastava et al. (2006) investigated the use of CRFs in Hindi POS tagging. They used lexical features and spelling features in their experiments. Dandapat et al. (2004) investigated the use of an HMM-based POS tagger in Bengali. They used tagged and untagged data by partially supervised learning. First the tagged data was processed by supervised learning in their method, then through multiple iterations they processed the untagged data and updated the transition and emission probabilities of their supervised HMM-based system. They also used a morphological analyser in their system and concluded that the accuracy of the morphological analyser must be increased and typographical errors of untagged data must be corrected for better POS tagging. Later, Dandapat et al. (2007) compared supervised and semi-supervised HMM-based models with MEMM-

based models for Bengali and found that, with a small amount of training data, MEMM-based models perform better but with a large amount of training data the performances are comparable. Ekbal and Bandyopadhyay (2008) developed POS taggers for Bengali, Hindi and Telugu using SVMs and HMMs. They found that SVMs perform better than HMMs in all three languages. In 2007, as part of the Shallow Parsing for South Asian Languages (SPSAL) workshop at IJCAI-07, a competition on POS tagging and chunking for south Asian languages (e.g. Hindi, Bengali and Telugu) was conducted, where different participants applied a wide range of learning techniques such as HMM, MEMM, decision trees, CRFs and Naive Bayes. The best POS tagging accuracy of 77.61% in Bengali was achieved using an MEMM-based model Dandapat et al. (2007). For Hindi and Telugu the best accuracies of 78.66% and 77.37% were achieved by CRF-based methods PVS and Karthik (2007).

POS tagging with Spanish-English code-mixed data was first explored by Solorio and Liu (2008b). They examined heuristic and machine learning methods. They use two monolingual taggers, a Spanish tagger and an English tagger, in their experiments. Their heuristic method used confidence thresholds and a few other indicators (e.g. tags that indicate a foreign word and the lemma of a word) from the two different taggers. They also combined a dictionary-based language identifier with their heuristic method and observe a performance boost. In their machine learning methods, the monolingual POS taggers were used again to extract features (e.g. the word, POS tags, the lemma and confidence scores according to both Spanish and English taggers) for their experiments. Among different machine learning methods (e.g. SVM, Naive Bayes, decision trees), the highest accuracy blue achieved by an SVM classifier. The following features are employed in their SVM-based experiments: (1) the word itself, (2) English POS tag, (3) English POS tagger lemma, (4) English POS tagger confidence, (5) Spanish POS tag, (6) Spanish POS tagger lemma and (7) Spanish POS tagger confidence.

Vyas et al. (2014) implemented a pipeline approach for POS tagging in English-

Hindi code-mixed data. Word-level language identification, text normalisation, transliteration and POS tagging are performed sequentially with the help of off-the-shelf tools. They divided the text into contiguous maximal word chunks which are in the same language. After that, language-specific POS taggers were applied to predict the POS labels of those word chunks. They identify that normalisation and transliteration are two challenging problems in this pipeline approach. This pipeline approach is also investigated by Jamatia and Das (2014) for non-romanised English-Hindi tweets. However, they identified that the combination of two different POS taggers leads to lower accuracy for their data.

Jamatia and Das (2014) explored POS tagging in Hindi-English tweets. Most of their data is in Hindi script. The number of code-mixed tweets is small, only 400 out of 1,700 tweets and the remaining tweets were monolingual Hindi tweets. For monolingual Hindi tweets, they trained an off-the-shelf POS tagger with 1,200 monolingual tweets and tested it on 100 tweets. They achieved 86% accuracy with this method. They performed experiments with different learning algorithms (e.g. SMO, Naive Bayes and decision tree) with prefix and suffix character n-grams, current, previous and next words as features. Among these, random forest performed best. For code-mixed tweets, they applied the same learning algorithms with the same features for POS tagging in code-mixed data and found that random forest achieved best accuracy (63.65%). They also used two different POS taggers to predict the POS label for words (i.e. English POS tagger for an English word and Hindi Post tagger for a Hindi word) like Vyas et al. (2014). However, they identify that the combination of two different POS taggers leads to lower accuracy compared to their decision tree based method for their data. The following are the features that were employed in their experiments: (1) The word itself, (2) previous three words, (3) previous 3 tags, (4) next 3 words, (5) 4-gram prefix, (6) four-gram suffix and (7) word length threshold ≤ 4. Combining monolingual taggers and language identification is a common trend in code-mixed POS tagging, either language identification decisions are made to obtain correct POS tag (Vyas et al., 2014; AlGhamdi et al., 2016) or

they are used to extract features (Jamatia et al., 2015; Sequiera et al., 2015a; Schulz and Keller, 2016). Ghosh et al. (2016) investigated code-mixed POS tagging using CRFs that incorporate of-the-shelf parsers and linguistic resources. As features they used prefix, suffix, the word and its neighbour and multiple doctrinaires and other handcrafted indicators (e.g. the word contains digit or not, is the word represents quantities). However, for acronyms, names and universal symbols they used some post processing rules. Feature extraction based POS tagging is also investigated by Pimpale and Patel (2016) by using Naive Bayes, multilayer Perceptrons and Decision Trees. Similar work (Jamatia et al., 2015) can be found where hand-crafted features were used with CRF, SVM, Naive Bayes and Random Forests (RF) to perform English-Hindi code-mixed POS tagging. Similar approaches were also investigated by Gupta et al. (2017); Ramesh and Kumar (2016) using CRF.

In the above works, monolingual POS taggers were used in two ways: (1) code-mixed sentences were broken into language specific chunks and then language specific POS taggers are applied to these chunks (Vyas et al., 2014), and (2) code-mixed sentences were sent to the taggers, similar to (Solorio and Liu, 2008b). Monolingual taggers are typically trained on full monolingual sentences. If language-specific text fragments are presented to such monolingual taggers, the taggers treat these fragments as full sentences. At the start and at the end of the input, the prediction of such taggers may become biased to some specific patterns (e.g. NN and NNP) that have been observed frequently at the start and at the end end tag sequence of sentences during training. Any approach that uses two monolingual POS taggers to process language-specific chunks (Vyas et al., 2014) may have suffered from this effect. On the other hand, when a code-mixed sentence is sent to both of the taggers as it is (Solorio and Liu, 2008b), each tagger may fail to tag foreign words with the special foreign word (FW) tag. The decision of an individual tagger might be affected at a code-mixed point. However, Solorio and Liu (2008b) use the information obtained from the taggers as features in their SVM-based experiments. Therefore, it is possible that their system repairs tagging errors of the monolingual taggers.

Furthermore, a pipeline approach (Vyas et al., 2014) suffers from error propagation from a particular module to the next module. As Vyas et al. (2014) use transliteration as a key module in their pipeline, transliteration errors may have affected the performance of POS tagging in their experiments. Transliteration from social media data is itself a challenging task due to spelling variations.

Recently, RNN models have been shown to have good performance in POS tagging of formal content, but for code-mixed POS tagging these methods are not well explored. Wang et al. (2015) have investigated the use of bidirectional LSTM on the Penn Treebank WSJ test set and achieved state-of-the-art performance. Zennaki et al. (2015) used RNN on low resourced languages and found comparable results with state-of-the-art POS taggers (Das and Petrov, 2011; Duong et al., 2013). Among neural approaches to code-mixed POS tagging, Patel et al. (2016) used recurrent newural network (RNNs, LSTMs and GRUs) with word embeddings for Hindi-English code-mixed content and found that the GRU baed model has outperformed the other models. Bhat et al. (2018) presented a treebank of Hindi-English code-mixed tweets and proposed a neural stacking model for parsing. They also performed POS tagging as a intermediate task in the stacking model.

5.3 Research Questions

In recent research (Solorio and Liu, 2008b; Vyas et al., 2014; Ghosh et al., 2016; Jamatia et al., 2015; Gupta et al., 2017; Ramesh and Kumar, 2016) it has been seen that code-mixed POS tagging can be improved by exploiting monolingual POS tagger. Given this approache, we will try to find the answers to the following questions in this chapter.

- How well will an single classifier like SVM or a first-order CRF with handcarfted features works for trilingual romanised and code-mixed content?
- Which one is the best approach for code-mixed POS tagging - Pipeline or Stacking ? Do these methods perform better than SVM and CRF taggers?

- Training/developing monolingual taggers requires effort when low-resourced languages are considered. Can a recurrent neural sequence labelling method (e.g. Long Short Term Memory) with neural embeddings perform better than the previous methods?

- What is the effect of the use of language labels in POS tagging?

5.4 Experiments

We divide the experiments into four parts. We implement baselines for POS tagging in Section 5.4.1. In Section 5.4.2.1 we implement pipeline systems. In Section 5.4.2.2 we present our stacking systems. In section 5.4.3 we present our LSTM based systems. We perform five fold cross-validation with the data and report average cross-validation accuracy.

5.4.1 Baseline

We investigate the use of hand-crafted features and features that can be obtained from monolingual POS taggers (stacking). We perform experiments with different combinations of these feature sets. The following are the features used in our experiments.

1. **Hand-crafted Features:** Following Barman et al. (2014a), we use prefix and suffix character-n-grams (n = 1 to 5), presence in dictionaries, length of the word, capitalisation information and the previous and the next word as hand-crafted features.

2. **Stacking Features:** These features are obtained from the output of a POS tagging system. These features are tokens, predicted labels, and prediction confidence of a POS tagging system.

3. **Combined Features:** This feature set is a union of previous two feature sets.

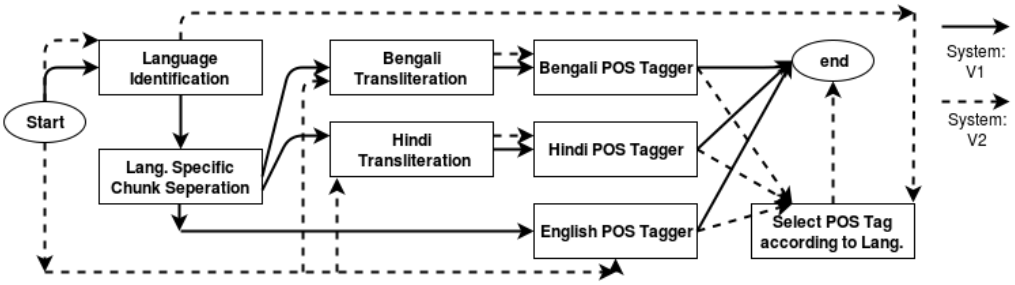

Figure 5.1: Pipeline systems: system V1 and V2 (Section 5.4.2.1).

We use our LID classifier using handcrafted features and SVM. Its predictions are used in the POS tagging experiments below. This method only uses the *code-mixed romanised data* and hand-crafted features. We try an linear kernel SVM and a linear chain CRF classifier (see Table 5.1). In terms of average cross-validation accuracy, the SVM classifier (85.00% for $C = 0.00097$) performs better than the CRF classifier (83.89%) in optimised settings.

5.4.2 Exploiting Monolingual Taggers

5.4.2.1 Pipeline

Following Vyas et al. (2014), the training data for this method is *monolingual non-romanised*. First, it uses an LID system (trained on romanised data) to identify language-specific chunks. After that it applies monolingual POS taggers to the relevant language chunks to produce the output. The component POS taggers are trained on monolingual non-romanised data.

Code-mixed romanised data passes through a pipeline of LID, transliteration and POS tagging modules. For example, for Bengali-English romanised code-mixed content, the LID module produces Bengali and English chunks, and the Bengali chunks are transliterated into Bengali script and are sent to a Bengali tagger. The English chunks are sent to an English tagger as they are. The final output combines the results from the individual taggers. To implement this method we carry out the following steps:

1. We perform transliteration based on language using Google Transliteration[3] for Hindi and Bengali tokens. (Vyas et al. (2014) use an in-house tool).

2. For the next step of the pipeline, we train monolingual POS taggers for Bengali and Hindi using the SNLTR Bengali and Hindi corpus[4] with TreeTagger[5] (Schmid, 1994a). For English we use the default English model which is available with the TreeTagger package. We also use a lightweight Bengali and Hindi stemmer to provide a stemmed lexicon to TreeTagger during training. We use these taggers to make predictions on English, transliterated Bengali and transliterated Hindi chunks.

The black lines in Figure 5.1 show the pipeline of this method (V1). The three training data sets for the three POS taggers follow different tag sets, we map these tags to the universal POS tags after prediction.[6] We achieve 71.12% average cross-validation accuracy with this method (V1) (third row of Table 5.1).

In method V1, the TreeTagger models are trained on full monolingual sentences. If language specific text fragments are presented to such monolingual taggers, the taggers may treat these fragments as full sentences. At the start and at the end of the input, the prediction of such taggers may become biased to some specific patterns (e.g. NOUN + PUNCT) that have been observed frequently as a start and an end tag sequence of sentences during training. To avoid this problem we implement a variant (V2) of this system in which we present full sentences (that may contain junk transliteration) to each POS tagger. We perform transliteration as the first component of the system. We present the transliterated content in Bengali script to the Bengali tagger, original romanised content to the English tagger and transliterated content in Hindi script to the Hindi tagger. Finally, we choose from the outputs of these three taggers based on the language prediction by the SVM

[3]https://developers.google.com/transliterate

[4]http://nltr.org/snltr-software/

[5]http://www.cis.uni-muenchen.de/ schmid/tools/TreeTagger/

[6]We also implement a system where all the tags in the SNLTR corpus are converted to universal POS tags before training. This variant does not outperform the current system.

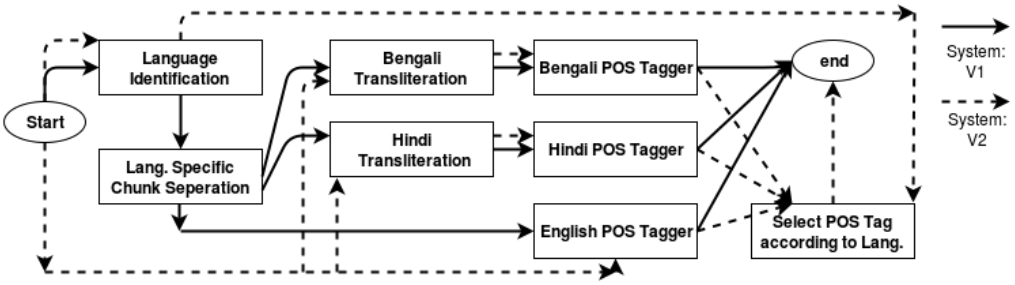

Figure 5.2: Stacked systems: system S1 and S2 (Section 5.4.2.2).

classifier for the original (romanised) content. The pipeline of this system (V2) is shown by the dotted lines in Figure 5.1. We achieve 71.27% average cross-validation accuracy in this method (V2) (fourth row of Table 5.1).

5.4.2.2 Stacking

This method uses *non-romanised monolingual and romanised code-mixed data* with hand crafted, stacking and combined features. This method follows the approach of Solorio and Liu (2008b) with necessary adjustments. In this method, romanised code-mixed content is transliterated blindly in all languages and is presented to different POS taggers (trained with non-romanised monolingual data), as in method V2. The romanised words and the output from the monolingual taggers are used as features to train an SVM classifier on romanised code-mixed content. To keep our methodology as similar as possible to Solorio and Liu (2008b) we follow the steps described below:

1. We train a Bengali and a Hindi TreeTagger (Schmid, 1994a) using the SNLTR corpus with default settings as described in Section 5.4.2.1.

2. We transliterate each token of a sentence into Hindi and Bengali irrespective of its language using Google Transliteration as in system V2.

3. After transliteration we send each transliterated output to the respective Tree-Tagger, i.e. we send the original sentence to the English TreeTagger, Bengali transliterated output to the Bengali TreeTagger and the Hindi transliterated output to the Hindi TreeTagger.

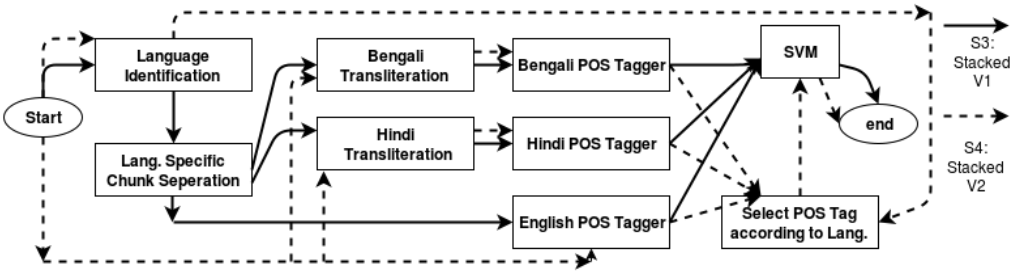

Figure 5.3: Pipeline Systems in Stacking: S3 (stacked-V1) and S4 (stacked-V2)

After that we follow the stacking approach of Solorio and Liu (2008b). Here, we stack an SVM classifier on top of the predictions generated by the TreeTaggers. We train a linear kernel SVM with stacking features and optimise parameter C in five fold cross-validation. The black lines in Figure 5.2 show the pipeline of this system (S1). The average cross-validation accuracy of this system is shown in the fifth row of Table 5.1. Average cross-validation accuracy of 86.57% is found by this method (S1). Given the setup, we further experiment by using the combined features from romanised and transliterated tokens and also consider SVM language predictions as a feature. We observe that combining these features boosts the accuracy. After trying combinations of these features the best accuracy (87.59%) is achieved by adding all features together (S2) (sixth row of Table 5.1). The architecture of the system is shown by the dotted lines in Figure 5.2. We also investigate the use of pipeline systems in stacking. The idea is to use all the predictions from a pipeline system and feed them into an SVM classifier. The stacked version of V1 (stacked-V1) achieves 85.99% and the stacked version of V2 (stacked-V2) achieves 85.83% average cross-validation accuracy with SVM using combined features. The black lines in Figure 5.3 show the pipeline of S3, stacked-V1 and the dotted lines show the pipeline of S4, stacked-V2. These methods do not outperform our implementation of Solorio and Liu (2008b)'s method S1 or its extended version S2.

5.4.3 Long Short Term Memory (LSTM)

The basic neural model for POS tagging is the same as the word + character-level LSTM that was used in the previous chapter for LID. We apply two versions of the

Type	Systems	Acc.
Baseline	SVM	85.00%
	CRF	83.89%
Pipeline	V1: Vyas	71.12%
	V2: Extn. of V1	71.27%
Stacking	S1: Solorio	86.57%
	S2: Extn. of S1	87.59%
	S3: Stacked-V1	85.99%
	S4: Stacked-V2	85.83%
LSTM	language features	**92.71**%

Table 5.1: Average cross-validation accuracy of POS tagging systems.

model: (1) without and (2) with language identification. To include language labels as features we use the one-hot output of our neural language classifier. During word representation the output of the language identifier network is concatenated with the word and character level representation:

$$v_t = h_M^c \oplus x_t \oplus l_t \tag{5.1}$$

where, l_t is the output of our language identifier system, x_t is the pre-trained word embedding and h_M^c is the word representation from the character-level LSTM. We use categorical cross-entropy as our loss function to this network. Dropout (Hinton et al., 2012) is used in different parts of our architectures. Dropout (with a keep probability of 0.5) is applied after the embedding layer, after bi-LSTM and before the output layer of the network. As prepossessing steps, maximum word length is set to 30 characters, maximum sentence length (i.e. the temporal dimension of LSTM) is set to 78 words. All words having less than 30 characters are padded with a special token to achieve same length for all words, similar thing is done with sentences to achieve characters the same sentence length over the data. The hyper parameters of this model is shown in Table 5.3. When using language identification as extra feature we achieve 92.71% and without language information the word + character level LSTM performs reasonably well (91.60%). These results are shown in Table 5.2, 5.4 and 5.5.

Model	Performance
Without LID Features	91.60
With LID Features	92.71

Table 5.2: Word + Character-level LSTM with and without language features.

Hyper Paramters	Value
Learning rate	0.05
LSTM hidden unit	256
Dropout rate	0.5
Batch size	64
Epochs	10
word vector dimension (d^w)	100
character vector dimension (d^c)	100

Table 5.3: Hyper parameters for word + character-level LSTM

Label	Precision	Recall	F-Score
punct	1.00	1.00	1.00
adj	0.89	0.87	0.88
adp	0.89	0.82	0.85
adv	0.90	0.86	0.88
conj	0.85	0.90	0.87
det	0.89	0.90	0.89
noun	0.93	0.94	0.93
num	0.93	0.95	0.94
pron	0.91	0.92	0.91
prt	0.89	0.95	0.92
verb	0.93	0.92	0.92
x	0.94	0.96	0.95

Table 5.4: Performance of the LSTM based model with language features.

Label	Precision	Recall	F-Score
punct	1.00	1.00	1.00
adj	0.91	0.81	0.86
adp	0.88	0.81	0.84
adv	0.91	0.80	0.85
conj	0.85	0.84	0.84
det	0.89	0.89	0.89
noun	0.90	0.94	0.92
num	0.94	0.93	0.94
pron	0.90	0.91	0.90
prt	0.89	0.93	0.91
verb	0.91	0.90	0.91
x	0.95	0.94	0.94

Table 5.5: Performance of the LSTM based model without language features.

5.5 Analysis and Discussion

We perform manual analysis on different categories as followings: (i) pipeline system (e.g. V2), (ii) stacking system (e.g. S2) and (iii) LSTM-based system (e.g. word + Character-level LSTM with language features and `fasttext` skip-gram embedding and with language features).

5.5.1 Statistical Significance Testing

For statistical significance testing we use two-sided bootstrap re-sampling (Efron, 1979) by implementing the pseudo-code of Graham et al. (2014). We find that the small improvement of V2 over V1 is statistically significant. Among other systems, we find that LSTM and stacked systems are significantly better than the monolingual tagger combinations (V1 and V2) and the improvement of LSTM over S2 is also statistically significant.

5.5.2 Effect of LID as Pre-processing module

In SVM-based LID, the most frequent error category is the confusion of Hindi words as Bengali words. We believe that the reason behind this is the small number of Hindi tokens in our training data. Most of these errors occur for tokens which are lexically identical in Hindi and Bengali, e.g. *'na'*, *'chup'*, *'sale'* and *'toh'*. To quantify the error propagation from SVM language prediction we repeat the experiments of V1, V2 and S2 with the gold language labels and observe that the performance of each system is increased (Table 5.6). Table 5.6 shows the increase of performance of these systems with original language labels over SVM classifiers predicted language labels. We observe that original language labels boost the accuracy of each system in this experiment. As for the LSTM based system, we find that language information is useful when performing POS tagging in code-mixed data, the LSTM based network improves 1.11% after considering the language features.

Systems	Gold LID	SVM LID
V1	72.09	71.12
V2	72.07	71.27
S2	88.92	87.59

Table 5.6: POS tagging accuracy of V1, V2 and S2 with gold-level language labels and SVM-based language labels.

5.5.3 Stacked vs Pipeline Systems

A reason for the poor accuracy of V1 and V2 is the difference between training and test data. The TreeTaggers are trained on monolingual non-romanised formal content while the test data is romanised code-mixed content. Secondly, error propagation through transliteration and LID might increase the error rate in this method. The accuracy of these systems improved (12.98% for V1 and 12.97% for V2) when we engage these systems in stacking using in-domain training data (see stacked-V1 and stacked-V2 in Table 5.1). We find that choosing the tagger(s) based on LID does not help in stacking approaches (e.g. stacked-V1 and stacked-V2), but using all taggers to generate features for the stacked classifier results in higher accuracy (e.g. S1 and S2). We find that a stacked approach (e.g. S2) trained with monolingual and code-mixed training data outperforms the other approaches of Table 5.1. This method (S2) follows the approach of Solorio and Liu (2008b), but includes three additional features: transliterated words, character n-gram and language prediction of our SVM language classifier.

5.5.4 Monolingual vs Code-mixed Sentences

We test the accuracy on code-mixed sentences and on monolingual sentences. V2 achieves 70.49% accuracy on code-mixed sentences and 72.20% on monolingual sentences. S2 achieves 83.42% on code-mixed sentences and 86.23% on the monolingual sentences. LSTM achieves 92.06% in code-mixed and 92.95% on monolingual sentences. All these systems perform better for monolingual sentences than their performance on code-mixed sentences. This result supports the hypothesis that per-

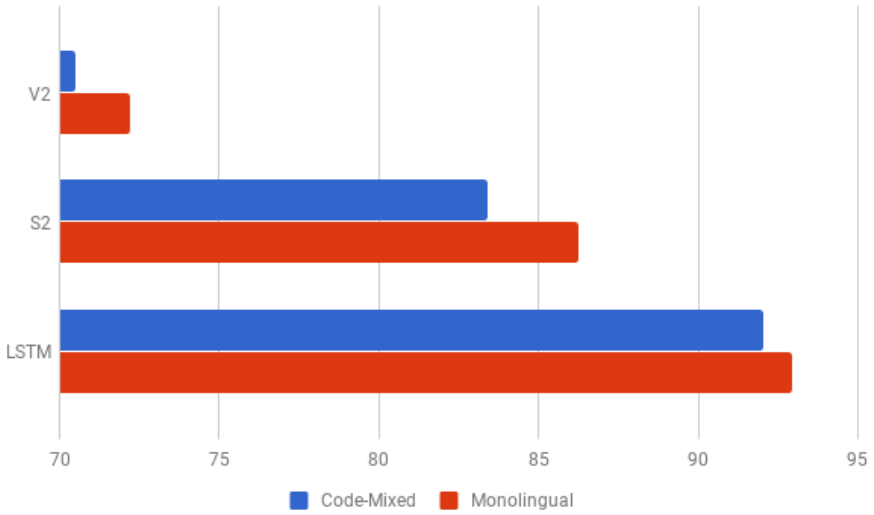

Figure 5.4: Performance of different systems on monolingual and code-mixed sentences where V2 = pipeline system, S2 = stacked system and LSTM = LSTM-based system.

forming POS tagging is harder on code-mixed sentences than it is on monolingual sentences. The performance of different systems is depicted in Figure 5.4.

5.5.5 Code-mixing Points

We also observe that POS tagger accuracy suffers at the code-mixed points. We consider a token as a code-mixed point (token-0) if the language of the token has been changed compared to the language of the previous token (named enity, acronym and, universal tokens are not included). Figure 5.5 shows the result of our analysis, where +1 means one token to right of a code-mixed point and -1 means one token to the left. It can be seen that all tested methods perform poorly at code-mixed points. Performance of these systems increases as the distance to code-mixed points rises. LSTM with `fasttext` skip-gram embeddings with language features achieves highest accuracy in code-mixed points and surroundings than that of the other models. The average word-level accuracy of LSTM-based system at code-mixed point is 90.06%, at previous token the accuracy is 95.70%, at the next token the accuracy is 92.06% It can bee seen from Figure 5.5 that the LSTM based system achieved better accuracy than other systems. We also find that 20.69% code-mixed

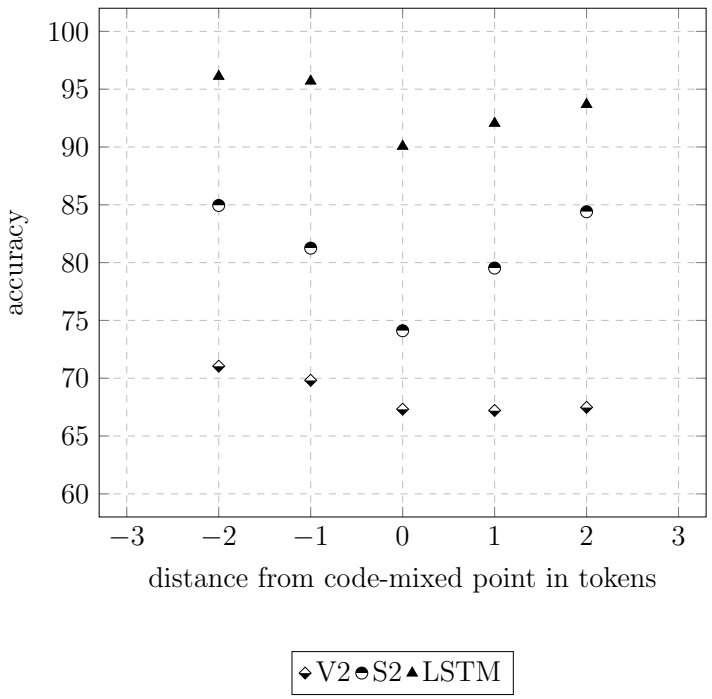

Figure 5.5: POS accuracy at code-mixed points and surroundings.

points consist of ambiguous tokens which might be a reason for the low performance.

5.5.6 Error Categories

The most common error pattern produced by all systems is adj-noun, i.e. English adjectives that are classified as NOUN. The number of these errors decreases with the better performing models, as expected. We observe that most of the chat-specific tokens (e.g. emoticons) are misclassified by V2. This system is trained with formal content. We also observe that LSTM-based system identifies correctly the sequence of code-mixed nouns while other systems suffers on that. Another prominent error category is verb-noun, i.e. verbs are classified as nouns. This error occured most of time for Bengali verbs. For example:

... kora jaina ... vaba jaina ...

Some of the verb-noun type errors occurred when two consecutive Bengali verbs are present (in the above example). We also observe that, when such patterns appeared at the start of a sentence (social media content is well-equipped with non-grammatical usage) all of our systems mis-classified such patterns. Among other

Labels	punct	adj	adp	adv	conj	det	noun	num	pron	prt	verb	x
punct	3958	0	0	0	0	0	0	1	0	0	0	0
adj	0	1597	4	8	5	7	172	2	9	1	29	2
adp	1	2	1032	8	19	33	46	1	52	24	37	0
adv	0	19	2	602	7	2	39	0	4	3	23	0
conj	0	6	7	5	615	0	17	0	11	15	10	0
det	0	17	8	5	6	748	12	1	12	16	8	1
noun	1	109	21	27	21	33	7252	2	86	28	143	28
num	0	5	0	0	0	0	1	140	1	1	0	0
pron	0	8	22	1	34	16	55	1	2112	36	21	0
prt	0	3	5	2	8	2	17	2	12	1169	14	0
verb	0	22	62	7	7	3	187	0	26	22	3748	2
x	1	1	0	2	0	0	18	0	1	1	1	562

Table 5.7: Confusion Matrix for the LSTM with `fasttext` skip-gram embeddings with language features.

errors, another prominent category is noun-verb, where nouns are mis-classified as verbs. For example:

> ... gach chas korbi ...

Here, 'chas' which originally means 'cultivation' is lexically similar to a Bengali word 'chas' which means 'want or ask'. For such ambiguous words all of our system performed poorly. Word ambiguity is in fact a major challenge when handling code-mixed social media content. The confusion matrix of our best performing system is described in Table 5.7.

5.6 Conclusion

We have performed POS tagging with trilingual code-mixed and romanised social media content and explored three different approaches: (i) SVM and CRF, (ii) pipeline, (iii) stacking and (iv) LSTM with `fasttext` skip-gram embedding. We also observe that the POS tagging accuracy is affected at code-mixed points for all systems. We find that a stacking approach (S2) that uses the combined feature set outperform monolingual tagger-based systems. We also observe that the performance of a pipeline system heavily depends on previous modules (e.g LID). The best pipeline approach achieves the lowest accuracy, even if it lags behind from the single

classifier based baseline systems (e.g. SVM and CRF). Considering these facts we answer the research questions described in Section 5.3:

- An SVM or a CRF with useful features (e.g. character n-grams, length, presence in dictionary) can achieve a reasonable accuracy (over 80%). Moreover, these systems can outperform monolingual tagger-based pipeline methods (71%).

- According to our experiments a stacking approach is better than a pipeline approach when dealing with code-mixed social media content. We find that when SVM is used on top the pipeline systems, it increases the performance drastically.

- The LSTM-based system achieves the highest accuracy which indicates that without using monolingual taggers it is possible to achieve state-of-the art accuracy with neural word and character embedding and LSTMs.

- Language information can help a POS tagging system. Section 5.5.2 shows that the use of gold level language tags improves the performance of monolingual tagger combinations and the stacking approach. For LSTM based systems the language information is also helpful. This is shown in Table 5.4 and Table 5.5. It can be observed that for most of the POS labels the F1 scores have been improved after the addition of language information.

In the next chapter, we perform LID and POS tagging jointly. The motivation behind that is to see whether the joint learning approach outperforms the individual tagging approach, which is a key research question of this thesis.

Chapter 6

Multitask Learning

Joint modelling or multi task learning (MTL) is an interesting topic in NLP. It has been investigated in the last decade using graphical models to perform multiple NLP tasks, for example, word segmentation and POS tagging (Zhang and Clark, 2008), noun phrase chunking (Sutton et al., 2007), name entity and relation extraction (Yu and Lam, 2010). However, in recent years, the use of neural models has gained popularity. Unlike graphical approaches (e.g. dynamic or factorial CRF and HMMs), there is no need to extract handcrafted features, using word embeddings, researchers have explored a number of neural MTL techniques (Collobert and Weston, 2008; Hatori et al., 2012; Søgaard and Goldberg, 2016) that can perform multiple tasks together with encouraging efficiency and accuracy.

Recent research work related to code-mixing are focused on performing different NLP tasks, for example, LID (Chittaranjan et al., 2014; Barman et al., 2014b; Shirvani et al., 2016; Jaech et al., 2016), POS Tagging (Jamatia et al., 2015; Sequiera et al., 2015a; Ghosh et al., 2016; Pimpale and Patel, 2016; Gupta et al., 2017), NER (Banerjee et al., 2016), sentiment analysis (Patra et al., 2018) but exploration for a unified approach that models a number of tasks together is not yet well investigated. Code-mixed content presents a great opportunity to test MTL approaches because NLP tasks are interdependent for code-mixed data. For example, let us consider the following snippet taken from the English-Hindi-Bengali code-mixed data:

Example 6.0.1. *...*

Mdridul/ ***bn/noun/b-per***

kar/ ***bn/noun/i-per***

tumi/ ***bn/pron/other***

ok/ ***bn/pron/other***

die/ ***bn/verb/other***

dio/ ***bn/verb/other***

...

Each line in the above example has the following format: the first token is the word, the second token is the language of the word, the third token is the POS tag and the last token is the name entity tag. Tokens are separated using '/' character. The word 'ok' can be considered as a English word but it is a Bengali word which means 'him'. For a POS tagger that attempts to perform tagging, language information is beneficial for this word. This is also true for the word 'die', which is a Bengali word and it means 'give'. Similarly, for a named entity recognition (NER) system, language and POS information is also beneficial while performing NER in code-mixed social media content. For example, let us consider the first two words of this snippet. Here, 'Mridul' is the first name and 'kar' is the surname but in Bengali 'kar' can also mean 'whose' or 'to do' depending on the context. To tag the name entity correctly, the POS information of the word 'kar' is necessary.

As discussed in Chapter 5, separate taggers are often arranged in a pipeline (Vyas et al., 2014), where output from the lower level tagger (e.g. language tagger) is either treated as input to the higher level tagger or decisions are made according to the lower label tag to maintain the data flow. This approach is risky because any error from the lower level tagger can flow through the pipeline and affect the performance of the higher level tagger. Further, a joint tagging scheme with label concatenation (Sequiera et al., 2015a) (e.g. labels of different tasks are concatenated to obtain a single label) increases the label sparsity and the computational complexity of a model. Another approach is stacking (Solorio and Liu, 2008a; Barman et al., 2016),

where a final task is accomplished by a classifier using the features obtained from lower level taggers. This approach is good when the number of tasks is small (e.g. LID, POS tagging), however, when the number of tasks is large, a stacking approach requires significant resources to train lower level taggers. To cope with the situation MTL is necessary so that multiple task can be learned simultaneously through a single model.

6.1 Chapter Organization

In this chapter, we investigate the use of a graphical model (FCRF) with handcrafted features and three neural MTL approaches to model LID and POS tagging jointly. This chapter is organized in the following way: Section 6.2 describes related work in MTL. Section 6.3 describes the research questions. The experiments and analysis are presented in Section 6.4 and 6.5. Finally we conclude in Section 6.6 by answering the research questions and other findings.

6.2 Background

Joint modelling or multi-task learning (MTL) is a well researched area of NLP and machine learning. Before the boom of neural NLP, MTL was dominated by the use of probabilistic graphical models (McCallum et al., 2003; Duh, 2005; Sutton et al., 2007; Li et al., 2011; Wang and Kan, 2013). The use of dynamic CRFs and factorial hidden Markov (HMM) models was very popular. Tasks were mainly monolingual and closely related. For example, the task of POS tagging and chunking was investigated by McCallum et al. (2003) using dynamic CRFs. Duh (2005) used factorial HMM to performed POS tagging and noun phrase chunking. Wang and Kan (2013) perform word segmentation and POS tagging using factorial CRFs. The use of such graphical models requires hand-crafted feature extraction. When the number of tasks is large the complexity of such a model grows exponentially.

During inference, the computation of marginal probabilities is also expensive in terms of computational resources for such models.

Recently, several neural MTL approach have been proposed to perform multiple tasks, for example, POS tagging, language modeling and chunking (Godwin et al., 2016; Søgaard and Goldberg, 2016), joint training on multiple treebanks (Guo et al., 2016), learning multiple semantic dependency graphs, (Peng et al., 2017; Fan et al., 2017), learning syntactic and discourse parsing together (Zhao and Huang, 2017). An important study of deep learning based MTL is proposed by Collobert and Weston (2008) where a common representation for input words were shared to solve different traditional NLP tasks such as POS tagging, chunking, name entity recognition, language modelling and semantic role labeling using convolutional neural network-based model. Hatori et al. (2012) investigated the joint training of word segmentation, POS tagging and dependency parsing. Similar studies can be found in Chinese dependency parsing (Li et al., 2014). Dong et al. (2015) performed multiple language translations from a single source language. The share one recurrent layer across multiple output layers. Søgaard and Goldberg (2016) used deep bidirectional RNNs to perform POS tagging, chunking and CCG supertagging. They used multiple RNNs layers to model multiple tasks and find that using different layers for different tasks is more effective than using the same layer in jointly learning closely-related tasks. Also successively growing a neural network (Hashimoto et al., 2016) for multiple tasks is another MTL approach can be found in literature, where multiple lower-level tasks (e.g. POS tagging, parsing) have been done through the augmentation of recurrent layers and finally to perform the higher-level task of textual entailment.

Among multilingual works, Ammar et al. (2016) investigated the dependency parsing in a multilingual environment, which also involves a multitasking scenario. Their model performs multilingual parsing by using (i) multilingual word clusters and embeddings, (ii) word-level LID information and (iii) language specific POS tags. A similar study has also been carried out by Bhat et al. (2017) where different

strategies for code-mixed parsing were evaluated using monolingual annotated data. Later, they proposed a neural stacking method to perform parsing in Hindi-English code-mixed tweets (Bhat et al., 2018). In their method again, multiple LSTM layers are used to perform multiple tasks (e.g. POS tagging, parsing). Chen et al. (2016), on the other hand, performs MTL with Mandarin-English code-switch data for speech recognition as a primary task and LID as an auxiliary task. They shared a common hidden layer to the output layers of the two task. They concluded that in this setting the primary task (speech recognition) is benefited from the second task. The work of Winata et al. (2018a) is close to our work where the authors have performed language modelling and POS tagging with Chinese-English code-mixed data using two LSTM layers. They use word-vectors and POS annotation from Penn Treebank as an input to their model. The output from the POS LSTM goes to the POS output layer and the outputs of the two LSTMs are summed up and go through a softmax layer which is designed for language modelling.

Before going to the next section let us discuss some important aspects of neural MTL. These aspects are mainly architecture related. Based on these concepts we formulate our research questions in the next section.

- **Sharing Common Representation:** Sharing a common representation across multiple tasks is often known as hard parameter sharing (Caruna, 1993; Baxter, 1997). In this settings the model is forced to learn a common set of features that are useful for all the tasks and mitigates the risk of over-fitting for a particular task. This approach is adapted by many researchers (Søgaard and Goldberg, 2016; Hashimoto et al., 2016). Figure 6.1 is a high level diagram of this architecture:

- **Multiple Layers for Multiple Tasks:** In this architecture, each task has its own recurrent layer with its own parameters. Sometimes the distance between the parameters of different task models is regularized to have similar parameters (Ruder et al., 2017). This model is investigated by Hashimoto

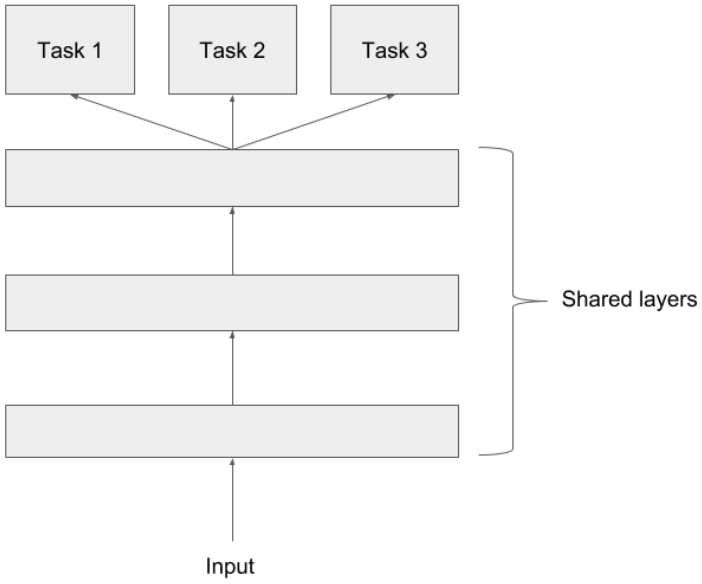

Figure 6.1: Sharing a common representation across multiple tasks. This approach is also known as hard parameter sharing.

et al. (2016) and Bhat et al. (2018). Figure 6.2 is a high level diagram of this architecture. Figure 6.3 depicts the architecture of Hashimoto et al. (2016) where multiple tasks have been performed using multiple LSTM layers. Figure 6.4 depicts the architecture of Bhat et al. (2018) POS tagging and parsing have been performed using multiple LSTM layers.

- **Output Propagation:** In the previous architecture (Figure 6.2), it can be observed that the output of the first task has been used as feature to the second task. This is known as output propagation. This allows back-propagation from one task into another task's model. Weight updates to a lower level task act as regularization on the model of a higher level task (Zhang and Weiss, 2016). It can be organized in many ways. For example, Hashimoto et al. (2016) use the the outputs of the lower-level task to be appended with the inputs of higher-level task (Figure 6.3). On the other hand, Bhat et al. (2018) use the word representation obtained from POS LSTM layers as input to the parsing LSTM layer.

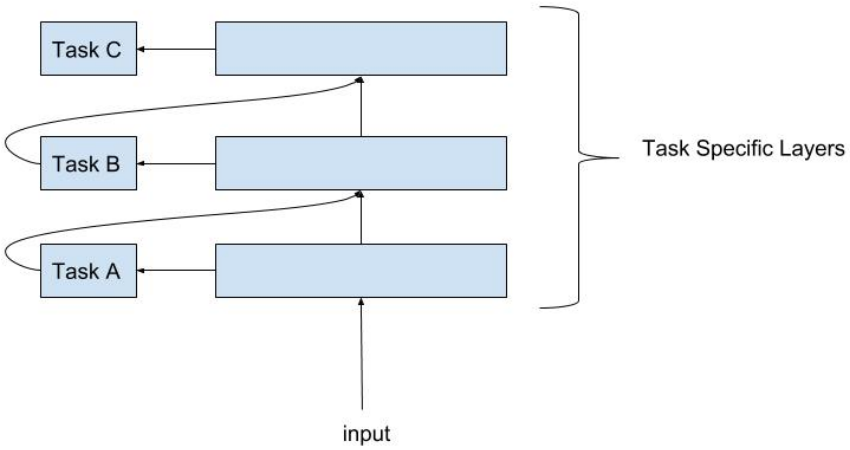

Figure 6.2: Multiple Layers for Multiple Tasks: A high-level architecture.

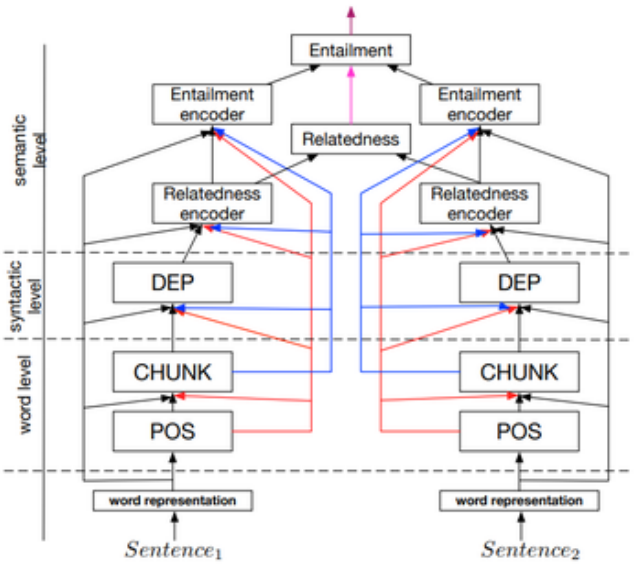

Figure 6.3: Neural architecture of Hashimoto et al. (2016). This performs multiple lower-level tasks to complete a higher-level task where POS = POS tagging DEP = dependency parsing and CHUNK = chunking. These tasks are accomplished by the use of task-specific separate LSTMs. Picture credit (Hashimoto et al., 2016).

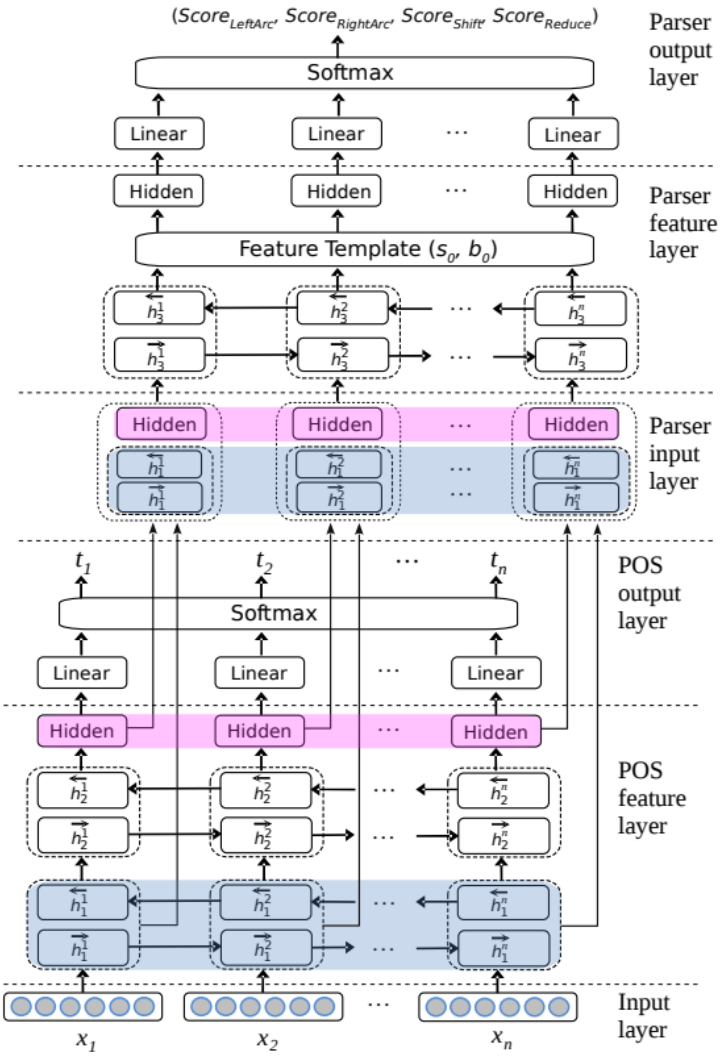

Figure 6.4: Neural architecture of Bhat et al. (2018). This performs POS tagging and parsing. This tasks are accomplished by the use of task-specific separate LSTMs. Picture credit (Bhat et al., 2018).

6.3 Research Questions

Multitask learning for code-mixed social media content is relatively less explored area. In our work, we focus mainly on neural MTL methods but we also implement a FCRF based baseline to perform LID and POS tagging. When more than one word-level task can be performed (e.g. LID and POS tagging) for code-mixed social media data, it is interesting to investigate the following questions:

- How well does a graphical model (baseline), Factorial CRF (FCRF), with handcrafted features and information from monolingual taggers handle the joint modelling of LID and POS with code-mixed data?
- Can a neural MTL outperform the FCRF with handcrafted features?
- Does joint learning of LID and POS leads to better system i.e. Can a MTL-based neural network outperform individual neural tagging accuracy (e.g. neural LID and neural POS)?
- In the recent literature, multiple recurrent layers have been assigned to perform multiple tasks (Hashimoto et al., 2016; Bhat et al., 2018). Eventually, increasing the number of recurrent layers increases the number of model parameters and training time. On the other hand, when dealing with closely related tasks, output propagation and sharing a common representation leads to good performance (Godwin et al., 2016). In this context we want to investigate - can we use output propagation and shared representation without using multiple recurrent layers to perform MTL?

6.4 Experiments

We divide the experiments into two parts. We implement a baseline system for LID and POS tagging using FCRF in Section 6.4.1. In Section 6.4.2 we implement

Systems	LID Features	TreeTagger Features	LID Acc.	POS Acc.
F1	Y	N	89.37	81.77
F2	N	Y	90.60	85.28
F3	Y	Y	**92.49**	**85.64**

Table 6.1: Performance of FCRF in LID and POS tagging with different features sets. Reported results are average cross-validation accuracy.

three neural MTL systems to perform the same tasks. We perform five fold cross-validation with the data and report average cross-validation accuracy.

6.4.1 Factorial Conditional Random Fields (FCRF)

We jointly model the two tasks (e.g. language identification and POS tagging) using a 2-level factorial CRF (FCRF) (Sutton et al., 2007). We use *romanised code-mixed training data* to train the model. In linear-chain CRF, there is only one input level ($x = x_{1:T}$) and one output level ($y = y_{1:T}$). The conditional probability in a linear-chain CRF is expressed by the following equation:

$$p(y|x) = \frac{1}{z(x)} \prod_{t=1}^{T} \psi_t(y_t, y_{t-1}, x_t) \tag{6.1}$$

where, ψ_t represents clique[1] potential functions and is expressed by the following:

$$\psi_t(y_t, y_{t-1}, x_t) = exp \sum_{k=1}^{K} \lambda_k f_k(y_t, y_{t-1}, x_t). \tag{6.2}$$

Here, K is the number of feature functions (f_k). The denominator $z(x)$ is the partition function, which is the sum over all 'y's and it is expressed by the following equation:

$$z(x) = \sum_{y} \prod_{t=1}^{T} \psi_t(y_{t,l}, y_{t-1,l}, x_t) \tag{6.3}$$

A factorial CRF combines multiple linear-chain CRFs, one for each output level. Unlike linear-chain CRFs, an FCRF deals with a vector of labels. In our case, the

[1] A clique in an undirected graph is formed with two vertices if there exists an edge connection between them.

vector contains two labels, a language label ($y^1 = y^1_{1:T}$) and a POS label ($y^2 = y^2_{1:T}$). The inputs ($x = x_{1:T}$) are shared among these output labels (e.g. $y^1_{1:T}$ and $y^2_{1:T}$) and the output labels also have interconnections (e.g. y^1_i and y^2_i $\forall i = 1, 2, ..., T$). The conditional probability is expressed by the following equation:

$$p(y|x) = \frac{1}{z(x)} \prod_{t=1}^{T} \prod_{l=1}^{L} \psi_t(y_{t,l}, y_{t-1,l}, x_t) \varphi_t(y_{t,l}, y_{t,l+1}, x_t) \tag{6.4}$$

where L is the number of levels (in our case $L = 2$), ψ_t represents transitions in each level (e.g. y^1_1 to y^1_2) and φ_t represents contemporal connections between two levels (e.g. y^1_1 to y^2_1). The denominator $z(x)$ is the partition function. We implement this FCRF using the GRMM toolkit We use three different feature sets in our experiments.

Feature Name	**Features Examples (with *value* = 1)**
G (char-n-gram)	a, m, a, r, am, ma, ar, ama, mar, amar
D (Dictionary)	<dict-train-bn>
L (Length ranges)	<4-6>
C (Capitalization)	-
P_1N_1 (Context)	<p1-je>,<n1-prothom>

Table 6.2: LID features generated for a word '*amar*' which is a part of a text fragment: '*je amar prothom*'.

1. **LID Feature Set (F1):** We keep most of the features (e.g. character n-grams, capitalisation information, length, presence in dictionaries, previous and next word) the same as for the LID experiments. Table 6.2 describes these features.

2. **TreeTagger Feature Set (F2):** The features obtained from the TreeTagger boost the accuracy of stacked systems, so we decided to include these features in this experiment. We use romanised and transliterated words, prediction from three TreeTaggers, confidence scores and the lemmas as features. This feature set is the exact feature set which is used by system S1.

3. **Combined Feature Set (F3):** This feature set is the combination of F1 and

F2. F3 can be obtained when both types of data (i.e. romanised code-mixed and non-romanised monolingual) are available.

In cross-validation we find that average language tagging accuracy is 89.37% and average POS tagging accuracy of the F1 feature set is 81.77%. After adding TreeTagger features, system F2 outperforms system F1, with 90.60% LID accuracy and 85.28% POS tagging accuracy. Finally, the combination of the previous two feature sets (F3) achieves 92.49% accuracy in LID and 85.64% in POS tagging (last row of Table 6.1).

6.4.2 Long Short Term Memory (LSTM)

We build our model on top of the word + character level LSTM network that has been used in LID and POS tagging and achieved the highest word-level accuracy. We employ three approaches: (1) joint labelling approach, (2) cascaded labelling approach and (3) multi-level labelling approach. We perform two tasks jointly, LID and POS tagging. This model considers words and the characters of a words as input to the network. The details of the model can be found in Chapter 4. In this experiment we either modified the output layers or added extra task-specific LSTM layers to achieve an MTL settings. As loss function we used categorical cross entropy for each task. The total loss is calculated by summing over all the individual losses. As features to the LSTM network, we use word embeddings and character embeddings from `fasttext` skip-gram (Joulin et al., 2016; Bojanowski et al., 2016) that have been used in our LID experiments (described in in Chapter 4). Dropout (drop out rate 0.5) is used in different parts of our architectures. Dropout is applied after each embedding layer, after the bi-LSTM and before each output layer of the network. As prepossessing steps, maximum word length is set to 30 characters, the maximum sentence length (i.e. the temporal dimension of LSTM) is set to 78 words. All words having less than 30 characters are padded with a special token to achieve the same length for all words, a similar thing is done with sentences to achieve same

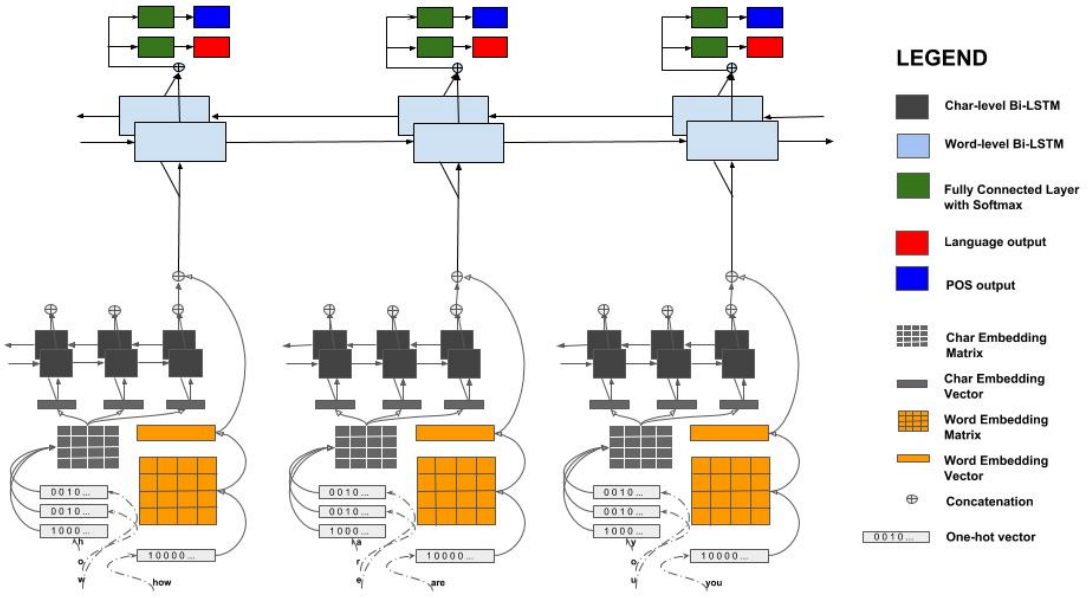

Figure 6.5: Joint Labelling Approach: Model architecture unrolled across three time steps.

sentence length over the data.

$$h_{1:T}^{\tilde{word}} = BiLSTM(x_{1:T}, c_{1:M}^{x_1}, c_{1:M}^{x_2}, ..., c_{1:M}^{x_T}) \tag{6.5}$$

For simplicity, let us define a function $BiLSTM(.)$, which is in fact, the word + character-level LSTM that has been used in LID and POS tagging. This model takes an input sequences of words ($w = w_1, ..., w_T$) and the characters of each word w_t, $c_{1:M}^{w_t} = \{c_1^{w_t}, c_2^{w_t}, ..., c_m^{w_t}, ..., c_M^{w_t}\}$, where M is maximum word length in our training data. This generates a representation, $h_{1:T}^{\tilde{word}}$ for the whole sequence of words (Equation 6.5). For MTL we use this representation ($h_{1:T}^{\tilde{word}}$) in following three ways:

6.4.2.1 Joint Labelling Approach

In this approach we use the output of the word + char-level LSTM to predict multiple tasks jointly. This method follows the hard-parameter sharing technique of Caruna (1993). This is a well studied method in NLP (Søgaard and Goldberg, 2016) and is used to reduce the risk of over-fitting. The architecture of this model is shown

Models	LID	POS
FCRF	92.49	85.64
Word+ Chracter-level LSTM	97.98	-
Word+ Chracter-level LSTM + Language Ferature	-	92.71
Joint Labelling Approach	96.88	91.87
Cascaded Approach	99.51	97.82
Multi-level Approach	99.53	97.99

Table 6.3: Performance of MTL approaches and individual taggers.

in Figure 6.5. In this method, a single hidden representation of a sequence is used to predict multiple tasks at a time, i.e. a common representation is shared across multiple output layers. For a number of tasks, the model is forced to learn features that are useful for all tasks. Thus the representation is more generalized and avoids the risk of over-fitting to a particular task. In our model we use the output of the $BiLSTM(.)$ (see Equation 6.5) to be connected to all output layers. Following are the equations for the multiple output layers:

$$
\begin{aligned}
y^1_{1:T} &= softmax(W_{y^1} \tilde{h^{word}_{1:T}} + b_{y^1}) \\
y^2_{1:T} &= softmax(W_{y^2} \tilde{h^{word}_{1:T}} + b_{y^2}) \\
&... \\
y^K_{1:T} &= softmax(W_{y^K} \tilde{h^{word}_{1:T}} + b_{y^K})
\end{aligned}
\tag{6.6}
$$

where, $y^1_{1:T}, y^2_{1:T}, ...y^K_{1:T}$ are the labels for K number of word-level tasks and $b_{y^1}, b_{y^2}, ..., b_{y^K}$ are task-specific bias vectors. Applying this approach we achieve 96.88% in LID and 91.87% in POS Tagging (Table 6.3).

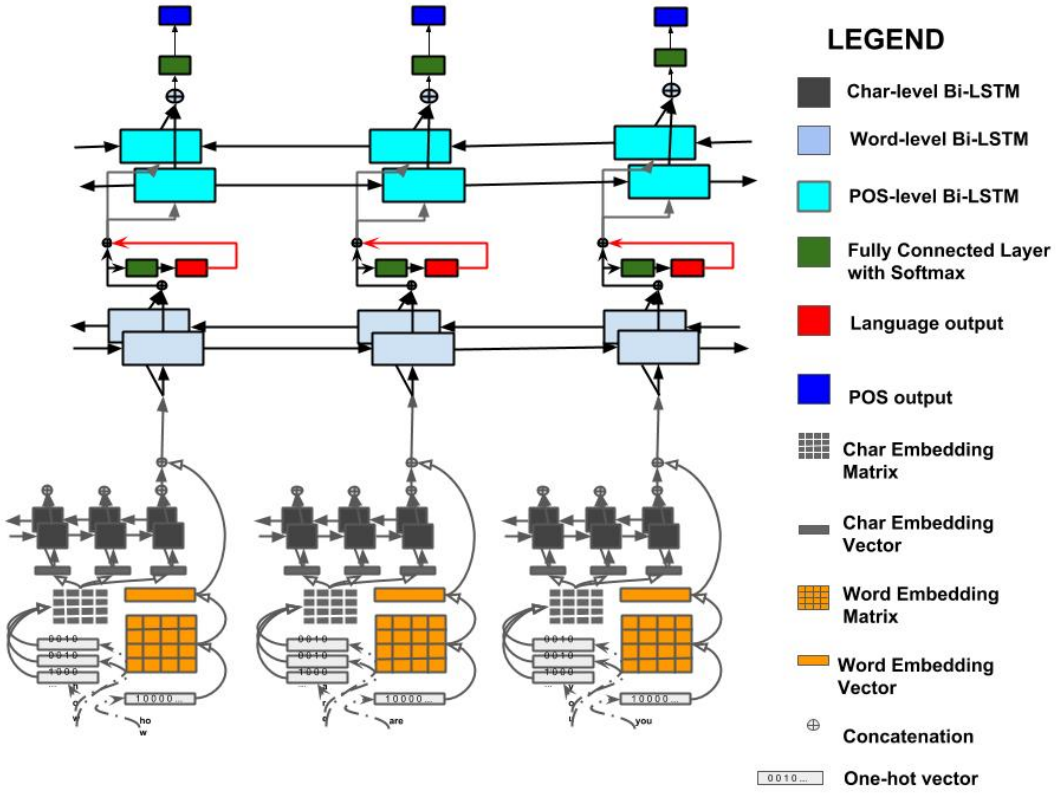

Figure 6.6: Multi-level Approach: Model architecture unrolled across three time steps.

6.4.2.2 Multi-level Approach

In this method of multitasking, task-specific layers are built like stacking for a number of related task. Hashimoto et al. (2016) proposed a hierarchical model where low level tasks, like POS tagging, chunking, dependency parsing are used to compute textual entailment of two sentences. The layers share a common representation along with the outputs of the previous layers as input to a layer. They used multiple RNN layers to perform multiple tasks, e.g. POS tagging, chunking and CCG super tagging, where three different RNN layers have been used to perform three different tasks. In this approach, stacking a recurrent model on top of another increases the number of parameters and the computational resources drastically. In our experiment we use this approach in the following way:

$$
\begin{aligned}
y^1_{1:T} &= softmax_{1:T}(W_{y^1} h^{\tilde{word}}_{1:T} + b_{y^1}) \\
h^{\tilde{y^2}}_{1:T} &= BiLSTM_{y^2}(h^{\tilde{word}}_{1:T} \oplus y^1_{1:T}) \\
y^2_{1:T} &= softmax_{1:T}(W_{y^2} h^{\tilde{y^2}}_{1:T} + b_{y^2}) \\
&... \\
h^{\tilde{y^K}}_{1:T} &= BiLSTM_{y^K}(h^{\tilde{word}}_{1:T} \oplus y^1_{1:T} \oplus y^2_{1:T} \oplus ... \oplus y^{K-1}_{1:T}) \\
y^K_{1:T} &= softmax_{1:T}(W_{y^K} h^{\tilde{y^k}}_{1:T} + b_{y^K})
\end{aligned}
\tag{6.7}
$$

Where, $y^1_t, y^2_t, ...y^K_t$ are the labels for K number of word-level tasks and $b_{y^1}, b_{y^2}, ..., b_{y^K}$ are task-specific bias vectors. $h^{\tilde{word}}_{1:T}$ is the common representation, shared across all output layers. Further, the output of the first task ($y^1_{1:T}$) is concatenated with common representation ($h^{\tilde{word}}_{1:T}$) and treated as an input to a second level LSTM which is designed for the next task. The output from the second level ($y^2_{1:T}$) LSTM gets concatenated with the first output ($y^1_{1:T}$) and the shared hidden representation

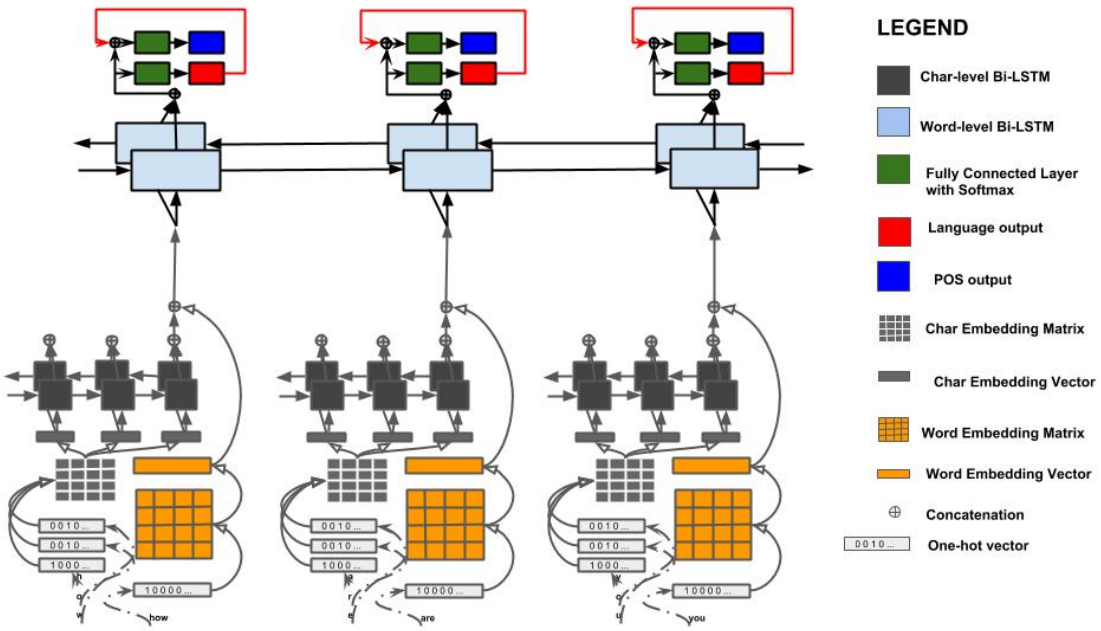

Figure 6.7: Cascaded Approach: Model architecture unrolled across three time steps.

($h_{1:T}^{\tilde{word}}$) and treated as input to the third task. In this way, we keep the original representation learned by LSTM ($h_{1:T}^{\tilde{word}}$) as a common factor and pass the low-level task outputs to the high-level task through a separate LSTM layers. This method resulted in highest word-level accuracy in LID (99.53%) and POS tagging (97.82%) (Table 6.3). The architecture of this model is shown in Figure 6.6.

6.4.2.3 Cascaded Approach

Sharing a common representation (Caruna, 1993; Baxter, 1997; Hashimoto et al., 2016; Søgaard and Goldberg, 2016) across layers has shown its effectiveness in many NLP tasks. On the other hand, the information passing across related task is also necessary (Hashimoto et al., 2016) when dealing with closely related task. However, the stacking approach of multiple LSTMs for multiple tasks increases the number of parameters and requires computational resources to train. To mitigate this, we propose a cascaded approach. In this approach we do not use multiple LSTMs to model multiple tasks, rather at each output layer, the original representation is concatenated with the previous layer outputs. In this way we maintain the output

propagation across layers and hard-parameter sharing. Formally, any label $y_t^i \in \{y_t^1, y_t^2, ..., y_t^K\}$ at time step t, where K is the number of total tasks we define y_t^i as following:

$$y_t^1 = softmax(W_{y^1} h_t^{\tilde{word}} + b_{y^1})$$
$$y_t^2 = softmax(W_{y^2} [h_t^{\tilde{word}} \oplus y_t^1] + b_{y^2})$$
$$...$$
$$y_t^K = softmax(W_{y^K} [h_t^{\tilde{word}} \oplus y_t^1 \oplus y_t^2 ... \oplus y_t^{K-1}] + b_{y^K})$$

where $\oplus$ is the concatenation operation and $h_t^{\tilde{word}}$ is the common representation shared by all layers. Here, no recurrent layers are applied to model multiple task. The original representation ($h_t^{\tilde{word}}$), which is obtained from word + character-level LSTM is shared across output layers (y^i), further, each output layer (y^i) shares its output to all higher-level output layers ($y^j, j > i$). At each output layer, all previous task outputs are concatenated with the original representation ($h_t^{\tilde{word}}$) and pass through a fully connected layer, which have a $softmax(.)$ activation function. We achieve 99.51% in LID and 97.82% in POS tagging using this approach. The architecture of this model is shown in Figure 6.7.

6.5 Analysis and Discussion

We observe that each neural joint modelling approach outperforms the baseline FCRF results. Moreover, two of the neural joint modelling approaches, (i) cascaded and (ii) multi-level, perform better than the joint labelling approach. We perform manual analysis of the three joint labelling approaches:

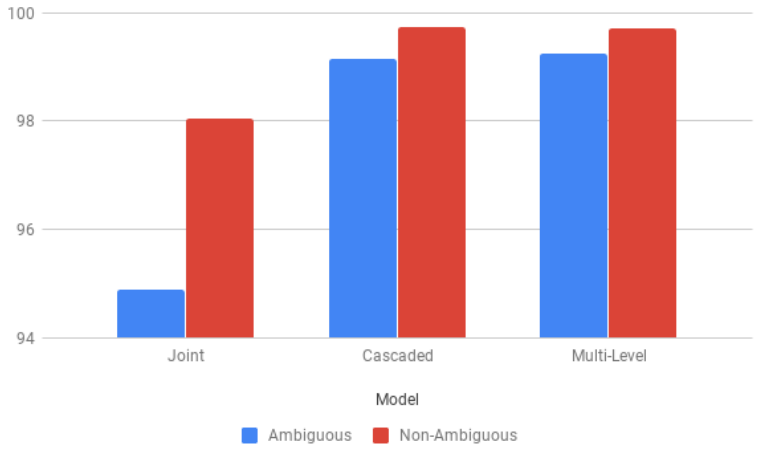

Figure 6.8: Performance of different systems for ambiguous and non-ambiguous tokens in LID. approach.

6.5.1 Statistical Significance Testing

For statistical significance testing we use two-sided bootstrap re-sampling (Efron, 1979) by implementing the pseudo-code of Graham et al. (2014). We find that among all three approaches, cascaded and multi-level approaches outperform joint labelling approach in LID and POS tagging. We find no statistical significance between the difference of the performance of the cascaded and multi-level approach. Using cascaded and multi-level approach we achieve a near perfect (99.50+%) word-level accuracy for LID in our data set. For POS tagging, it can be observed that cascaded and multi-level approach outperforms the joint approach significantly, in our data set we achieve more than 5% boost for POS tagging.

6.5.2 Ambiguous vs Non-ambiguous Words

We perform analysis for LID and POS tagging for the ambiguous and the non-ambiguous tokens. We find that all joint-modelling approaches perform relatively less well for ambiguous tokens in LID and POS tagging. Figure 6.8 shows the performance of different systems in LID for these two categories. It can be also observed that for non-ambiguous words, the performance of the three systems is over 98%. However, for cascaded and multi-level approaches the difference in performance of

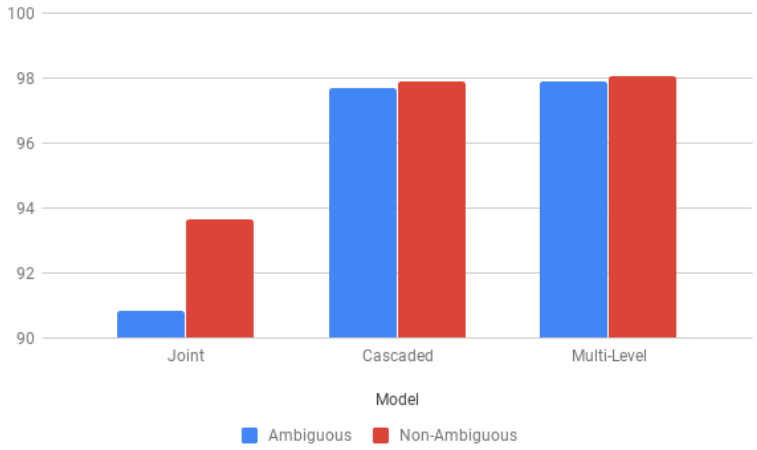

Figure 6.9: Performance of different systems for ambiguous and non-ambiguous tokens in POS tagging.

these two categories is very small, for cascaded it is (0.59%) and for multi-level it is (0.45%). We also evaluate the performance of these approaches for POS tagging. We observe a similar pattern as for LID, see Figure 6.9. Cascaded and muti-level approach outperform the joint labelling approach in POS tagging, both for ambiguous and non-ambiguous words. Further, the performance difference between cascaded and multi-level approach is much less for ambiguous and non-ambiguous words. These two approaches even outperform individual LID and individual POS tagging systems according to this metric. This indicates that the cascaded and the multi-level approach have learned to predict the correct tag for such ambiguous words.

6.5.3 Code-Mixing Points

A code-mixed point in a sentence refers to a token where language changes occurs (e.g. English to Bengali, Hindi to English). We consider a token as a code-mixed point (token-0) if the language of the token has been changed compared to the language of the previous token. Labels such as 'ne', 'acro', 'univ' and 'mixed' are not considered as changes of language. Figure 6.11 shows the performance of different systems for POS tagging. We observe the performance of the joint labelling approach

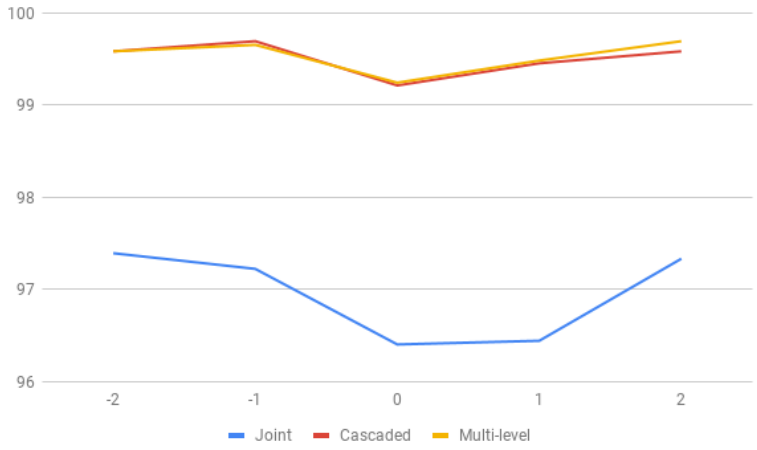

Figure 6.10: Performance of different systems at code-mixed points in LID where joint = joint approach, cascaded = cascaded approach, multi-level = multi-level approach.

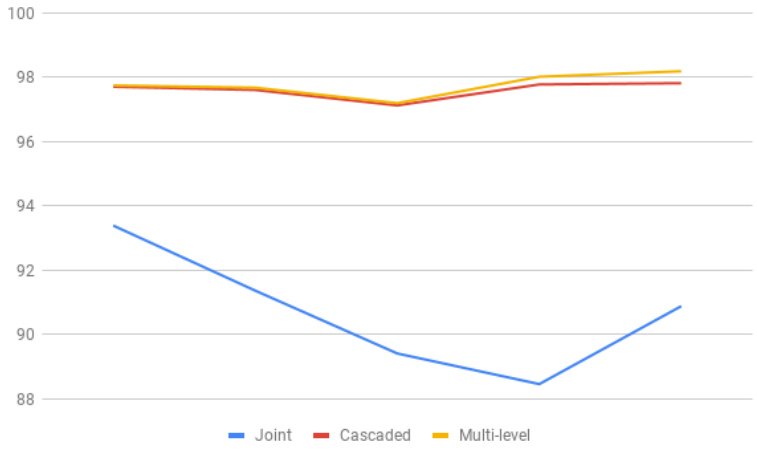

Figure 6.11: Performance of different systems at code-mixed points in POS tagging.

suffers at code-mixed point and one token after the code-mixed point. Figure 6.10 shows the result of our analysis for LID, where +1 means one token to right of a code-mixed point and -1 means one token to the left. It can be observed that the LID accuracy suffers at the code-mixed points (96%) for joint labelling approach but for cascaded and multi-level approach LID accuracy is high (over 99%). Both of these methods, cascaded and multi-level achieves similar accuracy at code-mixed point and neighbouring.

Label	Joint	Cascaded	Multi-level
en	0.96	0.99	0.99
bn	0.98	1.00	1.00
hi	0.93	1.00	1.00
mixed	0.56	0.99	0.99
univ	0.99	1.00	1.00
ne	0.79	0.98	0.98
acro	0.66	0.93	0.93

Table 6.4: Label wise F1 score for LID: Performance of three MTL approaches.

Label	Joint	Cascaded	Multi-level
PUNCT	1.00	1.00	1.00
ADJ	0.85	0.97	0.97
ADP	0.85	0.96	0.97
ADV	0.86	0.96	0.96
CONJ	0.87	0.96	0.96
DET	0.89	0.96	0.96
NOUN	0.92	0.98	0.98
NUM	0.95	0.99	0.99
PRON	0.91	0.97	0.97
PRT	0.92	0.97	0.97
VERB	0.91	0.98	0.99
X	0.94	0.97	0.97

Table 6.5: Label wise F1 score for POS: Performance of three MTL approaches.

6.5.4 Model Comparisons

In our experiments, the baseline system (FCRF) is always outperformed by the MTL approaches and the individual neural taggers. This indicates that the neural approaches (individual and MTL) are better than a feature-based graphical model. The joint labelling approach is outperformed by the individual neural LID and POS tagging systems and the other multi-tasking methods (Table 6.3). Which indicates that learning common features through hard parameter sharing for two different tasks (i.e. LID, POS tagging) might not always be helpful. This forces the model to learn from a common set of features that are helpful for all tasks, the common features might not be sufficient for the individual tasks. On the other land, the cascaded and multi-level approaches overcome this problem by sharing the outputs of one task to the next task. In this way, the second task always gets some extra information which can be beneficial. Moreover, through output propagation, changes in the weights of a lower level model affects the tagging decisions of a higher level model. The Multi-level approach considers multiple recurrent layers for multiple tasks, it also propagates the output of one task to the the next level recurrent layer. However, when the number of tasks is large, using multiple recurrent layers can require greater computational resources. This approach also increases the number of model parameters. The cascaded approach, on the other hand, uses a single recurrent layer with multiple output layers, each output layer propagates the output to the next output layer. The number of parameters does not increase drastically, as there are no task-specific recurrent layers in this model. Using only different output layers with output propagation and hard parameter sharing (e.g. a common representation from the recurrent layer), it is possible to achieve similar performance to a multi-level approach. It can be observed from Table 6.6 that the cascaded approach is the most efficient among all three MTL approaches, if (1) the number of parameters and (2) training time and (3) performance is considered. The difference between the performance of cascaded and multi-level approach is not significant. Using one million fewer parameters and almost three and half hours less

Data Set	Models	No. of Parameters	Training Time	LID	POS
A	Joint	2,396,709	8:51:03	96.88	91.87
	Cascaded	2,396,813	8:56:38	99.51	97.82
	Multi-Level	3,988,005	12:30:38	99.53	97.99
B	Joint	2,391,699	3:44:56	97.60	86.84
	Cascaded	2,392,217	3:45:52	99.56	97.36
	Multi-Level	3,995,283	4:50:01	99.60	97.58

Table 6.6: Performance of three different MTL approaches in two code-mixed data sets, where A: code-mixed data set of Barman et al. (2016) and B: code-mixed data set of Jamatia et al. (2015). Reported results are average of five-fold cross-validation accuracy.

training time, the cascaded model can perform as well as the multi-level approach.

Table 6.4 and 6.5 shows the label wise F1 scores for LID and POS tagging for these three approaches. We find that for LID the joint labelling approach is always outperformed by the other two MTL approaches for each language label. For POS tagging, we observe a similar pattern, except punctuation's the cascaded and the multi-level approaches achieves better performance for each label than that of the joint labelling approach.

We also test these approaches in a second data set which is obtained from Jamatia et al. (2015). Like our data set, this data set is annotated with language and POS tags. We find similar results, i.e. the joint labelling approach is always outperformed by the cascaded and multi-level approaches and the difference in the performance of cascaded and multi-level approaches is not statistically significant. These results are shown in Table 6.6.

6.6 Conclusion

We have performed joint labelling of LID and POS tagging for code-mixed data in this chapter. We compare a baseline approach (FRCF) and three neural MTL approaches and found that all of MTL approaches performs better than the FCRF with handcrafted features.

We also find that in MTL settings it is not always guaranteed that it will perform

better than the individual neural tagging approach. The joint labelling approach lags behind in LID and POS tagging by 1.10% and 1.16% than individual tagging approaches. This depends on the architectural decisions, cascaded and multi-level models on the other hand always outperform individual tagging accuracy.

The cascaded and multi-level approach both use hard parameter sharing and output propagation. The multi-level approach uses two different RNN layers for the two tasks. However, the cascaded approach does not use multiple task-specific recurrent layers and still it can achieve a similar accuracy to the multi-level approach. Further, it uses much fewer parameters and much less training time than the multi-level approach. Hence, it is possible to use output propagation and shared representation without using multiple task-specific recurrent layers in MTL settings.

In the next chapter we revisit the research questions of this thesis and try to answer them. Also, we discuss our observation of different systems and propose the future direction of this research in the next Chapter.

Chapter 7

Conclusions

In this thesis we explored LID and POS tagging with code-mixed data: as individual tagging systems and as joint tagging systems. We have used standard ML techniques, e.g. SVM, CRF, and FCRF with handcrafted features and explored neural methods, e.g. LSTM with fasttext Skip-Gram word embeddings. We now, revisit the research questions of Chapter 1 to summarise our findings.

> *1. Is it possible to achieve better performance than feature based ML (SVM and CRF) by using a neural approach for LID and POS tagging of code-mixed social media content?*

It is possible to outperform feature based learners (e.g. SVM and CRF) with an LSTM based system that uses fasttext Skip-Gram word and character embeddings. For LID we have used two neural systems: (1) the first one uses only word embeddings (2) and the second one is a novel LSTM based architecture that takes account of words and characters simultaneously. The second model is a combination of two LSTM-based systems. The first LSTM takes a sequence of characters (e.g. word is converted to a sequence of character) to obtain a representation for a word. This model is shared across each time step of a sentence to generate word representations on the fly. The generated word representations are concatenated with pre-trained word embeddings and used as inputs to the second level LSTM. The word representation from the second level LSTM is used to classify the words into relevant LID

tags. For LID, we find that the use of the word + character-level LSTM performs better than word-level LSTM and that significantly improves the tagging accuracy for named entities (*ne*), acronyms (*acro*) and word-level code-mixing (*mixed*). It is worth mentioning that despite having low frequencies, (*mixed* 0.26%, ne 2.59% and *acro* 0.84%), it is possible to achieve reasonable performance for these categories. Using the word + character-level LSTM model we achieve is 71% for *mixed*, 86% for *ne* and 82% for *acro* in terms of F1 scores.

2. How well can we exploit monolingual taggers to perform POS tagging for code-mixed data? Further, can a neural approach be effective?

When monolingual taggers are arranged in a pipeline and the data flow is controlled by a language identifier, the chances of error propagation are high. Further, language specific chunking might result in a chunk that is inappropriate for a monolingual tagger. On the other hand, when monolingual taggers act as feature generators in a stacking system, it is possible to outperform tagger combination-based systems. The stacking approach performs better than a graphical model (FCRF) with a similar feature set. For POS tagging, we find that the word + character-level LSTM outperforms the feature-based learners (SVM, CRF, FCRF and stacking approaches) and monolingual tagging combinations. Further, adding language information into the word + character-level LSTM improves the POS tagging accuracy for the neural method.

3. How well does a neural MTL approach perform for code-mixed content? Does MTL achieve better accuracy than individual taggers for code-mixed data?

In our experiments, the cascaded and the multi-level approach outperforms the individual neural taggers and joint labelling MTL approaches for LID and POS tagging. We find that it is not always guaranteed that an MTL system (e.g. joint labelling approach) will perform better than the individual neural tagging approach.

The joint labelling approach lags behind in LID and POS tagging by 1.10% and 1.16% than individual neural tagging approaches. In joint labelling approach the network is forced to learn some features that are common to the two tasks, it fails to learn those features which individually contribute to each task. On the other hand, the cascaded and multi-level approaches pass the output of one task to the input of another task, which leads to better representation. It should be noted that the cascaded MTL architecture is able to achieve similar performance to a multi-level MTL architecture, by using fewer parameters and less training time. Our experiments show that when dealing with code-mixed content and with multiple tasks, it is beneficial if a MTL approach (appart from a joint labelling approach) is taken. Using a cascaded MTL architecture we achieve a near perfect word-level accuracy for LID (99+%) and POS tagging (97+%). We also find that language and POS tagging performance suffers at code-mixed points for the joint labelling approach and feature-based learners. The word ambiguity caused by code-mixing affects the feature-based learners but neural MTL approaches are not so adversely affected by the ambiguities.

7.1 Use Cases

LID and POS tagging is prerequisite for many NLP applications. For example, transliteration of social media code-mixed content is highly dependent on the LID information. NLP tools that deals with social media romanised transliteration requires word-level LID. Other NLP modules, for example, sentiment analysis, is benefited by the POS information. When dealing with social media romanised content and performing such tasks (e.g. sentiment analysis), it is always good to have a multi-tasking model that can perform low-level (e.g. LID and POS) tasks efficiently than several small models in a pipeline.

7.2 Future Work

The research presented in the thesis can be made in multiple directions. Three ideas are described in the following sections that we consider as a future extension of this research:

7.2.1 Improving Performance at Code-mixed Points

During analysis we observed that the performance of the NLP tasks (e.g. LID and POS Tagging) is relatively lower at code-mixed points than the other positions. In our future work we plan to investigate and improve the approaches to mitigate this issue.

7.2.2 Unified LID and POS Tagging for Indian Languages

An approach can be taken for a unified LID and POS tagging system for code-mixed social media content in Indian languages. In future, we plan to include a number of Indian languages (language and POS annotated) to extend our work in LID and POS tagging. It will be an interesting matter to see how well individual and MTL approaches perform for LID and POS tagging when (1) the number of languages is increased (2) different POS tag sets for different languages are used and (3) same POS tag set (Petrov et al., 2011) for all languages are used.

7.2.3 LID for Similar Languages

Many of the Indian languages originated from Indo-European and Dravidian language families. Therefore, there exists similarity among languages that belong to the same language family (e.g. Hindi and Marathi). Moreover, due to the short geographical distances, some languages share similar scripts, share a lot of vocabulary and also have similar grammatical structure (e.g. Bengali, Assamese). In social media content, classifying such languages can be challenging because Romanisation is involved. When languages are very similar due to their origin, it will be

an interesting to see that how well our methods performs.

7.2.4 Word-Level MTL: Inclusion of Multiple Tasks

The effectiveness of a cascaded MTL architecture can be tested in many environments. For example, inclusion of multiple word-level tasks for code-mixed data (on top of LID and POS tagging). These may include chunking, shallow parsing, NER and word-level sentiment analysis. Also, these tasks can be arranged for multiple language pairs, for example, multiple Indian languages. We also plan to extend the framework to perform multiple sentence level tasks. For example, the outputs of multiple tasks can be concatenated to form a single representation, this representation can be used in another LSTM as input that operates on the sentence level and has multiple output layers to perform multiple tasks (e.g. sentiment analysis).

Bibliography

Alex, B. (2008). *Automatic detection of English inclusions in mixed-lingual data with an application to parsing*. PhD thesis, School of Informatics, The University of Edinburgh, Edinburgh, UK.

AlGhamdi, F., Molina, G., Diab, M., Solorio, T., Hawwari, A., Soto, V., and Hirschberg, J. (2016). Part of speech tagging for code switched data. In *Proceedings of the Second Workshop on Computational Approaches to Code Switching*, pages 98–107.

Ammar, W., Mulcaire, G., Ballesteros, M., Dyer, C., and Smith, N. A. (2016). Many languages, one parser. *arXiv preprint arXiv:1602.01595*.

Auer, P. (1984). *Bilingual conversation*. John Benjamins Publishing.

Auer, P. (2013). *Code-Switching in Conversation: Language, Interaction and Identity*. Routledge.

Baldwin, T. and Lui, M. (2010a). Language identification: The long and the short of the matter. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 229–237. Association for Computational Linguistics.

Baldwin, T. and Lui, M. (2010b). Language identification: The long and the short of the matter. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 229–237. Association for Computational Linguistics.

Banerjee, S., Chakma, K., Naskar, S. K., Das, A., Rosso, P., Bandyopadhyay, S., and Choudhury, M. (2016). Overview of the mixed script information retrieval (msir) at fire-2016. *Organization (ORG)*, 67:24.

Bar, K. and Dershowitz, N. (2014). The tel aviv university system for the code-switching workshop shared task. In *Proceedings of the First Workshop on Computational Approaches to Code Switching*, pages 139–143.

Barman, U., Das, A., Wagner, J., and Foster, J. (2014a). Code mixing: A challenge for language identification in the language of social media. In *Proceedings of the First Workshop on Computational Approaches to Code Switching. EMNLP 2014, Conference on Empirical Methods in Natural Language Processing*, pages 13–23, Doha, Qatar. Association for Computational Linguistics.

Barman, U., Wagner, J., Chrupała, G., and Foster, J. (2014b). Dcu-uvt: Word-level language classification with code-mixed data. In *Proceedings of the First Workshop on Computational Approaches to Code Switching*, pages 127–132.

Barman, U., Wagner, J., and Foster, J. (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.

Baxter, J. (1997). A bayesian/information theoretic model of learning to learn via multiple task sampling. *Machine learning*, 28(1):7–39.

Baykan, E., Henzinger, M., and Weber, I. (2008). Web page language identification based on urls. *Proceedings of the VLDB Endowment*, 1(1):176–187.

Belazi, H. M., Rubin, E. J., and Toribio, A. J. (1994). Code switching and x-bar theory: The functional head constraint. *Linguistic inquiry*, pages 221–237.

Bergsma, S., McNamee, P., Bagdouri, M., Fink, C., and Wilson, T. (2012). Language identification for creating language-specific twitter collections. In *Proceedings of the second workshop on language in social media*, pages 65–74. Association for Computational Linguistics.

Bhat, G., Choudhury, M., and Bali, K. (2016). Grammatical constraints on intra-sentential code-switching: From theories to working models. *arXiv preprint arXiv:1612.04538*.

Bhat, I. A., Bhat, R. A., Shrivastava, M., and Sharma, D. M. (2017). Joining hands: Exploiting monolingual treebanks for parsing of code-mixing data. *arXiv preprint arXiv:1703.10772*.

Bhat, I. A., Bhat, R. A., Shrivastava, M., and Sharma, D. M. (2018). Universal dependency parsing for hindi-english code-switching. *arXiv preprint arXiv:1804.05868*.

Black, E., Jelinek, F., Lafferty, J., Mercer, R., and Roukos, S. (1992). Decision tree models applied to the labeling of text with parts-of-speech. In *Speech and Natural Language: Proceedings of a Workshop Held at Harriman, New York, the Fifth*

DARPA Workshop on Speech and Natural Language, pages 117–121, San Mateo, CA, USA. Morgan Kaufmann Publishers, Inc.

Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2016). Enriching word vectors with subword information. *CoRR*, abs/1607.04606.

Brants, T. (2000). TnT: A statistical part-of-speech tagger. In *Proceedings of the Sixth Conference on Applied Natural Language Processing*, pages 224–231. Association for Computational Linguistics.

Cárdenas-Claros, M. and Isharyanti, N. (2009). Code-switching and code-mixing in internet chatting: Between'yes,"ya,'and'si'-a case study. *The Jalt Call Journal*, 5(3):67–78.

Carter, S., Weerkamp, W., and Tsagkias, M. (2013). Microblog language identification: Overcoming the limitations of short, unedited and idiomatic text. *Language Resources and Evaluation*, 47(1):195–215.

Caruna, R. (1993). Multitask learning: A knowledge-based source of inductive bias. In *Machine Learning: Proceedings of the Tenth International Conference*, pages 41–48.

Cavnar, W. B. and Trenkle, J. M. (1994). N-gram-based text categorization. In Pavlidis, T., editor, *Proceedings of SDAIR-94, Third Annual Symposium on Document Analysis and Information Retrieval*, pages 161–175.

Çetinoğlu, Ö., Schulz, S., and Vu, N. T. (2016). Challenges of computational processing of code-switching. *arXiv preprint arXiv:1610.02213*.

Chang, J. C. and Lin, C. (2014). Recurrent-neural-network for language detection on twitter code-switching corpus. *CoRR*, abs/1412.4314.

Chen, M., Pan, J., Zhao, Q., and Yan, Y. (2016). Multi-task learning in deep neural networks for mandarin-english code-mixing speech recognition. *IEICE TRANSACTIONS on Information and Systems*, 99(10):2554–2557.

Chen, X., Xu, L., Liu, Z., Sun, M., and Luan, H.-B. (2015). Joint learning of character and word embeddings. In *IJCAI*, pages 1236–1242.

Chittaranjan, G., Vyas, Y., Bali, K., and Choudhury, M. (2014). Word-level language identification using crf: Code-switching shared task report of msr india system. In *Proceedings of The First Workshop on Computational Approaches to Code Switching*, pages 73–79.

Collobert, R. and Weston, J. (2008). A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM.

Cutting, D., Kupiec, J., Pedersen, J., and Sibun, P. (1992). A practical part-of-speech tagger. In *Proceedings of the Third Conference on Applied Natural Language Processing*, pages 133–140. Association for Computational Linguistics.

Damashek, M. (1995). Gauging similarity with n-grams: Language-independent categorization of text. *Science*, 267(5199):843.

Dandapat, S., Sarkar, S., and Basu, A. (2004). A hybrid model for part-of-speech tagging and its application to Bengali. In *International Conference on Computational Intelligence*, pages 169–172.

Dandapat, S., Sarkar, S., and Basu, A. (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 Association for Computational Linguistics Companion Volume Proceedings of the Demo and Poster Sessions*, pages 221–224, Prague, Czech Republic. Association for Computational Linguistics.

Das, A. and Gambäck, B. (2014). Identifying languages at the word level in code-mixed indian social media text. In *11th International Conference on Natural Language Processing*, page 378.

Das, A. and Gambäck, B. (2015). Code-mixing in social media text: the last language identification frontier?

Das, D. and Petrov, S. (2011). Unsupervised part-of-speech tagging with bilingual graph-based projections. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 600–609. Association for Computational Linguistics.

Dewaele, J.-M. (2010). *Emotions in Multiple Languages*. Palgrave Macmillan.

Dey, A. and Fung, P. (2014). A Hindi-English code-switching corpus. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, pages 2410–2413, Reykjavik, Iceland. European Language Resources Association (ELRA).

Diab, M., Fung, P., Ghoneim, M., Hirschberg, J., and Solorio, T. (2016). Proceedings of the second workshop on computational approaches to code switching.

In *Proceedings of the Second Workshop on Computational Approaches to Code Switching*.

Dong, D., Wu, H., He, W., Yu, D., and Wang, H. (2015). Multi-task learning for multiple language translation. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, volume 1, pages 1723–1732.

Duh, K. (2005). Jointly labeling multiple sequences: A factorial hmm approach. In *Proceedings of the ACL Student Research Workshop*, pages 19–24. Association for Computational Linguistics.

Dunning, T. (1994). *Statistical identification of language*. Computing Research Laboratory, New Mexico State University.

Duong, L., Cook, P., Bird, S., and Pecina, P. (2013). Simpler unsupervised pos tagging with bilingual projections. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 634–639.

Efron, B. (1979). Bootstrap Methods: Another Look at the Jackknife. *The Annals of Statistics*, 7(1):1–26.

Ekbal, A. and Bandyopadhyay, S. (2008). Web-based Bengali news corpus for lexicon development and POS tagging. *POLIBITS, ISSN 1870*, 9044(37):20–29.

Elworthy, D. (1999). Language identification with confidence limits. *arXiv preprint cs/9907010*.

Fan, X., Monti, E., Mathias, L., and Dreyer, M. (2017). Transfer learning for neural semantic parsing. *arXiv preprint arXiv:1706.04326*.

Foster, J., Çetinoglu, Ö., Wagner, J., Le Roux, J., Hogan, S., Nivre, J., Hogan, D., Van Genabith, J., et al. (2011). #hardtoparse: POS tagging and parsing the twitterverse. In *Proceedings of the Workshop On Analyzing Microtext (AAAI 2011)*, pages 20–25.

Gambäck, B. and Das, A. (2014). On measuring the complexity of code-mixing. In *Proceedings of the 11th International Conference on Natural Language Processing, Goa, India*, pages 1–7.

Gardner-Chloros, P. (1991). Language selection and switching in strasbourg.

Ghosh, S., Ghosh, S., and Das, D. (2016). Part-of-speech tagging of code-mixed social media text. In *Proceedings of the Second Workshop on Computational Approaches to Code Switching*, pages 90–97.

Gimpel, K., Schneider, N., O'Connor, B., Das, D., Mills, D., Eisenstein, J., Heilman, M., Yogatama, D., Flanigan, J., and Smith, N. A. (2011). Part-of-speech tagging for Twitter: Annotation, features, and experiments. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers-Volume 2*, pages 42–47. Association for Computational Linguistics.

Giménez, J. and Màrquez, L. (2004). SVMTool: A general POS tagger generator based on support vector machines. In *In Proceedings of the 4th International Conference on Language Resources and Evaluation*, pages 43–46.

Godwin, J., Stenetorp, P., and Riedel, S. (2016). Deep semi-supervised learning with linguistically motivated sequence labeling task hierarchies. *arXiv preprint arXiv:1612.09113*.

Goldszmidt, M., Najork, M., and Paparizos, S. (2013). Boot-strapping language identifiers for short colloquial postings. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 95–111. Springer.

Gottron, T. and Lipka, N. (2010). A comparison of language identification approaches on short, query-style texts. In *In Advances in Information Retrieval, 32nd European Conference on IR Research (ECIR 2010)*, pages 611–614. Springer.

Graham, Y., Mathur, N., and Baldwin, T. (2014). Randomized significance tests in machine translation. In *Proceedings of the ACL 2014 Ninth Workshop on Statistical Machine Translation*, pages 266–274.

Grefenstette, G. (1995). Comparing two language identification schem es.

Grothe, L., De Luca, E. W., and Nürnberger, A. (2008). A comparative study on language identification methods. In *LREC*. Citeseer.

Guo, J., Che, W., Wang, H., and Liu, T. (2016). Exploiting multi-typed treebanks for parsing with deep multi-task learning. *arXiv preprint arXiv:1606.01161*.

Gupta, D., Tripathi, S., Ekbal, A., and Bhattacharyya, P. (2017). Smpost: Parts of speech tagger for code-mixed indic social media text. *arXiv preprint arXiv:1702.00167*.

Han, B., Cook, P., and Baldwin, T. (2012). Automatically constructing a normalisation dictionary for microblogs. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 421–432. Association for Computational Linguistics.

Hashimoto, K., Xiong, C., Tsuruoka, Y., and Socher, R. (2016). A joint many-task model: Growing a neural network for multiple nlp tasks. *arXiv preprint arXiv:1611.01587*.

Hatori, J., Matsuzaki, T., Miyao, Y., and Tsujii, J. (2012). Incremental joint approach to word segmentation, pos tagging, and dependency parsing in chinese. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*, pages 1045–1053. Association for Computational Linguistics.

Hidayat, T. (2012). An analysis of code switching used by facebookers: a case study in a social network site. Student essay for the study programme "Pendidikan Bahasa Inggris" (English Education) at STKIP Siliwangi Bandung, Indonesia, `http://publikasi.stkipsiliwangi.ac.id/files/2012/10/08220227-taofik-hidayat.pdf`.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*.

Hong, L., Convertino, G., and Chi, E. H. (2011). Language matters in twitter: A large scale study. In *ICWSM*.

Hsu, C.-W., Chang, C.-C., and Lin, C.-J. (2010). A practical guide to support vector classification. Technical report. Department of Computer Science, National Taiwan University, Taiwan, `https://www.cs.sfu.ca/people/Faculty/teaching/726/spring11/svmguide.pdf`.

Hughes, B., Baldwin, T., Bird, S., Nicholson, J., and MacKinlay, A. (2006). Reconsidering language identification for written language resources.

Jaech, A., Mulcaire, G., Ostendorf, M., and Smith, N. A. (2016). A neural model for language identification in code-switched tweets. In *Proceedings of The Second Workshop on Computational Approaches to Code Switching*, pages 60–64.

Jain, N. and Bhat, R. A. (2014). Language identification in code-switching scenario. In *Proceedings of the First Workshop on Computational Approaches to Code Switching*, pages 87–93.

Jamatia, A. and Das, A. (2014). Part-of-speech tagging system for Indian social media text on Twitter. In *Social-India 2014, First Workshop on Language Technologies for Indian Social Media Text, at the Eleventh International Conference on Natural Language Processing (ICON-2014)*, volume 2014, pages 21–28.

Jamatia, A. and Das, A. (2016). Task report: Tool contest on pos tagging for codemixed indian social media (facebook, twitter, and whatsapp) text@ icon 2016. *the proceeding of ICON 2016*.

Jamatia, A., Gambäck, B., and Das, A. (2015). Part-of-speech tagging for code-mixed english-hindi twitter and facebook chat messages. In *Proceedings of the International Conference Recent Advances in Natural Language Processing*, pages 239–248.

Johnson, S. (1993). Solving the problem of language recognition. In *Technical Report*. School of Computer Studies, University of Leeds.

Joshi, A. K. (1982). Processing of sentences with intra-sentential code-switching. In *Proceedings of the 9th conference on Computational linguistics-Volume 1*, pages 145–150. Academia Praha.

Joulin, A., Grave, E., Bojanowski, P., and Mikolov, T. (2016). Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*.

Kachru, B. B. (1976). Toward structuring code-mixing: An indian perspective.

Kachru, B. B. (1978). Code-mixing as a communicative strategy in india. *International dimensions of bilingual education*, pages 107–124.

Kachru, B. B. (1983). *The indianization of English: the English language in India*. Oxford University Press.

Kachru, B. B. (1986). *The alchemy of English: The spread, functions, and models of non-native Englishes*. University of Illinois Press.

King, B. and Abney, S. (2013). Labeling the languages of words in mixed-language documents using weakly supervised methods. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1110–1119, Atlanta, Georgia. Association for Computational Linguistics.

Kralisch, A. and Mandl, T. (2006). Barriers to information access across languages on the internet: Network and language effects. In *System Sciences, 2006.*

HICSS'06. Proceedings of the 39th Annual Hawaii International Conference on, volume 3, pages 54b–54b. IEEE.

Laboreiro, G., Bošnjak, M., Sarmento, L., Rodrigues, E. M., and Oliveira, E. (2013). Determining language variant in microblog messages. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, pages 902–907. ACM.

Lafferty, J., McCallum, A., and Pereira, F. C. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. pages 282–289.

Li, D. (2000). Cantonese-English code-switching research in Hong Kong: a Y2K review. *World Englishes*, 19(3):305–322.

Li, X., Wang, X., and Yao, L. (2011). Joint decoding for chinese word segmentation and pos tagging using character-based and word-based discriminative models. In *Asian Language Processing (IALP), 2011 International Conference on*, pages 11–14. IEEE.

Li, Z., Zhang, M., Che, W., Liu, T., and Chen, W. (2014). Joint optimization for chinese pos tagging and dependency parsing. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 22(1):274–286.

Lignos, C. and Marcus, M. (2013). Toward web-scale analysis of codeswitching. In *87th Annual Meeting of the Linguistic Society of America*.

Lui, M. and Baldwin, T. (2011). Cross-domain feature selection for language identification. In *Proceedings of 5th international joint conference on natural language processing*, pages 553–561.

Lui, M. and Baldwin, T. (2012). langid. py: An off-the-shelf language identification tool. In *Proceedings of the ACL 2012 system demonstrations*, pages 25–30. Association for Computational Linguistics.

Lui, M. and Baldwin, T. (2014). Accurate language identification of twitter messages. In *Proceedings of the 5th workshop on language analysis for social media (LASM)*, pages 17–25.

Mandal, S., Das, S. D., and Das, D. (2018). Language identification of bengali-english code-mixed data using character & phonetic based lstm models. *arXiv preprint arXiv:1803.03859*.

Mann, G. S. and McCallum, A. (2008). Generalized expectation criteria for semi-supervised learning of conditional random fields. In *Proceedings of ACL-08: HLT*, pages 870–878, Columbus, Ohio. Association for Computational Linguistics.

Mann, G. S. and McCallum, A. (2010). Generalized expectation criteria for semi-supervised learning with weakly labeled data. *The Journal of Machine Learning Research*, 11:955–984.

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

McCallum, A., Rohanimanesh, K., and Sutton, C. (2003). Dynamic conditional random fields for jointly labeling multiple sequences. In *NIPS-2003 Workshop on Syntax, Semantics and Statistics*.

McNamee, P. (2005). Language identification: a solved problem suitable for undergraduate instruction. *Journal of Computing Sciences in Colleges*, 20(3):94–101.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

Milroy, L. and Muysken, P., editors (1995). *One speaker, two languages: Cross-disciplinary perspectives on code-switching*. Cambridge University Press.

Murthy, K. N. and Kumar, G. B. (2006). Language identification from small text samples. *Journal of Quantitative Linguistics*, 13(01):57–80.

Muysken, P., Díaz, C. P., Muysken, P. C., et al. (2000). *Bilingual speech: A typology of code-mixing*, volume 11. Cambridge University Press.

Myers-Scotton, C. (1995). A lexically based model of code-switching. *One speaker, two languages: Cross-disciplinary perspectives on code-switching*, pages 233–256.

Myers-Scotton, C. (1997). *Duelling languages: Grammatical structure in codeswitching*. Oxford University Press.

Myers-Scotton, C. and Jake, J. L. (1995). Matching lemmas in a bilingual language competence and production model: Evidence from intrasentential code switching.

Nakagawa, T., Kudo, T., and Matsumoto, Y. (2001). Unknown word guessing and part-of-speech tagging using support vector machines. In *NLPRS*, pages 325–331. Citeseer.

Nguyen, D. and Doğruöz, A. S. (2013). Word level language identification in online multilingual communication. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing (EMNLP 2013)*, pages 857–862, Seattle, Washington, USA. Association for Computational Linguistics.

Owoputi, O., O'Connor, B., Dyer, C., Gimpel, K., Schneider, N., and Smith, N. A. (2013). Improved part-of-speech tagging for online conversational text with word clusters. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT 2013)*, pages 380–390, Atlanta, Georgia. Association for Computational Linguistics.

Padró, M. and Padró, L. (2004). Comparing methods for language identification. *Procesamiento del lenguaje natural*, 33.

Palanisamy, A. and Devi, S. L. (2006). HMM based POS tagger for a relatively free word order language. *Research in Computing Science*, 18:37–48.

Patel, R. N., Pimpale, P. B., and Sasikumar, M. (2016). Recurrent neural network based part-of-speech tagger for code-mixed social media text. *arXiv preprint arXiv:1611.04989*.

Patra, B. G., Das, D., and Das, A. (2018). Sentiment analysis of code-mixed indian languages: An overview of sail_code-mixed shared task@ icon-2017. *arXiv preprint arXiv:1803.06745*.

Peng, H., Thomson, S., and Smith, N. A. (2017). Deep multitask learning for semantic dependency parsing. *arXiv preprint arXiv:1704.06855*.

Petrov, S., Das, D., and McDonald, R. (2011). A universal part-of-speech tagset. *arXiv preprint arXiv:1104.2086*.

Pimpale, P. B. and Patel, R. N. (2016). Experiments with pos tagging code-mixed indian social media text. *arXiv preprint arXiv:1610.09799*.

Poplack, S. (1980). Sometimes i'll start a sentence in spanish y termino en espanol: toward a typology of code-switching1. *Linguistics*, 18(7-8):581–618.

Prager, J. M. (1999). Linguini: Language identification for multilingual documents. *Journal of Management Information Systems*, 16(3):71–101.

Pratapa, A., Bhat, G., Choudhury, M., Sitaram, S., Dandapat, S., and Bali, K. (2018). Language modeling for code-mixing: The role of linguistic theory based synthetic data. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1543–1553.

PVS, A. and Karthik, G. (2007). Part-of-speech tagging and chunking using conditional random fields and transformation based learning. In *Proceedings of the*

IJCAI-2007 Workshop on Shallow Parsing for South Asian Languages (SPSAL-2007), pages 21–24.

Ramesh, S. H. and Kumar, R. R. (2016). A pos tagger for code mixed indian social media text-icon-2016 nlp tools contest entry from surukam. *arXiv preprint arXiv:1701.00066*.

Ray, P. R., Harish, V., Sarkar, S., and Basu, A. (2003). Part of speech tagging and local word grouping techniques for natural language parsing in Hindi. In *Proceedings of the 1st International Conference on Natural Language Processing (ICON 2003)*.

Rijhwani, S., Sequiera, R., Choudhury, M., Bali, K., and Maddila, C. S. (2017). Estimating code-switching on twitter with a novel generalized word-level language detection technique. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1971–1982.

Ruder, S., Bingel, J., Augenstein, I., and Søgaard, A. (2017). Learning what to share between loosely related tasks. *arXiv preprint arXiv:1705.08142*.

Samih, Y., Maharjan, S., Attia, M., Kallmeyer, L., and Solorio, T. (2016). Multilingual code-switching identification via lstm recurrent neural networks. In *Proceedings of the Second Workshop on Computational Approaches to Code Switching*, pages 50–59.

San, H. K. (2009). Chinese-English code-switching in blogs by Macao young people. Master's thesis, The University of Edinburgh, Edinburgh, UK. `http://hdl.handle.net/1842/3626`.

Sankoff, D. and Poplack, S. (1981). A formal grammar for code-switching. *Papers in Linguistics - International Journal of Human Communication*, 14:3–46.

Scannell, K. P. (2007). The crúbadán project: Corpus building for under-resourced languages. In *Building and Exploring Web Corpora: Proceedings of the 3rd Web as Corpus Workshop*, volume 4, pages 5–15.

Schmid, H. (1994a). Part-of-speech tagging with neural networks. In *Proceedings of the 15th Conference on Computational Linguistics - Volume 1*, COLING '94, pages 172–176, Stroudsburg, PA, USA. Association for Computational Linguistics.

Schmid, H. (1994b). Probabilistic part-of-speech tagging using decision trees. In *Proceedings of the International Conference on New Methods in Language Processing (NeMLaP-1)*, pages 44–49, Manchester, UK.

Schulz, S. and Keller, M. (2016). Code-switching ubique est-language identification and part-of-speech tagging for historical mixed text. In *Proceedings of the 10th SIGHUM Workshop on Language Technology for Cultural Heritage, Social Sciences, and Humanities*, pages 43–51.

Sequiera, R., Choudhury, M., and Bali, K. (2015a). Pos tagging of hindi-english code mixed text from social media: Some machine learning experiments. In *Proceedings of the 12th International Conference on Natural Language Processing*, pages 237–246.

Sequiera, R., Choudhury, M., Gupta, P., Rosso, P., Kumar, S., Banerjee, S., Naskar, S. K., Bandyopadhyay, S., Chittaranjan, G., Das, A., et al. (2015b). Overview of fire-2015 shared task on mixed script information retrieval. In *FIRE Workshops*, volume 1587, pages 19–25.

Shafie, L. A. and Nayan, S. (2013). Languages, code-switching practice and primary functions of facebook among university students. *Study in English Language Teaching*, 1(1):187–199. `http://www.scholink.org/ojs/index.php/selt`.

Sharma, A. and Motlani, R. (2015). Pos tagging for code-mixed indian social media text: Systems from iiit-h for icon nlp tools contest.

Shirvani, R., Piergallini, M., Gautam, G. S., and Chouikha, M. (2016). The howard university system submission for the shared task in language identification in spanish-english codeswitching. In *Proceedings of The Second Workshop on Computational Approaches to Code Switching*, pages 116–120.

Shrivastava, M., Melz, R., Singh, S., Gupta, K., and Bhattacharyya, P. (2006). Conditional random field based POS tagger for Hindi. Presentation slides of the First National Symposium on Modeling and Shallow Parsing of Indian Languages (MSPIL), `http://www.cfilt.iitb.ac.in/~mspil-06/mspilpresn.zip` accessed 2015-04-09.

Sibun, P. and Reynar, J. C. (1996). Language identification: Examining the issues.

Singh, S., Gupta, K., Shrivastava, M., and Bhattacharyya, P. (2006). Morphological richness offsets resource demand-experiences in constructing a pos tagger for hindi. In *Proceedings of the COLING/ACL on Main conference poster sessions*, pages 779–786. Association for Computational Linguistics.

Søgaard, A. and Goldberg, Y. (2016). Deep multi-task learning with low level tasks supervised at lower layers. In *Proceedings of the 54th Annual Meeting of the*

Association for Computational Linguistics (Volume 2: Short Papers), volume 2, pages 231–235.

Solorio, T., Blair, E., Maharjan, S., Bethard, S., Diab, M., Ghoneim, M., Hawwari, A., AlGhamdi, F., Hirschberg, J., Chang, A., et al. (2014a). Overview for the first shared task on language identification in code-switched data. In *Proceedings of the First Workshop on Computational Approaches to Code Switching*, pages 62–72.

Solorio, T., Blair, E., Maharjan, S., Bethard, S., Diab, M., Gonheim, M., Hawwari, A., AlGhamdi, F., Hirshberg, J., Chang, A., and Fung, P. (2014b). Overview for the first shared task on language identification in code-switched data. In *Proceedings of the First Workshop on Computational Approaches to Code-Switching. EMNLP 2014, Conference on Empirical Methods in Natural Language Processing*, Doha, Qatar. Association for Computational Linguistics.

Solorio, T. and Liu, Y. (2008a). Learning to predict code-switching points. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 973–981. Association for Computational Linguistics.

Solorio, T. and Liu, Y. (2008b). Part-of-speech tagging for English-Spanish code-switched text. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1051–1060. Association for Computational Linguistics.

Sowmya, V., Choudhury, M., Bali, K., Dasgupta, T., and Basu, A. (2010). Resource creation for training and testing of transliteration systems for Indian languages. In *Proceedings of the Language Resource and Evaluation Conference (LREC) 2010*, pages 2902–2907. European Language Resources Association.

Sutton, C., McCallum, A., and Rohanimanesh, K. (2007). Dynamic conditional random fields: Factorized probabilistic models for labeling and segmenting sequence data. *Journal of Machine Learning Research*, 8(Mar):693–723.

Toutanova, K. and Manning, C. D. (2000). Enriching the knowledge sources used in a maximum entropy part-of-speech tagger. In *Proceedings of the 2000 Joint SIGDAT conference on Empirical methods in natural language processing and very large corpora: held in conjunction with the 38th Annual Meeting of the Association for Computational Linguistics-Volume 13*, pages 63–70. Association for Computational Linguistics.

Tromp, E. and Pechenizkiy, M. (2011). Graph-based n-gram language identification on short texts. In *Proc. 20th Machine Learning conference of Belgium and The Netherlands*, pages 27–34.

Vogel, J. and Tresner-Kirsch, D. (2012). Robust language identification in short, noisy texts: Improvements to liga. In *Proceedings of the 3rd International Workshop on Mining Ubiquitous and Social Environments*, pages 1–9.

Voss, C. R., Tratz, S., Laoudi, J., and Briesch, D. M. (2014). Finding romanized arabic dialect in code-mixed tweets. In *LREC*, pages 2249–2253.

Voutilainen, A. (1995). A syntax-based part-of-speech analyser. In *Proceedings of the Seventh Conference on European Chapter of the Association for Computational Linguistics*, EACL '95, pages 157–164, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Vyas, Y., Gella, S., Sharma, J., Bali, K., and Choudhury, M. (2014). POS tagging of English-Hindi code-mixed social media content. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 974–979, Doha, Qatar. Association for Computational Linguistics.

Wang, A. and Kan, M.-Y. (2013). Mining informal language from chinese microtext: Joint word recognition and segmentation. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 731–741.

Wang, P., Qian, Y., Soong, F. K., He, L., and Zhao, H. (2015). Part-of-speech tagging with bidirectional long short-term memory recurrent neural network. *arXiv preprint arXiv:1510.06168*.

Winata, G. I., Madotto, A., Wu, C.-S., and Fung, P. (2018a). Code-switching language modeling using syntax-aware multi-task learning. *arXiv preprint arXiv:1805.12070*.

Winata, G. I., Wu, C.-S., Madotto, A., and Fung, P. (2018b). Bilingual character representation for efficiently addressing out-of-vocabulary words in code-switching named entity recognition. *arXiv preprint arXiv:1805.12061*.

Winkelmolen, F. and Mascardi, V. (2011). Statistical language identification of short texts. In *ICAART (1)*, pages 498–503. Citeseer.

Xafopoulos, A., Kotropoulos, C., Almpanidis, G., and Pitas, I. (2004). Language identification in web documents using discrete hmms. *Pattern recognition*, 37(3):583–594.

Xia, M. X. (2016). Codeswitching language identification using subword information enriched word vectors. In *Proceedings of The Second Workshop on Computational Approaches to Code Switching*, pages 132–136.

Yamaguchi, H. and Tanaka-Ishii, K. (2012). Text segmentation by language using minimum description length. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*, pages 969–978. Association for Computational Linguistics.

Yu, X. and Lam, W. (2010). Jointly identifying entities and extracting relations in encyclopedia text via a graphical model approach. In *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*, pages 1399–1407. Association for Computational Linguistics.

Zaidan, O. F. and Callison-Burch, C. (2011). The arabic online commentary dataset: an annotated dataset of informal arabic with high dialectal content. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers-Volume 2*, pages 37–41. Association for Computational Linguistics.

Zennaki, O., Semmar, N., and Besacier, L. (2015). Unsupervised and lightly supervised part-of-speech tagging using recurrent neural networks. In *Proceedings of the 29th Pacific Asia Conference on Language, Information and Computation*, pages 133–142.

Zhang, Y. and Clark, S. (2008). Joint word segmentation and pos tagging using a single perceptron. *Proceedings of ACL-08: HLT*, pages 888–896.

Zhang, Y. and Weiss, D. (2016). Stack-propagation: Improved representation learning for syntax. *CoRR*, abs/1603.06598.

Zhao, K. and Huang, L. (2017). Joint syntacto-discourse parsing and the syntacto-discourse treebank. *arXiv preprint arXiv:1708.08484*.

Appendix A

SVM-based LID Results for Nepali-English Data.

Following are the results of our SVM-based LID systems for Nepali-English data. In the First Workshop of Computational Approaches to Code-Switching, EMNLP, 2014 (Solorio et al., 2014a), our SVM-based LID system achieved the highest token-level accuracy in the word-level language identification task of Nepali-English. The goal of the task is to generate word-level language predictions (six labels - *lang1, lang2, ne, mixed, ambiguous and other*) for mixed lingual user generated content.

Features	**Accuracy**	**Features**	**Accuracy**
G	96.02	GD	96.27
GL	96.11	GDL	96.32
GC	96.15	GDC	96.20
GLC	96.21	**GDLC**	**96.40**

Table A.1: Average Cross-Validation Accuracy for SVM-6-Way based Prediction for Nepali-English Training, Data Set, G = Char-N-Gram, L = Binary Length Features, D = Dict.-Based Labels and C = Capitalization features. For more detail, please visit the Language Identification Chapter.

Tweets		
	Token Level	*Tweet Level*
Nepali-English	0.963	0.958
Surprise Genre		
	Token Level	*Post/Comment Level*
Nepali-English	0.856	0.775

Table A.2: Test set results (overall accuracy) for Nepali-English and Spanish-English Tweet Data and Surprise Genre