"ye word kis lang ka hai bhai?" Testing the Limits of Word level Language Identification

Spandana Gella, Kalika Bali, Monojit Choudhury
Microsoft Research India
{t-spgell, kalikab, monojitc}@microsoft.com

Abstract

Language identification is a necessary prerequisite for processing any user generated text, where the language is unknown. It becomes even more challenging when the text is code-mixed, i.e., two or more languages are used within the same text. Such data is commonly seen in social media, where further challenges might arise due to contractions and transliterations. The existing language identification systems are not designed to deal with code-mixed text, and as our experiments show, perform poorly on a synthetically created code-mixed dataset for 28 languages.We propose extensions to an existing approach for word level language identification. Our technique not only outperforms the existing methods, but also makes no assumption about the language pairs mixed in the text - a common requirement of the existing word level language identification systems.This study shows that word level language identification is most likely to confuse between languages which are linguistically related (e.g., Hindi and Gujarati, Czech and Slovak), for which special disambiguation techniques might be required.

1 Introduction

As the World Wide Web is constantly being inundated by user-generated content in many languages, automatic *Language Identification* (LI) of text fragments has become an extremely important problem. Since mid 90s, this problem has attracted the attention of researchers (Prager, 1999), and by 2005 it was thought to be a solved problem (McNamee, 2005). However, there is a renewed interest in LI over last few years (Lui and Baldwin, 2011; Tromp and Pechenizkiy, 2011; Bergsma et al., 2012; Lui and Baldwin, 2012; Goldszmidt et al., 2013; King and Abney, 2013; Carter et al., 2013; Lui et al., 2014) primarily due to a surge in user-generated content in social media, given the rising popularity of Twitter, Facebook and several other online social networking platforms.

Short length, presence of non-standard spelling variations, idiomatic expressions and acronyms are the most commonly cited reasons that make LI for social media content, such as tweets, a challenging problem (Carter et al., 2013; Goldszmidt et al., 2013). However, there are two other frequently observed phenomena in social media (and also in other forms of user-generated content such as blogs and discussion forums) which have received very little attention: *Code-mixing* and *Transliteration*.

Code-mixing, loosely speaking, is the process of mixing more than one language in a single conversation. In the context of social media, code-mixing can be defined as using more than one, usually only two, languages in a single tweet or post. For instance, in the following comment posted on Facebook, the user has *mixed* Hindi (in italics) and English words in a single sentence.

love affection *lekar salose se* sunday *ke din chali aarahi* divine *parampara ko age badhha rahe ho*

Translation: With love and affection, (you) are carrying forward the divine tradition that has been going on on Sundays for years.

This example also illustrates the phenomenon of *transliteration*; the Hindi text is not written using the Devanagari script; the user has transliterated it in Roman script. Code-mixing and transliteration are not mere challenges for LI, rather these phenomena demand a new definition of LI. It is no longer sufficient to identify the language of a tweet or microblog or even a sentence. One needs to identify the language of every word.[1] Transliteration adds to the complexity of the problem because there is no notion of correct transliteration (the same word can be spelled as *badhha, badha, badhaa* and so on), which then can be further contracted according to the norms of computer mediated communication.

How common are these phenomena? In a separate study by us (to appear, reference removed for anonymity), we found that 50% of the sentences on a Hindi film discussion forum are code-mixed (mainly between Telugu-English and Hindi-English); we also found that 17% of the posts on Facebook public pages of Indian celebrities have code-mixing, and more than 90% of the Indian language texts were Romanized. In fact, code-mixing and transliteration are not unique to social media; Gupta et al (2014) have shown that 6% of Bing queries from India has Roman transliterated Hindi words mixed with English terms. Neither is code-mixing unique to Indian languages. In a recently conducted shared task on LI for codeswitched text[2], tens of thousands of tweets were released that had code-mixing between English and Spanish, English and Nepali, English and Mandarin, and between standard and dialectal Arabic. Thus, LI at word level in the presence of code-mixing and transliteration (wherever relevant) is an essential task that needs to be solved before any further analysis of user-generated text in social media.

In this paper, we study the performance of word-level LI for code-mixed text of several state-of-the-art and off-the-shelf LI systems. Since these systems were not designed for tackling code-mixed text, the performance, as expected, is less than satisfactory. We propose extensions to an existing LI technique for language labeling at word level. In the absence of sufficient data across many languages annotated with language labels for words, we created a synthetic dataset by mixing natural language text fragments in 28 languages. Our dataset included three cases of transliterated text, for Hindi, Bengali and Gujarati. The proposed extensions outperform all the existing LI systems evaluated by a significant margin for all 28 languages.

Previous work on LI at word level presumes an a priori knowledge of the *two languages* mixed and the task mostly reduces to a binary classification problem (Solorio and Liu, 2008a; Nguyen and Dogruoz, 2013; Gella et al., 2013). However, this is not the case in practice. The languages to be identified are usually not known in advance and the set of potential language labels can span all the languages represented on the World Wide Web. The primary contribution of the current work is to highlight the challenges in such a scenario and present a few techniques to deal with these.

The rest of the paper is organized as follows. In Sec. 2 we review related work with special reference to the existing LI systems evaluated in this work. Sec. 3 describes the synthetic dataset creation. In Sec. 4 we propose different extensions and baselines for word level LI, and in the next section, we present the results of the evaluation experiments. Sec 6 discusses the various implications of these results. We conclude with some future directions in Sec 7.

2 Related Work

Code-Mixing (CM) or the embedding of linguistic units such as phrases, words and morphemes of one language into an utterance of another language (Gumperz, 1982; Myers-Scotton, 1993; Myers-Scotton, 2002) is a well-studied linguistic phenomenon of multilingual speech communities. As mentioned earlier,the wide-spread use of computer mediated communication like email, chat, and more recently, on social media like Facebook and Twitter (Herring, 2003; Cardenas-Claros and Isharyanti, 2009; Paolillo, 2011) has ensured that code-mixed data is fairly prevalent on the web. In the case of social-media content where there are additional complications due to contractions, non-standard spellings, and non-standard constructions as well as mixing of scripts, processing of the data poses major chal-

[1]Sometimes, in certain languages, even the same word can be composed of parts taken from different languages. E.g., the word *housinalli* is composed of an English root *house* and Kannada suffix *nalli* meaning "in the house". This kind of usage is not uncommon for morphologically rich languages, but is beyond the scope of the current work.

[2]`http://www.emnlp2014.org/workshops/CodeSwitch/call.html`

lenges. Further, many languages that use non-Roman scripts, like Hindi, Bangla, Chinese, Arabic etc. are often present in a Romanized form. (Sowmya et al., 2010). Not a lot of work has been done on computational models of code-mixing (Solorio and Liu, 2008a; Solorio and Liu, 2008b; Nguyen and Dogruoz, 2013) , primarily due to the paucity of CM data in conventional text corpora which makes data-intensive methods hard to apply. (Solorio and Liu, 2008a) in their work on English-Spanish CM use models built on smaller datasets to predict valid switching points to synthetically generate data from monlingual corpora. While natural language processing like POS tagging, normalization, etc remain hard problems to solve, any processing of code-mixed text, first needs to deal with the identification and labeling of the parts of text which are in different languages.

Monolingual language identification has been worked on intensively in NLP where the task is to assign a language to every document according to the language it contains. There are existing methods which show high accuracies on large (Xia et al., 2009) and short (Tromp and Pechenizkiy, 2011; Lui and Baldwin, 2012) documents when tested against a small set of languages. However, for Code-mixed text, especially those involving transliterations and orthographic variation, this is far from a solved problem. In their work, (King and Abney, 2013) use a weakly supervised n-gram based model trained on monolingual data for labeling languages in a mixed-language document. In their experiments with language-labels of words in multilingual online discussions using language models and dictionaries, (Nguyen and Dogruoz, 2013) show that spelling variations, similarity between words in the two languages, as well as Named Entities, are the biggest source of errors, though they show some improvement with the incorporation of context. Gella et al. (2013) built the system with the best performance for the shared task on language identification and back transliteration for English mixed with Hindi, Bangla and Gujarati in FIRE 2013 (Saha Roy et al., 2013). The system uses a source language with a confidence probability for each word similar to (King and Abney, 2013), and incorporates context information through a code-switching probability. The thing to be noted in all these cases is that the LI systems have a priori information on the languages that they have to disambiguate,making this essentially a binary classification task.

Some of the commonly used as well as state-of-the-art LI techniques used that we evaluate in our work are:

`linguini`: This was one of the early systems proposed by (Prager, 1999) to identify multiple languages and their proportions in a single document. This is based on vector space model, where the languages in a document are determined by the similarity scores of the document vector with the language feature vector.

`langid.py`: `langid.py`[3] is an off-the-shelf language identification tool trained over naive bayes classifier with a multinomial event model over a mixture of byte 1,2,3,4-grams. It supports 97 languages and has shown high accuracy scores over both long and short documents.

polyglot: This system is designed to detect multilingual documents, their constituent languages and to estimate the portion of each constituent language. This was achieved with a generative mixture model combined with the document representation developed for monolingual language identification. (Lui and Baldwin, 2011).

`CLD2`: `CLD2`[4] is a compact language detection tool designed to target mono/multi lingual web pages of at least 200 characters. It uses Naive Bayes classifier over unigrams, with quadgrams as features.

3 Synthetic Dataset for Code-Mixed Text

In order to conduct a comprehensive evaluation of LI systems on code-mixed text, we need a dataset of short text fragments, which we shall refer to as *document* (even though they are no longer than a sentence), spanning a large number of languages. Ideally, a good proportion of these sentences should feature code-mixing between various language pairs. Most importantly, the words in every sentence must be annotated with the appropriate language label, so that we can automatically evaluate the performance of various LI systems on this collection.

We are not aware of any such publicly available dataset. There are monolingual LI test-benches, but none of these features code-mixing at the word level. The dataset used by (King and Abney, 2013) has language mixing at inter-

[3]https://github.com/saffsd/langid.py

[4]https://code.google.com/p/cld2/

sentential level; typically there are stretches of 20 or more words in a single language before switching. However, tweets and social media posts are shorter, and code-mixing in such context often means embedding one to three word phrases of one language inside a sentence in another language (such as the example cited in Sec. 1). Recently, around 500 code-mixed Web search queries with language labels for words were released as a part of the FIRE shared task on transliterated search (Roy et al., 2013). The dataset contains Bengali-English, Hindi-English and Gujarati-English mixed queries, where all the Indian language words are Romanized. We shall refer to this dataset as the **FIRE-data**, which will be used later for some of our experiments.[5]

Code-mixed text in only four languages is not sufficient for a realistic evaluation of Web-scale systems. Nevertheless, it is also a very effort intensive task to gather and annotate code-mixed text for a large number of language pairs. Hence, we decided to create a synthetic code-mixed dataset spanning 28 languages listed in Table 2.

3.1 Design Principles

Artificially generating code-mixed text from monolingual text data is a non-trivial problem for several reasons. First, code-mixed text is not a random mixture of two languages; rather there are strict syntactic and semantic rules that govern the code-switching points as well as the structure of the sentence. Second, while theoretically one could mix any subset of languages, in practice code-mixing is commonly observed only between certain pairs of languages which have a sizeable bilingual population with active Web presence. Therefore, we came up with the following design principles for creating a synthetic dataset:

1. *Every document consists of words from at most two languages.* This is based on a practical observation that people rarely mix more than two languages in a single sentence. Whenever they apparently mix more than two languages, words of one of the languages can almost always be explained away as *borrowing* rather than mixing.

2. *All the documents are in a single script, which we chose as the Roman script because of its popularity.* If in a code-mixed text, each language is written in their native script (say Hindi words in Devanagari and English words in Roman script), LI becomes a trivial problem. Therefore, we choose languages which either use Roman script (like most of the West European languages) or languages which are quite commonly Romanized on the Web such as the Indian languages.

3. *Whenever code-mixing happens, one of the languages is always English.* This principle is also based on the empirical observation that code-mixing happens mostly between English and other languages (Vyas et al., 2014). However, this is not necessary; there are examples of code-mixing between Turkish and Dutch, Arabic and French, Chinese and Malay, and so on. Since selecting a representative set of language pairs is difficult, and mixing between all pairs would not only lead to impractical cases, but also make it cumbersome to analyze and represent the experimental data, we decided to only experiment with mixing between English and other languages. Note that this choice does not limit the generality of the conclusions of this study as none of the algorithms exploit the fact that one of the languages is English.

4. *The length of the documents vary from 4 to 12 words, roughly equivalent to 20-135 characters.* This choice is again motivated by the typical length of tweets and social media posts.

5. *There is only one codeswitching point per document.* Thus, for every code-mixed document, the words from each language appear together (see Table 1 for examples). This is not true in reality. For instance, the code-mixed sentence cited in Sec. 1 has 5 codeswitching point. Nevertheless, code-mixing does not allow a random set of words chosen from two languages to be permuted in any fashion. The distribution of words as well as codeswitching points are governed by a complex interplay of the syntactic rules of the two languages and semantic constraints; while some of the constraints are known, how they interact with each other and between language pairs are open questions. In absence of such knowledge, we thought as a first study, it will be a better to have as much coherent text fragments in a document as possible. This is achieved through this one codeswitching principle, of course, at the expense of losing some generality.

[5]As a part of the Code-switching shared task at EMNLP 2014, large datasets for English-Spanish, English-Mandarin, English-Nepali and Dialectal-Standard Arabic have been released recently. But those were not available during this study.

3.2 Dataset Creation

As a testbench for our experiments, we created a synthetic dataset following the above design priciples. The monolingual text samples have been collected from the dataset used by (King and Abney, 2013). These are texts from the Wikipedia and various other resources, such as The Universal Declaration of Human Rights, Rosetta project and Jehovah's Witness website[6]. We selected 25 languages, including English, from this sample, all of which use Roman script.

We created 954 short documents for each language in the following manner. First, a real valued random variable γ is sampled uniformly at random from the interval $[0, 1]$. γ defines the fraction of words in the document that are in English. Another integer valued random variable L is sampled, again uniformly at random from the interval $[4, 12]$. L is the number of words in the document. We define two integer dependent variables $X = \text{round}(\gamma L)$ and $Y = \text{round}((1-\gamma)L)$. Here, $\text{round}(\cdot)$ is the rounding-to-nearest-integer function. A sequence of X words are chosen from the English monolingual corpus (i.e., an X-gram is sampled from the corpus following the X-gram frequency distribution of English). Let us denote this sequence (potentially null) as Z_{en}. Similarly, a sequence of Y words are sampled, again at random from the text corpus of the other language. Let this (again potentially null) sequence be denoted by Z_{l^*}. Finally, a binary variable s is sampled from $\{0, 1\}$. If $s = 1$, we generate the document $Z_{en}Z_{l^*}$, else we generate the document $Z_{l^*}Z_{en}$. Thus, we created 22896 documents spanning 25 languages, most of which are code-mixed with English.

For transliterated text in Gujarati, Bengali and Hindi, we used the test and development sets of the **FIRE-data**. This dataset has naturally code-mixed text mostly derived from Web search queries, and we will report the performance of our system directly on this data.

In Table 1 we show few sample test documents.

3.3 Limitations

It is important to understand the limitations of this synthetic dataset, so that we can appropriately draw our conclusions on the basis of the experimental results on this testbench. As we shall see in the next section, none of the LI algorithms proposed here explicitly assume that at most two languages are mixed and there is only one codeswitching point. Therefore, we believe that these features of the synthetic dataset are not expected to lead to any misleading conclusions. On the other hand, real code-mixed data can have richer distributional patterns, for instance often single words or short phrases are embeded, which if appropriately exploited can lead to better performance. This dataset, however, lacks contractions and non-standard spellings because most of the text fragments come from Wikipedia. Therefore, performance of the algorithm on tweets or social media data could be lower than what we observe here, unless the training data is appropriately modified.

4 Language Identification Algorithms

Following the ideas of (King and Abney, 2013) and (Gella et al., 2013), we first build binary classifiers for each language. The classifier for a language l takes a word as input and returns a score between 0 and 1, which can be loosely interpreted as some sort of probability that the input word is of language l. We combine the outputs of these 28 classifiers and the context of a word to decide the final label of the word. We use character n-gram based classifiers as they are more efficient, robust to noisy data and save us the effort of collecting lexicons.

In this section, first we describe the binary classifier, followed by two simple baselines and one simple disambiguation strategy. We also describe a method to compute the accuracy if we had an optimal disambiguation strategy.

4.1 Binary classifier for Words

For each language l, we built a binary classifier using character n-gram features. We experimented with character n-grams for $n = 1$ to 5, and two standard classifiers – *Naive Bayes* and *Logistic Regression*, for which we used the Mallet tool box (McCallum, 2002) set to the default settings. The positive examples consisted of a random set of words of l and an equal number of negative examples were constructed by choosing words from all the other 27 languages. The training data for 25 languages were collected from the respective Wikipedias; since user-generated transliterated content is not available on Wikipedia, for the three Indic languages the training data was

[6]http://www.watchtower.org

Language	L	γ	**Sentence**
Spanish	8	0.50	do nicely Switch off *efectivos tanto entre los*
Turkish	11	0.45	*safhalarnda paraszdr lk retim mecburidir Teknik* be delighted to meet them
French	12	0.25	Vasili knew this *une libert plus grande Considrant que les tats Membres*
Hindi	5	0.60	to reinvent Michigan *chandini badarava*
Bangla	9	0.60	*swadhinatar sutre smritituku pasei* sailing in from another room
Latvian	5	0.10	*ierobeojumu un apmakstu periodisku atvainjumu*
Swahili	11	0.90	*ya Magharibi kwenye pwani la Bahari Atlantiki Imepakana na Benin* finger

Table 1: Sample of synthetic test data; the non-English part is in italics

obtained from the development set of the **FIRE-data**. Table 2 shows the number of positive training examples used for each language. An equal number of negative examples were used as well.

We found that for all languages, logistic regression gave the best performance when we use a combination of n-gram features for all $n = 1$ to 5. The accuracy on the training set of the best binary classifiers varied from 0.866 for Catalan to 0.992 for Gujarati with an average of 0.941. These findings and numbers are in good agreement with those mentioned in (King and Abney, 2013; Gella et al., 2013).

Language	**Size**	**Language**	**Size**
Catalan (ca)	908	Czech (cs)	1318
Danish (da)	1054	German (de)	1108
Estonian (et)	1304	Finnish (fi)	1435
French (fr)	1038	Irish (ga)	1002
Hunharian (hu)	1379	Indonesian (in)	1055
Italian (it)	1342	Latvian (lv)	1519
Lithuanian (lt)	1258	Maltese (mt)	1369
Dutch (nl)	1160	Polish (pl)	1652
Portuguese (pt)	1228	Romanian (ro)	1278
Slovak (sk)	1410	Slovene (sl)	1136
Spanish (es)	905	Swahili (sw)	931
Swedish (sv)	1051	Turkish (tr)	1363
Hindi (hi)	9486	Bangla (bn)	3140
Gujarati (gu)	384	English (en)	3276

Table 2: Languages tested and number of unique words in the training samples used for training the binary classifiers for words.

4.2 Baselines

Let $d = w_1, w_2 \dots w_{|d|}$ be a document, where w_i's are the words. The task of word-level LI is to assign each word w_i with a language label $\lambda(w_i)$ chosen from $\mathbf{L} = \{l_1, l_2 ... l_{28}\}$, the set of all languages. Let us denote the confidence score of the classifier for language l_j on input w_i as $conf\_score(w_i, l_j)$.

`MaxWeighted`: In this method we simply assign the label of the language classifier that has the highest confidence score.

$$\lambda(w_i) = \text{argmax}_{j=1}^{n}\, conf\_score(w_i, l_j)$$

Thus, this is a completely context agnostic model.

`Random`: In this model, a randomly chosen label from the possible set of labels is assigned to the w_i, where the possible set of labels is obtained by setting a threshold value t on the confidence scores of the classifiers.

$$\lambda(w_i) = rand\{l_j : conf\_score(w_i, l_j) \geq t\}$$

where $rand\{\cdot\}$ selects a random element from the set.

4.3 CoverSet Method

The `CoverSet` method is based on the assumption that code-mixing, whenever it happens, happens only with a few languages, though it assumes no hard upper bound on the number of languages that can be mixed in a document. Thus, we want to choose as few labels from $\mathbf{L}$ as possible so as to label or *cover* all the words w_1 to $w_{|d|}$, without compromising on accuracy. Since this in essence is similar to the *minimum set cover* problem, we call this technique the `CoverSet`.

Like `Random`, we define a hard threshold t, such that the only labels that are considered valid for w_i are those for which $conf\_score(w_i, l_j) \geq t$. Let us define this subset of languages as $\mathbf{L}^i \subset \mathbf{L}$. We define the *minimal coverset* of d as $\mathbf{L}_d^*$ if and only if the following two conditions are satisfied:

1. $\mathbf{L}_d^* \cap \mathbf{L}^i \neq \phi \quad \forall 1 \leq i \leq 28$
2. There does not exist a set $\mathbf{L}_d'$ that satisfies condition 1, and $|\mathbf{L}_d'| < |\mathbf{L}_d^*|$.

Once the *minimal coverset* is determined through iterative search, λw_i is assigned the label from the set $\mathbf{L}_d^* \cap \mathbf{L}^i$ that has the highest confidence score. Note that there can be more than one *minimal coverset* for d, in which case, we do the labeling for all the sets and the one that yields the highest $\sum_i conf\_score(w_i, \lambda(w_i))$ is selected as the final output.

4.4 Optimal Labels

In order to estimate the maximum achievable accuracy in such an extreme LI problem with our language classifiers, we designed the following optimal method. For each word w_i, we first compute the set of possible labels $\mathbf{L}^i$ based on a threshold t. If the actual (gold standard) label of w_i, $\lambda^*(w_i) \in \mathbf{L}^i$, then we assign $\lambda(w_i) = \lambda^*(w_i)$, else any random label is selected for $\mathbf{L}^i$. Note that this is not a practical system because it assumes the knowledge of the gold standard labels; but it will be studied to understand the maximum achievable accuracy of a word label LI system.

5 Experiments and Results

We evaluated the four existing LI systems as well as the four proposed word level LI strategies on the synthetic testbench. Since the existing LI systems output language labels only for the entire document, we evaluated them on document labeling accuracy, and only on the 25 non-Indic languages because they cannot handle transliterated content. However, the strategies that we proposed were evaluated on both document and word labeling accuracies for all the languages.

5.1 Document label accuracy

Table 3 we present the document label accuracies for the 8 systems. For a given document, **Lang1** refers to the language which has the higher proportion of words. The other language is denoted as **Lang 2**. In other words, if for a document $\gamma > 0.5$, **Lang1** is English and **Lang2** is the other language, else **Lang1** is the other language and **Lang2** is English.

The 4 existing systems output a single language as the label for an input document. But `linguini`, `polyglot` and `langid.py` also output a second or more languages with some scores whenever the system identifies that the document could consist of more than one language. Thus, for these systems, we compare their first or primary output with **Lang1** and the second output, whenever present, with **Lang2** to compute the respective accuracies. For our approaches, we count the word labels and assign the language with the highest and the second highest number of labels in the document as **Lang1** and **Lang2** respectively. We tuned the parameter t for three of the proposed systems for maximum word labelling accuracy. These values are also shown in the Table 3, for which the document labeling accuracies have been reported.

System	**Lang1**	**Lang2**	**Code Mixing**		
			P	R	F
`linguini`	0.589	0.084	0.817	0.846	0.831
`polyglot`	0.827	0.308	0.990	0.426	0.590
`langid.py`	0.781	0.058	0.790	1.000	0.885
`CLD2`	0.817	NA	NA	NA	NA
`Random`-0.8	0.635	0.235	0.814	0.991	0.894
`MaxWeighted`	0.832	0.468	0.814	0.993	0.895
`CoverSet`-0.5	0.888	0.717	0.868	0.991	0.920
`Optimal`-0.5	0.968	0.882	0.910	0.990	0.953

Table 3: Document label accuracies for the existing and the proposed LI systems. The value of t shown after hyphen.

We also report the precision (P), recall (R), f-score (F) of the systems in identifying code-mixed documents. While computing these values, we did not consider the language labels, but only whether the output was a single (implying monolingual document) or multiple (implying code-mixed document) languages.

System	**Hindi**	**Gujarati**	**Bangla**	**English**
`Random`-0.8	0.170	0.103	0.180	0.812
`MaxWeighted`	0.419	0.105	0.399	0.797
`CoverSet`-0.5	0.569	0.017	0.394	0.814
`Optimal`-0.5	0.792	0.714	0.719	0.864

Table 4: Language accuracies on **FIRE-data**

Table 4 presents the document labeling accuracies for the 3 Indic languages evaluated on the **FIRE-data**. The numbers are lower than the corresponding figures for the other languages.

5.2 Word Labeling Accuracy

We define the word labeling accuracy as the fraction of words (tokens) in the entire synthetic dataset (including Indic language data) that have been labeled with the correct language. Figure 1 plots the word labeling accuracy of the four systems for different values of t. Since MaxWeighted does not depend on t, its value is constant (0.67). As expected, Optimal presents an upper-bound on the accuracy, which is close to 0.94, and the Random baseline provides a lower-bound on the accuracies, which varies between 0.3 and 0.5. Performance of CoverSet is always better than MaxWeighted, and reaches 0.82 for optimal value of $t = 0.5$.

L	A	Top n confused languages				
it	.815	en (.096)	ca (.018)	ro (.012)	pt (.010)	es (.009)
es	.733	pt (.071)	en (.060)	it (.038)	ca (.027)	ro (.013)
mt	.889	en (.039)	nl (.007)	fr (.005)	ro (.004)	it (.004)
tr	.809	en (.094)	hi (.015)	bn (.014)	da (.011)	lv (.007)
fr	.681	en (.201)	ca (.029)	ro (.017)	pt (.017)	it (.008)
hu	.884	en (.024)	bn (.011)	hi (.009)	da (.009)	pt (.007)
ca	.618	en (.199)	es (.040)	ro (.025)	pt (.025)	fr (.023)
sk	.690	cs (.164)	en (.062)	sl (.031)	pl (.009)	bn (.006)
lt	.869	en (.032)	lv (.021)	bn (.012)	id (.007)	pt (.006)
de	.775	en (.122)	nl (.041)	da (.016)	sv (.007)	fr (.006)
ga	.811	en (.112)	pt (.018)	fr (.012)	hu (.007)	bn (.006)
sl	.737	en (.065)	sk (.063)	cs (.030)	bn (.017)	hi (.015)
pl	.871	en (.043)	sk (.013)	ro (.010)	mt (.009)	pt (.006)
ro	.740	en (.127)	it (.027)	fr (.021)	ca (.019)	es (.010)
nl	.730	en (.153)	de (.039)	da (.032)	hu (.007)	sv (.007)
pt	.800	en (.070)	es (.050)	it (.023)	ca (.015)	bn (.007)
sv	.803	en (.082)	da (.061)	de (.008)	nl (.007)	it (.006)
cs	.599	sk (.218)	en (.069)	sl (.030)	hu (.018)	mt (.011)
da	.702	en (.177)	sv (.060)	nl (.015)	mt (.011)	de (.007)
fi	.681	et (.169)	en (.037)	hi (.022)	sv (.020)	bn (.011)
et	.848	en (.050)	hi (.024)	ro (.009)	nl (.009)	id (.009)
lv	.848	en (.041)	lt (.026)	pl (.013)	sl (.008)	id (.006)
sw	.714	en (.069)	tr (.039)	hi (.032)	bn (.026)	pl (.017)
id	.810	hi (.045)	en (.040)	bn (.016)	es (.014)	tr (.014)
hi	.738	bn (.148)	en (.040)	id (.014)	gu (.006)	nl (.006)
bn	.878	en (.042)	hi (.041)	pt (.003)	it (.003)	id (.003)
gu	.712	hi (.146)	bn (.087)	en (.017)	id (.006)	mt (.005)
en	.897	pt (.008)	it (.008)	fr (.007)	hu (.006)	ga (.006)

Table 5: Confusion matrix between languages. L:Language, A: Accuracy

5.3 Error Analysis

Figures 2a and 2b show the variation of the word labeling accuracy of three of the proposed systems (results for Random ommited) with the parameters L (document length) and γ (fraction of words in English). Again, as expected, performance of Optimal is not affected by variation of these parameters, and that of CoverSet is also quite robust though we see slight decrease in performance when L drops below 6. Performance of MaxWeighted is mostly unaffected by L, but improves linearly with γ, which implies that there is a high tendency for this system to label words as English.

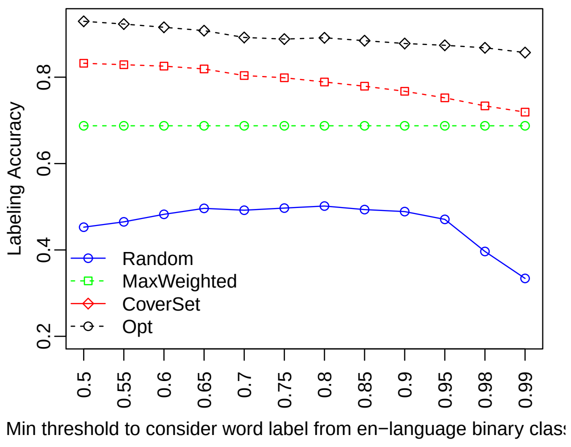


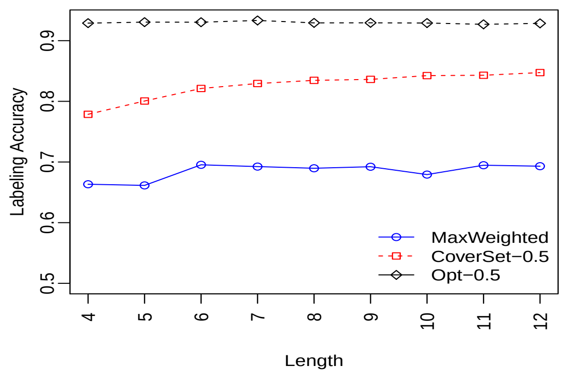
Figure 1: Word labeling accuracy vs. t

We also computed the language label confusion matrix for these strategies. Table 5 shows the five most common confused languages for each of the languages, along with the fraction of tokens so confused by the CoverSet approach. From the table we can infer that English is the least confused language and Czech, Catalan and Finnish are the most confused languages. This table also shows that (Czech, Slovak), (Gujarati, Hindi), (Hindi, Bangla) and (Finnish, Estonian) are some of the most confused language pairs.

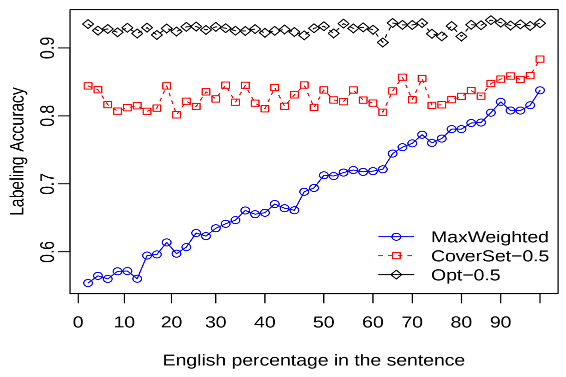
6 Discussion

As can be seen from the results presented in the previous section, three of the four extensions proposed by us outperform all of the existing LI systems (Table 3). While polyglot and CLD2 are comparable to the three proposed extensions in labelling accuracies for Lang1, they fail to a varying degree in labelling Lang2. This is not unexpected as none of the existing systems are designed to identify and label languages in a code-mixed text. The low accuracies of the Random0.8 are also not surprising as this algorithm randomly assigns labels from a set without taking any context information into consideration.

All the three extensions are robust to length, i.e.,



(a) Varying Length



(b) Varying English Percentage

Figure 2: Effect of L and γ on the performance.

the number of words in a text fragment (Fig. 2a). The `Optimal` method provides an upper-bound on the accuracy that can be achieved using the current set of binary classifiers. `CoverSet` uses context information and achieves higher accuracies for labeling both Lang1 and Lang2 confirming the importance of context in identifying languages in code-mixed text. Further, `CoverSet` technique is also stable with respect to γ. Thus, from the results the `CoverSet` technique seems the most promising for identifying and labeling languages in a code-mixed text.

It is interesting to note that the language-pairs that are confused the most (e.g., Hindi-Gujarati, Slovak-Czech etc.) are linguistically related and very close to each other. In fact, the very poor performance of the `CoverSet` meythod for Gujarati as opposed to that of `Optimal` in Table 4 is due to the fact that most of the Gujarati words were also identified as Hindi words. Hence,these are likely to be confused by the LI systems. It therefore seems that different techniques and features would be required to disambiguate between these languages.

The fall in accuracies for all the proposed extensions when tested on **FIRE-data** could be attributed to the fact that this data is not synthetic and shows all the variations we might expect in real code-mixed data, but more importantly the data is in Roman transliteration. As previous studies have shown (Sowmya et al., 2010; Ahmed et al., 2011) there is a lot of variation in transliterations of Indian languages due to one-to-many mappings at character level, regional variations and lack of any universally applicable standard conventions. The difficulty for LI systems to disambiguate transliterated Indian languages, thus, increases manifold. It should be noted, however, that the overall trend followed by the proposed extensions remains the same as that on the synthetic data for other languages.

7 Conclusion

In this paper we considered language identification task for short code-mixed documents containing one or two languages. We analyzed the word-level LI accuracy of some off-the-shelf systems on code-mixed synthetic dataset. We extended these algorithms to identify word-labels across 28 languages. The results show that our extensions outperform the existing systems significantally. However, most of the methods used consider a binary classification for a language pair.

An important area to work on is the code-mixing of transliterated text where standard techniques do not work and lexicons are not available. Our results also show that linguistically close or related languages are more difficult to disambiguate and further work is required to specifically address this problem.The biggest bottleneck, as always, remains the unavailability of suitable datasets and we will continue to explore new ways to collect, generate and annotate code-mixed data. In conclusion, we would like to mention that language identification is far from a solved problem; issues like code-mixing, transliteration and non-standard usage not only make it a more difficult problem, but also demands fundamental modifications in the definition of LI. It is an area that is likely to attract a lot more interest from researchers in the near future.

References

U. Ahmed, K. Bali, M. Choudhury, and V. B. Sowmya. 2011. Challenges in designing input method editors for indian languages: The role of word-origin and context. In *Proceedings of IJCNLP Workshop on Advances in Text Input Methods*.

S. Bergsma, P. McNamee, M. Bagdouri, C. Fink, and T. Wilson. 2012. Language identification for creating language-specific Twitter collections. In *Proceedings of the Second Workshop on Language in Social Media*, pages 65–74.

MS Cardenas-Claros and N Isharyanti. 2009. Code-switching and code-mixing in internet chatting: Between yes, ya, and si a case study. In *The JALT CALL Journal, 5*.

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

Spandana Gella, Jatin Sharma, and Kalika Bali. 2013. Query word labeling and back transliteration for indian languages: Shared task system description. In *FIRE Working Notes*.

Moises Goldszmidt, Marc Najork, and Stelios Paparizos. 2013. Boot-strapping language identifiers for short colloquial postings. In *Proc. of ECMLPKDD*. Springer Verlag.

J. Gumperz. 1982. *Discourse Strategies*. Oxford University Press.

Parth Gupta, Kalika Bali, Rafael E. Banchs, Monojit Choudhury, and Paolo Rosso. 2014. Query expansion for mixed-script information retrieval. In *Proc. of SIGIR*, pages 677–686. ACM Association for Computing Machinery.

S. Herring. 2003. *Media and Language Change: Special Issue*.

Ben King and Steven Abney. 2013. Labeling the languages of words in mixed-language documents using weakly supervised methods. In *Proceedings of NAACL-HLT*, pages 1110–1119.

Marco Lui and Timothy Baldwin. 2011. Cross-domain feature selection for language identification. In *In Proceedings of 5th International Joint Conference on Natural Language Processing*. Citeseer.

Marco Lui and Timothy Baldwin. 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.

Marco Lui, Jey Han Lau, and Timothy Baldwin. 2014. Automatic detection and language identification of multilingual documents. *Transactions of the Association for Computational Linguistics*, 2:27–40.

Andrew Kachites McCallum. 2002. Mallet: A machine learning for language toolkit. *http://mallet.cs.umass.edu*.

P. McNamee. 2005. Language identication: A solved problem suitable for undergraduate instruction. *Journal of Computing Sciences in Colleges*, 20(3).

C. Myers-Scotton. 1993. *Dueling Languages: Grammatical Structure in Code-Switching*. Claredon, Oxford.

C. Myers-Scotton. 2002. *Contact linguistics: Bilingual encounters and grammatical out-comes*. Oxford University Press, Oxford.

Dong Nguyen and A. Seza Dogruoz. 2013. Word level language identification in online multilingual communication. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 857–862.

John C. Paolillo. 2011. conversational codeswitching on usenet and internet relay chat. *Language@Internet*, 8(3).

John M Prager. 1999. Linguini: Language identification for multilingual documents. In *Systems Sciences, 1999. HICSS-32. Proceedings of the 32nd Annual Hawaii International Conference on*, pages 11–pp. IEEE.

Rishiraj Saha Roy, Monojit Choudhury, Prasenjit Majumder, and Komal Agarwal. 2013. Overview and datasets of fire 2013 track on transliterated search. In *FIRE Working Notes*.

T Solorio and Y. Liu. 2008a. Learning to predict code-switching points. In *Proceedings of the Empirical Methods in natural Language Processing*.

T Solorio and Y. Liu. 2008b. Parts-of-speech tagging for english-spanish code-switched text. In *Proceedings of the Empirical Methods in natural Language Processing*.

V. B. Sowmya, M. Choudhury, K. Dasgupta Bali, T., and A. Basu. 2010. Resource creation for training and testing of transliteration systems for indian languages. In *Proceedings of the LREC 2010*.

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

Yogarshi Vyas, Spandana Gella, Jatin Sharma, Kalika Bali, and Monojit Choudhury. 2014. Pos tagging of english-hindi code-mixed social media content. *Proceedings of the First Workshop on Codeswitching, EMNLP*.

Fei Xia, William D Lewis, and Hoifung Poon. 2009. Language id in the context of harvesting language data off the web. In *Proceedings of the 12th EACL*, pages 870–878.