Mining Hindi-English Transliteration Pairs from Online Hindi Lyrics

Kanika Gupta[1*], Monojit Choudhury[2], Kalika Bali[2]
[1]NI Systems (India) Pvt. Ltd. Bangalore, [2]Microsoft Research Labs India
[1]Bannerghata Road, Bangalore India, [2]"Vigyan", 9 Lavelle Road, Bangalore India
Email: kanika.gupta@ni.com, monojitc@microsoft.com, kalikab@microsoft.com

Abstract

This paper describes a method to mine Hindi-English transliteration pairs from online Hindi song lyrics. The technique is based on the observations that lyrics are transliterated word-by-word, maintaining the precise word order. The mining task is nevertheless challenging because the Hindi lyrics and its transliterations are usually available from different, often unrelated, websites. Therefore, it is a non-trivial task to match the Hindi lyrics to their transliterated counterparts. Moreover, there are various types of noise in lyrics data that needs to be appropriately handled before songs can be aligned at word level. The mined data of 30823 unique Hindi-English transliteration pairs with an accuracy of more than 92% is available publicly. Although the present work reports mining of Hindi-English word pairs, the same technique can be easily adapted for other languages for which song lyrics are available online in native and Roman scripts.

Keywords: Transliteration data, Data Mining from web, Hindi-English

1. Introduction

Due to the popularity of local language songs, a huge collection of song lyrics is available on the Web for languages like Hindi, Bengali, Telugu, Tamil, Arabic, Chinese, Japanese and Korean, to name a few. The lyrics for such languages, which use non-Roman script, are often present in both the native script and Roman transliterated form. Therefore, this data can be potentially mined to obtain transliteration pairs between the native and the Roman scripts, which can be used for training and evaluation of transliteration systems.

In this work, we describe a method to mine Hindi-English transliteration pairs from online Hindi song lyrics. The technique is based on the observations that lyrics are transliterated word-by-word, maintaining the precise word order. The mining task is nevertheless challenging because of two reasons. First, the Hindi lyrics and its transliterations are usually available from different, often unrelated, websites. Therefore, it is a non-trivial task to match the Hindi lyrics to their transliterated counterparts. Second, there are various types of noise in lyrics data that needs to be appropriately handled before songs can be aligned at word level.

Transliteration has been a focus of research because of its extensive applications in MT, IR and Input Method Editors (IME) (Sowmya et al., 2010). On the other hand, there are very limited publicly available datasets for training and testing transliteration systems. The major contributions of this work are in (a) identifying that song lyrics is a rich and readily available source of transliterated content which can be efficiently mined to gather large amounts of high quality training data, and (b) publicly sharing a 30823 Hindi-English transliteration pairs dataset. Although the present work reports mining of Hindi-English word pairs, the same technique can be easily adapted for other languages for which song lyrics are available online in native and Roman scripts.

2. Related Work

Transliteration broadly refers to sound preserving transcription of a word or name from one language into the script of another language (Knight and Graehl, 1998). For example, the Hindi word मान 'value' can be transliterated into English as *man* or *maan*. It is a useful technique for translating out-of-vocabulary words and named entities in MT and Cross-lingual IR. For certain languages, where typing in the native script is not very popular in the cyberspace, transliteration is also used as an input mechanism (Animesh et al., 2008; Ehara and Kumiko, 2008; Sowmya et al., 2010). In such cases, the user types the native language words and sentences (usually) in Roman script, and a transliteration engine automatically converts the Roman input back to the native script. This input mechanism, commonly referred to as IME, is popularly used for all Indian languages including Hindi, Bangla, Tamil, Telugu, etc., and also, Arabic, Chinese, Japanese and Korean to name a few. There are several commercially available transliteration based IMEs for Indic and other languages. Examples include Microsoft Indic Language Transliteration (http://specials.msn.co.in/ilit/), Quillpad (http://www.quillpad.in/) and Google Transliteration (http://www.google.com/transliterate/).

It is important to make a distinction between *forward* and *backward transliteration*. While the former refers to transliterating a word of language A (say Hindi) into the script of language B (say English, in which case the script is Roman), the latter is the reverse process of getting back the word in the native script, given its transliteration in a foreign script. Thus, the process of generating *maan* or *man* from the word मान, is forward transliteration, whereas the process of generating मान given *man* is

[*] This work was done while the author was working at Microsoft Research Labs India.

backward transliteration. Note that forward and backward transliterations ideally require different datasets for training. For instance, to train an English-to-Hindi backward transliteration engine one would need transliterations pairs such as "*man,* मान", where the original word is in Hindi, and its transliteration in English is a representation of the sound (/man/ in IPA) of the Hindi word using the English script. On the other hand, for English-to-Hindi forward transliteration engine, one would need instances like "*man*, मैन", where the original word of English origin – "man", and the transliterated word is representation of its sound (/mæn/ in IPA) in Hindi script, i.e., मैन.

Most of the modern transliteration engines are based on machine learning approaches (see, e.g., Li and Kumaran (2009), Khapra and Bhattacharyya (2010), and references therein) and therefore, their performances depend on the quantity and quality of the training data. Klementiev and Roth (2006) proposed one of the first methods to mine transliteration pairs, only named entities (NEs), from comparable corpora. Starting from an article-aligned comparable corpus in English and Russian, they extracted NEs from English articles, and identified a set of potential Russian transliterations for those NEs by using an English-Russian transliteration classifier that was trained on a small seed corpus. The extracted pairs were then re-ranked based on the frequency distribution of the NEs in the English and Russian articles. After re-ranking, the candidate pairs whose score was above a threshold were used to further retrain the classifier. The process was repeated with the newly trained classifier to discover more NE transliteration pairs.

Saravanan and Kumaran (2008) applied the above model for English and Tamil, and concluded that frequency distribution is not a reliable feature for mining infrequent NEs. Udupa et al. (2008, 2009) further extended the model by using CLIR techniques to mine comparable documents followed by an Extended Weighted HMM (EW-HMM) based classifier adopted from (He, 2007) to rank the NE pairs. They tested their approach on Tamil, Kannada, Hindi and Russian, on one side and English on the other. The Named Entities Workshop 2010 had a shared task on mining NE transliteration pairs from linked Wikipedia titles between English and Arabic, Chinese, Hindi, Tamil and Russian (Kumaran et al., 2010). The participants were provided with 1000 training instances in each language pair as seed data.

Note that IME requires backward transliteration from English/Roman script to other languages, whereas MT and IR are benefitted by NE transliteration. However, NE transliteration data is not particularly useful for training general purpose backward or forward transliteration engines. This is because usually there are standard spellings for NEs in a language, which limit the extent of variation in transliteration as compared to open domain all-word transliteration used in IME. We are not aware of any previous work on mining general domain all-word transliteration pairs from the Web. Sowmya et al. (2010) described a set of controlled user experiments through which approximately 25000 transliteration pairs were collected each for Hindi, Bangla and Telugu on one hand and English on the other. This data has 6090 unique Hindi-English transliteration pairs. Around 10000 English-Hindi NE transliteration pairs were released during NEWS 2009 transliteration generation shared task (Li and Kumaran, 2009).

We are not aware of any previous work on mining domain-independent all-word transliteration pairs from the Web. Nevertheless, there are two known datasets for training/testing of English-Hindi transliteration systems. Sowmya et al. (2010) described a set of controlled user experiments through which approximately 25000 transliteration pairs were collected each for Hindi, Bangla and Telugu on one hand and English on the other. This data has 6090 unique Hindi-English transliteration pairs. In another attempt to create such data, around 10000 English-Hindi NE transliteration pairs were released during NEWS 2009 transliteration generation shared task (Li and Kumaran, 2009).

3. Mining Hindi Lyrics Corpus

We mine Hindi-English transliteration pairs from online Hindi song lyrics. The data is intended for training Hindi-to-English [2] forward and English-to-Hindi backward transliteration engines. Figure 1 shows the schematic of our approach. There are three major steps involved: creation of a corpus of Devanagari and Roman song lyrics by mining the Web, alignment of the Devanagari and Roman songs, and finally, alignment at the word level between Devanagari lyrics and their Roman transliterations, which in turn generates the Hindi-English transliteration pairs.

3.1 Crawling of song lyrics

Bollywood[3] or the Mumbai-based Hindi film industry produces the largest number of movies in the world every year. Approximately 1000 movies, including documentaries and non-commercial films, are produced every year. Almost all Bollywood movies feature several songs. These songs are very popular across the globe, and in India they are the most searched items[4]. There are several popular websites that collect and host Bollywood song lyrics along with online radios and videos.

We crawled seven popular lyrics websites and collected 21519 and 51686 song lyrics, and 3.3 and 9 million words in Devanagari and Roman scripts respectively. Table 1 lists the websites and number of songs and words collected from those. We observed that the Roman

[2] Since Hindi is written in Devanagari script, and English in the Roman script, often we will refer to the Hindi words as Devanagari words and the English ones as the Roman transliterations, or just Roman words for short.

[3] http://en.wikipedia.org/wiki/Bollywood

[4] http://www.google.com/intl/en/press/zeitgeist2010/regions/ in.html

transliterations of the songs present on the two websites, viz. smriti.com and lyricsindia.net, were in ITRANS (Chopde, 2009), which were generated automatically from the Devanagari lyrics. These songs are not useful for extracting transliteration data because they are not natural transliterations, but transcodings. Therefore, we removed those from the corpus and for further analysis, only the remaining 27391 songs in Roman script have been considered. The corpus has 34640 and 74094 unique words in Devanagari and Roman scripts respectively.

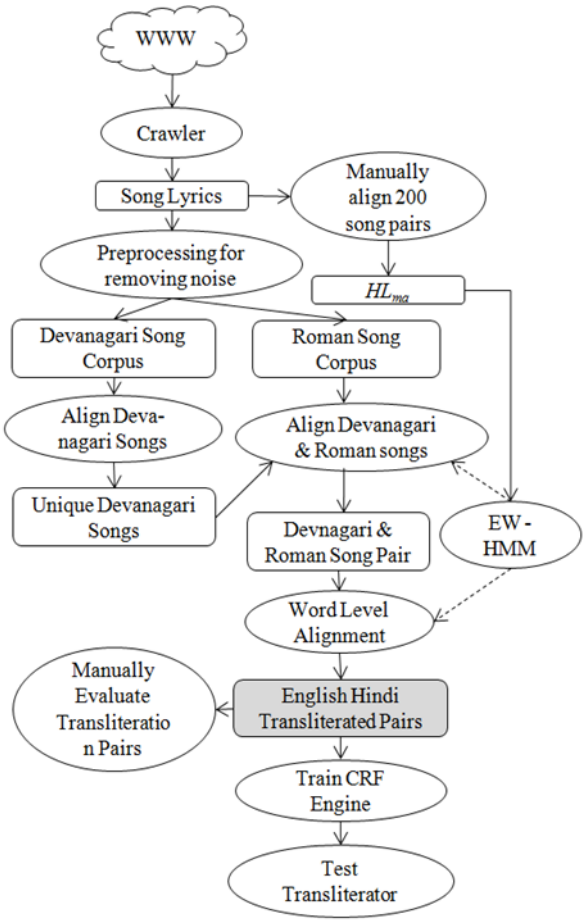

Figure 1: Schematic of the approach

200 unique Devanagari songs were randomly chosen from the corpus along with their Roman transliterations. These songs were manually aligned at the word level to extract 4651 unique Hindi-English transliteration pairs consisting of 3834 unique Hindi words. This data, which we shall refer to as HL_{ma}, has been used as a seed corpus to train the initial classifier.

3.2 Preprocessing for noise removal

Manual inspection of the lyrics revealed the existence of various kinds of noise in the data that must be removed or normalized across the songs before attempting for song or word level alignments. The most common types of noise are:

a) Transcriptions of solfeggio and vocalizations, which have no standard representation (e.g., hoolalala, hoooo lalala, hoo la la la);

b) Repetitions of lines or phrases in the song are sometimes explicitly mentioned (e.g., dum dar dum dar jasn jasn dum**,** dum dar dum dar jasn jasn dum), sometimes mentioned only once with an integer denoting the number of repetitions (e.g., dum dar dum dar jasn jasn dum – 2), sometimes with ellipses (dum dar dum dar jasn jasn dum**,** dum dar …), and sometimes repetitions are altogether omitted;

c) Line-breaks, punctuations and even paragraph-breaks can vary widely across the different versions of the same lyrics;

d) Unintentional errors, such as spelling mistakes and omission of spaces, are also quite frequent.

	Roman		Devanagari	
Websites	**song**	**word**	**song**	**word**
smriti.com	9.7	1.77	9.7	1.77
lyricsindia.net	11.6	1.57	11.6	1.53
10lyrics.com	5.1	0.89	0.2	0.03
lyricsmasti.com	6.6	1.64		
hindilyrix.com	11.4	2.03		
musicmaza.com	4.2	0.72		
giitaayan.com	2.9	0.45		
Total	**51.7**	**9.00**	**21.5**	**3.34**

Table 1: List of websites crawled and number of songs ($\times 10^3$) and words ($\times 10^6$) collected

The first three types of noise create the most serious problems for song and word level alignments. Non-standard transcriptions for vocalization etc. are quite difficult to standardize. However, as a pre-processing step, we remove the other two types of noise from the corpus as follows:

- We check for exact repetition of lines within a song. For all such repetitions, only the first occurrence of a line is retained; all other occurrences are deleted from the lyrics.
- All lines that are valid prefixes of some other line within a song are deleted.
- After executing the above two steps in succession, we replace all punctuations, line-breaks and paragraph-breaks within the song by spaces.

This preprocessed set of songs has been used for further analysis.

4. Alignment at Song Level

Headings Since the songs have been crawled from different and unrelated websites, we expect several versions of the same song to be present in both Devanagari and Roman scripts. It is not possible to identify different versions of the same songs by comparing the song titles, because titles may widely vary

across websites. For instance, the same song is refered to as "*Dheemi Dheemi*" in one website, "*dhiimii dhiimii bhiinii bhiinii*" in two other websites, and "*Dheemi Dheemi Khushboo Hai Tera Badan*" on a fourth website. Sometime two different songs might also have the same title. For instance, there are two different songs from the movies "*Chandni Chowk to China*", and "*My Name is Khan*" with the same title "*Tere naina*".

A direct comparison of first few words of the songs is also an unreliable strategy because, more often than not, songs begin with vocalizations or solfeggios which are very noisy. Approximate string matching approaches, such as edit distance based similarity metric or use of classifier for identifying transliteration equivalents (as in Klementiev and Roth, 2006), are also impractical because in order to identify possible matches we need to compare each song to 50000 other songs. Approximate string matching algorithms or classifiers are too slow to scale up for such large number of long strings.

Note that, after preprocessing of the lyrics, different Devanagari versions of the same song are expected to be identical except for, perhaps, the presence of some noise of the first and fourth kind. On the other hand, their Roman counterparts can still be significantly different from each other due to natural spelling variations generated during forward transliteration. It is important as well as useful for us to capture these variations. Therefore, we break the problem of song alignment into the following two sub-problems: First, we identify all the Devanagari versions of the same song. Since they are expected to be identical, we retain only one of the versions for further processing. The second sub-problem is to align the Roman songs to one of the unique Devanagari songs discovered in the previous step.

4.1 Aligning Devanagari Songs

Since word level comparison between every pair of songs is inefficient and ineffective, for every Devanagari song we first identify a small subset of other songs that could possibly align to it. This subset is computed as follows: Taking idea from document representation and comparison strategies in IR, we define a vector space model for representing Devanagari songs, where a song S is represented by a 1000 dimensional vector $\mathbf{v}_S$, such that

$$\mathbf{v}_S[i] = count(S, w_{i+50})/length(S)$$

Here, $count(S, w_{i+50})$ is the number of occurrences of the $(i + 50)th$ most frequent word of the corpus in S, and $length$(S) is the number of words in S. The offset 50 is added to the index i because we consider the 50 most frequent words in the corpus as stop words.

The similarity between two songs is defined as the cosine of the angle between their vector representations. Since cosine computation is very fast, we are able to compute the similarity between every pair of Devanagari songs. We observed that the cosine similarity between two versions of the same song is always greater than 0.9, though a very high cosine similarity need not always indicate that the songs are identical. Nevertheless, for most of the songs, there were very few (<6) candidates with cosine similarity >0.9; this allowed us to compute word level edit distances between a song and all its potential matches. The costs of word insertion, deletion or substitution were all set to 1. Two words were considered equal if they matched exactly. Thus, for example, the word level edit distance between एक मैं और एक तू and एक राधा ओ एक मीरा is 3. Two songs S and S' were declared identical if their word level edit distance was less than $(length(S) + length(S'))/4$. Through this process we discovered 10397 unique Devanagari songs; this implies that there are on an average 2.15 versions of each Devanagari song in the corpus.

All the assumptions and parameters made here ensured a high precision for song alignment, possibly compromising recall. This is important because by aligning two unrelated Devanagari songs, we might lose one unique candidate from the dataset, which is unacceptable. However, on the other hand, if two versions of the same song are not aligned then some of the computations in the subsequent steps will be repeated unnecessarily, but this will not affect the quality or quantity of the mined data.

4.2 Aligning Roman Songs

Initially we tried a vector space representation approach for aligning the Roman songs to one of the unique Devanagari songs. Recall that each Devanagari song is represented by a 1000 dimensional vector, where the projection of $\mathbf{v}_S$ on the ith dimension is the normalized frequency of the word w_{i+50} in S. In order to map the Roman songs into the same vector space, we need to identify the transliterations of w_{i+50}'s in the Roman songs. We used an EW-HMM model (Udupa et al., 2009) trained on the seed corpus HL_{ma} to spot the transliteration equivalents of w_{i+50}'s. It was assumed that a Roman word r is a transliteration equivalent of w_{i+50} if EW-HMM(r, w_{i+50}) > t, where t is a user defined threshold. Please refer to Sec. 5.1 for the details of EW-HMM training and threshold selection. Suppose, through this process $r_{i,1}$, $r_{i,2}$, …, and $r_{i,ki}$ are identified as possible transliterations of w_{i+50}, then the vector corresponding to a Roman song S' is defined as

$$\mathbf{v}_{S'}[i] = [\Sigma_{j=1 \text{ to } ki}\ count(S', r_{i,j})]/length(S')$$

In an ideal situation, where every Roman transliteration is derived from a unique Hindi word and where there is no other noise in the data, it is easy to show that if S' is a transliteration of S, then $\mathbf{v}_S = \mathbf{v}_{S'}$. However, as we shall see in Sec. 5.1, the output of the EW-HMM classifier is very noisy for all values of t. A large number of transliteration equivalents, $r_{i,1}$ *to* $r_{i,ki}$, were obtained for every Devanagari word, most of which were actually not a correct transliteration. Consequently, the cosine similarities between the vectors of Roman and

Devanagari songs were distributed quite evenly between 0 and 0.5, and it was impossible to choose a reliable subset of Devanagari transliterations for each Roman song. We also tried to further re-rank the $r_{i,j}$'s by comparing bigram and trigram frequency distribution of w_{i+50} and $r_{i,j}$'s in the Devanagari and Roman songs, but it did not improve the results.

The technique that finally worked is based on a heuristic hash function that maps a Devanagari/Roman song to a string of n Devanagari/Roman characters. This string is generated by concatenating up to n first letters of each word in a song, except for those beginning with vowels, ल /*l* and ह /*h*. Vowels, *h* and *l* are included in the string because their transliterations are very noisy and often the vocalizations begin with these letters. For example, if n = 5, then the song "*Hoo lalala Hoo lalalalala lalala Oh ho hoo lalala Ek bagiya mein rehti hai ek maina Poochhti hai ki bolo kya hai kehna*" will be mapped to the string "*bmrmp*". We set n to 20.

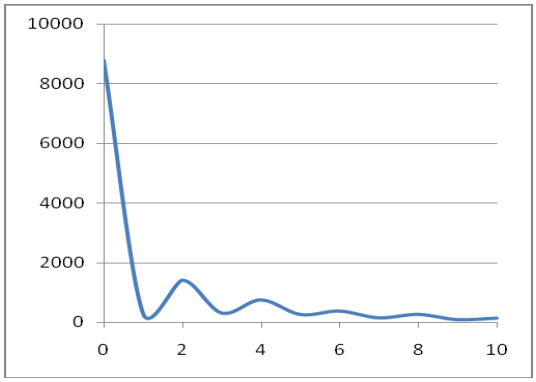

Figure 2: Plot of edit-distances between the strings (x-axis) vs. number of song pairs (y-axis)

The strings for the Roman songs are then compared against those for the Devanagari songs using the standard edit distance algorithm, where cost of deletion, insertion and substitution were all set to 1. For every Devanagari letter (e.g., क) we manually created a list of Roman letters (e.g., *k*, *c*, *q*, *x*) that were considered transliteration equivalents.

Figure 2 shows the distribution of the edit distances.We observe that for large number of Devanagari-Roman song pairs, this value is 0; we also observed that the edit distance was always within 10 for pairs of songs which were real transliterations of each other. Hence, for every Roman song, we extracted all the Devanagari songs for which the edit distance between the hashed strings were within 10. If this list was long, we chose up to 10 closest matches. Each Roman song was then compared against this list of Devanagari songs at word level. This process ofword level comparison, which not only allows us to identify matching songs, but as a byproduct also generates the word-level alignments, is presented in next section.

5. Word-level Alignment

Alignment of the words between a Devanagari song and its (possible) Roman transliterated version has been carried out using an edit distance based approximate string matching algorithm, where each words is treated as a character. The similarity between two words is measured using a HMM-based classifier for identifying transliteration equivalents.

5.1 Classifier for identifying transliteration equivalents

Udupa et al. (2009) described the use of EW-HMM for generating hidden alignments between character sequences of a word and its transliterations, and subsequently using it as a classifier for identifying transliteration pairs. We train an implementation of the same EW-HMM on the HL_{ma} dataset and another 3654 unique pairs from (Sowmya et al., 2010) data. The remaining 2436 Hindi-English transliteration pairs in (Sowmya et al., 2010) data, which we shall refer to as SD_{test}, has been used for testing. Figure 3 shows the distribution of the confidence scores output by the EW-HMM, which ranges from 0 to 4, on the SD_{test} data. Ideally, since all the pairs in the test set are valid transliteration equivalents, the system should output a very high confidence score (close to 4). However, we observe that 40% of the valid pairs have scores less than 2.5. On the other hand, sometimes the system returns very high score for word pairs which are not really transliteration equivalents. Therefore, it is tricky to choose a threshold t above which a pair can be considered as valid transliteration equivalents. For our mining task, we set t to 2.5. Thus, the recall of the system is expected be around 60%. The precision of the classifier was found to be around 80%.

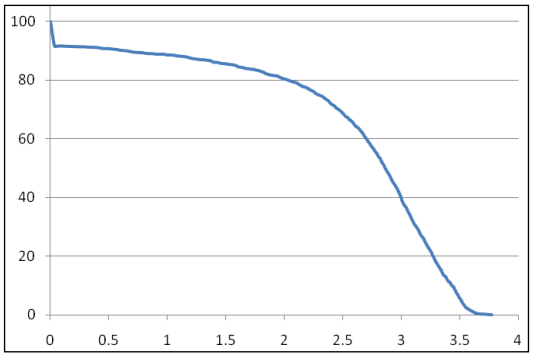

Figure 3: Plot of confidence score from EW-HMM (x-axis) vs. % of word pairs with a score greater than or equal to that value (y-axis)

5.2 Approximate string-matching algorithm

We use the same edit distance algorithm as was used for aligning Devanagari songs (Sec 4.1); however, here two words are assumed to match if their EW-HMM score is greater than 2.5. The alignment algorithm is run between a Roman song and the corresponding Devanagari transliterations identified by the heuristic algorithm (Sec. 4.2). We assume a Roman song *S'* to be the transliteration of a Devanagari song *S* if the edit distance between them is less than (*length*(*S*) + *length*(*S'*))/4. We observe that this