
Deepak Kumar Gupta  
Comp. Sc. & Engg. Deptt.  
IIT Patna, India  
deepak.mtmc13@iitp.ac.in

Shubham Kumar  
Comp. Sc. & Engg. Deptt.  
IIT Patna, India  
shubham.ee12@iitp.ac.in

Asif Ekbal  
Comp. Sc. & Engg. Deptt.  
IIT Patna, India  
asif@iitp.ac.in

ABSTRACT
In this paper, we describe the system that we developed as part of our participation to the FIRE-2014 Shared Task on Transliterated Search. We participated only for Subtask 1 that focused on labeling the query words. The entire process consists of the following subtasks: language identification of each word in the text, named entity recognition and classification (NERC) and transliteration of the Indian language words written in non-native scripts to the corresponding native Indic scripts. The proposed methods of language identification and NERC are based on the supervised approaches, where we use several machine learning algorithms. We develop a transliteration framework which is based on the modified joint source channel model. Experiments on the benchmark setup show that we achieve quite encouraging performance for both pairs of languages. It is also to be noted that we did not make use of any deep domain-specific resources and/or tools, and therefore this can be easily adapted to the other domains and/or languages.

Keywords
Language Identification, NERC, Transliteration, Ensemble, Modified Joint-Source Channel Model

1. INTRODUCTION
Recent decade has seen an upsurge in the social networking and e-commerce sector witnessing an enormous growth in the volume of data flowing out of media networks, which can be used by public and private organizations alike to gain valuable insights. New forms of communication, such as micro-blogging, Tweets, status, reviews and text messaging have emerged and become ubiquitous. These messages often are written using Roman script due to various socio-cultural and technological reasons [4]. Many languages such as South and South East Asian Languages, Arabic, Russian etc. make use of indigenous scripts while writing in text forms. The process of phonetically representing the words of a language in a non-native script is called transliteration.

Transliteration, especially into Roman script, is used abundantly on the Web not only for documents, but also for user queries that intend to search for these documents. These problems were addressed in the FIRE-2013 Shared Task on Transliteration [5]. More recent studies show that building computational models for the social media content is more challenging because of the nature of the mixing as well as the presence of non-standard variations in spellings and grammar, and transliteration [1]. The work that we present here is in connection to the shared task that is being conducted as an continuation to the previous year.

1.1 Task Description
This year, two subtasks on Transliterated Search were conducted: first one is the query labeling, and the second task is related to the ad-hoc retrieval task for Hindi film lyrics. We participated for the first task which is described very briefly as follows:

Subtask 1: Query Word Labeling
Suppose that a query q: w1 w2 w3 . . . wn is written in the Roman script. The words, w1 w2 etc., could be standard English words or transliterated from another language L. The task is to label the words as E or L depending on whether it is an English word, or a transliterated L-language word. And then, for each transliterated word, provide the correct transliteration in the native script (i.e., the script which is used for writing L). The task also required to identify and classify the named entities of types person, location, organization and abbreviation.

2. METHODOLOGY
The overall task for query labeling consists of three major components, viz. Language Identification, NERC and Transliteration. It is to be noted that we did not make use of any domain-specific resources and/or tools for the sake of their domain-independence. Below we describe the methodologies that we followed for each of these individual modules.

2.1 Language Identification
The problem of language identification concerns with determining the language of a given word. The task can be modeled as a classification problem, where each word has to be labeled either with one of the three classes, namely Hindi (or Bengali), English and Mixed (denotes the mixed characters of English and non-Latin language scripts). Our proposed method for language identification is supervised in nature. In particular we develop the systems based on four
different classifiers, namely random forest, random tree, support vector machine and decision tree. We use the Weka implementations\footnote{http://www.cs.waikato.ac.nz/ml/weka/} for these classifiers. In order to further improve the performance we construct an ensemble by combining the decisions of all the classifiers using majority voting. We followed the similar approach for both Hindi-English and Bangla-English pairs. The features that we implemented for language identification are described below in brief:

1. Character n-gram: Character n-gram is a contiguous sequence of n character extracted from a given word. We extract character n-grams of length one (unigram), two (bigram) and three (trigram), and use these as the features of the classifiers.

2. Context word: Local contexts help to identify the type of the current word. We use the contexts of previous two and next two words as features.

3. Word normalization: Words are normalized in order to capture the similarity between two different words that share some common properties. Each capitalized letter is replaced by ‘A’, small by ‘a’ and number by ‘0’.

4. Gazetteer based feature: We compile a list of Hindi, Bengali and English words from the training datasets. A feature vector of length two (representing the respective gazetteer for the language pair: Hindi-English or Bangla-English). Now for each token we set a feature value equal to ‘1’ if it is present in the respective gazetteer, otherwise ‘0’. Hence for the words that appear in both the gazetteers, the feature vector will take the values of 1 in both the bit positions. Recent studies also suggest that gazetteer based features can be effectively used for the language identification\footnote{http://www.cs.waikato.ac.nz/ml/weka/}.

5. InitCap: This feature checks whether the current token starts with a capital letter.

6. InitPunDigit: We define a binary-valued feature that checks whether the current token starts with a punctuation or digit.

7. DigitAlpha: We define this feature in such a way that checks whether the current token is alphanumeric.

8. Contains# symbol: We define the feature that checks whether the word in the surrounding context contains the symbol #.

Last three features help to recognize the tokens which are mixed in nature (i.e., do not belong to Hindi, English and Bangla). Some of the examples are: 2mar, #lol, (rozana etc.

2.2 Named Entity Recognition and Classification

Named Entity Recognition and Classification (NERC) in an unstructured texts such as facebook, blogs etc. are more challenging compared to the traditional news-wire domains. Here the task was to identify named entities (NEs) and classify them into the following categories: Person, Organization, Location and Abbreviation. We use machine learning model to recognize the first three NE types, and for the last one we used heuristics. It is to be noted that there were many inconsistencies in annotation, and hence we pre-processed the datasets to maintain uniformity. In order to denote the boundary of NEs we use the BIO encoding scheme\footnote{B, I and O denote the beginning, intermediate and outside the NEs.}.

1. Local context: Local contexts that span the preceding and following few tokens of the current word are used as the features. Here we use the previous two and next two tokens as the features.

2. Character n-gram: Similar to the language identification we use n-grams of length up to 5 as the features.

3. Prefix and Suffix: Prefix and suffix of fixed length character sequences (here, 3) are stripped from each token and used as the features of classifier.

4. Word normalization: This feature is defined exactly in the same way as we did for language identification.

5. WordClassFeature: This feature was defined to ensure that the words having similar structures belong to the same class. In the first step we normalize all the words following the process as mentioned above. Thereafter, consecutive same characters are squeezed into a single character. For example, the normalized word AAAaaa is converted to Aa. We found this feature to be effective for the biomedical domain, and we directly adapted this without any modification. Detailed sensitivity analysis might be useful to study its effectiveness for the current domain.

6. Typographic features: We define a set of features depending upon the Typographic constructions of the words. We implement the following four features: AllCaps (whether the current word is made up of all capitalized letters), AllSmall (word is constructed with only uncapitalized characters), InitCap (word starts with a capital letter) and DigitAlpha (word contains digits and alphabets).

2.3 Transliteration

A transliteration system takes as input a character string in the source language and generates a character string in the target language as output. The transliteration algorithm\footnote{[2]} that we used here is conceptualized as two levels of decoding: segmenting source and target language strings into transliteration units (TUs); and defining appropriate mapping between the source and target TUs by resolving different combinations of alignments and unit mappings. The TU is defined based on the regular expression.

For K aligned TUs (X: source TU and T: target TU), we have

\[ P(X, T) = P(x_1, x_2, \ldots, x_k, t_1, t_2, \ldots, t_k) \]
We implement a number of transliteration models that can generate the original Indian language word (i.e., Indic script) from the given English transliteration written in Roman script. The Indic word is divided into TUs that have the pattern C^*M, where C represents a consonant or a consonant or conjunct and M represents the vowel modifier or matra.

We implement a number of transliteration models that can generate the original Indian language word (i.e., Indic script) from the given English transliteration written in Roman script. The Indic word is divided into TUs that have the pattern C^*M, where C represents a consonant or a consonant or conjunct and M represents the vowel modifier or matra. An English word is divided into TUs that have the pattern C^*V^*, where C represents a consonant and V represents a vowel [2]. The process considers contextual information in both the source and target sides in the form of collocated TUs, and compute the probability of transliteration from each source TU to various target candidate TUs and chooses the one with maximum probability. The most appropriate mappings between the source and target TUs are learned automatically from the bilingual training corpus. The training process yields a kind of decision-tree list that outputs for each source TU a set of possible target TUs along with the probability of each decision obtained from a training corpus. The transliteration of the input word is obtained using direct orthographic mapping by identifying the equivalent target TU for each source TU in the input and then placing the target TUs in order. We implemented all the six models as proposed in [2]. Based on some experiments on the held out datasets we selected the following three models to submit our runs:

Model-I: This is a kind of monogram model where no context is considered, i.e. \( P(X,T) = \prod_{k=1}^{K} P(<x,t>_k) \)

Model-II: This model is built by considering next source TU as context.

\[ P(X,T) = \prod_{k=1}^{K} P(<x,t>_k | x_{k+1} \) \]

Model-III: This model incorporates the previous and the next TUs in the source and the previous target TU as the context.

\[ P(X,T) = \prod_{k=1}^{K} P(<x,t>_k | <x,t>_k-1, x_{k+1} \) \]

The overall transliteration process attempts to produce the best output for the given input word using Model-III. If the transliteration is not obtained then we consult Model-II and then Model-I in sequence. If none of these models produces the output then we consider a literal transliteration model developed using a dictionary. This process is shown below:

**Input: Token (t) which is labeled as L in Language identification**

**Output: Transliteration (T) of given token**

1. T <- Model-I (t)
   
   1.1 If T contains null value
   
   1.1.1 T <- Model-I with Alignment (t)

2. T <- Model-II (t)
   
   2.1 T <- Model-III (t)\(^4\)

3. T <- Model-II with Alignment (t)

4. T <- Model-III (t)\(^5\)

\(^3\)Denotes Hindi(H) or Bengali(B)

\(^4\)each model takes input token and divide it into several non native TU and give the native TU for each of them

\(^5\)each model with Alignment takes input token and align the source TU with target TU

3. EXPERIMENTS AND DISCUSSIONS

3.1 Datasets

We submitted runs for the two pair of languages, namely Hindi-English and Bangla-English. For language identification the FIRE 2014 organizers provided three documents for Hindi-English pair and two documents for Bangla-English pair. For each of the language pairs, the individual documents are merged together into a single file for training. The training sets consist of 1,004 (20,658 tokens) and 800 sentences (27,969 tokens) for Hindi-English and Bangla-English language pairs, respectively. The test sets consist of 32,270 and 25,698 tokens for Hindi-English and Bangla-English, respectively. For training of transliteration algorithm we make use of 54,791 Hindi-English and 19,582 Bangla-English parallel examples. Details of these datasets can be found in [6].

3.2 Results and Analysis

In this section we report the results that we obtained for query word labeling. We submitted three runs which are defined as below:

- **Run-1:** For language identification and NERC, we construct the ensembles using majority voting. If the token is labeled as any native language (Hindi or Bangla) then we perform the transliteration for that token.

- **Run-2:** In this run we perform language identification by majority ensemble, and NERC by SMO. The word labeled as native language is transliterated accordingly.

- **Run-3:** In this run both the language identification and NERC are carried out using SMO. Transliteration is done following the same method.

<table>
<thead>
<tr>
<th>Run ID</th>
<th>LF</th>
<th>LR</th>
<th>LF</th>
<th>EP</th>
<th>ER</th>
<th>EP</th>
<th>LA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run-1</td>
<td>0.920</td>
<td>0.843</td>
<td>0.880</td>
<td>0.883</td>
<td>0.352</td>
<td>0.907</td>
<td>0.886</td>
</tr>
<tr>
<td>Run-2</td>
<td>0.922</td>
<td>0.843</td>
<td>0.881</td>
<td>0.884</td>
<td>0.331</td>
<td>0.907</td>
<td>0.886</td>
</tr>
<tr>
<td>Run-3</td>
<td>0.882</td>
<td>0.841</td>
<td>0.861</td>
<td>0.88</td>
<td>0.896</td>
<td>0.888</td>
<td>0.870</td>
</tr>
</tbody>
</table>

**Table 1:** Result for language identification of Bangla-English. Here, LP-Language precision, LR-Language recall, LF-Language FScore, EP-English precision, ER-English recall, EF-English FScore, LA-Labeling accuracy
Table 2: Results for language identification of Hindi-English

<table>
<thead>
<tr>
<th>Run ID</th>
<th>LP</th>
<th>LR</th>
<th>LF</th>
<th>EP</th>
<th>ER</th>
<th>EF</th>
<th>LA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run-1</td>
<td>0.921</td>
<td>0.895</td>
<td>0.905</td>
<td>0.89</td>
<td>0.908</td>
<td>0.899</td>
<td>0.879</td>
</tr>
<tr>
<td>Run-2</td>
<td>0.921</td>
<td>0.893</td>
<td>0.907</td>
<td>0.89</td>
<td>0.908</td>
<td>0.899</td>
<td>0.878</td>
</tr>
<tr>
<td>Run-3</td>
<td>0.905</td>
<td>0.865</td>
<td>0.885</td>
<td>0.86</td>
<td>0.886</td>
<td>0.873</td>
<td>0.857</td>
</tr>
</tbody>
</table>

The systems were evaluated using the metrics as defined in [5]. Overall results for language identification are reported in Table 1 and Table 2 for Bangla-English and Hindi-English pairs, respectively. Evaluation shows that the first two runs achieve quite same level of F-scores. Experimental results for transliteration are reported in Table 3 and Table 4 for Bangla-English and Hindi-English, respectively. Comparisons with the other submitted systems show that we achieve the performance levels which are in the upper side. For Hindi we obtain the precision, recall and F-score of 92.10%, 89.5% and 90.80%, which are very closer to the best performing system (only inferior by less than one point F-score). For English, our system attains the precision, recall and F-score values of 89.00%, 90.80% and 89.90%, respectively. This is lower than the best system only by 0.2% F-score points. For Bangla-English pair our system also performs with impressive F-score values. In a separate experiment we evaluated the model of NERC. We obtain the precision, recall and F-score values of 61.00%, 44.25% and 51.25%, respectively for Hindi-English pair using the ensemble framework. For Bangla-English pair it yields the precision, recall and F-score values of 54.25%, 43.25% and 48.25%, respectively.

Our close investigation to the results show that our model developed for language identification suffers due to very short forms of the words, ambiguities, erroneous words and mixed wordforms. Short words refer to the tokens which are written in their short forms. Ambiguities arise because of the wrong spelling. Also, the alphanumeric wordforms (i.e., mixed) often contribute to the overall errors. The errors are shown in Table 5.

The transliteration model makes most of the errors because of spelling variation (e.g., Pahalaa vs. पहला). Inconsistent annotation is another potential source of errors.

4. CONCLUSION

In this paper we presented a brief overview of the system that we developed as part of our participation in the query labeling subtask of transliteration search. The proposed method classifies the input query words to their native language and back-transliterate non-English words to their original script. We have used several classification techniques for solving the problem of language identification and NERC. For transliteration we have used a modified joint source-channel model. We submitted three runs. Comparisons with the other systems show that our system achieves quite encouraging performance for both the pairs, Hindi-English and Bangla-English.

Our detailed analysis suggest that language identification module suffers most due to the presence of very short wordforms, ambiguities and alphanumeric words. Errors encountered in the transliteration model can be reduced a lot by developing a method for spelling variations.

5. REFERENCES


transliteration systems for Indian languages. Pages 2902–2905.