ABSTRACT
This paper presents the BMSCE team’s participation in ‘FIRE Shared Task on Transliterated Search subtask-1’. Our Language Identification system is based on the n-grams approach and uses a tri-gram language identifier trained over a shared and collected training set to classify the language of document at the word level. We use a rule based approach blended with simple dictionary search to back transliterate the Romanized Kannada word. We participated in the Bengali-English, Guajarati-English, Kannada-English, Malayalam-English and Tamil-English language pairs and reverse transliteration for Kannada. We use n-grams to identify the label of the language and a rule based approach is applied to obtain the back transliterations of Romanized Kannada text.

In n-grams approach, we use the tri-gram approach proposed by Canvar [4] to identify the language of the word. We make use of a dictionary based approach to back transliterate a word into its native script. Approximate string matching algorithms have also been employed to select the best matching transliteration.

2. LANGUAGE IDENTIFICATION
2.1 Data Set
The training data with the words being annotated with their language was given as a part of the subtask. However, since languages like Kannada and Tamil did not have sufficient amount of training data, for each language, we collected monolingual corpora for English and the Indic languages from several Wikipedia pages, Facebook public pages and different websites and blogs. Only the specific language Unicode was extracted from these websites by indicating the range of valid Unicode block for that language. For example, while collecting monolingual corpora for Hindi, Unicode block for Guajarati was specified to be \u0A81-\u0AF3 and \u0AF3-\u0AF1. By this method, the word embedding of other languages, punctuations and symbols was avoided in the training data. Table 1 shows the amount of training data was used.

<table>
<thead>
<tr>
<th>Language Pair</th>
<th>English Word Count</th>
<th>Indic Language Word Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bengali-English</td>
<td>9645</td>
<td>22661</td>
</tr>
<tr>
<td>Guajarati-English</td>
<td>9645</td>
<td>9352</td>
</tr>
<tr>
<td>Kannada-English</td>
<td>25452</td>
<td>31899</td>
</tr>
<tr>
<td>Malayalam-English</td>
<td>9645</td>
<td>22607</td>
</tr>
<tr>
<td>Tamil-English</td>
<td>9645</td>
<td>10967</td>
</tr>
</tbody>
</table>

2.2 Normalization
As Indic languages can be romanized in many ways, the user content is often subject to variations in spellings. Therefore, the collected documents in the native language were romanized to a standard form by using a Python module called unidecode. Since
the module maps each native language Unicode to an alphabet in English, the vowel following immediately after the consonant is deleted. For instance, க in Tamil becomes k instead of ka. Such problems were handled with a specific Python script.

2.3 Implementation
The standardized data is then used as the training data and a model is created by using tri-gram Language Identifier. All unigrams are labelled to be English words. A set for most frequent bigrams for every language was maintained and words were labelled accordingly.

3. Back Transliteration
3.1 Data Set
We back transliterated the romanized native language words only for Kannada-English word pairs. A list of all the unique words in the corresponding language file was maintained and this dictionary was referred during the back transliteration procedure. The native language of the words was preserved. A dictionary of 3709 words was employed.

3.2 Implementation
We use handcrafted rules to map the roman alphabets to Kannada Unicode characters which give a coarsely transliterated word. The raw transliteration will then be fine-tuned using the dictionary approach. Table 2 shows the algorithm for back transliterating the romanized Kannada word. rawTransliterate() is a method that implements the crude transliteration of the word based on the pre-determined mapping.

As variations in the spellings while transliterating a Kannada text into English is common, we try to recreate a possible list of words that the user would have intended. For example, the user would have intended the word kADu but would have transliterated word as kadu. To tackle this problem, we capitalize the ‘t’s and ‘d’s as the spelling variations are frequent for these consonants. The resulting words are stored in a list. The vowels within the words of the list are capitalized, one at a time and the resulting words are added to the list. The list obtained by the end of all the iterations, represents a possible set of words the user had meant.

An approximation string matching algorithm is then utilized to obtain a set of close matches in the dictionary with the specified or higher probability. We use the get_close_matches() method from the difflib module in Python. If no close match is found, the word itself is assumed to be the close match. The best match is obtained by calculating the fuzzy ratio of the close matches with the coarsely transliterated word and the word with highest fuzzy ratio is chosen as the best match. The longest common sequence between the best match and the raw transliterated word is found. The unmatched part is also found. The matched and unmatched part together is output as the back transliteration of the word.

4. EXPERIMENT RESULTS
A single run for six different language pairs was submitted and the results of language classification for all the language pairs are recapitulated in Table 3.

The back transliteration was implemented only for Kannada-English language pair. Table 4 shows the transliteration accuracy for Romanized Kannada words in the Kannada-English language pair.

5. ERROR ANALYSIS
Tri-gram approach helps identifying the language of the word to some extent. But, the algorithm fails when the word belongs to both the languages. For example, the most frequently used bigrams in English such as me, in, is, us, etc. are also the commonly used bigrams in the Indic languages. The n-grams approach does not consider the context cues such as the previous word and the next word and thus fails to recognize the code mixes in the document. Also, as the tri-gram approach converts the words into lowercase, no useful information can be drawn about the Named Entities. Due to these reasons, the accuracy for the some of the language pairs has decreased. A sequence classifier that considers the context cues would improve the accuracy of the system.
Table 3. Language label analysis for all five language pairs

<table>
<thead>
<tr>
<th>Language Pair</th>
<th>LP</th>
<th>LR</th>
<th>LF</th>
<th>EP</th>
<th>ER</th>
<th>EF</th>
<th>LA</th>
<th>EQMF All</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bengali-English</td>
<td>0.721</td>
<td>0.652</td>
<td>0.685</td>
<td>0.697</td>
<td>0.867</td>
<td>0.773</td>
<td>0.754</td>
<td>0.249</td>
</tr>
<tr>
<td>Gujarati-English</td>
<td>0.985</td>
<td>0.756</td>
<td>0.856</td>
<td>0.037</td>
<td>0.833</td>
<td>0.071</td>
<td>0.746</td>
<td>0.173</td>
</tr>
<tr>
<td>Kannada-English</td>
<td>0.881</td>
<td>0.906</td>
<td>0.894</td>
<td>0.691</td>
<td>0.671</td>
<td>0.681</td>
<td>0.806</td>
<td>0.218</td>
</tr>
<tr>
<td>Malayalam-English</td>
<td>0.865</td>
<td>0.836</td>
<td>0.851</td>
<td>0.48</td>
<td>0.757</td>
<td>0.588</td>
<td>0.696</td>
<td>0.217</td>
</tr>
<tr>
<td>Tamil-English</td>
<td>0.912</td>
<td>0.574</td>
<td>0.705</td>
<td>0.719</td>
<td>0.943</td>
<td>0.816</td>
<td>0.799</td>
<td>0.122</td>
</tr>
</tbody>
</table>

LP - Token level Precision, LR - Recall and LF - F-Measure for the Indic languages in the language pair
EP - Token level Precision, ER - Recall and EF - F-Measure for English in the language pair
EQMF - Exact Query Match Fraction

Table 4. Transliteration accuracy for Kannada-English language pair

<table>
<thead>
<tr>
<th>Language Pair</th>
<th>TP</th>
<th>TR</th>
<th>TF</th>
<th>ETPM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kannada-English</td>
<td>0.512</td>
<td>0.531</td>
<td>0.521</td>
<td>433/732</td>
</tr>
</tbody>
</table>

TP - Token level Precision, TR - Recall, TF - F-Measure for Transliteration
ETPM - Exact Transliteration Pair Match as defined [5]

The back transliterations works well if the word or the closest words exist in the dictionary. But, as languages like Kannada are agglutinative in nature and can have several inflections [6] which will be difficult to implement. Due to the lack of resources the transliteration accuracy is not remarkable. As not all combinations of consonant replacements have been employed, transliteration errors occur frequently. For example, the replacement of the ‘n’s with ‘N’s has not been employed, the user intended kaNNu word cannot be obtained.

6. CONCLUSION AND FUTURE WORK
In this paper, we discussed the n-gram approach to identify the language of the word and a rule based approach to back transliterate a romanized Kannada word to its native script. Although a tri-gram based language identifier gives reasonable accuracy, it fails to use the context cues to identify the language of the word. We plan to implement a sequence based classifier that would classify the word based on the previous and the next word. Instead of using only tri-grams, we improvise on using {1,2,3,4,5,6}-grams trained on different machine learning algorithms such as MaxEnt, Naïve Bayes, SVM etc. We also set our sight on developing a model that uses the linguistic theory of Code Switches [7].

Our back transliteration system is prone to errors due to insufficient data. Instead of employing hand crafted rules to transliterate a word, we plan to implement a learned model to determine the mapping of the Roman alphabets to Unicode of the native language.
7. ACKNOWLEDGMENTS
We thank Technical Education Quality Improvement Programme (TEQIP) phase II, BMSCE and State Project Facilitation Unit (SPFU), Karnataka for supporting our research work.

8. REFERENCES


