IIIT-H System Submission for FIRE2014 Shared Task on Transliterated Search

Conference Paper · January 2015

DOI: 10.1145/2824864.2824872

6 authors, including:

Irshad Ahmad Bhat
International Institute of Information Technology, Hyderabad
11 PUBLICATIONS **26** CITATIONS
SEE PROFILE

Vandan Mujadia
International Institute of Information Technology, Hyderabad
10 PUBLICATIONS **25** CITATIONS
SEE PROFILE

Aniruddha Tammewar
International Institute of Information Technology, Hyderabad
11 PUBLICATIONS **33** CITATIONS
SEE PROFILE

Riyaz Ahmad Bhat
Sher-i-Kashmir Institute of Medical Sciences
21 PUBLICATIONS **77** CITATIONS
SEE PROFILE

IIIT-H System Submission for FIRE2014 Shared Task on Transliterated Search

Irshad Ahmad Bhat
LTRC, IIIT-H
Hyderabad, Telengana
irshad.bhat@research.iiit.ac.in

Vandan Mujadia
LTRC, IIIT-H
Hyderabad, Telengana
vandan.mujadia@research.iiit.ac.in

Aniruddha Tammewar
LTRC, IIIT-H
Hyderabad, Telengana
uttam.tammewar@research.iiit.ac.in

Riyaz Ahmad Bhat
LTRC, IIIT-H
Hyderabad, Telengana
riyaz.bhat@research.iiit.ac.in

Manish Shrivastava
LTRC, IIIT-H
Hyderabad, Telengana
m.shrivastava@iiit.ac.in

ABSTRACT

This paper describes our submission for *FIRE 2014 Shared Task on Transliterated Search*. The shared task features two sub-tasks: Query word labeling and Mixed-script Ad hoc retrieval for Hindi Song Lyrics.

Query Word Labeling is on token level language identification of query words in code-mixed queries and back-transliteration of identified Indian language words into their native scripts. We have developed letter based language models for the token level language identification of query words and a structured perceptron model for back-transliteration of Indic words.

The second subtask for *Mixed-script Ad hoc retrieval for Hindi Song Lyrics* is to retrieve a ranked list of songs from a corpus of Hindi song lyrics given an input query in Devanagari or transliterated Roman script. We have used edit distance based query expansion and language modeling followed by relevance based re-ranking for the retrieval of relevant Hindi Song lyrics for a given query.

Categories and Subject Descriptors

I.2.7 [**Artificial Intelligence**]: Natural Language Processing—*Language parsing and understanding*

Keywords

Language Identification, Transliteration, Information Retrieval, Language Modeling, Perplexity

1 INTRODUCTION

This paper describes our system[1] [2] for the *FIRE 2014 Shared Task on Transliterated Search*. The shared task features two sub-tasks: *Query word labeling* and *Mixed-script Ad hoc retrieval for Hindi Song Lyrics*. Subtask-I addresses the problem of language identification of query words in code-mixed queries and transliteration of Indic words into their native script. The task focuses on token level language identification in code-mixed queries in English and any of the given 6 Indian languages viz Bengali, Gujarati, Hindi, Malayalam, Tamil and Kannada. The overall task is to 1) label words with any of the following categories: lang1, lang2, mixed, Named Entities (NE), and other, and 2) transliterate identified Indic words into their native scripts. We submit predictions and transliterations for the queries from the following language pairs: Hindi-English, Gujarati-English, Bengali-English, Kannada-English, Malayalam-English and Tamil-English.

Our Language Identification system uses letter based conditional probabilities (henceforth CP); character level n-gram CP for individual words and character level n-gram CP for word-level context for each query word. While for back-transliteration of Indic words, we first transliterate these words to WX-notation[3] (transliteration scheme for representing Indian languages in ASCII) and then use Indic Converter tool-kit[4] (developed in-house) to convert these words to their native script.

The second subtask for *Mixed-script Ad hoc retrieval for Hindi Song Lyrics* is to retrieve a ranked list of songs from a corpus of Hindi song lyrics given an input query in Devanagari or transliterated roman script. The documents used for retrieval may be in Devanagari or in roman script. The task focuses on the retrieval of relevant documents irrespective of the script encoding of query or documents. This requires us to normalize the available documents to a single script; in our case Roman. Further we need to process the query to account for all the possible spelling variations for each query term. We have used edit distance based query expansion and language modeling followed by relevance based re-ranking for the retrieval of relevant Hindi Song lyrics for a given query.

As stated in the shared task description, "A challenge that search engines face while processing transliterated queries and documents is that of extensive spelling variation. For instance, the word **dhanyavad** ('thank you' in Hindi and many other Indian languages) can be written in Roman script as dhanyavaad, dhanyvad, danyavad, danyavaad, dhanyavada, dhanyabad and so on." Further, the complexity of search may increase if faced with multiple information sources having different script encoding. The modern Indian Internet user's query might easily be interspersed with words from

[1] https://github.com/irshadbhat/litcm

[2] https://github.com/vmujadia/Subtask-2-Mixed-script-Ad-hoc-retrieval-for-Hindi-Song-Lyrics

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

[4] https://github.com/irshadbhat/python-converter-indic

FIRE '14, December 05-07, 2014, Bangalore, India

DOI: http://dx.doi.org/10.1145/2824864.2824872

their native languages transliterated in Roman or in native UTF-8 encoded scripts along with English words. Both the problems are tackled separately in this shared task. Subtask-I deals with the identification of language of origin of a query term for a code-mixed query, while Subtask-II deals with the retrieval from information sources encoded in different scripts.

The rest of the paper is organized as follows: In § 2, we discuss the data, our methodology and the results for the Subtask-I. § 3 is dedicated to the Subtask-II. In this section, we discuss our approach towards the task and present our results. We conclude in § 4 with some future directions.

2 QUERY WORD LABELLING

The task is to identify and label the words as English language or a transliterated L-language word and back-transliteration of identified Indic words into their native scripts.

2.1 Data

The Language Identification and Transliteration shared task in the Code-Switched (henceforth CS) data is meant for language identification and transliteration in the following 6 language pairs (henceforth LP) namely, Hindi-English (H-E), Gujarati-English (G-E), Bengali-English (B-E), Tamil-English (T-E), Kannada-English (K-E) and Malayalam-English (M-E). For each language – L the following data sets are provided:

- Monolingual corpora of English, Hindi and Gujarati in their native scripts.
- Word lists with corpus frequencies for English, Hindi, Bengali and Gujarati.
- Word transliteration pairs for Hindi-English, Bengali-English and Gujarati-English which are used for training and testing transliteration systems.

Apart from the above data sets, we further crawled Romanized data for each Indian language from various song lyric websites. The Romanized data is used to build language models for each language. Table 1 shows the training data statistics.

Language	# of Lines	Avg. Sentence Length	Avg. Token Length
Hindi	5,31,558	7.11	6.41
English	10,00,000	20.05	6.04
Gujarati	2,824	6.40	7.42
Tamil	2,06,030	4.65	7.94
Malayalam	1,06,006	4.31	8.95
Bengali	68,388	5.10	5.93
Kannada	25,890	5.17	7.38

Table 1: Statistics of Training Data for Language Modeling

For each language pair a test set containing up to 1000 lines of code-mixed data and a development set containing 100 to 800 lines of code-mixed Gold standard (with true labels and transliterations) data are also provided for prediction and tuning the parameters of the models. Table 2 shows some input queries and desired outputs.

2.2 Methodology

We divided Subtask-1 into two stages, Language Identification and Transliteration. In Language Identification stage, we train separate smoothed n-gram based language models for each language in an LP. We compute the letter based conditional probabilities; character-level CP of individual words and character-level CP of word-level context for each word using these language models at inference time. A linear combination of the two conditional probabilities is then used to predict the labels. For back-transliteration of Indic words to their native scripts we first transliterate these words to WX-notation using a structured perceptron model and then use the Indic-converter tool-kit to convert these words to their native scripts.

2.2.1 Language Identification

The Language Identification task is modelled using letter based n-gram models. We train separate character-level n-gram models on unique words in the training corpora of each language. Character level n-gram models assign CP to a word irrespective of its context (preceding and following words). In order to built a more accurate Language Identification system, it is important to also take word-level context into account. But in our case it will rarely increase the accuracy because of the out-of-vocabulary (OOV) problem due to the rich morphology of the given Indian Languages and the spelling variations of Indic words in Roman script. Suppose a word '*khushboo*' occurred several times in the training corpora of Hindi language along with its various spelling variations like '*khushbo*', '*khushbu*', '*khoshboo*' and '*khushibu*'. If the test case contains the same word but with some other spelling variation like '*khushbuu*', the character level CP of this new word will still be very accurate, even though the character-level language model never saw this word in the training data. But the context based word-level language model will give very low score to this word because it will find this word out of vocabulary. The out of vocabulary words will not only affect their own CP but also the CP of their neighboring words. In case of Indian languages in Roman script, the number of OOV words will be very large, thus degrading the performance of context based word-level language model.

In order to tackle the OOV problem in context based word-level language models, we adapt a different approach to incorporate the word-level context into the language identification system. We train an n-gram character-level language model on the word-level context of a word. Consider the sentence '*piyush sharma ke haseen sapney*'. To compute the context based word-level CP of the word say '*haseen*' with 2-gram history, we merge the resulting n-gram i.e. '*sharma ke haseen*' into a single word – '*sharmakehaseen*' and then use the trained character based language model with word level context to obtain the desired word-level CP.

Using these language models we compute the letter based conditional probabilities for individual words and word-level context of a word and use a linear combination of these two scores to predict labels.

2.2.1.1 N-gram Language Models

Given a word w, we compute the conditional probability corresponding to k[5] classes c_1, c_2, ... , c_k as:

$$p(c_i|w) = \ p(w|c_i) * p(c_i) \quad (1)$$

The prior distribution $p(c)$ of a class is assumed to be uniform. Each training set is used to train a separate letter-based language model to estimate the probability of word w. The language model $p(w)$ is implemented as an n-gram model using the IRSTLM-Toolkit [2] with Kneser-Ney smoothing. The language model is defined as:

[5] In our case value of k is 2 as there are 2 languages in an LP

Input query	Outputs
sachin tendulkar number of centuries	sachin\H tendulkar\H number\E of\E centuries\E
palak paneer recipe	palak\H=पालक paneer\H=पनीर recipe\E
mungeri lal ke haseen sapney	mungeri\H=मुंगेरी lal\H=लाल ke\H=के haseen\H=हसीन sapney\H=सपने
iguazu water fall argentina	iguazu\E water\E fall\E argentina\E

Table 2: Input query with desired outputs for Hindi-English code-mixed queries

$$p(w) = \prod_{i=1}^{n} p(l_i | l_{i-k}^{i-1}) \quad (2)$$

where l is a letter and k is a parameter indicating the amount of context used (e.g., *k=4* means 5-gram model).

2.2.2 Transliteration

Back-transliteration of Indic words is carried out in two steps. First we transliterate these words to WX using a structured perceptron model and then convert WX to the native scripts. WX is a transliteration scheme for representing Indian languages in ASCII. It is a one-to-one UTF-8 to ASCII mapping for Indian languages and thus does not effect the performance of transliteration system. We use WX to work with ASCII rather than UTF-8 for computational efficiency.

We model the transliteration system as a structure prediction problem with global feature representation. Our transliteration model is basically a second order Hidden Markov Models (SHMM) formally represented in Equation 3. We denote the sequence of letters in a word in source script as boldface **s** and the sequence of hidden states which correspond to letter sequences in the target script as boldface **t**. A basic HMM model has the following parameters:

$$P(\mathbf{s}; \mathbf{t}) = \arg\max_{t_1 \cdots t_n} \prod_{i=1}^{n} \underbrace{P(t_i | t_{i-1}, t_{i-2})}_{\text{Transition Probabilities}} \underbrace{P(s_i | t_i)}_{\text{Emission Probabilities}} \quad (3)$$

where $s_i \cdots s_n$ is a letter sequence in the source script, and $t_i \cdots t_n$ is the corresponding letter sequence in the target script.

Instead of maximum likelihood estimates, we use structured perceptron of Collins [1] to learn the model parameters. With structured perceptron local contextual features can be made relevant to the whole sequence of target letters, since the parameters are updated with respect to the whole sequence instead of a single letter or observation. It also allows us to use feature based emissions. We replace the basic multinomial emissions $P(s_i|t_i)$ with the feature based emissions $(\theta \cdot f(s,t))$; where $f(s,t)$ is a feature function and θ are the model parameters. The feature template used to learn the emissions is shown in Table 3. We use second-order viterbi to decode the best letter sequence in the target script while learning the parameters as well as at the time of testing.

2.2.2.1 Character Alignment

Like any other supervised machine learning approach, supervised machine transliteration requires a strong list of transliteration pairs to learn its model parameters. We use the provided transliteration pairs for building the transliteration system. We character align the transliteration pairs for training and testing the transliteration models. We use Giza++ tool-kit [6] for the alignment task. Giza++ produces three types of alignments from the transliteration pairs: $1 \rightarrow 1$, $1 \rightarrow Many$ and $1 \rightarrow \emptyset$. There can also be $\emptyset \rightarrow 1$ alignments where target string characters are left unaligned. Out of these four letter alignments, we modified the $\emptyset \rightarrow 1$ alignments. Keeping these alignments as such at the training time would mean we have to introduce $\emptyset$ in the test strings before decoding. We modify these alignments by merging the target character with the previous aligned pair if it is not the first character, otherwise it is merged with the succeeding aligned character pair. Table 4 shows a sample source to target alignments before and after the merging.

Ngram	Features
Unigrams	$l_{i-4}; l_{i-3}; l_{i-2}; l_{i-1}; l_i; l_{i+1}; l_{i+2}; l_{i+3}; l_{i+4}$
Bigrams	$l_{i-4}l_{i-3}; l_{i-3}l_{i-2}; l_{i-2}l_{i-1}; l_{i-1}l_i; l_il_{i+1}; l_{i+1}l_{i+2}; l_{i+2}l_{i+3}; l_{i+3}l_{i+4}$
Trigrams	$l_{i-4}l_{i-3}l_{i-2}; l_{i-3}l_{i-2}l_{i-1}; l_{i-2}l_{i-1}l_i; l_il_{i+1}l_{i+2}; l_{i+1}l_{i+2}l_{i+3}; l_{i+2}l_{i+3}l_{i+4};$
Tetragrams	$l_{i-4}l_{i-3}l_{i-2}l_{i-1}; l_{i-3}l_{i-2}l_{i-1}l_i; l_{i-2}l_{i-1}l_il_{i+1}; l_{i-1}l_il_{i+1}l_{i+2}; l_il_{i+1}l_{i+2}l_{i+3}; l_{i+1}l_{i+2}l_{i+3}l_{i+4};$

Table 3: Feature template used for learning the emission parameters.

(a)	Roman (Hindi)	a	a	s	h	i	r	v	a	d	∅
	WX (Hindi)	A	∅	S	∅	i	r	v	a	d	a
(b)	Roman (Hindi)	a	a	s	h	i	r	v	a	d	
	WX (Hindi)	A	∅	S	∅	i	r	v	a	da	
(a)	Roman (Hindi)	∅	u	n	a	n	i				
	WX (Hindi)	y	U	n	A	n	I				
(b)	Roman (Hindi)		u	n	a	n	i				
	WX (Hindi)		yU	n	A	n	I				

Table 4: Roman (Hindi) → WX (Hindi) alignment pairs for transliteration: (a) before merging and (b) after merging.

2.3 Experiments

The stages mentioned in Section § 2.2 are used for the language identification and transliteration task for each language in an LP. In order to predict correct labels for Language Identification, we trained 7 character-level (5-gram) language models (one for each language) on unique words of a language and 7 letter based (5-gram) language models with word-level context (5-gram) of a word. We computed the character-level 5-gram probabilities of individual words and character-level 5-gram probabilities of word-level context (5-gram) of each word for each language in an LP. A simple linear combination of these two probabilities with equal weights (unit coefficients) is used to predict the labels. Unit coefficients are used because of the unavailability of annotated data to learn them leaving scope for further improvement. Table 5 shows language identification accuracy on Gold standard development data for each LP.

We carried out a series of experiments to learn the best feature template for learning the emission parameters in structured perceptron. We transliterated Indic words to WX using the structured perceptron model and then converted WX to Indian native scripts for each LP, using the Indic-converter tool-kit.

Language Pair → Eval. Metric ↓	Bengali English	Gujarati English	Hindi English	Kannada English	Malayalam English	Tamil English
LP	0.835	0.986	0.83	0.939	0.895	0.983
LR	0.83	0.868	0.749	0.926	0.963	0.987
LF	0.833	0.923	0.787	0.932	0.928	0.985
EP	0.819	0.078	0.718	0.804	0.796	0.991
ER	0.907	1	0.887	0.911	0.934	0.98
EF	0.861	0.145	0.794	0.854	0.86	0.986
TP	0.011	0.28	0.074	0	0.095	0
TR	0.181	0.243	0.357	0	0.102	0
TF	0.021	0.261	0.122	0	0.098	0
LA	0.85	0.856	0.792	0.9	0.891	0.986
EQMF All(NT)	0.383	0.387	0.143	0.429	0.383	0.714
EQMF–NE(NT)	0.479	0.413	0.255	0.555	0.525	0.714
EQMF–Mix(NT)	0.383	0.387	0.143	0.437	0.492	0.714
EQMF–Mix and NE(NT)	0.479	0.413	0.255	0.563	0.675	0.714
EQMF All	0.004	0.007	0.001	0	0.008	0
EQMF–NE	0.004	0.007	0.001	0	0.008	0
EQMF–Mix	0.004	0.007	0.001	0	0.008	0
EQMF–Mix and NE	0.004	0.007	0.001	0	0.008	0
ETPM	72/288	259/911	907/2004	0/751	90/852	0/0

Table 6: Token Level Results [a]

[a] LP, LR, LF: Token level precision, recall and F-measure for the Indian language in the language pair. EP, ER, EF: Token level precision, recall and F-measure for English tokens. TP, TR, TF: Token level transliteration precision, recall, and F-measure. LA: Token level language labeling accuracy. EQMF: Exact query match fraction. – : without transliteration. ETPM: Exact transliterated pair match

Language Pair	Accuracy (%)
Hindi-English	88.08
Bengali-English	90.02
Gujarati-English	89.34
Tamil-English	92.53
Telugu-English	92.27
Kannada-English	91.89

Table 5: Language Identification accuracies on development data

2.4 Results and Discussion

Each language identification and transliteration system is evaluated against Testing data as mentioned in § 2.1. All the results are provided on token level. Tables 6 show results of our language identification and transliteration system.

Systems are evaluated separately for each tag in an LP using *Recall*, *Precision* and *F1-Score* as the measures for the Indian language in the language pair, English tokens and transliteration. Systems are also evaluated on token level language labeling accuracy using the relation *'correct label pairs/(correct label pairs + incorrect label pairs)'*. Evaluation is also performed on EQMF (Exact query match fraction), EQMF but only considering language identification and ETPM (Exact transliterated pair match).

3 HINDI SONG LYRICS RETRIEVAL

This section describes our system for the Subtask-II which is about the Hindi song lyrics retrieval. The proposed system is similar to that of Gupta et al.[3]. They have used Apache Lucene IR platform for building their system, while we build our system from the scratch following a typical architecture of a search engine. The overall system working and design is depicted in Figure 1. As with any search engine, our system also hinges on the posting lists normalized using TF-IDF metric. The posting lists are created based on the structure of the provided songs lyrics documents.

As is stated in the Subtask-II, input query to the system can be given in Devanagari script or it may be in transliterated Roman script with a possible spelling variation, e.g., repeated letters, recurring syllables etc. Also, the documents (∼60000) that are provided for indexing contain lyrics both in Devanagari and Roman scripts with some similar noise. Prior to document indexing, we had to run through a preprocessing step in order to tackle these issues. In the following, we discuss the process of data cleaning and data normalization.

3.1 Data Normalization

Data normalization is an important part of our retrieval system. It takes cares of noise that could affect our results. The goal of normalization was to make the song lyrics uniform across documents and to clean up the unnecessary and unwanted content like HTML/XML tags, punctuation marks, unwanted symbols etc.

In some of the documents, song lyrics contain more information about the pronunciation like **jahaa.N bharam nahii.n aur chaah nahii.n**, where symbol "**.**" signifies some phonetic value of it surrounding letters. In such documents even the case of a letter has a phonetic essence. However, there are only few files with such additional information. The other documents may represent the word **jahaa.N** simply as **jaha** or **jahaan**. We also observed that "D" mostly corresponds to "dh" in normal script, but the mapping from its uppercase to lowercase does not always need an insertion of an additional "h" following it (due to spell variations). So, we decided to ignore this information and removed all the special symbols and converted the whole data into lowercase. As we already mentioned, the training documents are in both Devanagari and Roman scripts, for simplicity we convert all the Devanagari lyrics into Roman using the transliteration tool discussed in § 2.2.2.

The next step of the conversion handles the problem of characters in a word repeating multiple times depending on the writing styles. Consider the following two queries **had kar di ap ne** vs **hadd kar dii aap ne**, both these queries are same but their writing pattern is different. In order to handle such variation, we replace the letters which appear continuously with a single character representing them e.g. "$mann$" → "man"; "$jahaa$" → "$jaha$". This also helps when some word in a song is elongated at the end e.g. "saara jahaaaaaan", but in the lyrics the elongation may not be present **saara jahaan**. So, to match these two, we converge both of them

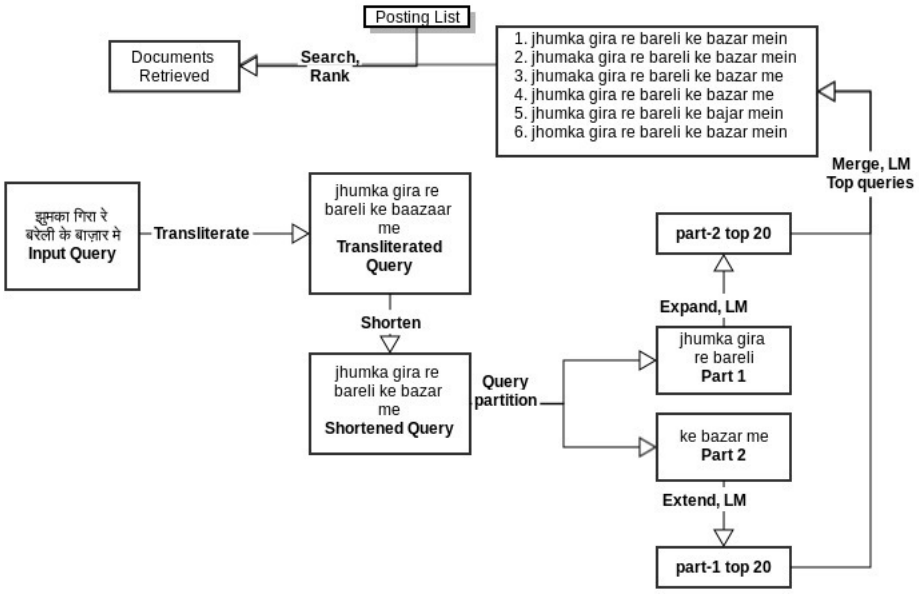

Figure 1: Mixed-script Ad hoc retrieval for Hindi Song Lyrics System Work flow

to **sara jahan**.

3.2 Posting list and Relevancy

As we mention in the previous section, we transliterated all the lyrics documents uniformly to Roman script. Once transliterated, these documents are then used to create posting list. In order to create the posting list, we first identify the structure of these documents and decide appropriate weights for each position in the structure. The terms in the title, for example, will receive the highest weight, while the terms in the body will receive relatively lower weights. In posting list, we have indexed the following patterns separately:

- title of the song,
- first line of song,
- first line of each stanza,
- line with specific singer name,
- each line of chorus,
- song line which has number at last(indication for no. of times repetition), and
- all other remaining lines.

The appropriate weights to these patterns are assigned in the following order: *Title of the song > First line of song > First line of stanzas > Each line of chorus > Song line which has number at last (indication for number of times repetition) > line with specific singer name > All other remaining lines of the song*. This weighted factors help us to compute effective relevance.

The identification and extraction of title field was quite trivial, as there are very few variations in the title field of each song in the corpus. There were, however, huge variations in other patterns or fields across documents like starting of new stanzas, prominent singer name and number of times each line of a song is repeated. Finally, the term-documents frequency counts in the posting list are normalized using TF-IDF metric, which is a standard metric in IR to assign more mass to less frequent words [4].

3.3 Query Expansion

Query expansion (QE) is the process of reformulating a seed query to improve retrieval performance in information retrieval operations [8]. In the context of Indian language based transliterated song lyrics retrieval, query expansion includes identifying script of seed query and expanding it in terms of spelling variation to improve the recall of the retrieval system.

As explained in the task description, there might be several ways of writing the query with same intention. For instance, in Hindi and many other Indian languages, the word **dhanyavad** "thank you" can be written in Roman script as **dhanyavaad**, **dhanyvad**, **danyavad**, **danyavaad**, **dhanyavada**, **dhanyabad** and so on. Such variation will have an adverse effect on the retrieval of relevant documents. So its in important, that such variation should be normalized, which is what is discussed in the next section.

3.3.1 Edit distance for Query expansion

In order to address the above problem, we used Levenshtein distance, a fairly simple and commonly used technique for edit distance. First we created a dictionary of all the unique words present in the converted data. While processing a query, for each word we find all possible words (biased by some rules), within an edit distance of 2, present in the dictionary. To limit the number of possible words, we apply some rules on edit operations which were created by a careful observation of the data. The general form of rules is shown below:

> **If 'x' is present in a word, it can be replaced with 'y' , it can be deleted, or the word can be split at its index.**

We also consider word splitting in this module e.g. **lejayenge** can be split into **le jayenge**. After limiting the in-numerous possibilities by using manually created rules, we still get 30 possibilities for each word on average. If we consider all the possibilities, even if the length of the query is small, say 5 words, the possible variations of such query would grow up to 30^5, which is a very large number to process. To handle this problem, we apply another strat-

egy based on language modeling to limit and rank query expansion. The next section throws details on this strategy.

3.4 Language Modeling

In order to limit the number the possible queries generated using edit distance, we rank them in the order of importance by using scores given by a language model.

3.4.1 Language model

We use SRILM toolkit[7] to train the language Model on cleaned lyrics corpus. We keep the language model loaded, and as we give a sentence to this loaded model, it returns probability and perplexity scores. We use the perplexity score on the generated queries (variations of the original query) to rank them.

To limit the number of new queries and to speed up the process of ranking, we first break the input query into multiple parts. We set the length of splits to be equal to 4 words. As mentioned earlier, we get around 25 to 30 word suggestions on average for each query term by edit distance method. Therefore, the number of possible variations for each split could reach up to 25^4 i.e., around 400K variations for each 4-word part of the query. These generated queries are then finally ranked using perplexity scores using the language model. We observed that the overall scoring and ranking process of around 400K variations takes less than a second. Then, for each part we consider top 20/25/30 variations, depending on the number of parts in the query. Using these top variations from each part, we generated all the possible combinations, which are of length of the actual query. We further rank them using the language model to get the top 20 queries for search and retrieval.

3.5 Search

After applying the query expansion techniques on the test query, we take its top 15-20 spell variations. For each variation, song retrieval system generates most relevant 20 song document ids according to their relevance to the query.

For a given query, our searching module first searches for all the possible n-grams, where $\mathbf{n} \leq$ **length of the query**, in posting list. It then retrieves the results based on cosine distance of the boolean and vector space retrieval model[4]. As per the sub task-2 definition this searching module gives score from 0-4, 4 for title or first line phrase match with given query, 3 for exact phrase match match with given query other than title, 2 for phrase match match with given query, 1 for any keyword match with given query and 0 for irrelevant song.

3.6 Results and Analysis

Table 7 shows the comparative results of 4 teams using various metrics. Although, we tried basic techniques, we could still manage to achieve reasonable results. While doing error analysis, we found that in the data cleaning we could not clean the whole data properly. There were some type of HTML/XML tags that could not be removed as they were attached with their neighboring words. We also observed that some words in query were very different in terms of orthography from the original words in the lyrics, but they sound similar in terms of their pronunciation. This could not be handled by our restrictive (2 edits) edit distance approach. We tried to make as many rules as possible to be used in the edit distance module, but still they were not exhaustive enough to cover all the possibilities. Another improvement in our system would be to use a better Language Modeling toolkit such as RNNLM[5].

TEAM	bits-run-2	iiith-run-1	bit-run-2	dcu-run-2
NDCG@1	0.7708	0.6429	0.6452	0.4143
NDCG@5	0.7954	0.5262	0.4918	0.3933
NDCG@5	0.6977	0.5105	0.4572	0.371
Map	0.6421	0.4346	0.3578	0.2063
MRR	0.8171	0.673	0.6271	0.3979
RECALL	0.6918	0.5806	0.4822	0.2807

Table 7: Subtask-II Results

4 CONCLUSIONS

In this paper, we described our systems for the two subtasks in the Fire Shared task on Transliterated Search. For the Subtask-I on language identification and transliteration, we have implemented a letter based language model for language identification system and a structured perceptron based transliteration system. To summarize, we achieved reasonable accuracy with a linear combination of character level CP and word level CP for language identification system. Structured perceptron model uses the letter alignments learned from GIZA++ tool-kit to transliterate a Romanized Indic word to its native script.

In Mixed-script Ad hoc retrieval task we used very common techniques like edit distance, query expansion, Language modeling and the standard document retrieval algorithms. The most simple yet helpful method was the shortening of words which tackles the problem of recurring characters. Thus our system which uses very basic tricks and methodologies is competitive in terms of the results.

5 References

[1] Michael Collins. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. pages 188–193, 2006.

[2] Marcello Federico, Nicola Bertoldi, and Mauro Cettolo. Irstlm: an open source toolkit for handling large scale language models. In *Interspeech*, pages 1618–1621, 2008.

[3] Parth Gupta, Kalika Bali, Rafael E Banchs, Monojit Choudhury, and Paolo Rosso. Query expansion for mixed-script information retrieval. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*, pages 677–686. ACM, 2014.

[4] Christopher D Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to information retrieval*, volume 1. Cambridge university press Cambridge, 2008.

[5] Tomas Mikolov, Stefan Kombrink, Anoop Deoras, Lukar Burget, and J Cernocky. Rnnlm-recurrent neural network language modeling toolkit. In *Proc. of the 2011 ASRU Workshop*, pages 196–201, 2011.

[6] Franz Josef Och and Hermann Ney. Giza++: Training of statistical translation models, 2000.

[7] Andreas Stolcke et al. Srilm-an extensible language modeling toolkit. In *INTERSPEECH*, 2002.

[8] Olga Vechtomova and Ying Wang. A study of the effect of term proximity on query expansion. *Journal of Information Science*, 32(4):324–333, 2006.