Mixed Script Ad hoc Retrieval using back transliteration and phrase matching through bigram indexing: Shared Task report by BIT, Mesra

Nimesh Ghelani, Sujan Kumar Saha[*] [*], Amit Prakash[†]
Dept. of Computer Science and Engineering
Birla Institute of Technology, Mesra, India
nimeshghelani@gmail.com, sujan.kr.saha@gmail.com, aprakash@bitmesra.ac.in

ABSTRACT

This paper describes an approach for Mixed-script Ad hoc retrieval, a subtask as part of FIRE 2015 Shared Task on Mixed Script Information Retrieval. We participated in subtask 2 of the shared task, where a statistical model was used to carry out back transliteration to Devanagari script. To perform the search, bigram based index of the documents were used and search was performed using pivot terms in the query.

Categories and Subject Descriptors

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

Keywords

transliteration, information retrieval

1. INTRODUCTION

A large number of languages are written using indigenous scripts. Internet has allowed anyone to easily post content to websites, which is usually written in Roman script due technical reasons. This process of phonetically representing the words of a language in a non-native script is called transliteration. The situation, where both documents and queries can be in more than one scripts, and the user expectation could be to retrieve documents across scripts is referred to as Mixed Script Information Retrieval.

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.

Subtask 2 of the shared task focuses on Mixed-script Ad hoc retrieval, where given a query in Roman or Devanagari script, the task was to fetch ranked documents based on their relevance. The documents consisted of mixed script content which are related to song lyrics, movie reviews, or astrology. To solve this task, Devanagari script was chosen as the base script. A back transliteration module was built using a frequency based statistical model on partitioned letter group matching. It was built using the training data of transliterated Devanagari-Roman word pairs. Entire query and documents were back transliterated to Devanagari script using this module. The search was performed on bigram indexes of documents[1]. Spelling variations were handled using LCS (Longest Common Subsequence) based similarity. Pivot terms based on high IDF (Inverse Document Frequency) values[1] were chosen from the query terms. Search was carried around these pivot terms to perform phrase match in documents. Further boosting and reordering of results were performed using heuristics based on intent and document titles.

The rest of the paper discusses the detailed methodology in section 2, followed by the results in section 3 and finally the conclusion in section 4.

2. METHODOLOGY

The proposed method consists of 2 modules: The Back Transliteration module, and the Searching module.

The Transliteration module is used for transliterating words expressed in Roman script to Devanagari script. This is needed because the search module assumes every word in the query and documents to be in Devanagari script. The reason Devanagari is used as the base script over Roman is its concrete implication of phonemes solely from the respective letters. In other words, each phoneme is strongly attached to its corresponding letters rather than the whole word, or the neighboring phonemes. This helps in the Search module which involves computing similarity scores between two similar sounding words.

2.1 Back Transliteration

Back Transliteration is performed using a statistical model which is trained on a list of Devanagari-Roman transliterated word pairs.

2.1.1 Training

Training was performed on a list of 36,947 transliterated Hindi words (Roman script) and their Devanagari representation.[2]

Given a list of Devanagari-Roman word pair, each Roman word W_r is represented as characters $\wedge r_1 r_2 .. r_n \$$. The $\wedge$ and

[*]Advisor
[†]Helped out with result analysis

the \$ are added as characters to mark the beginning and the end of the string, respectively. Subsequent operations will treat these markers just like any other character of the word. This is done in order to separate vowels which start the word from the ones which contribute to the phoneme by succeeding a consonant. The corresponding Devanagari word W_d has no $\wedge$ and \$ markers.

The Roman word W_r is partitioned (refer expression 1) into groups of characters where length of each individual group is between 2 and $nGramLimitR$ (assigned to 3 in the system). The corresponding Devanagari word W_d is also partitioned (refer expression 2) into groups where length is between 1 and $nGramLimitD$ (assigned to 3 in the system). The lower limit for Devanagari is 1 due to single Devanagari letters often having a consistent sound contribution to the whole word.

$$W_r = wgR_1...wgR_i...wgR_n, 2 \leq |wgR_i| \leq nGramLimitR \tag{1}$$

$$W_d = wgD_1...wgD_i...wgD_m, 1 \leq |wgD_i| \leq nGramLimitD \tag{2}$$

For each possible partitions of both words where $n = m$, wgD_i is added as a possible phoneme equivalent of wgR_i. A global dictionary is maintained with key as Roman letter groups, and value as another dictionary with key as Devanagari letter groups and value as integer counts. This global dictionary is updated over all the Roman-Devanagari word pairs.

Partitioning is done using a straightforward approach with exponential time complexity. This is practical due to the lower constant factor associated and since the word length is usually small enough.

The proposed approach produces lot of noise due to blind position based letter group matching. However, with enough training data, the relevant Devanagari letter groups bubble up over the outliers.

The model generated by the training yields for any Roman letter group lgR, a list of tuples (lgD_i, f_i), denoting that the Devanagari letter group lgD_i was mapped to lgR, f_i times. The value $score(lgD_j, lgR) = f_j / \sum f_i$ is used as the confidence score for lgD_j being a suitable candidate for replacing lgR.

2.1.2 Transliteration

This module determines possible transliterations of an input word in Roman script to Devanagari script. It produces multiple results and attempts to rank them on the basis of their confidence scores.

Similar to the Roman script words in training phase, the input word W_r is surrounded by the $\wedge$ and \$ markers. Partitioning is also performed in the same way as the partitioning of Roman words during training (refer expression 3). For any partition, each Roman letter group is matched to the Devanagari letter group with maximum score, using the dictionary generated during the training. These matched letter groups are concatenated to get the final result W_d, with a consolidated confidence score (equation 4) for the result.The result with maximum confidence score for a partition is returned as the best Devanagari candidate for the given Roman word.

$$\begin{aligned} W_r &= lgR_1...lgR_i...lgR_n, 2 \leq |lgR_i| \leq nGramLimitR \\ W_d &= lgD_1...lgD_i...lgD_n \end{aligned} \tag{3}$$

$$score_{consolidated} = \prod_{i=1}^{n}(|lgR_i| * score(lgD_i, lgR_i)) \tag{4}$$

2.2 Data Structures

2.2.1 Word Dictionary (wordDict)

The $wordDict$ supports two operations, $insert$ and $query$.

$insert(w)$ inserts the word w into the dictionary.

$query(w, threshold, maxResults)$ returns at most top $maxResults$ words present in the dictionary whose similarity with w is greater than $threshold$.

$wordDict$ is implemented using a hash table, with keys as words. $query$ is done by iterating over all the keys and finding its similarity with the query word using Longest Common Subsequence (LCS) (equation 5. All words w_i in the dictionary such that $similarity(w, w_i) \geq threshold$, are sorted in decreasing order of their similarity score with query w, and top $maxResults$ words are returned.

$$similarity(w_1, w_2) = LCS(w1, w2)/max(|w1|, |w2|) \tag{5}$$

2.3 Preprocessing and Indexing the documents

Around 60,000 documents constitutes the search pool.

Documents to be searched are mixed script text documents. Since the base script was chosen to be Devanagari, transliteration is performed wherever required, to ensure the entire document is in Devanagari script. Indexing is done separately for the document titles and the contents.

For each word w, $before(w)$ and $after(w)$ is built, which is an augmented $wordDict$ of words occurring just before and after w in any document, respectively. $before(w)$ and $after(w)$ represent set of tuples $(w_x, (doc_1, ..., doc_i, ..., doc_n))$. The doc_i refers to the document in which w_x occurred before/after w. These are built both for the content and title of the documents.

The searching module makes uses the IDF (Inverse Document Frequency) values of the words. IDF values are calculated for each Devanagari word (original or transliterated) in the document content and title. $IDF(w) = log\frac{N}{1+n}$, where the word w occur in n documents out of N total documents.

A global $wordDict$ is inserted with all the Devanagari words (original or transliterated) found in the documents. It is denoted as $docWordDict$. It serves as the unigram index of documents.

2.4 Searching

Input $\rightarrow$ Query as a list of words in mixed script.
Output $\rightarrow$ List of documents sorted by their relevance score.

Any query word in Roman script is replaced by transliterating it to Devanagari script using the transliteration module (using the highest scored result).

2.4.1 Pivot Selection

A pivot term is selected from the query terms, around which the rest query is expanded over the documents. The selection criteria used for the pivot terms are their IDF values. Top N_{pivots} (assigned to 3 in the system) distinct query terms, sorted in decreasing order of their IDF values, are chosen as pivot terms.

In some cases, there are Roman words whose correct Devanagari representation are present in the documents, but the representations produced by the module are not. This is due to one of the two reasons, 1. The result produced by the transliteration module is incorrect, 2. There are multiple correct Devanagari representations for that Roman word. Since the transliteration module always tends to produce result sounding as close to the correct result, *docWordDict*'s fuzzy query is used to fetch similar words present in the documents with similarity score above a reasonable threshold. For a word w, if the most similar word found has a similarity score of above 0.95, that word's IDF value is concluded as the IDF value for w. Otherwise, the maximum IDF value of the similar words is chosen.

2.4.2 Pivot expansion

A bigram traversal query is performed on each of the N_{pivot} pivots and scores are independently added to vote for relevance of the results.

For each pivot term, a candidate word list is fetched from the *docWordDict*'s query. The candidate words have a similarity score above *similarityThreshold* (assigned to 0.7 in the system). Processing multiple similar words instead of one accounts for incorrectness in the transliteration module and multiple similar sounding representations of the pivot term in the documents. Bigram traversal (next section) is performed on each of the candidate words and results are combined into the result set for that pivot term. If two separate query for same pivot term (different candidate word) returns a score for a same document, the maximum score is considered in the combined result.

Result sets of pivot terms are combined by adding the scores for overlapping documents, voting for their relevance.

2.4.3 Candidate word bigram traversal

Input $\rightarrow$ *word*, pivot, original query.
Output $\rightarrow$ List of documents with relevance score.

The idea is to traverse across bigrams to efficiently match variable length phrases in the document content. Traversal is performed across the left and right of the pivot position in the original query separately, whose results are later combined.

If the original query has terms $q_1 q_2 ... q_n$, each q_i will refer to a list of words w_{j,q_i} such that $\mathit{similarity}(q_i, w_{j,q_i}) \geq \mathit{similarityThreshold}$. A traversal can be defined from w_{x,q_i} to w_{y,q_j} such that, $\mid i - j \mid = 1$. The state variables for w_{y,q_j} during traversal are computed from the variables of w_{x,q_i}. It should be noted that a state is defined by w_{y,q_j} as well as the path taken to reach that word, but for the sake of short variable names, it is omitted but remains true.

The state variables consist of the result document set ($\mathit{doc\_set}(w)$), and confidence score ($\mathit{score}(d, w)$) along with phrase count ($\mathit{count}(d, w)$) corresponding to each document (d) in the result document set ($\mathit{doc\_set}(w)$).

The first traversal always begins from w_{x,q_p} (q_p is the current pivot). The initial result document set $\mathit{doc\_set}(w_{x,q_p})$ consists of documents which have w_{x,q_p} in their body. The initial confidence score for a document d is $\mathit{score}(d, w_{x,q_p}) = \mathit{similarity}(w_{x,q_p}, q_p)$. The initial phrase count $\mathit{count}(d, w_{x,q_p})$ is simply the number of times w_{x,q_p} occur in the document body. $\mathit{bigram\_doc\_set}(x, y)$ represents the set of documents in which bigram xy occur. $\mathit{bigram\_count}(d, x, y)$ is the number of times bigram xy occur in document d.

Traversing from w_{x,q_i} to w_{y,q_j}, assuming $i+1 = j$, the state variables for w_{y,q_j} are calculated as shown in equation 6. This traversal can also be interpreted as an effort to match the bigram $q_i q_j$ in the query by matching similar bigram xy in the document. The new result document set consist of documents from $\mathit{doc\_set}(w_{x,q_i})$ which have the bigram xy present in its content. For a document d, the new count is updated by the minimum of its count in previous state and the number of times bigram xy occurs in the content of d. A normalized value of this count (Equation 7), along with the similarity between q_j and w_{y,q_j} (Refer equation 5) and the score of d in previous state are used to calculate the new score for document d. The count is normalized in order to have a regulated effect on scoring.

$$\begin{aligned}
\mathit{doc\_set}(w_{y,q_j}) &= \mathit{doc\_set}(w_{x,q_i}) \\
&\quad \cap \mathit{bigram\_doc\_set}(w_{x,q_i}, w_{y,q_j}) \\
\mathit{count}(d, w_{y,q_j}) &= \mathit{min}(\mathit{count}(d, w_{x,q_i}) \\
&\quad , \mathit{bigram\_count}(d, w_{x,q_i}, w_{y,q_j})) \\
\mathit{score}(d, w_{y,q_j}) &= \mathit{similarity}(q_j, w_{y,q_j}) \\
&\quad + \mathit{score}(d, w_{x,q_i}) \\
&\quad * \mathit{normalizeCount}(\mathit{count}(d, w_{y,q_j}))
\end{aligned} \tag{6}$$

$$\mathit{normalizeCount}(x) = \mathit{min}(2, 1 + \frac{x-1}{3}) \tag{7}$$

For $j + 1 = i$, just swapping the parameters in *bigram_doc_set* and *bigram_count* is required. Using the above traversal rules, traversal starts from *word*, towards left and right separately. If the pivot query term is q_p, then $\mathit{scoreLeft}(d) = \mathit{score}(d, w_{x,q_L})$, such that it is non-zero and L is as small as possible. Similarly, $\mathit{scoreRight}(d) = \mathit{score}(d, w_{y,q_R})$, such that it is non-zero and R is as large as possible. The query phrase from q_L to q_R, was thus matched in document d. It should be noted that this may not be true as traversals were performed through bigrams and not larger n-grams. However, it serves as a decent assumption. Both the left and right scores are combined using $(leftScore + rightScore) * (R - L + 1)$ to serve as a final score for document d. All relevant documents with non-zero scores are preserved.

2.4.4 Boosting results

Once the relevant documents with their scores are computed, some boosting heuristics are applied to reorder results. For sake of clarity, this result set is denoted by $resultSet$.

The entire search algorithm is repeated, instead this time just on document titles. This result set is denoted by $titleResultSet$. Score of any document in $resultSet$ also present in $titleResultSet$ is added by $titleMatchScore * 1.5$. Lastly, intent boosting is performed, where $intentBoost$ (0.2 in the system) is added to scores of documents whose document class (lyrics, movie reviews, etc) are explicit in the query and document title. Class is simply determined by matching class terms to document titles.

After sorting the results, 10 documents with highest scores are selected.

3. RESULTS AND ERROR ANALYSIS

The results obtained for the submitted runs are summarized in Table 1. For comparison, best score among all the teams are stated in parenthesis. Relative to other teams, the best overall NDCG@1, MAP and MRR scores were obtained, while the overall NDCG@5, NDCG@10 and RECALL were second best. The scores for cross-script NDCG@1, NDCG@5, MAP and RECALL were second best, and the rest were at third.

Subtask 2 results		
	Overall Score	Cross-script
NDCG@1	0.7567 (0.7567)	0.3400 (0.4233)
NDCG@5	0.6837 (0.6991)	0.3350 (0.3964)
NDCG@10	0.6790 (0.7160)	0.3678 (0.4358)
MAP	0.3922 (0.3922)	0.2960 (0.3060)
MRR	0.5890 (0.5890)	0.3904 (0.4233)
RECALL	0.4735 (0.4921)	0.4551 (0.5058)

Table 1: Results obtained along with best score among all teams (in parenthesis)

Producing only 10 results per query for submission affected the recall and slightly the MAP. Just rerunning the evaluation on 20 results per query, increased the overall recall to 0.5038, and the cross script recall to 0.4751. The overall MAP was slightly increased to 0.4073, and cross script MAP to 0.3037.

These results give an insight on where the system fails and the possible improvements. The search module heavily relies on the transliteration module for accurate transliteration. The transliteration module is, however, based on a non sophisticated statistical model, which sometimes hurts the overall score.

While calculating IDF values for words, similar words are treated differently, which is a bad choice because some variation of a low IDF word might actually get a high IDF value. This directly affects the pivot selection, where IDF value plays a crucial role. Pivot selection also needs improvisation so that it tries to cover the entire query instead of some fixed number of pivots. Eg, for a long query with large number of high IDF terms, selecting fix number of pivots might leave out important parts of the query.

4. CONCLUSION AND FUTURE WORK

In this paper, an approach for retrieving relevant documents from a mixed script document collection, was discussed and analysed. A frequency based letter group mapping model for back transliteration was used to perform search on a bigram representation of the documents. Pivot selection was done to identify important parts of query, around which the search was expanded.

There is a lot of scope for future work. The search module will directly benefit from a better back transliteration module. Sophisticated transliteration models can be used to test its improvement on the search module. A script specific rule based similarity method can be applied for finding similar sounding words with different spellings. There are many constants used throughout the algorithm, whose values were chosen based on heuristics and assumptions. Their tuning will significantly contribute to optimal behaviour of the system.

The problem with the current method of IDF was discussed in the previous section. A potential solution would be to cluster similar sounding words and calculate IDF values of the clusters. Better IDF values would allow for incorporating them into the document relevance scoring. Document scoring also needs some deep analysis and improvisations.

5. REFERENCES

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

[2] K Gupta, M Choudhury, K Bali. Mining Hindi-English Transliteration Pairs from Online Hindi Lyrics.In Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12),2012, 2459-2465.

[3] P Gupta, K Bali, R E Banchs,M Choudhury, P Rosso. Query Expansion for Mixed-script Information Retrieval. In Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval, 2014, 677-686.