Levenshtein Distance based Information Retrieval

Veena G, Jalaja G
BNM Institute of Technology,
Visvesvaraya Technological University

Abstract— In today's web based applications information retrieval is gaining popularity. There are many advances in information retrieval such as fuzzy search and proximity ranking. Fuzzy search retrieves relevant results containing words which are similar to query keywords. Even if there are few typographical errors in query keywords the system will retrieve relevant results. A query keyword can have many similar words, the words which are very similar to query keywords will be considered in fuzzy search. Ranking plays an important role in web search; user expects to see relevant documents in first few results. Proximity ranking is arranging search results based on the distance between query keywords. In current research information retrieval system is built to search contents of text files which have the feature of fuzzy search and proximity ranking. Indexing the contents of html or pdf files are performed for fast retrieval of search results. Combination of indexes like inverted index and trie index is used. Fuzzy search is implemented using Levenshtein's Distance or edit distance concept. Proximity ranking is done using binning concept. Search engine system evaluation is done using average interpolated precision at eleven recall points i.e. at 0, 0.1, 0.2…..0.9, 1.0. Precision Recall graph is plotted to evaluate the system.

Index Terms— stemming, stop words, fuzzy search, proximity ranking, dictionary, inverted index, trie index and binning.

1 INTRODUCTION

Information retrieval (IR) is the method of obtaining necessary information from a collection of large data set. User usually searches by providing the keyword or query. Queries can be a single or multiple keywords. In information retrieval, search for a query will not show single result instead many results which match the query will be shown. In Information Retrieval ranking the result set is very much important as the user will be interested in getting required information from first few documents of result set.

While entering characters there may be some typographical errors (typos), fuzzy search finds similar keywords and displays results for the predicted keywords. If the user wants to search for a documents containing keyword Vanaja but by mistake he or she types Wanaja then because of fuzzy search [1] the system will be able to retrieve the document containing Vanaja. Similarly if user wants to search for documents containing keyword Gouri but by mistake he or she types Gowri the system will be able to display proper document containing Gouri. Since words Vanaja and Wanaja are similar, similarly words Gouri and Gowri are similar.

In case a query contains more than one term, then considering proximity [1] (the distance between keywords) is very important. To illustrate the importance of proximity let us consider the query "knowledge management". Systems that do not take proximity into account return general documents in which all the two terms knowledge and management are individually important, but the document does not necessarily contain information about knowledge management. On the other extreme an exact phrase match would make sure that the document retrieved matches the query but implementing such phrase matching search would result in not displaying many relevant results. A proximity ranking, ranks query results based on distance between query keywords.

Pre-processing is an important step for fast retrieval of search results. Pre-processing involves text extraction from data set files, stop word removal, stemming, unique words extraction and Index creation. Indexes are very important in any search engine. Combination of indexes like Trie Index[2] and Inverted List index[3] is used in current research. Trie Index is a special kind of tree; unique words are inserted into Trie Index which helps in fuzzy search. Inverted Index contains a word ID and list of Document ID i.e. all those documents which contains the word. Along with document ID list of position ID will be stored, i.e. where and all the word is present in the document. Storing position ID is necessary for ranking search results based on proximity ranking.

Fuzzy search is based on finding similar words from the dictionary. Levenshtein distance or edit distance in combination with Trie index finds similar words faster. Edit distance here refers to number of single character operations such as insertion, replacement or deletion need to be done in order to transform one word to another word. For example edit distance between "bin" and "pin" is one, since replacing character 'b' by 'p' word "bin" can be converted to "pin". Based on the length of the word a threshold for edit distance is determined all similar words within the threshold distance will be considered for fuzzy search.

Proximity ranking is implemented based on binning concept. Inverted index contains document ID which is associated with list of bin ID. The document is divided into bins, the number of bins varies from document to document but each bin contains equal number of words. For proximity ranking Inverted index is searched to find if two or more query words are within the same bin or in adjacent bin. If the query words are in same bin or adjacent bin the document is ranked higher otherwise the document is ranked lower.

In next section, problem statement is described. In section 3 previous works on fuzzy search, indexing and term proximity are reviewed. Section 4 describes the proposed method that builds fuzzy search and proximity ranking. Experimental re-

sults are shown in Section 5. Section 6 concludes and discusses future work.

2 Problem Statement

Implementation of search system with advanced search features such as fuzzy search with proximity ranking. Existing search systems provide different kinds of ranking such as page ranking, ranking based on number of citations of the documents and ranking based on term frequency and inverse document frequency (tf-idf). Proximity ranking is very important since it determine the relevance of the answers as search queries usually contain related keywords and user is mostly looking for documents which have query keywords together.

3 Related Work

3.1 Fuzzy Search

K-Grams[4] are a sequence of k characters. Consider a word "course", if $ character is used to denote the beginning and end of the word, all 3 - gram character set of $course$ are {$co, cou, our, urs, rse, se$}. A k – gram index contains all k – grams for all words in the dictionary. For each k – gram (k_i) all words which contains k_i as their substring is identified and a posting list will be done.

Fig. 1. Posting List for the 3 – gram Col

Fig. 1.shows the posting list for the 3 – gram Col. K – Gram algorithm does not find all possible spelling errors. For example consider the word SODMY which must be corrected to SOUMY, the trigrams for SODMY are SOD, ODM, DMY all the three trigrams contains the error D thus the word SOUMY will not be found using K – gram algorithm because it does not contain any of the trigrams. Levenshtein distance[5] or edit distance is the number of single character operations required to transform one string to another. Damerau- Levenshtein distance[6] is the same as Levenshtein distance with minor modification, the single character operations allowed in case of Levenshtein distance is insertion, deletion and substitution whereas in case of Damerau- Levenshtein distance along with above mentioned operations transpositions of adjacent characters are allowed.

3.2 Indexing

For efficient and faster retrieval of search results indexing plays an important role. Inverted Index is most commonly used index; in this index each word is mapped to a list of documents where the word is present. Trie Index is a form of tree data structure, every node except leaf node consists of many branches each branch represents a particular character of the word. Forward Index is a type of index in which a document is mapped to list of words present in the document. As mentioned in [7] Hyb indexing outperforms inverted indexing by a factor of 15 – 20 in worst case. Hyb indexing is a variation of inverted lists in which data in compressed.

3.3 Proximity Ranking

BM25[8] is a model of information retrieval which computes the score for document based on query terms by considering term frequency and inverse document frequency without considering proximity between query keywords. In [9] proposed a method to change the document score based on proximity of query terms. In [10] proposed a method for proximity through spans. Depending on query terms spans are identified dynamically, the span need not be of fixed length and spans need not contain every query term. In [1] the common phrases in the document are identified and they are stored as part of Trie data structure query is segmented into various segments using phrase information and documents are ranked based on phrase matching, one of the limitations with this approach is that the phrases need to be identified in advance as part of preprocessing step.

4 Proposed Method

4.1 Proposed system Architecture

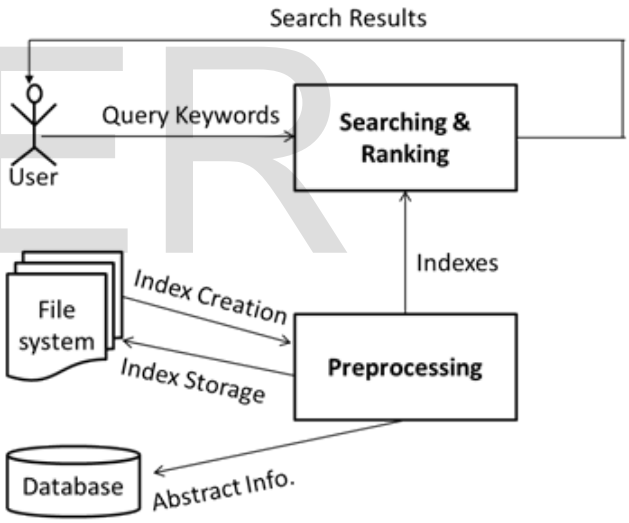

Fig. 2. System Architecture

Fig. 2. shows the proposed system architecture, pre-processing module is meant for creating and retrieval of indexes. Searching and ranking module is responsible for searching suitable documents using query keywords and indexes, this module ranks documents based on proximity distance between query keywords. User uses the system to search relevant documents. File system stores data set as well as index files. Database holds abstract information of each file.

Preprocessing

Text data from each of data set file is read special characters are ignored, stop words [11] are ignored, duplicate words are ignored and stemming [12] is performed. Words are arranged alphabetically. Each unique word is given a word Id. The col-

lection of word Id and word name corresponds to Dictionary. Dictionary will be stored in file system during this phase. It will be retrieved to main memory during the start of search server. Dictionary will also be represented as Trie data structure since using Trie data structure helps in faster retrieval of similar words for fuzzy search. Trie data structure will be stored in file system during this phase. It will be retrieved to main memory during the start of search server. During this phase an inverted list is created by reading in each file of data set word by word if the word in the dictionary exists then in which bin the word falls will be noted. The bin position helps in proximity ranking. The inverted index is created as a mapping between word Id and list of document Id's i.e. the documents in which the word exists and list of bin Id's i.e. the bin position at which the word is present in the document.

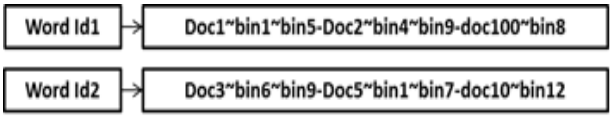

Fig. 3. Inverted Index

Fig. 3. shows inverted index format. Preprocessing module creates inverted index and stores it in a file on file system. It will be retrieved to main memory during the start of search server.

Searching and Ranking.

User can enter one query keyword or more than one query keyword for searching. When user enters more than one query keyword proximity ranking will be enabled. Fuzzy search is based on Levenshtein distance or edit distance.

$$D(i,0) = i, \quad \ldots..(1)$$

$$D(0,j) = j, \quad \ldots..(2)$$

$$D(i,j) = min\begin{cases} D(i-1,j)+1 \\ D(i,j-1)+1 \\ D(i,j-1)+\begin{cases}1 & if\ x(i) \neq y(j) \\ 0 & if\ x(i) = y(j)\end{cases}\end{cases} \quad \ldots..(3)$$

Fig. 4. Equations for Levenshtein distance

Fig. 4. Shows equations for Levenshtein distance, equation (1) and (2) represent base case equations for recursive function. In equation (3) the first term represent insertion cost of a character, 2nd term represent deletion cost of a character and third term represent replacement cost if two characters being compared are different.

	ε	B	I	N
ε	0	1	2	3
P	1	1	2	3
I	2	2	1	2
N	3	3	2	(1)

Fig. 5. Example for Levenshtein distance calculation

Fig. 5. Shows the application of Levenshtein distance equation, the edit distance between "BIN" and "PIN" is calculated as 1 i.e. if single character is replaced one word can be converted to another word. By replacing 'P' by 'B' word "PIN" can be converted to "BIN". Words are considered as similar if the edit or Levenshtein distance is less than threshold value. The threshold value can be determined by using the word length. Since Trie index is used for finding the edit distance if prefix (p) of a word exceeds the threshold value for edit distance then all words which contains p as prefix will be skipped from checking the edit distance which in turn help in determining similar words faster.

Algorithm Name: findSimilarWords

Input:

word, // Query word
node, // Trie node initially it will point to root of Trie
tr // threshold distance of word

Output:

list of similar words

Steps:

```
// declarations
 editDist[ ], orgDist[ ]  is an array of length (word.length +1)
 Initially orgDist has values 0,1,2,...
1   For each childNode of node
2     editDist [0] = orgDist[0] +1
3       for j =1 to word.length
4         editDist [j] = Apply Levenshtein distance equation
5       childNode.orgDist  = editDist
6       if(   childNode. orgDist[word.length] > tr )
7         break
8       findSimilarWords(word, childNode, tr)
9   if(node.orgDist[word.length] <= tr)
10    add the wordId from Trie to similar word list.
```

Above algorithm findSimilarWords, is used to find list of similar words for a given query word with threshold distance "tr". During preprocessing step all unique words in data set are represented using Trie data structure. "node" represent root node of a Trie. Child nodes of a Trie are traversed, while traversing if the edit distance of a node which represent the prefix of certain word in dictionary exceeds threshold distance then the remaining child nodes are skipped from checking for similar words. If the edit distance is within threshold distance then the word will be added to result list of similar words.

Algorithm Name: proximityRanking

Input:
wordIdList, // List of similar word Id lists
docIdList //List of documents containing similar words as that of query keywords
Output:
Ranked list
Steps:
1 For each list in wordIdList
2 for each docId in docIdList
3 obtain binId from inverted list
4 if(words are in same bin or words are in adjacent bin)
5 add the docId to the top of rankedList.
6 else
7 add the docId to the end of rankedList.

Above algorithm proximityRanking, is used to rank documents based on distance between query keywords. The entire set of documents which have all query keywords or words similar to query keywords are considered for ranking. Bin Id list is obtained from inverted list. If all words are in same bin or adjacent bin then the document is ranked higher else the document is ranked lower.

Algorithm Name: search
Input:
queryWordList, // List of query keywords
Output:
Search Result
Steps:
1 remove stop words from query word list
2 apply stemming to each query keyword
3 for each keyword in queryWordList
3 find threshold edit distance
4 similarWordList = findSimilarWords(keyword, node, threshold)
5 documents with phrases = proximityRanking (similarWordList, wordIdList) .
6 for each keyword
7 find documents without phrases
8 Result = (documents with phrases) union (documents without phrases)

Above algorithm search, is used to search relevant documents. Initially preprocessing of query keywords is done, this involves removal of stop words from the keyword list. For each query keyword stemming is performed. To find list of similar words to each query keyword threshold distance is calculated based on length of query keyword. Similar words are found using Levenshtein distance. Inverted lists are intersected to determine documents which contain all query keywords. Proximity ranking is applied to find documents with phrases. Documents without phrases i.e. Documents containing few query keywords but they do not form a phrase will be identified. The result will be displayed as union of documents with phrases and documents without phrases.

5 Experimental Results

Evaluation of search system is done using recall and average interpolated precision. Cran dataset is used for evaluating system which contains 1400 documents, 225 queries and set of relevant document number for each query. Lucene is an open source search system which is being compared with the current search system being developed (IFSWPR). Initially queries are given as input to each lucene and IFSWPR systems. Precision and Recall for each of the queries is calculated for each system. Interpolated precision for each query is calculated using the following formula.

$$P_{int}(r) = \max(P(r')) \text{ where } r' >= r$$

Interpolated precision at recall point (r) is maximum precision of all those points for which recall value (r') is greater than or equal to (r). Interpolated precision is determined at eleven golden points i.e. at 0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0. Average interpolated precision is calculated by considering interpolated precision of each of 225 queries. Average interpolated precision versus recall graph is plotted for both systems.

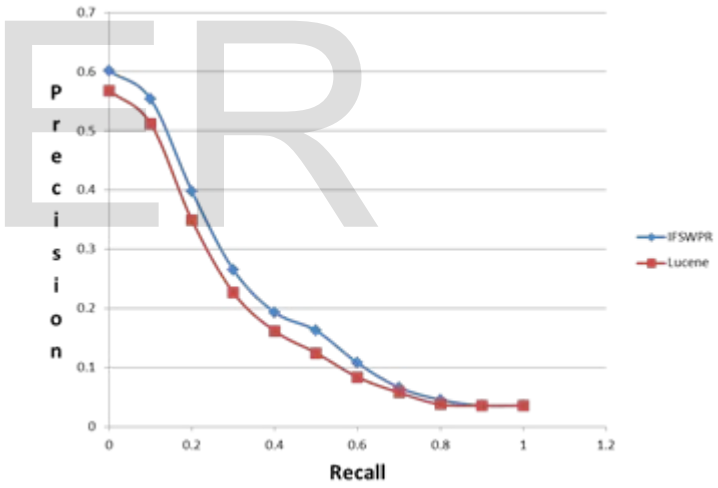

Fig. 6. Graph representing Average interpolated precision versus recall of IFSWPR and Lucene systems.

Fig. 6. shows graph of average interpolated precision versus recall of IFSWPR and lucene systems. Since IFSWPR system used modified inverted list by storing position of a word in a document using bin Id this system will retrieve more number of relevant documents within first few results when compared to lucene system. Other than cran data set English Wikipedia data set is used for testing fuzzy search and proximity ranking.

6 Conclusion

IFSWPR system is developed to provide advanced search features like fuzzy search and proximity ranking. Fuzzy search helps user in retrieving relevant results even if there are few

typographical errors in the query keywords. Proximity ranking ranks records based on distance between query keywords. Levenshtein distance is used in fuzzy search. Proximity ranking makes use of modified inverted list by storing word positions in the form of bin Id. There are other advanced search features like auto completion, page ranking etc. which will be considered in future for implementation.

7 REFERENCES

[1] Inci Cetindil, J. Esmaelnezhad, T. Kim and Chen Li "Efficient Instant-Fuzzy Search with Proximity Ranking,," IEEE 30th International conference on data engineering year 2014.
[2] G. Li, J. Wang, C. Li, and J. Feng, "Supporting efficient top-k queries in type-ahead search," in SIGIR, 2012, pp. 355–364
[3] M. Persin, J. Zobel, and R. Sacks-Davis, "Filtered document retrieval with frequency-sorted indexes," JASIS, vol. 47, no. 10, pp. 749–764, 1996
[4] A. Singhal. "Modern information retrieval: A brief overview." Bulletin of the IEEE Computer Society Technical Committee on Data Engineering,
[5] S. Chaudhuri and R. Kaushik, "Extending autocompletion to tolerate errors," in SIGMOD Conference, 2009, pp. 707–718.
[6] Damerau-Levenshtein edit distance explained http://scarcitycomputing.blogspot.in/2013/04/damerau-levenshtein-edit-distance.html
[7] H. Bast and I. Weber, "The completesearch engine: Interactive, efficient, and towards ir & db integration," in CIDR, 2007, pp. 88–95.
[8] H. Zaragoza, N. Craswell, M. J. Taylor, S. Saria, and S. E. Robertson, "Microsoft cambridge at trec 13: Web and hard tracks," in TREC, 2004..
[9] Y. Rasolofo and J. Savoy. "Term proximity scoring for keyword-based retrieval systems." In ECIR, page 207, 2003.
[10] Krysta M. Svore,Pallika H. Kanani, Nazan Khan "How Good is a Span of Terms? Exploiting Proximity to Improve Web Retrieval" SIGIR'10, 2010, pp. 154-161.
[11] Stopword Lists "http://www.ranks.nl/stopwords"
[12] The Porter Stemming Algorithm "http://tartarus.org/martin/PorterStemmer/"