Spell Suggester for Search Queries

Venkata Krishna Kota[*], Umamaheswaran S[**]

[*] Central Research Laboratory, Bharat Electronics Limited, Bangalore, India
[**] Central Research Laboratory, Bharat Electronics Limited, Bangalore, India

Abstract- Spell Suggestion plays a vital role in search applications. Lucene is an open source java library for building search applications. Lucene offers many built in features to develop rapid search applications. It offers spell suggestions for single term queries. In this paper, an algorithm is proposed for multi term spell suggestions and it is implemented.

Index Terms- component; Information Retrieval, Spell Suggestion

I. Introduction

With the availability of huge data and the necessity of retrieving useful information from that data in makes search applications popular. There is thirsty need for search applications in all areas where data exist and information is needed. Open source search libraries and tools offer basic building blocks using which users can develop fully fledged search applications[3].

Lucene is an open source search library. Apart from core search features like indexing and searching, it provides much functionality to enrich the search performance and user experience. Analyzers for various languages, support for various query types, hits ranking, index term and query term expansion, advanced search, faceted search, spell checking, hit highlighting, stemming, location aware search are few of them[2].

Spell checker is an important module which helps user in making correct queries. It will enrich the user experience. Lucene has spell check module. It offers spell checking for single term queries. In this paper, a spell suggester is developed using lucene to provide query suggestions to multi term queries.

II. Related Work

Lucene API provides much functionality to build search engine applications. Lucene offers a Spell Checker module to provide query suggestions to the misspelled queries. To find the suggestions for the dictionary of terms are required. Some applications uses query history as the dictionary from which it gets the query suggestions. Lucene uses the Spell Checker dictionary. It is constructed from the lucene's inverted index. In turn inverted index will be built by analyzing the textual content in the documents. Lucene uses string distance algorithms to calculate the similarity between the query and its suggestion. Lucene uses Jaro Winkler Distance , Levenstein Distance and NGram Distance to find the string distance[1,4].

If we analyze lesser textual content and build inverted index using it, it contains fewer terms only. So the Spell Checker index developed from that inverted index also contain fewer terms and when we try to get suggestions for the misspelled term then appropriate suggestions will not come because of fewer terms.

When we build the inverted index from documents of one domain, its spell checker index contains terms related to that domain only. When we apply queries of some other domain, because of lack of terms related to the domain of query, it may not provide accurate query suggestions. So it is better to build spell check index from the inverted index constructed from relevant domain specific documents.

Lucene's spell checker provides suggestions for single term queries. In this paper a Multi Term Query Suggester module is developed to provide suggestions to multi term queries.

III. Methodology

Multi Term Spell Suggestion is aimed to develop based on the building blocks of Single term spell suggestions available in Lucene. A corpus of English movie reviews is taken as the dataset for this experiment. All the documents in that corpus are analyzed and indexed using lucene functionalities. Index is tested using Luke tool[5]. Spell Index is generated from that inverted index using SpellChecker. SuggestionComparator is developed to compare given two suggestions to know the best suggestion in them.

Algorithm for SuggestionComparator is given below

```
1. Algorithm SuggesionComparator(suggestion1, suggestion2,indexSearhcer, corpusIndex, queryParser,termProximityThreshold)
2. //Description: SuggestionComparator compares 2 suggestions and returns Positive number if suggestion2 is better suggestion, Negative or 0 if suggestion1 is better than suggestion2
3. //Input:
      a. Suggestion1: one suggestion for the user query
      b. Suggestion2: another suggestion for the given query
      c. indexSearcher: Lucene's Index Searcher
      d. corpusIndex: Lucene index constructed with large number of text content
      e. queryParser: Lucene's QueryParser
      f. termProximityThreshold: proximity threshold to check the terms within the specified distance to each other
4. //Output: An Integer Number
5. {
6. Get HitCount h1 for suggestion1 from corpusIndex using queryParser and indexSearcher
7. Get HitCount h2 for suggestion2 from corpusIndex using queryParser and indexSearcher
8.  If  h1!=h2  then return h2-h1
9.  Else
10. {
11. Prepare a proximity query ‘pq1’ from suggestion1 with
    term proximity equal to ‘termProximityThreshold’
    using queryParser
12. Prepare a proximity query ‘pq2’ from suggestion2with
    term proximity equal to ‘termProximityThreshold’
    using queryParser
13. Get HitCount hc1 for pq1 from corpusIndex using
    indexSearcher
14. Get HitCount hc2 for pq2 from corpusIndex using
    indexSearcher
15. return  hc2-hc1
16. }
17. }
```

Multi Term Spell Suggester is developed using the following algorithm.

```
1.  Algorithm MultiTermSpellSuggester (query,
    SingleTermSpellSuggester, spellIndex, corpusIndex,
    indexSearcher, queryParser, suggestionCount,
    minHitThreshold, queryTermExtractor,
    suggestionComparator)
2.  //Description: MultiTermSpellSuggester analyzes the
    user’s query and identifies misspelled terms in it.  Then
    it returns the best spell suggestion for the given query
3.  //Input:
    a. query: user’s query for which spell suggestions
       need to be given
    b. SingleTermSpellChekcer: SpellChecker provided
       by lucene
    c. spellIndex: Lucene’s Spell Checker Index
    d. corpusIndex: Lucene index constructed with large
       number of text content
    e. indexSearcher: Lucene’s IndexSearcher
    f. queryParser: Lucene’s Query Parser
    g. suggestionCount: a configuration parameter.
       Describes the number of spell suggestions need to
       be considered for each term in the given query
    h. minHitThreshold: a configuration parameter. To
       classify a term as correctly spelled or misspelled. If
       that term have hitCount less than
       ‘minHitThreshold’ then classify it as misspelled
       term.
    i. queryTermExtarctor: Lucene’s
       QueryTermExtractor
    j. suggestionComparator: an instance of
       SuggestionComparator
4.  //output: Spell Suggestion for the given query
5.  {
6.  Parse the query using queryParser and build
    luceneQuery
7.  Extract the weighted terms from the luceneQuery using
    queryTermExtractor
8.  FOR each weighted term ti, DO
9.  {
10. Get document frequency ‘df_ti’ for that term in the
    corpusIndex using IndexReader associated with the
    index Searcher.
11. IF  df_ti < minHitCount  THEN
12. {
13. Get singleTermSpellSuggestions for ‘ti’ using lucene
    SpellCheck module and add them to list ‘sugs_ti’
14. }
15. ELSE
16. {
17. Add ‘ti’ to list ‘sugs_ti’
18. }
19. }
20. //Now each query term has a set of suggestions for it
21. Find all possible suggestions with combinations such
    that each combination is formed by replacing the terms
    in the query with one of their corresponding suggestions
22. Add all such suggestions to the ‘suggestions’ list
23. Add query to suggestions list
24. Sort the ‘suggestions’ list using the
    suggestionComparator such that the best suggestion
    stands first in the list
25. Return the best suggestion, which is the first element in
    the list after sorting
26. }
```

MultiTermSpellSuggester is implemented. It is tested with various queries and got satisfactory results.

IV. Results

A corpus of documents containing reviews of English movies is analyzed using Lucene API and inverted index is constructed. Constructed Inverted index is tested using Luke tool. Spell index is constructed from that inverted index using Lucene SpellChecker construction utilities. Lucene’s spell check features for single term queries are tested and made sure that spell index is constructed correctly. The MultiTermSpellSuggester algorithm is implemented. It is tested with various multi term queries and got satisfactory results. Some of the test cases are described in Table 1.

Table 1 Results Table A

Query Type	**Query**	**Suggestion**
Single Term Query	scwargenegar	schwargenegger
Field based Single term	content: scwargenegar	content: schwargenegger
Phrase Query	“arnol scwargenegar”	“arnold schwargenegger”
Field based Phrase Query	content: “arnol scwargenegar”	content: “arnold schwargenegger”

Multi Term Query	content:”arnod scwageneggar” terminator actoon	content:”arnold schwargenegger” terminator action
Boolean Query	arnol AND scwagenegar	arnold AND schwargenegger
Query with Operators	arnol +scwagenegar	arnold +schwargenegger
Boosted Query	arnol^4 scwagenegar	arnold^4 schwargenegger
Proximity Query	“arnol scwargenegar”~5	“arnold schwargenegger”~5

A. *for parameters ‘SuggestionThreshold’=5, ‘minHitCount’=5*

V. Conclusion

Results A corpus of documents containing reviews of English movies is analyzed using Lucene API and inverted index is constructed. Constructed Inverted index is tested using Luke tool. Spell index is constructed from that inverted index using Lucene SpellChecker construction utilities. Lucene’s spell check features for single term queries are tested and made sure that spell index is constructed correctly. The MultiTermSpellSuggester algorithm is implemented. It is tested with various multi term queries and got satisfactory results. Some of the test cases are described in Table 1 (for ‘SuggestionThreshold’=5, ‘minHitCount’=5).

References

[1] Andrzej Bialecki, Robert Muir and Grant Ingersoll, “Apache Lucene 4”, SIGIR 2012 workshop on Open Source Information Retrieval, 2012.

[2] Eric Hatcher, Otis Gospodnetic and Michael McCandless, “Lucene in Action”, Second Revised Edition, Manning Publications, 2010.

[3] Christopher D Manning, Prabhakar Raghavan and Hinrich Schutze, “Introduction to Information Retrieval”, Cambridge University Press, 2008.

[4] Lucene Search Library, available at, http://lucene.apache.org

[5] Luke tool, available at, http://code.google.com/p/luke

Authors

First Author – Venkata Krishna Kota received his B.Tech degree in Computer Science and Information Technology from Jawaharlal Nehru Technological University in 2005 and M.E degree in Computer Science from Anna University in 2008. He is working as Member (Research Staff) at Central Research Laboratory (CRL), Bharat Electronics Limited (BEL), Bangalore. His research interests are Information Retrieval and Complex Event Processing.

Second Author – S. UMAMAHESWARAN is currently a senior research staff at Central Research Laboratory of Bharat Electronics Ltd., Bangalore. He has more than 15 Years of Experience in the field of embedded systems. After Graduating from Madurai Kamarajar University with BE in Computer science and engineering, he completed his M E in Computer science from National Institute of Technology, Tiruchirappalli in 1998. His research interests are in embedded systems, real time operating systems, Tactical Radios and Mobile ad hoc networks. He is also a registered Ph. D Scholar in Indian Institute of Technology, Kharagpur.