Information Retrieval and Web Search Engines

Homework 5: Adding Spell Checking, AutoComplete and Snippets to Your Search Engine

REPORT

Name - Mukund Murrali **USC ID - 1264824477**

Steps followed to complete the assignment

The steps followed to complete the assignment are detailed in the following sections.

Spelling Correction

Spelling Correction functionality was implemented using Peter Norvig's spelling correction program. This spelling correction requires a corpus of words as input (big.txt) which acts as the training data for performing spell correction.

- **Creation of big.txt using Apache Tika**

  The words that form the big.txt come from the news website Reuters. I used Apache Tika to parse the html content of all the html files in the reuters news directory to get the words list. The logic that I used can be seen here,

  ```
  List<String> words = Arrays.asList(htmlContent.split("\\W+"))
  ```

  These words were then written to the big.txt file.

- **Peter Norvig's spell correction**

  Norvig's spell correction program uses the text file (''big.txt") generated in the previous step to get set of words using edit distance 1 and edit distance 2. The operations used to compute the edit distance are, a) deletion, b) transposition, c) alteration (replace), d) insertion. To optimize performance, the set of words i.e., the serialized dictionary can be saved on a file instead of parsing the big.txt during every single execution.

  In order to create the serialized_dictionary.txt, I did a dummy run to call the SpellCorrector code using a php script which has the content,

  ```
  <?php
  ini_set('memory_limit', '1024M');
  include 'SpellCorrector.php';
  echo SpellCorrector::correct('octabr');
  //it will output october
  ?>
  ```

  After executing this code, the serialized_dictionary.txt file will be generated. This can be used for subsequent calls to the SpellCorrector code.

- **Adding Spelling Correction functionality to the client code**

  For every word that is input in the search field, after hitting submit, I call the SpellCorrector::correct function (for each word entered) which will provide the spell corrected word.

  ```
  $correction = SpellCorrector::correct($terms[$i]);
  ```

  Using this, I implemented the client side functionality as follows,

  Showed the result for misspelled word. Just below the text box, I displayed the correct spelling which is clickable. Upon clicking the correct word, I perform a search and displayed the new results.

Auto Completion

Auto Complete functionality was implemented enabling Solr's inbuilt auto complete features.

- **SuggestComponent of Solr**

  The SuggestComponent in Solr provides users with automatic suggestions for query terms. I used this to build a powerful auto-suggest feature in my search application. I followed the steps given in the AutoComplete document to enable this.

  I added a search component to solrconfig.xml and tell it to use the SuggestComponent using FuzzyLookupFactory. After adding the search component, I added a request handler definition that incorporates the "suggest" search component defined previously.

- **Adding Auto Complete functionality to the client code**

  I used JavaScript, jQuery and Ajax to add Auto Complete to the client. For every letter that the user types in the search box, an Ajax call will be placed to Solr's suggest function.

  ```
  var suggestionArr=
  JSON.parse(JSON.stringify(data)).suggest.suggest[currTerm].suggestions;
  ```

  This will return a set of words (in my case I have limited the result to be of size 5 in the suggest.count config mentioned above). I displayed the result array of words as a drop-down in the text box. Each result word is clickable, meaning the text box will be replaced by the word the user clicks.

  For n-word queries, the previous word/words is/are appended to the current letter entered as the request to the suggest function.

Snippets

In order to generate snippets, I used Apache Tika to parse the html content of the news websites into sentences.

- **Sentence parsing using Apache Tika**

Using Apache Tika, I parsed the html content (using regular expression in Java) of all the html files in the reuters news directory to get the sentences. I used the following regular expression to split the html content to sentences,

```
((?<=[a-zA-Z0-9_])\\.\\s[A-Z]+)|\\n")
```

I then generated a list of the parsed html files which is used during client processing.

- **Adding Snippet functionality to the client code**

  For each search result, I looked for a string match (identified position index) of the query terms within the parsed web page. I then returned the first sentence that provides a match. If no match is found, then no snippet is returned.

  For multiple term queries, I followed the suggestions given in the homework document,

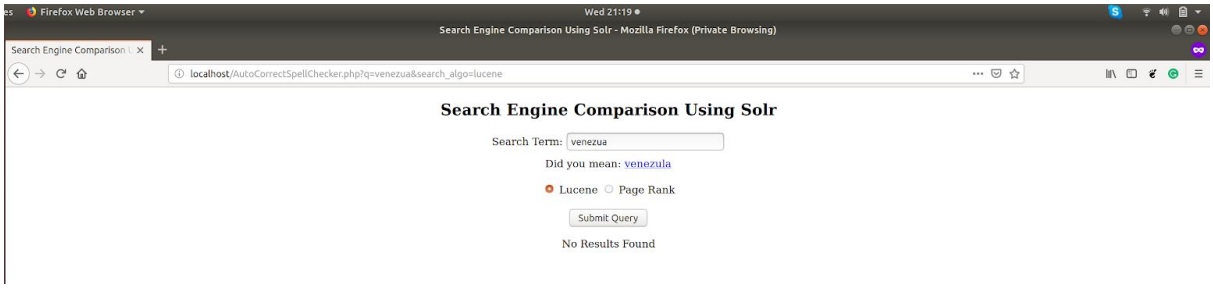
  a. Tried finding a sentence with all the terms together
  b. If not, returned the sentence that has all the terms in it, even if they are not together or in same order
  c. If none of the fore-mentioned are found, returned the first sentence with at least one query term in it
  d. If no match is found, then no snippet is returned

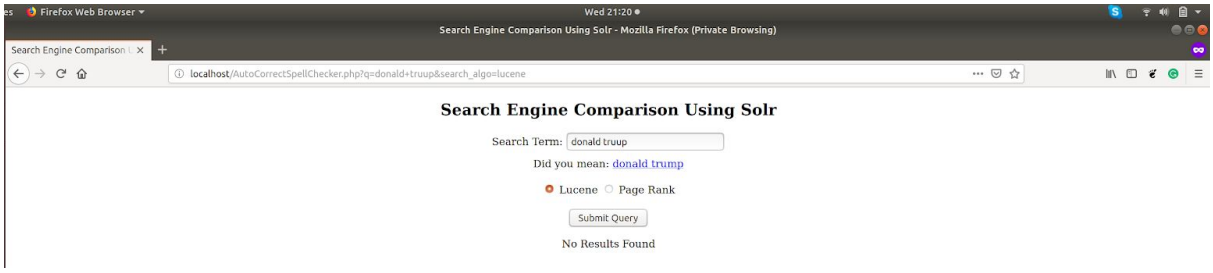
  Logic to trim the snipper to 160 characters or less,

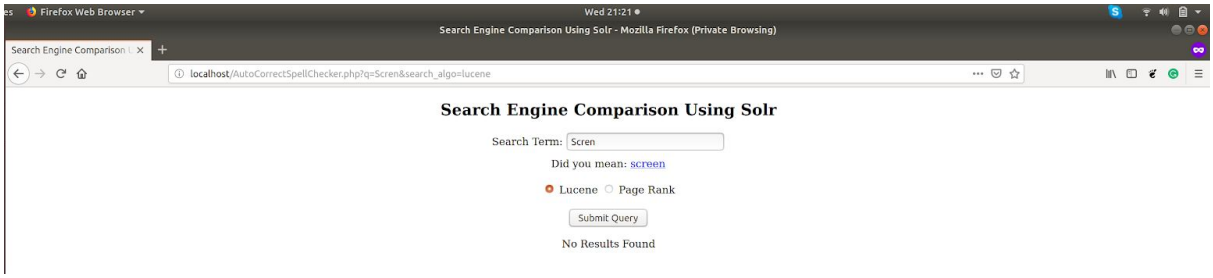
  a. If the sentence in which the word is located is less than 160 characters, then I displayed the whole sentence as snippet
  b. If not, I went to the position index of the word then started building the sentence on both the directions (using two pointers) until either one pointer reaches end/begin of the sentence and until 160 characters are reached

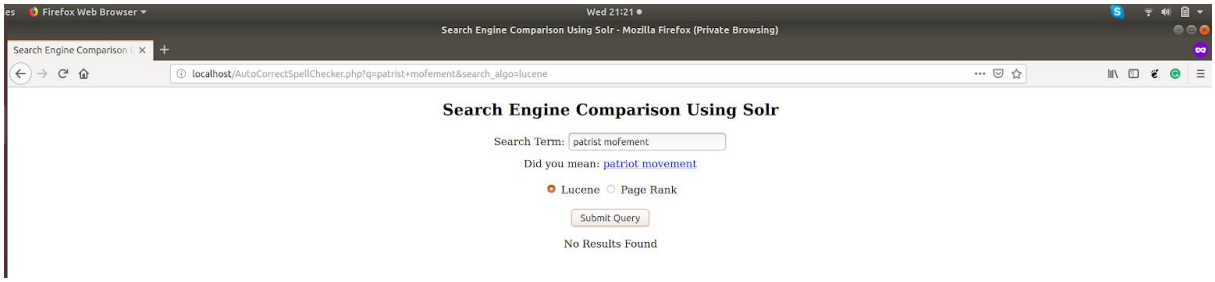
  I finally highlighted the query terms when displaying the Snippet.

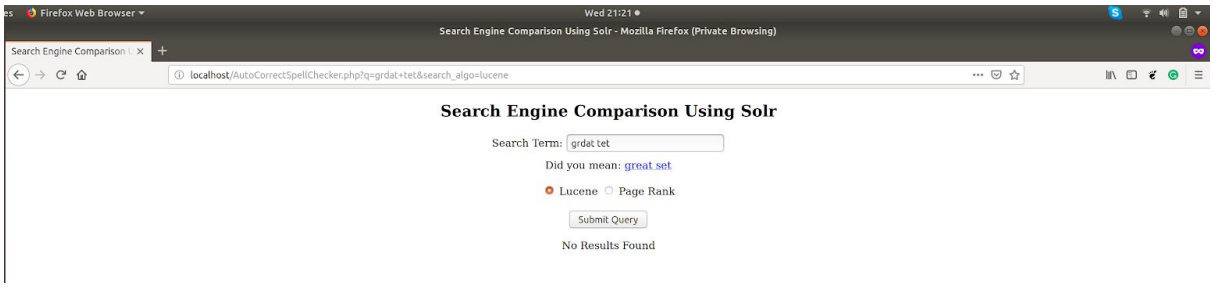
Examples of Spelling Correction











Examples of Auto Completion