Text Technologies for Data Science
Course Work 1 Report

Meghna Raje
S2305829

30th October 2022

1 Details on the methods used for tokenization and stemming

Methods used in the pre-processing module are as follows.

1. Element.Tree module is used for parsing the XML data
   (a) parse() and getroot() functions are used to initialise the tree
   (b) root tag is then used to iterate over elements and sub-elements of the XML file
   (c) data from the XML file is stored in a dictionary where the key is the document number and values are headlines and text body combined

   {Document Number 1: "Heading1 + text1",Document Number 2: "Heading2 + text2",...}

2. Tokenisation
   (a) All upper case characters in the line are first converted into the lower case using the function .lower()
   (b) Regrex package is used to remove everything other than lexical English alphabets, numbers, and spaces
   (c) lines are then split into tokens using the .split() function
   (d) tokens are then stored in a dictionary where key is the document number and values are list of tokens

3. Stop Words Removal
   (a) All the stop words are first stored in a form of a list
   (b) This list is then iterated over the token to remove stop words from them

   {'1': ['ft','14','91','correct','jubile'...],'2': ['ft','14','91','correct'...],...}

4. Stemming
   (a) stemming.porter2 package is used for getting the root form of the words
   (b) The root form of the word is then replaced in the token dictionary

2 Details on the implementation of the inverted index

1. Creating a vocabulary list
   (a) itertool package is used to get the list of unique words in the document These unique words are then sorted in ascending order
2. Creation of inverted index
   (a) Iterating over the outer loop for each word in the sorted vocabulary. Then, iterating over the inner loop for each element from the dictionary formed after pre-processing that is tokenization and removing stop words
   (b) If the word from the vocab is in the tokens list for a particular document then saving the doc number and its position into the new index dictionary

{'0': {'14': [66, 124, 134, 142],'15': [19, 30, 33],..}..}

3 Details on how you implemented the four search functions

1. There are four different functions for each type of search-
   (a) Phase Query
   (b) Proximity Query
   (c) Boolean Query
   (d) Phase and Boolean Query
   (e) Single term Query
2. After reading the search expression from "queries.lab2.txt" file, the query is separated from the query number and then pre-processing is applied to it. Then, it is passed to evaluatePostfix() function for search operation
3. Tasks performed by evaluatePostfix functions are as follows:
   (a) It creates two different stacks one for the operator and other for the term. The operators like AND, OR, and NOT are stored in the operator stack and everything else is stored in the term stack
   (b) After this first term is popped out from the term stack and the stack is empty later then the Single term Query function is called
   (c) If the operator stack and term stack both are not empty then the Boolean Query or Phase and Boolean Query functions are called depending on if there are quotes(") in the term or not
   (d) If the term stack is not empty but the operation stack is empty then the Phase Query or Proximity Query search functions are called based on checking if " or # is there in the terms
4. Tasks performed by each of the search functions
   (a) Phase Query
      i. This function takes two words as input. Firstly it removed " quotes from the word
      ii. After the quotes are removed then it retrieves the list of documents associated with those two words from the index dictionary
      iii. The result for both words is then converted into sets and the intersection for both sets is taken For each document in the intersection of the set result, the positions listed in each of the documents are compared with each other. If the difference between the position is equal to one then the document contains the phase in it and it is stored in the result list

(b) Proximity Query

i. Firstly it extracts the distance from one of the terms and stores it in a variable

ii. Later anything other alphabet is removed from both the terms and the list of documents that contain these terms are retrieved

iii. For each document containing both these terms, the distance between the positions is calculated and checked if it's less than the given distance. If it satisfies this condition then the result is added to the final list

(c) Boolean Query

i. Separate result is retrieved for both the words first

ii. If the operator is OR then the union between both the set of documents is carried out and the final result is returned

iii. If the operator is AND then the intersections is performed for both the documents

iv. If the operator is NOT then the documents associated with the term NOT are subtracted from the total list of documents and later on the corresponding operation is carried out depending on if it's "AND" or "OR"

(d) Phase and Boolean

i. Firstly phase query is applied for words which contain quote(") in it

ii. List of documents is retrieved for the other term which does not contain quotes

iii. Based on whether the operator is "AND", "NOT" or "OR", union or intersection is carried out on the documents obtained from steps i. and ii.

4 A brief commentary on the system as a whole and what you learned from implementing it

1. Implementing the system from scratch gave a good understanding of how a search engine works and increased my confidence in developing higher-level systems in the next phase of the course

2. Learning from the implementation:

   (a) Understanding of basic methods used for text processing

   (b) Overview on how information retrieval works

   (c) Good understanding of using the dictionary as a data structure

   (d) Strong knowledge of text laws

5 Challenges faced when implementing

1. Tried using a binary file for storing the index but while retrieving the data from the binary file the location position was changing to different values and was giving the wrong result for the index

2. Implementing stack for Query search was a bit challenging, popping out the right term is very important otherwise it will give the wrong output

6 Ideas on how to improve and scale implementation

1. The system performance can be improved by not loading the whole file in the Element tree for parsing the XML file

2. Tokenisation can be improved further by considering 's and hyphens in it

7 Challenge Question - Rerunning the same queries for Boolean and Ranked IR, but without applying stopping this time in indexing or queries

1. Retrieved results
   (a) For Boolean search query- In the phase query after removing stop words we are getting a few documents that contain terms like "income other tax" where income and tax are not phased here. Since we removed stop word between them they are getting considered as a phase which is a false case
   (b) For Ranked Ir- There is no difference in the ranking for both the cases.
2. Processing time for index and queries
   (a) Processing time for index creation increases almost 3 times
   (b) Processing time for query without stop word was 93ms and with stop word was 213ms. This implies that running the query by without applying stopping increases the response time by a factor of 2.2.
3. Size of the index
   (a) The index file size increased from 2.18MB to 3.55MB after including the stop words
   (b) The vocab size is increased from 226092 to 422543 which is almost the double