République Algérienne Démocratique Populaire
Ministère de l'Enseignement Superieur et de la Recherche Scientifique
Universite Ferhat Abbes de Setif 1

Faculté des Sciences

Département d'Informatique

Master 2: I.D.T.W

RIW Project Report

By:

Simango Psychology G1

Abdou Abarchi Ibrahim G2

2022/2023

Professor:

Dr Harrag

Introduction

Information retrieval is the process of obtaining information system resources that are relevant to an information need from a collection of those resources (wikipedia).

IR system can be defined as a software program that deals with the organization, storage, retrieval, and evaluation of information from document repositories, particularly textual information. The main components of the IR system are acquisition, preprocessing, indexing, and querying. For our IR system, we used python programming language to implement all the components as below.

Acquisition

The corpus we used for our system is an XML document (trec.5000.xml with 5000 sample articles), thus, we created a method (in***_data (filename, dtype, reg) =>list)*** which accepts a filename and type of document in order to read various types of files. For general types of files; we just used python file handling methods. For text files, we made an exception, to provide a regular expression on top of filename. To handle XML documents, we used *ElementTree* from *xml.etree* package. We looped through root children named 'DOC', finding the 'DOCNO', 'HEADLINE', and 'TEXT'; and then we appended text to headline. We then stored the document id and document text to a list and thus, the list became the return variable of the method.

Preprocessing

The first step we did was to substitute tabs, more than non-breaking spaces, and new lines to single non-breaking space. To accomplish we used the python *sub* method from *re* package. After some surveying, we noted that there exist a special word '&amp' which is used in place of the character '&', thus, we substituted it to that character. The next step was to decode some special characters which are used in natural language but have a different representation for instance inverted commas. To implement it, we used the *unidecode* method from *unidecode* package. After we changed word cases to lowercase to provide consistency. Next we removed punctuation marks by using a regular expression while noting that there exist hyphenated compound words. We then removed extra spaces before tokenization.

To tokenize, we imported *word_tokenize* from *nltk.tokenize* package. For normalization, we chose stemming and we used PorterStemer method from *nltk.stem.porter* package. To implement the whole prepossessing stage we created a method called, which takes a sentence and a Boolean variable to determine whether or not we remove stop words and normalize (stemming).

Indexing

To make a positional index, we chose to build one from scratch using two methods.

The first, ***index_data(in_data: list)=>dict***, to create a dictionary where keys are terms, and the values are dictionaries whose keys are documents id's and their values, positions of the term in the respective document (***{ term: { docid: [pos1, pos2,….], ….}……})***. Then we stored another reference dictionary to keep track of documents and number of terms. The reference dictionary is used on ranked retrieval where precise length of document is required and also total number of documents in the corpus.

The second method, ***out_data (data, filename, dtype, query, rnk)*** where, is for writing out the index and we chose to store it in json file.

To test different types of searches, we created two indexes corresponding to two different scenarios which are: cases where stopping and stemming are included in preprocessing step and cases otherwise.

***We created methods to write out results of each component (preprocessing, indexing) to text files for better visualization.*

Querying

Querying was based on two models: Boolean retrieval and tfidf ranked retrieval.

Firstly, we load the positional index into memory and then given queries.

Boolean Retrieval

The model serves to retrieve documents corresponding to Boolean operations.

AND*:* Retrieve documents which correspond to query of format word1 AND word2 and contains both word1 and word2.

OR: Retrieve documents corresponding to query of format word1 OR word2, and contains either one of them or both.

NOT: Retrieve all documents which do not contain mentioned term or word; meaning search by exclusion.

The three operations above caters for Boolean search.

Besides Boolean search we also created methods to perform proximity search of format #\d (word1, word2) where it returns all document which contain both words and the distance between the words is given.

Phrasal search returns documents which contains all words given in query and succeeding words have hierarchical positions; meaning difference in position of succeeding words is 1.

Tfidf Ranked retrieval

The concept is based on calculation of term frequency (tf) which means number of occurrences of a term in a given document (tf =>frequency) and inverse document frequency is calculated from document frequency, meaning number of document which contains the term in the corpus. Then, we calculated the weight of each term in the document by computing tf*idf: $w = (1 + \log_{10} tf) * \log_{10} (N/ df)$. This means for each term in the index, a corresponding weight has to be calculated thus just after loading the index, we called a method to build a structure based on the index to prevent unnecessary calculation during search. For each query, a document is retrieved and the given score is computed is cosine similarity method we defined, thus, document is ranked based on its corresponding score.

System

We did not take dates or numbers into considerations when preprocessing thus, the index contains a lot of junk nonetheless, the system performed well. We learned that, it is nigh impossible to implement a perfect system unless of course we anticipate all possible types of data which cater for special cases like represented characters which needs to be noted before preprocessing. The index still showed some punctuation characters and also we did not pay special attention to dates, thus, we think that we should seek other methods to allow that.

Challenges

It was a bit challenging when we used python since it is a new programming language to us, thus, it took a while and a bit of research on how to actually implement ideas we had. We had to pay attention to words format such as, compound words however that, too, gave some issues. Time is required to master python