Design Document for IR Assignment 1

Team Members:

- Akash Sebastian ID:2015A7PS967H
- Rakesh ID:2015A7PS052H
- R. Shriram ID: 2015A7PS011H
- R. Bhavitej ID: 2015A7PS021H

Application Architecture:

The corpus contains scrapped news articles from “The Hindu” and “Indian Express” from the dates 13th Sept 2017 to 22nd Sept 2017(10 days). The entire corpus contains 5436 documents of which 974 are Indian Express and 4462 are Hindu. The scrapper outputs each document in a certain format, where the first line contains the link from where the article was extracted from, the second contains the date at which this article was published in a particular website and the third contains the title of the document and the remaining document contains the article itself.

Each preprocessed word is pushed into a complex data structure, multimap_list[preprocessed_word][list of doc_ids the word occurs in][list of list[list of paragraph id it occurs in][frequency of the word in doc_id]]. After every document has been indexed the multimap_list is then pickled into “data.pkl”. For querying, the same preprocessing steps are carried out and a modified tf-idf is calculated to get best results.

Formula is given by:

- Tf-Idf_modified = ∑((log(tf)+1)*log(N/df)*pmax)

Where ∑ runs over all documents, tf is the term frequency, df is the document frequency and pmax is the max no of query words occurring in one paragraph. The outputs of the query are written to a json file and read on the front end(GUI).

The website is written using HTML, CSS, JavaScript and the backend in PHP. The command_read.txt is generated as soon as a search is inserted into the query and the search button is clicked. The format of the command_read.txt is on the first line there will be the vendor which the user is asking for articles from, next line has the number of results which should be retrieved(hard coded to ten for the purpose of this demo) and the last line has the query itself. This generated command_read.txt is read by the python script and it processes the query and writes the results into the output.json file. The output.json file is read by JavaScript through an AJAX call and the results are added via JavaScript into the markup to be displayed.

Preprocessing:

Each of these documents are preprocessed in various ways, and finally creating an index which is stored in “data.pkl”. Let’s have a closer look at the steps involved in preprocessing. Each document is word tokenized, but also taking care of things like “Mr. James” as one word and not two. Any stop words

in the document are removed. Each word is pos_tagged(parts of speech tagging) to give special attention to proper nouns. This is followed by stripping away punctuations. The words are then lemmatized based on our own algorithm, as porter stemming (nltk package) and lemmatizer (nltk package) gave poor results. Lemmatizing is done using nltk's synset which returns synonyms of a word from which the most matching prefix word is chosen. Ex. 'Happiest' synonyms are 'glad', 'happy' and 'well-chosen'. For which the best matching prefix here is happy.

Search Time:

The search time for each query is mentioned in the left side of the web page. It is updated based on the given search.