Report and documentation for our assignment

After discussion, we have decided to implement our program using Python. This is because python has a large range of libraries that are very useful for this assignment. In this report, all the libraries used for this program will be explained, along with documentation for how the program runs and functions. In general, this python file contains 8 functions. Each of these functions provides essential implementation and usage of libraries to form the desired concluding output.

Firstly, to gain the input and output of the system it must be able to read Web pages. It is expected to produce an appropriate formatted output, therefore we made use of the 'urllib.request.urlopen' that was imported from 'urllib.request'. We considered using this module because it defines functions and classes which aids in the process of opening URLs. In this case it is a 'HTTP' connection and could open the URL if 'url.request.urlopen()' is used. The content is stored in the variable 'htmlOutput' that will be used for further implementation. We have also considered to store the URLs into a text file named 'Websites.txt', in which the system can read the URLs and process them with the desired output displayed into another document file. This is a flexible approach, since the URL could be easily added or removed from the file. For this occasion, we decided to use 5 URL links that will be processed through main().

The next stage will be for the system to produce HTML Parsing. This is because the HTML tags will need to get rid of in order for the text to be analysed. Our desired output would be the result displayed in plain text, in which 'html.parser' and 'filter()' were used to produce this. The module 'html.parser' allows parsing of web files formatted in HTML and XHTML. HTML data is process and calls handler methods when start tags, end tags, comments, tags, and other mark-up elements are come across. This is used with 'filter()' which requires a 'function' and 'iterable' as its parameters. It constructs a filter object from those elements of 'content' for which 'htmlOutput' returns true.

To complete the 'Pre-processing' phase, the input text will be transformed into normal form. We import English stop words at this stage to be removed from the data that needs to be tokenized, lowering tokens size. As well program removes all punctuations replacing them with white spaces, this is achieved by .translate() command which takes argument as transitional table, table is created with str.maketrans() command. After, tokenisation is necessary as it will split the sentence or data into words using one method ***word_tokenize():*** in which special characters are also treated as separate tokens. Before this method could be used, it is required that *word_tokenize* to be imported from ***nltk.tokenize***. Having tokens, now program iterates list by removing non-alpha tokens, lower cases all tokens and removes all stop words, massively lowering the count of tokens. Note: at this stage program might fail if python cannot or do not have some encodings for characters, this was found during testing with Wikipedia pages. NLTK is important for this

pre-processing stage as it is providing a suite of text processing libraries for classification, tokenization, stemming, tagging and parsing.

Also requiring the NLTK library, Part-of-Speech (POS) Tagging forms the next step. This is a suitable POS tagger as it allows the result to be processed for the remaining steps. Our variable 'tags' is used to store the tokens that have been tagged using the '**nltk.pos_tag**' method. From the '**speech_Tag**' function, the non-processed tokens are tagged. The frequency of tags is processed, and common words are sorted out. Common words are achieved by frequency distribution commands from NLTK and sorted with .most_common() to see which tags repeat most frequently. By using the POS-tagger, it is expected that it processes a sequence of words and attaches a part of speech tag to each word. For instance, it could be the word 'for' is listed as 'IN' meaning it's a preposition.

To select the keywords, we considered ways to identify the words or phrases in the text that are most useful for indexing purposes. Words that are not useful, such as common words must be removed. For this part the function named 'keyWords' were used, that sorts out the most common words based on the first 3 words in the tag. Why 3, it was chosen because top 3 words will have most impact and most repetitions in the tag list. Therefore, it was necessary to use a POS Tagger to define the tags for each word. This system will clearly display the results with appropriate titles for each feature. It is evident in this function, that the number of items that are in the list displayed. This will vary depending on the URL that are chosen to be processed. Our 'keyWords' function uses 'len' to determine the length of items that are in the list, identifying that there was a change in output from the pre_processing() function.

Stemming is used for the analysis section of the program, which aims to treat various infected forms of a word in the same way. In this case, the library of '**nltk.stem**' is used so that '**PorterStemmer**' could be imported. In our system, the function 'stemming' is implemented that can remove duplicate values of stems using set() giving a close list of the processed tokens list. PorterStemer is a popular stemming algorithm, that is a sort of normalizing method. The idea is that the variations in a word could still deliver the same meaning. This allows the system to be more efficient as its shortens the lookup, for instance *fox* and *foxes* would mean the same thing.

Finally, when we combine all these features together, a complete system is formed where we have control over all the individual components. The python file also contains comments that aids with the understanding of the processes. Along with this, the text file named 'Websites' is required since the system reads the URLs from this. Once the system processes all stages, it is expected for 5 different text files to produce. These are named by the URL, to allow easy indication of the file. After successfully opening the text files, the Pre-Processing stage is shown first containing the number of items in the list. The second section is the speech tags, which is followed with the keywords selection in the third part. Finally, the stem list is shown in the text files to form a complete system.

The possible improvements to our solution could be the consideration of using an alternative Tagging system such as Tree Tagger. This could be the case, because using POS Tagging will output ambiguous words and unknown words. In addition, it may create 'lexical ambiguity' as in the process of tagging it can sometimes retrieve such words that have different tag categories when they are used in different context. This suggests that words could be assigned the same POS tags, even though they have different meaning in a different context. Another improvement would be to make the stop word more extensive, so that it covers a wide range of languages depending on the context of the websites. Furthermore, by using stemming in our system it cannot relate words which have different forms based on grammatical constructs. For instance, 'is', 'am' and 'be' all represent the same root verb of 'be'. These factors could all affect the overall output of the system, but each algorithm have its own advantages and disadvantages.