CE706 – Information Retrieval

Assignment 1: Indexing for web search

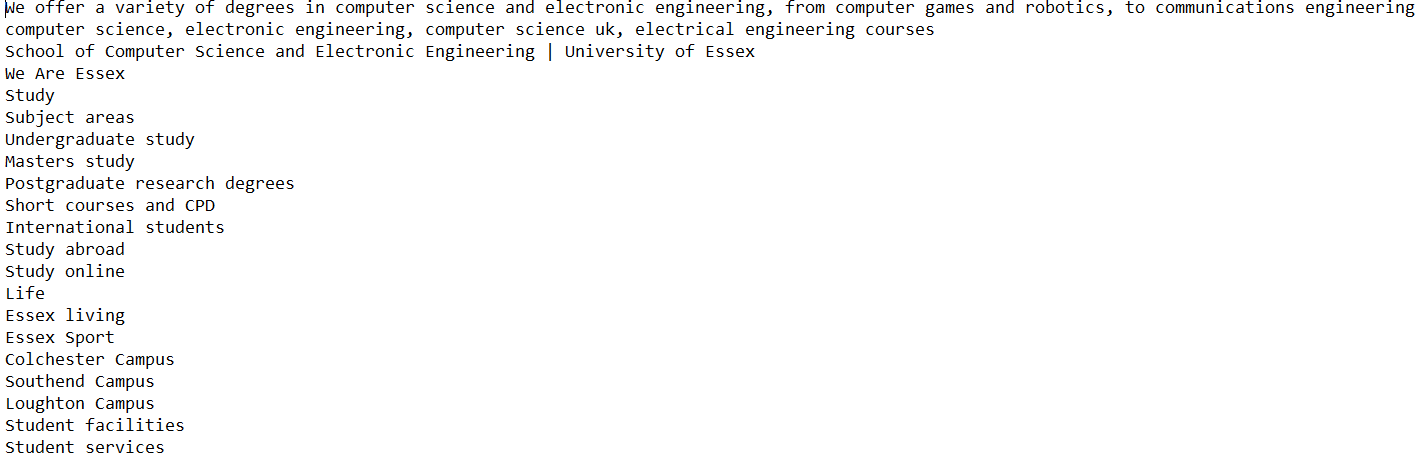
Pranavi Vasa: pv18403

Siddharth Shyamsunder: ss18812

URLs.txt contain list of urls. The program will go through and produce files according to the stages.

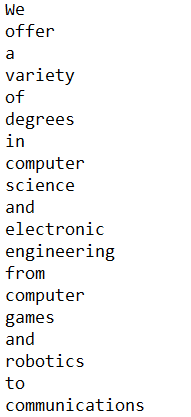
**HTML\_parsing:** First the program is calling get\_output() function, it will generate plain text from the URL. It is taking ‘url’ as input and processing the it to produce the plain test. It is using Beautiful Soup for reading the text. First it is removing the script from the text. After it is finding all the meta tags and storing them into a string. After it is getting plain text from the beautifulsoup url and it is removing all the unwanted spaces. In the end it is attaching meta data to the plain text of the page.

Output of this stage is two files corresponding the two url’s. Those files are “HTML\_Parsing0.txt” and “HTML\_Parsing1.txt”. Head of the file is shown below: (First two lines are meta tags data)

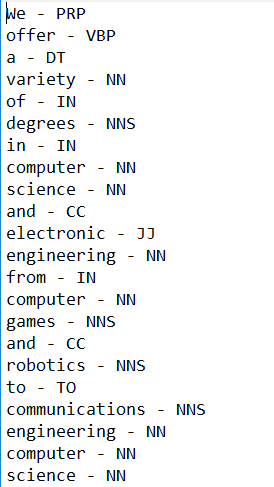


**Pre-processing:** Next program will call the pre\_processing() function. This function removes punctuations, lemmatization and tokenization.

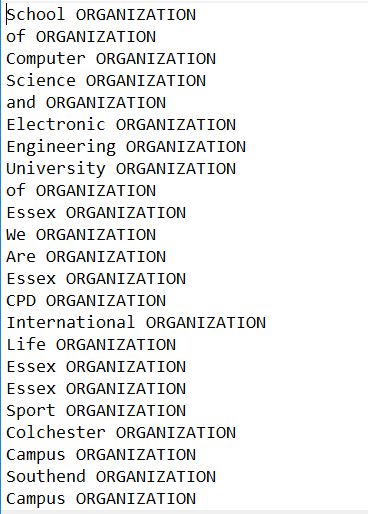
Output of this stage is two files corresponding the two url’s. Those files are “pre\_processing0.txt” and “pre\_processing1.txt”. Head of the file is shown below:



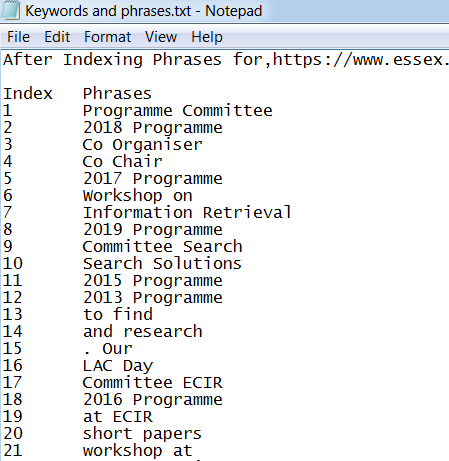
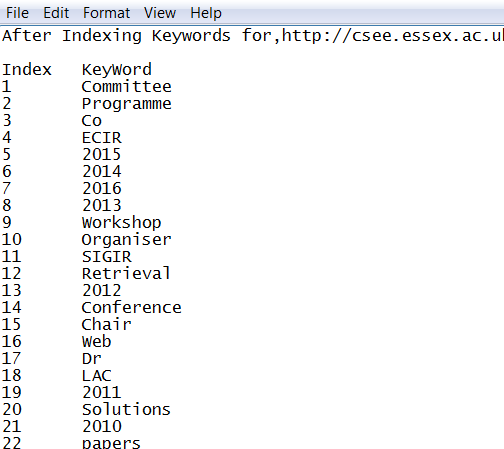
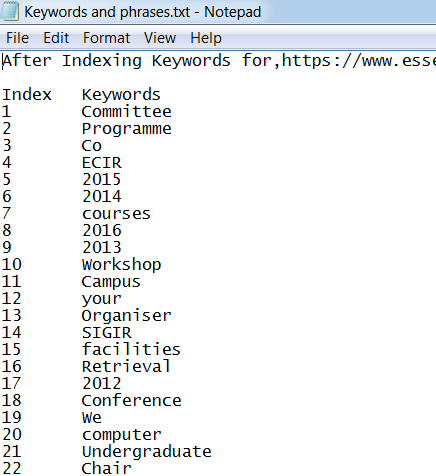
**Part-of-Speech Tagging:** Next program will call pos\_tag() method. This will produce two files corresponding to the two url’s. Those files are “POS\_tagging0.txt” and “POS\_tagging1.txt”. Head of the file is shown below:

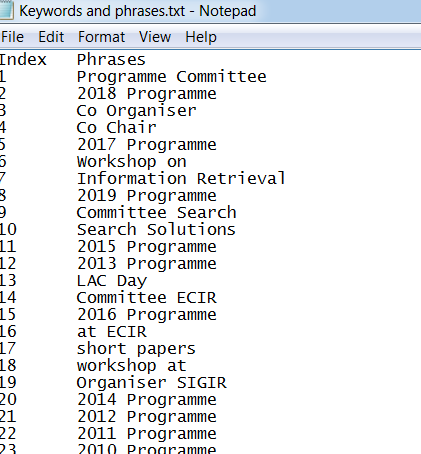


**NER Tagging:** Next program will call NER\_tagger() method. This will give the NER’s for the text using StanfordNER tagger and displaying only tagged terms. This will generate two files namely “NER\_tagging0.txt” and “NER\_tagging1.txt”. Head of the file shown below:



**Keywords and Phrases**: Our final step will be indexing the data according to Term Frequency and Inverse Document Frequency. Here we have created multiple functions to simulate the steps required to get the relevant words under a proper index. First we will extract the information from the URL’s. Secondly we will call the Compute TFDict method. This method helps with calculating the TF or term frequency Value that is number of occurrences by total number of words. Then we will call the CountDict method. This method is used to check the relevance of each word in the Documents, in other words how many documents does a keyword or phrase appear in. The next function ComputeIDFDict which we have crate will map the earlier relevance to the Inverse Document Frequency (IDF), which is nothing but the logarithmic value of number of documents in our case 2 by relevance of the word. Then we multiply our TF values with IDF Values in our function ComputeTFIDFDict and get the relevance factor for our words and phrases. Furthermore we sorted this list and as per the document tokens, we added the sorted index of keywords and phrases of each of our documents. In the end, we were able to get the below output.





CONCLUSION:

In this assignment we have extracted the text from the url’s. We have pre-processed the extracted text from the url. We have applied part-of-speech and NER tags to the extracted output. Calculated the tfidf weights of the terms and based on that extracted the key words and phrases. Here we have used the function to calculate the tfidf weight we can use predefined functions from the scikit learn. Also we have used 3 entity NER, further we could use 5 and 7 entity NER.