# Information Retrieval Report
# Assignment 1: Indexing for Web Search

Registration Nos. 1700927 and 1700239

February 2018

## 1 The Task

This report presents the design of a system which reads a list of HTML pages and converts them into structured knowledge. The proposed processing pipeline employs several techniques for extracting and processing the text from each URL, which are explained in detail in the ensuing section. The program finally delivers an indexed ranking per word that shows the tf.idf term and in which document each word is present.

## 2 Text Processing Pipeline

The proposed pipeline was programmed in Python mainly using the NLTK and other required libraries. The indexation process for web search is performed through the following steps:

1. Reading web sites: The program begins with the system asking the user to type all the URLs for which, the user would like to create an index. In this initial step, the system will store all these web pages in a list which will be used to request, read and extract the text from every URL stored.

2. Parsing HTML: The extracted documents do not only contain text but metatags, java script code and many other coded structures. In order to ensure that all valuable information is extracted from every URL listed, appropriate code was written such that all text contained into the website is rescued. The final text that would be carried forward for further processing would thus contain the plain text of the website and any other source of text stored, for example in metatags.

3. Pre-Processing: This is a critical aspect in the pipeline due to the fact that any information removed will affect the performance of the following steps. Thus, it was decided to carry out a minor pre-processing which involved three steps. The general idea was to avoid losing vital information.

i. Tokenization: It is the process of splitting a text into several parts such as words, punctuation, symbols and so forth. It is a critical step in the pipeline because this helps understand the elements of the text that are analyzed.

ii. Eliminating Symbols: Some symbols, shown below, were removed. It was done because by removing them, the performance in the pos-tag phase improved. Put it other way, the performance with these symbols was observed to be weak.

['dr', 'Dr', '@', '/', '©']

iii. Lemmatization: This process converts the word to its base form (without transforming into lower case). Lemmatization was employed so that vital information is not lost and at the same time, simplicity in searching the words is ensured.

Any other normalizing method, such as stemming, removing punctuation, other types of symbols and so forth, was not undertaken at this point since they remove vital information for the forthcoming steps. For example, lowering the case of all the letters in a token reduces the ability of the POS-tagging and name entity recognition methods to tag correctly, for example it was seen that POS-tag of Udo (NN) and udo(JJ) differs from each other.

4. Name Entity Extraction (NER) and Part-of-Speech Tagging: The functions *ne-chunck* and *pos-tag* were employed for this purpose, which does the work of recognizing entities and tagging each token.

5. Phrase Detection: To implement this process, it is necessary to create a series of grammar rules that the function *RegexpParser* will follow. These rules are displayed hereunder:

$$< NNP >< NNP > \tag{1}$$

$$< NNP >< IN >< NNP > \tag{2}$$

$$< NN.*|JJ > * < NN.* > \tag{3}$$

The grammar rule (1) was introduced since (3) fails to recognize persons' names separately and merges all the names as one single text. The same problem was detected for identifying some organization names, such as the name of universities (Example, University 'of' Essex). This is why (2) was introduced. Apart from the cases handled by (1) and (2), (3) performs with acceptable accuracy detecting meaningful construction of a sentence.

6. Post-Processing: Once the grammar rule is applied, it is required to transform the tree object [1] to a list type object. The code actually converts this tree object in to a list of list. The function *tree2conlltags* then creates a list for each word which contains the word, its POS-tag and the position of that word in the sentence.

   The process of normalizing the text, that was initiated at step 2 is now resumed. Stemming, conversion to lower case, removing the punctuations, symbols and digits is applied to get the final list.

7. Ranking: This process aims to implement the formula 4.

   $$TF.IDF = tf_t \times log(\frac{N}{df_t}) \tag{4}$$

   Term Frequency (IF)is simply the frequency with which a term or a word appears in a given document. Inverse Document Frequency (IDF), on the other hand, ranks the words in ascending order of their frequency of occurrence. The least frequently occurring word is given the highest weight. IDF is basically, the inverse function of the number of documents in which a word occurs [2]. Since, the tf-idf together checks for high term frequency and low document frequency, it helps in eliminating the words with lower importance. Breitinger, et.al. [1] show that 83 per cent of the recommender systems use tf-idf for assigning weights to terms.

   The final output is displayed as a dataframe object (a table form) containing the columns words (Dictionary of Words) and tf-idf index, separately for each URL.

## 3 Solution and the Functionality Implemented

This section discusses the functionality of the code implemented, such that it complements the discussion of Text Processing Pipeline in section 2.

After the URLs are read and parsed, tokeninzing is undertaken. During the tokenizing phase, two approaches were adopted: First, the text was normalized and then tokenized and second, tokenzing was done on the raw parsed text. The results of both approaches revealed that more information was retained using the second approach. This may be because, if the text is normalized first, all the punctuations and stopwords get removed. This perhaps makes it difficult to identify the commonly co-occurring words and relevant sentences, at a later stage.

Choosing appropriate grammar rules was a major challenge. We applied three different approaches to frame these rules before coming up with the one which is discussed in the previous section. The reasons why those rules were not adopted is because it increased the complexity and reduced the accuracy. The following rules were tried for the phrase detection task.

[1]The output generated on executing the grammar rules is an object of the type "tree".

```
{<NNP><NN><NN>}
{<NNP><IN><NNP>}
{<NNP><CC><NNP><NNP>}
{<NNP><NNP>}
{<NNP><NN>}
{<JJ><NNP><NNP>}
{<NN><NN>}
{<VBN><NNS>}
{<DT>?<JJ>*<NN>}
```

The grammar rule mentioned in section 2 (3) achieves higher accuracy in detecting sentence constructions than all the above-mentioned rules. Moreover, the arguments in favour the rule (3) finally used in the code is discussed in [3].

## 4 Possible Improvements and Extensions

The second URL is not simple html (it is a .php with frames and ¡Div¿ tags), so more sophisticated approach can be adopted in reading it. Just because, the URLs are read and parsed both the URL using simple parsing methods, the second URL produces some noise that affects the final output.

There could be a better way of dealing with punctuations and numbers in string format. This is because, despite listing all possible punctuations to be removed, some of them are still retained in the output. A regular expression can be explored that might help to remove the noise that still persists.

More grammar rules could have been explored to enhance the accuracy of the search engine. There is however, a trade-off between sophistication in grammar rules enhancing the accuracy and simple rules enhancing the efficiency. More work could have helped achieve a better balance on accuracy versus efficiency on the grammar rules.

## References

[1] Corinna Breitinger, Bela Gipp and stefan Langer *Recommender systems: A Literature Survey.* International Journal on Digital Libraries. 2015. Vol. 17, No.4. pages 305–338.

[2] Karen Spärck Jones. *A Statistical Interpretation of Term Specificity and Its Application in Retrieval.* Journal of Documentation. 1972. Vol. 28. pages 11–21.

[3] Su Nam Kim, Timothy Baldwin and Min-Yen Kan *Evaluating N-gram based Evaluation Metrics for Automatic Keyphrase Extraction.* In Proceedings of of the 23rd International Conference on Computational Linguistics (Coling 2010), Beijing, August 2010. pages 572–580.