CSE508 : Information Retrieval - Assignment 2

- *Shreyash Arya (2015097)*

PRE-PROCESSING:

Pre-processing followed is similar to the previous assignment by considering all the files except with the extension .html, .header, .footer and .musings as they do not contain any relevant information. The following pre-processing steps are taken (in mentioned sequence):

- Removing HTML tags using BeautifulSoup inbuilt module.
- Fixing the contractions (Eg. don't → do not) using contractions python library.
- Tokenization using NLTK's word tokenizer.
- Removing non-ascii characters using Unicode.
- Lowercasing the document data.
- Removing punctuations using regex (r'([^\w\s])|_+').
- Replacing numbers with words using the num2words library.
- Stemming using NLTK's Porter Stemmer.
- Removing stopwords using the NLTK's English stop-words dictionary.

> There are 467 documents after preprocessing and removing the non-required extension files.
> The vocabulary contains 38896 terms.
> Titles are extracted from the index.html present in the main folder and in the SRE folder.

DICTIONARY CREATION:

Total of nine dictionaries are made (stored as pickle files) to store the data for faster result generation:

1. 'vocab.pkl' → Contains the vocabulary terms.
2. 'prepro_files.pkl' → To store the pre-processed files from the previous step.
3. 'name_to_count.pkl' → The dictionary that maps document name to document ID.
4. 'count_to_name.pkl' → The dictionary that maps document ID to document name.
5. 'tf_dic.pkl' → TF values are stored for all the documents.
6. 'df_dic.pkl' → DF values are stored for all the documents.
7. 'tf_tit_dic.pkl' → TF values for the created title documents.
8. 'df_tit_dic.pkl' → DF values for the created title documents.
9. 'document_term_tfidf.pkl' → Contains TF-IDF values for all the terms in the vocab and there TF-IDF scores to each document.

QUESTIONS:

1) **Tf-Idf based document retrieval:** For each query, your system will output top k documents based on tf-idf-matching-score.

- The user inputs the query that is also pre-processed using similar steps as done for the documents.
- Then for each term in the query, TF-IDF values are calculated for all the documents and stored.
- The TF-IDF scores are calculated by giving special attention to the terms in the document title. The TF-IDF score is the combination of the title score and the body score: **Total_score = (α1 * title_score + α2 * body_score)**
    - α1 is taken as 0.6 and α2 is taken as 0.4 giving more weight to titles.
- Both body_score and the title_score contains the TF-IDF score which is calculated as TF_value * IDF_value.
    - The TF_value is measured by using the formula:
        - **(1 + (Frequency of word in the document / Total number of words in the document))** where 1 is added for the smoothing and so that the score doesn't become 0.
    - The IDF_value is defined as the
        - **(log-base10 [(Length of the corpus / (1 + DF value]))** where DF value is the number of documents in which the term appears and 1 is added for smoothing and so that the value doesn't become 0.
- Then for each document, the term total scores are added and sorted in descending order of the TF-IDF values to rank the top k (where k is taken as the user input).

2) **Tf-Idf based vector space document retrieval:** For each query, your system will output top k documents based on a cosine similarity between query and document vector.

- In this case, the query and documents are both considered as vectors and then the cosine similarity is calculated between the two. The k having the highest cosine similarity is returned.
- The query vector is created (of the length of the size of vocabulary = 38896) which contains the frequency of the term in the query (where the term matches the vocabulary word) and others are kept zero.
- The documents are also converted to vector as the query (with the same length) but contain the total (title+body) TF-IDF scores for each vocabulary term and for each document.
- Finally, the cosine similarity (which is already normalized by the length of the query and the document) is measured between the created vectors and the places where the terms are not present, the similarity score becomes zero in the vector and only the relevant query terms are considered for the TF-IDF values.
- Top k (user-input) results are returned.

INFERENCES:

- The vector similarity (case 2) will give the better results as compared to case 1 as the terms considered for the TF-IDF calculation are only the relevant terms (as other values in the vector becomes zero due to the absence in the query and the cosine similarity is also zero).
- For handling the numerical terms, the num2words library is used as the pre-processing part which takes care of all the numerical data.
- If the query contains the terms that are present in the title, then the results are biased to return those documents which contain the title terms. This is because the weight to the title_score is given more as compared to the body_score.
    - P.S. if we give no/less weight to the title_score then the results are returned according to the text present in the body of the document which is an obvious case.

Mention a case where 1,2 give different results. Explain why the difference occurs.

Consider the query: “All summer long, they roamed through the woods and over the plains, playing games and having fun. None were happier than the three little pigs, and they easily made friends with everyone. Wherever they went, they were given a warm welcome, but as summer drew to a close, they realized that folk were drifting back to their usual jobs, and preparing for winter. Autumn came and it began to rain. The three little pigs started to feel they needed a real home. Sadly they knew that the fun was over now and they must set to work like the others, or they'd be left in the cold and rain, with no roof over their heads. They talked about what to do, but each decided for himself. The laziest little pig said he'd build a straw hut.”

This query is taken from the ‘3lpigs.txt’ file.

Results:

Method 1:

```
Enter k for top k documents: 5

friends.txt     104.66715272
3lpigs.txt      104.663319964
game.txt        104.632069561
lmtchgrl.txt    104.490669706
3wishes.txt     104.36349551
```

Method 2:

```
Enter k for top k documents: 5
```

3lpigs.txt	0.026160113489043714
lmtchgrl.txt	0.02606002061115806
goldfish.txt	0.025955042202237533
lmermaid.txt	0.02595375715994727
wlgirl.txt	0.025903164011232195

As we can see that the results in both the methods are different (though the correct file is present in both the cases). This is because in the first case we blindly adding the TF-IDF scores without considering how the query terms are contributing to the final TF-IDF values while the vector-based method considers both TF-IDF values and how the different query terms correlate with the others present in the document.

Also, in the first case method, we can see that the top file returned is the ‘friends.txt’ which has the title “Friends: A Story” and hence more weight is given to the title even though other text terms are not relevant wrt to the query. In the second method, however, all the query terms are considered and hence even with considering the title weight, the correct document is returned at the top. This shows that the vector-based method captures the context with other words in the query and matches it with the best document.