Project-1 Comp 479/6791
Fall 2020

Dept. of Computer Science and Software Engineering
Concordia University

Submitted to: - Dr. Sabine Bergler

Submitted by: Yashika Khurana
Student id: 40094722

1. INTRODUCTION

Preprocessing of the text is one of the most important part of Information retrieval. Text preprocessing simply means to bring your text into a form that is predictable and analyzable. In this project I have applied the preprocessing techniques on the data **collection "REUTERS21578" Reuter's Corpus**.
The Reuter's corpus is a collection of newspaper articles/documents, which are stored in a number of .sgm files. I was working with this document and applying the preprocessing tasks on it**. Some of the preprocessing tasks applied to the documents include Lowercasing, Stemming (Porter stemmer), Stop words removal etc.** This project is divided into 6 blocks. Each block is dedicated to performing some preprocessing on the tasks and will generate block.json file.
This project also uses the pipeline, which means that every step can be executed in stand-alone fashion with the appropriate input and will generate output suitable as input for the next module. Pipeline steps section includes all the module steps applied and discussed the approach followed. Then in the Solutions section discussed about the functions included. In the final, Output files section discussed about the output of each block

2. PIPELINE DECISION STEPS

A pipeline means that every step can be executed in stand-alone fashion with the appropriate input and will generate output suitable as input for the next module. Modules have to obey certain formatting constraints and have to be inserted into the templates. The templates call each module a block. Simply add your code in the proper place in the appropriate script.
Each block has its own objectives to satisfy along with specific input and output structures.

2.1 Block 1 Reader

Downloaded Reuter's Corpus collection - reuters21578 contains various files including. sgm, .txt and .dtd. Our news collection documents is in .sgm files. So, the first block aim is to read all the .sgm files of the collections which have raw data. Input of this block takes path of the Reuters collections and output of this block include all the content of the .sgm files.
Approach: To get the folder file of the Reuters collection, **used the 'os' module** of the python and then checked whether extension of **the file is '.sgm'**, if it is .sgm then I read the document.
Please note that while reading the file I used **the encoding "encoding="ISO-8859-1""** otherwise it is giving error. I am using **Mac** and I researched about the encoding on supported by the Mac OS. If the file is opened in this encoding then only it was able to read

```
(venv) yashikakhurana@Yashikas-MacBook-Air-2 IR_P1 % python ./block-1-reader.py --path ./reuters21578/ -o block1.json
Traceback (most recent call last):
  File "./block-1-reader.py", line 32, in <module>
    for reuters_file_content in solutions.block_reader(path):
  File "/Users/yashikakhurana/PycharmProjects/IR_P1/solutions.py", line 20, in block_reader
    reuters_file_content= filename.read()
  File "/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/codecs.py", line 322, in decode
    (result, consumed) = self._buffer_decode(data, self.errors, final)
UnicodeDecodeError: 'utf-8' codec can't decode byte 0xfc in position 1519554: invalid start byte
```

that, otherwise if I used it gives me the following error. So, used the different encoding and it worked fine.

2.2 Block 2 Document Segmenter

Input for block 2 is the output of the block1.json, which includes all the Merged data of 22 .sgm files. The purpose of this block is to decide the scope of each news document. To split each .sgm file from the input, the best way was to split the each document by **"<!DOCTYPE lewis SYSTEM "lewis.dtd">"** which is the first line of every individual .sgm. So, used the 're' module to split all the .sgm files stored in the list.
So, now in the list we have every .sgm file content. The next step is to get the news documents. The approach I followed is to iterate the list which contains the all .sgm files, then extracting the **<REUTERS tag**. This tag will lead us to individual news document.The approach followed was to get the index of the **<REUTERS and </REUTERS> to get the first pair, then iterating over the it to reach till the end.** Then same for the next .sgm file, and it keep it going on until we extracts the news document.In this block I just extract the individual news document for the input of the next block

2.3 Block 3 Extractor

Text generated between the <REUTERS and </REUTERS> contains a lot tags eg <DATE>, <TOPICS>, <PLACES>, <PEOPLE>, <ORGS>, <EXCHANGES>, <COMPANIES>, <UNKNOWN>, <TEXT>, <TITLE>, <DATELINE>, <BODY>. Some of the **tags are noise and contains meta data**. So, the purpose of this block is to identify the important information and extract them. Also, to identify the tags which contains the important information in the news document which will be useful to construct an inverted index.
In this block I have used **the 'BeautifulSoup'** library which help us to get the information. So, by iterating all the <REUTERS. Firstly applied the **"html.parser"** because it was satisfying the purpose of the tags which need to be extracted. Secondly, then extracting the <TEXT> tag because there are some tags with **<TEXT TYPE="BRIEF"> and <TEXT TYPE="UNPROC">** which don't include any useful information and also don't have the <BODY> tag. So by filtering these tags, only useful **<TEXT>** tags are allowed for the

further purposing. In the third step, I extract the <REUTERS tag to get the id of the news document with the attribute 'NEWSID', which will help to uniquely identified the document and storing that id in the dictionary. In the fourth step, I extract the **<DATE>,<TITLE> and <BODY>** tag which is the important content of any news document and stored the result in the string. In the fifth step created the dictionary which have **the id as key and text as the concatenation of the date, title and body content.**

2.4 Block 4 Tokenizer

Tokenization is a step which splits longer strings of text into smaller pieces, or tokens. Larger chunks of text can be tokenized into sentences, sentences can be tokenized into words, etc. Further processing is generally performed after a piece of text has been appropriately tokenized. **Tokenization is also referred to as text segmentation or lexical analysis.**
The main purpose of the block is to take the output of block 3 and to split a long text into tokens using **NLTK library**. So by iterating the dictionary from the previous block which have the id and long text, First step I performed **is removing the punctuation from the text** before tokenizing it, because the if we don't remove the punctuations from the text, it will increase the no. of tokens generated. Punctuation removed from the text with the help of regex. Now the next step is to generate the tokens from the text, so with the help of NLTK library, used the functionality of **nltk.word_tokenize.** Finally made a tuple with the document it belongs to i.e. document id and token. For example (DocId, token)

2.5 Block 5 Stemmer

Text Normalization is the process of applying some linguistics models to tokens of text. Text tokens often have some minor difference but refer to same thing which needs to recognize such tokens and reduce them to the same common form. So, in this block applied **the text normalization by making the text lower case and then applied the Porter Stemmer**. The Porter stemming algorithm (or 'Porter stemmer') is a process for removing the commoner morphological and in flexional endings from words in English. Stemming is a straightforward normalization technique, most often implemented as a series of rules that are progressively applied to a word to produce a normalized form.
So, with the help **of nltk.stem.PorterStemmer** applied the Porter Stemmer on the tokens, which will be the input for the next block.

2.6 Block 6 Stop Words Removal

Stop words are just a set of commonly used **words** in any language. **Stop words are** commonly eliminated from many text processing applications because these **words can** be distracting, non-informative (or non-discriminative) and **are** additional memory overhead.

This block aims to remove the stop words. I used the **nltk.corpus.stopwords** to remove the stop words. Used **set(stopwords.words('english'))** which contains the stop words of the English language. So, the output of this block is the tuples of document id and tokens.

3. SOLUTIONS

In the solutions.py we have functions which execute all the blocks block_reader, block_document_segmenter, block_extractor, block_tokenizer, block_stemmer, block_stopwords_removal. All the code are written in the above mentioned function only. All the commands used to run the methods are mentioned in the appendix.

4. OUTPUT FILES

All the blocks generates the .json file. Block 1 will generate the block1.json and Block 2 will generate block2.json and so on. Final output will be the block6.json which contains the tuple of document id and tokens.

5. FILES SUBMITTED

The zip files contains all the output files including block1.json, block2.json, block3.json, block4.json, block5.json and block6.json for the first five news documents. It also includes the solutions.py which have all the functions code, this is written to process all the 22.sgm files.

APPENDIX:

Commands used to run each block:

Block 1:

```
python ./block-1-reader.py --path ./reuters21578/ -o block1.json
```

Block 2:

```
python ./block-2-document-segmenter.py -i block1.json -o
block2.json
```

Block 3:

```
python ./block-3-extractor.py -i block2.json -o block3.json
```

Block 4:

```
python ./block-4-tokenizer.py -i block3.json -o block4.json
```

Block 5:

```
python ./block-5-stemmer.py -i block4.json -o block5.json
```

Block 6:

```
python ./block-6-stopwords-removal.py -i block5.json -o
block6.json
```