SCC.413: Applied Data Mining

Text Summarisation Lab Session

In this lab you will learn how to create an Extractive Single-document automatic text summarizer to summarise text articles from Wikipedia in different languages.

Note: For this lab you'll be using Python3 and pip3.

By the end of this lab you should have learnt how to:

1- Get and parse an article from a URL, Wikipedia in this case.
2- Extract text paragraphs from the article in 1-.
3- Clean the extracted text to remove special characters and leading spaces.
4- Automatically detect the language of the Wikipedia article in 1- using **langdetect** python package.
5- Split the text into sentences using **NLTK**'s tokenizer.
6- Get relevant stop words list (see step 4-)
7- Calculate frequency for each word.
8- Learn how to assign scores to each sentence based on word frequencies in 7-.
9- Get a summary based on the scores in 8- by selecting the top n sentences (e.g. top three sentences with highest scores).

Now I explain each step with example so you can run that and get the results using Python3 on your machine.

1- Get and parse an article from a URL, Wikipedia in this case.

We start by defining which query to use in order to get a Wikipedia article. In this example we'll use the query '**Language'** in order to get a Wikipedia article discussing Language in general. Feel free to use other queries.

```
import urllib.request

query = 'Language'
from urllib.parse import quote
print(urllib.parse.quote(query))
scraped_data =
urllib.request.urlopen('https://en.wikipedia.org/wiki/'+urllib.parse.quote(
query))
article = scraped_data.read()
```

The above code will get an English Wikipedia article about Language and scrape its contents. Changing "**en**.wikipedia" to other languages code will display the article in different language even if the query is still in English. For example, to get the article in Arabic change **en** to **ar** as in: https://**ar**.wikipedia.org/wiki/Language. See here for more codes: https://en.wikipedia.org/wiki/List_of_ISO_639-1_codes

Now we have the file in html file format. In order to test that try saving the output to a file.

NOTE: trying to print the article's contents directly to the console might slow down your machine therefore we are printing to file instead:

```
print(article, file=open("rawArticle.txt", "a"))
```

If you open the file you'll notice it's full with HTML tags.

2- Extract text paragraphs from the article in 1-.

The next step is to parse the article's source code and get the textual material, which are needed to summarise the article. To do so we'll need to use **BeautifulSoup lxml** which can parse both XML and HTML contents. In our case the Wikipedia article is in HTML file format.

NOTE: you may need to pip install the following:

```
$ pip3 install beautifulsoup4
$ pip3 install lxml
```

We start by parsing the article.

Next we iterate trhough the article's source code and extract text in between the paragraphs tags .

The extracted paragraph's text will be then combined to form one big string (i.e. **article_text**).

```
import bs4

parsed_article = bs4.BeautifulSoup(article,'lxml')

paragraphs = parsed_article.find_all('p')

article_text = ""

for p in paragraphs:
    article_text += p.text
```

3- Clean the extracted text to remove special characters and leading spaces.

In this step we clean the text by removing any special characters and save the output to the variable **formatted_article_text** as clean plain text.

```
formatted_article_text = re.sub('[!@#$%^&*]g', ' ', article_text )
formatted_article_text = re.sub(r'\s+', ' ', formatted_article_text)
```

Try and print the output to a text file and examine the output. It should be clear from any HTML tags or special characters. As we will be dealing with other languages it is advised that we work with UTF-8 format so we can print Unicode characters as otherwise Python3 will return an encoding error.

```
print(formatted_article_text, file=open("cleanArticle.txt", "a",
encoding="utf8"))
```

4- Automatically detect the language of the Wikipedia article in 1- using langdetect python package.

This may require pip installing langdetect package:

```
$ pip3 install langdetect
```

The language detection process is done automatically. All you need to do is to call the method **detect** on the text you need to know the language for.
The method returns the code of the language rather than the language name. In the following code I added in a simple if statements to convert the code to the equivalent language.

Note: Remember you need to do that for any additional language.

```
#detect language
from langdetect import detect
lang = detect(formatted_article_text)

if lang == 'ar':
    lang = 'arabic'
if lang == 'en':
    lang = 'english'
if lang == 'es':
    lang = 'spanish';
print(lang)
```

5- Split the text into sentences using NLTK's tokenizer.

Now that we have the text and we know which language it's written in we start working on extracting the sentences. This is an important step towards summarising a document. This type of summarisation is called "Extractive Summarisation".

For this step you will need to use a **NLTK** tokenizer called **PUNKT** which is a tokenizer that uses an unsupervised algorithm to split the text into sentences, it's trained on a large sample of text and can tell the different between a '.' In Dr. Johns and an end of sentence full-stop. The code next will show you how to download punkt from **NLTK**.

If you haven't already, start by pip installing **NLTK**:

```
$ pip3 install nltk
```

Then using nltk punkt you can divide the text into sentences:

```
import nltk

nltk.download('punkt')
sentence_list = nltk.sent_tokenize(article_text)
```

try to print the list of sentences to file and examine the output.

Note: you don't always need to download punkt. This can be done once and in the next run you can replace that line with:

```
from nltk.tokenize import sent_tokenize
sentence_list = nltk.sent_tokenize(article_text)
```

6- Get relevant stop words list (see step 4-)

In Step 4- we detected the language of the article. This is now useful to know which stop-words list to use.

As in the previous step you'll have to download the stopwords lists from **NLTK**. Next we'll save the stopwords contents to a variable as follows:

```
nltk.download('stopwords')
stopwords = nltk.corpus.stopwords.words(lang)
```

Try to print the stopwords directly to the console to examine the stopwords list:

```
print(stopwords)
```

As in the previous step, you only need to download the stopwords from NLTK once, later you can just call the following line without the need to download:

```
stopwords = nltk.corpus.stopwords.words(lang)
```

7- Calculate frequency for each word.

This process involves tokenizing text into words, removing stopwords and calculating frequencies.

```
word_frequencies = {}
for word in nltk.word_tokenize(formatted_article_text):
    if word not in stopwords:
        if word not in word_frequencies.keys():
            word_frequencies[word] = 1
        else:
            word_frequencies[word] += 1

maximum_frequncy = max(word_frequencies.values())

for word in word_frequencies.keys():
    word_frequencies[word] = (word_frequencies[word]/maximum_frequncy)
```

In the script above, we loop through all the sentences and then corresponding words to first check if they are stop words. If not, we proceed to check whether the words exist in the **word_frequencies** dictionary. If the word is encountered for the first time, it is added to the dictionary as a key and its value is set to 1. Otherwise, if the word previously exists in the dictionary, its value is simply updated by 1.

8- Learn how to assign scores to each sentence based on word frequencies in 7-.

After calculating the weighted frequencies for all the words. We now calculate the scores for each sentence by adding weighted frequencies of the words that occur in that particular sentence.

```
sentence_scores = {}
for sent in sentence_list:
    for word in nltk.word_tokenize(sent.lower()):
        if word in word_frequencies.keys():
            if len(sent.split(' ')) < 30:
                if sent not in sentence_scores.keys():
                    sentence_scores[sent] = word_frequencies[word]
                else:
                    sentence_scores[sent] += word_frequencies[word]
```

In the script above, we first create an empty **sentence_scores** dictionary. The keys of this dictionary will be the sentences themselves and the values will be the corresponding scores of the sentences. Next, we loop through each sentence in the **sentence_list** (see point -5) and tokenize the sentence into words.

We then check if the word exists in the **word_frequencies** dictionary. This check is performed since we created the **sentence_list** list from the **article_text** object; on the other hand, the word frequencies were calculated using the **formatted_article_text** object, which doesn't contain any stop words, numbers, etc. We do not want very long sentences in the summary, therefore, we calculate the score for only sentences with less than 30 words (although you can tweak this parameter for your own use-case). Next, we check whether the sentence exists in the **sentence_scores** dictionary or not. If the sentence doesn't exist, we add it to the **sentence_scores** dictionary as a key and assign it the weighted frequency of the first word in the sentence, as its value. On the contrary, if the sentence exists in the dictionary, we simply add the weighted frequency of the word to the existing value.

9- Get a summary based on the scores in 8- by selecting the top n sentences (e.g. top three sentences with highest scores).

Now we have the **sentence_scores** dictionary that contains sentences with their corresponding score. To summarize the article, we can take top **n** sentences with the highest scores. The following script retrieves top 3 sentences and prints them on the screen.

```
#Create summary, threshold defines number of top sentences with highest
weight to include
import heapq
threshold = 3
summary_sentences = heapq.nlargest(threshold, sentence_scores,
key=sentence_scores.get)

#Merge sentences into one summary
summary = ' '.join(summary_sentences)
print(summary)
```

Extra:

Now try to rerun the summariser on other languages and see what results you get. Remember you can tweak the URL in 1- to convert the article into other languages. Also try using other queries or maybe try to read the text from a file on your machine rather than reading it online from Wikipedia.

Graph-based Summarisation system:

You may have noticed that the quality of the summaries are not great due to the simplicity of the approach used.

You can compare that to another summarisation system such as textRank which uses a graph-based algorithm summarisation technique (see reference below for more details on graph-based summarisation):

You can clone textRank from the following GitHub repository: https://github.com/summanlp/textrank

Start by installing textRank:

```
$ pip3 install summa
```

To test textRank on a text file stored locally on your machine:

```
$ textRank -t fileLocation
```

Text Rank Graph-Based Summariser reference: http://www.aclweb.org/anthology/W04-3252

"In short, a graph-based ranking algorithm is a way of deciding on the importance of a vertex within a graph, by taking into account global information recursively computed from the entire graph, rather than relying only on local vertex-specific information"