Text Mining Lab1

For these lectures, we will use different libraries (see the Annexe). Import them and make sure there are no errors during the import.
During this tutorial you will able to mine text data and extort knowledge from it. Analyse well the obtained results and discuss them in the report and send the report to the Professor.

Part I. Text processing

1. Tokenisation

Tokenisation involves three steps, which are breaking a complex sentence into words, understanding the importance of each word with respect to the sentence, and finally produce a structural description on an input sentence.

Import the libraries:

```
import pandas as pd
import numpy as np
import nltk
import os
import nltk.corpus
```

Sample text for performing tokenization :
text = “Your professor Nistor Grozavu is member of the ETIS laboratory. ETIS laboratory consist of 5 teams and has 150 researchers. Nistor main research work is data mining and machine learning. Machine Learning is also used in Text Mining and words researching. »

Importing word_tokenize from nltk :

```
from nltk.tokenize import word_tokenize
```

Passing the string text into word tokenize for breaking the sentences :

```
token = word_tokenize(text)
token
```

2. Finding frequency distinct in the text

```
from nltk.probability import FreqDist
```

To find the frequency of top k words :

```
fdist1 = fdist.most_common(k)
```

3. Stemming

Importing Porterstemmer from nltk library
Checking for the word ‘research’

```
from nltk.stem import PorterStemmer
```

```
pst = PorterStemmer()
pst.stem(“research”)
```

Checking for the list of words

```
stm = ["research", "researching", "researchers"]
for word in stm :
  print(word+ ":" +pst.stem(word))
```

Test for 2 other words.

4. Lemmatisation

In simpler terms, it is the process of converting a word to its base form. The difference between stemming and lemmatization is, lemmatization considers the context and converts the word to its meaningful base form, whereas stemming just removes the last few characters, often leading to incorrect meanings and spelling errors.

Importing Lemmatizer library from nltk :

```
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
```

```
print(“rocks :”, lemmatizer.lemmatize(“rocks”))
print(“corpora :”, lemmatizer.lemmatize(“corpora”))
```

Test for other words.

5. Stop Words

“Stop words” are the most common words in a language like “the”, “a”, “at”, “for”, “above”, “on”, “is”, “all”. These words do not provide any meaning and are usually removed from texts. We can remove these stop words using nltk library.

Importing stopwors from nltk library :

```
from nltk import word_tokenize
from nltk.corpus import stopwords
a = set(stopwords.words(‘english’))
text = “Cristiano Ronaldo was born on February 5, 1985, in Funchal, Madeira, Portugal.”
text1 = word_tokenize(text.lower())
print(text1)
stopwords = [x for x in text1 if x not in a]
print(stopwords)
```

6. Part of speech tagging (POS)

Part-of-speech tagging is used to assign parts of speech to each word of a given text (such as nouns, verbs, pronouns, adverbs, conjunction, adjectives, interjection) based on its definition and its context.

```
text = “vote to choose a particular student or a group (party) to represent them in Cergy University Conseil”

Tokenize the text :
tex = word_tokenize(text)
for token in tex:
print(nltk.pos_tag([token]))
```

7. Named entity recognition

It is the process of detecting the named entities such as the person name, the location name, the company name, the quantities, and the monetary value.

```
text = “Cergy Paris University President Francois Germinet introduced the new research directives at Saint Martin campus”
Importing chunk library from nltk :
from nltk import ne_chunk
Tokenize and POS Tagging before doing chunk:
token = word_tokenize(text)
tags = nltk.pos_tag(token)
chunk = ne_chunk(tags)
chunk
```

8. Chunking

Chunking means picking up individual pieces of information and grouping them into bigger pieces. In the context of NLP and text mining, chunking means a grouping of words or tokens into chunks.

```
text = “We saw the yellow dog”
token = word_tokenize(text)
tags = nltk.pos_tag(token)reg = “NP: {<DT>?<JJ>*<NN>}”
a = nltk.RegexpParser(reg)
result = a.parse(tags)
print(result)
```

9. Tf-IDF - the frequency calculation¶

The tf-idf (term frequency–inverse document frequency) calculation is used to calculate a proximity score between a search term and a document (this is what search engines do).

```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer(stop_words=stopwords)
tfs = tfidf.fit_transform(train['Text'])
list(tfidf.vocabulary_.keys())[:10] + [‘...']
```

10. Test on a dataset :

```
import os
if not os.path.exists('spooky.csv'):
    from pyensae.datasource import download_data
    download_data('spooky.csv',
                  url='https://raw.githubusercontent.com/GU4243-ADS/spring2018-project1-ginnyqg/
master/data/')
```

Part II. Clustering text data

1. Import wikipedia data.

The content of each Wikipedia article is stored wiki_list while the title of each article is stored in variable title.

```
import pandas as pd
import wikipedia
articles=['Data Science','Artificial intelligence','Machine Learning','European Central
Bank','Bank','Financial technology','International Monetary Fund','Basketball','Swimming','Tennis']
wiki_lst=[]
title=[]
for article in articles:
   print("loading content: ",article)
   wiki_lst.append(wikipedia.page(article).content)
   title.append(article)
print("examine content")
wiki_lst
```

2. Represent each article as a vector

Perform k-means clustering

```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
```

Cluster the data using k=2 to 10

Use the elbow method to find the best number of clusters by computing the Silhouette index.

Note. In order to have more documents par cluster, you can change the text dataset and use the 20-NewsGroup as follows :

Load 4 categories for example:

```
categories = ['alt.atheism', 'soc.religion.christian',
              'comp.graphics', 'sci.med']
```

We can now load the list of files matching those categories as follows:

```
from sklearn.datasets import fetch_20newsgroups
twenty_data = fetch_20newsgroups(subset='train', categories=categories, shuffle=True, random_s-
tate=42)
```

4. Evaluate the result

Since we have used only 10 articles, it is fairly easy to evaluate the clustering just by examining what articles are contained in each cluster. That would be difficult for a large corpus. A nice way is to create a word cloud from the articles of each cluster.

```
from wordcloud import WordCloud
result={'cluster':labels,'wiki':wiki_lst}
result=pd.DataFrame(result)
for k in range(0,true_k):
  s=result[result.cluster==k]
  text=s['wiki'].str.cat(sep=' ')
  text=text.lower()
  text=' '.join([word for word in text.split()])
  wordcloud = WordCloud(max_font_size=50, max_words=100, background_color="white").gen-
erate(text)
  print('Cluster: {}'.format(k))
  print('Titles')
  titles=wiki_cl[wiki_cl.cluster==k]['title']
  print(titles.to_string(index=False))
  plt.figure()
  plt.imshow(wordcloud, interpolation="bilinear")
  plt.axis("off")
  plt.show()
```

References :

1. https://towardsdatascience.com/clustering-documents-with-python-97314ad6a78d
2. https://towardsai.net/p/data-mining/text-mining-in-python-steps-and-examples-78b3f8fd913b
3. http://www.xavierdupre.fr/app/ensae_teaching_cs/helpsphinx3/notebooks/td2a_eco_NLP_tf_idf_ngrams_LDA_word2vec_sur_des_extraits_litteraires.html
4. https://www.expertsystem.com/natural-language-processing-and-text-mining/
5. https://www.nltk.org
6. https://www.edureka.co

Annexe

```
#! pip install wordcloud
#!pip install gensim
#!pip install pywaffle
#!pip install keras
#!pip install tensorflow
#import nltk
#nltk.download('stopwords')
#nltk.download('punkt')
#nltk.download('genesis')
#nltk.download(‘wordnet')
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import seaborn as sns
import matplotlib.pyplot as plt
from wordcloud import WordCloud
from IPython.display import display
```

```
import base64
import string
import re
import os
from collections import Counter
from time import time
# from sklearn.feature_extraction.stop_words import ENGLISH_STOP_WORDS as stopwords
from sklearn.metrics import log_loss
import matplotlib.pyplot as plt
from pywaffle import Waffle

from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.decomposition import NMF, LatentDirichletAllocation
import nltk

from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.collocations import *
try:
    stopwords = set(stopwords.words('english'))
except LookupError:
    import nltk
    nltk.download('stopwords')
    stopwords = set(stopwords.words('english'))
#stopwords
```