MOHAMMED V UNIVERSITY of Rabat
Faculty of Sciences

Computer Science department

Master in Data Engineering and Software Development

Natural Language Processing

entitled :

Text Summarization

Prepared by :

MAROUANE HAGOUCHE
LOUGUEMIHI IMANE

Supervisor :

Pr. ABDELHAK MAHMOUDI

Academic year 2020-2021

Contents

Chapter 1

Text summarization

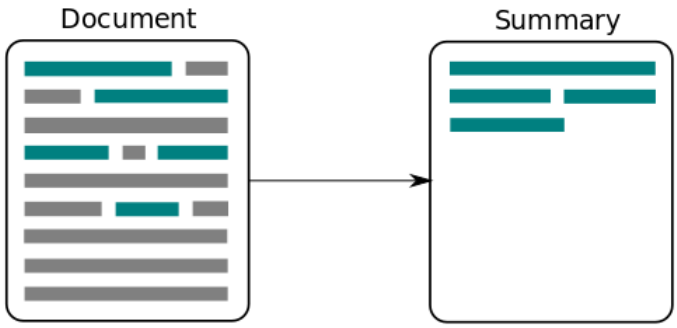
1.1 Introduction

The world of texts is a vast with the massive increase in the text information that we receive every day, we now have the Internet and various forms of social media that we consume to stay connected with the world with short messages and statuses. This increase in data applies to large documents as well. We as humans tend to have short attention, and this leads us to get bored when consuming or reading large text documents and articles. Text summarization system could be helpful in finding the most important contents of the text in a short time and it is an extremely important concept in text analytics that is used by businesses and analytical firms to shorten and summarize huge document.

1.2 Definitions

1.2.1 Text summarization

Text summarization refers to trying to extract important concepts and themes from a given full text or full texts. These extracting summaries from the original huge text done without losing vital information. The summary by human costs time and its need understanding and intelligence. It is very hard to summarize many texts manually and individually, so in order to solve this problem automatic summarization came. Automatic summarization actually means to select automatically important portions which are paragraphs or sentences from a full text or full texts.



Text Summarization visualization

Text summarization methods can be grouped into two main categories: Extractive and Abstractive methods

1. **Extractive Summarization**

   Extractive summarization is the traditional method developed first, a subset of the text from the original text is chosen to the summary (sentences, paragraphs or phrases), Extractive summarization can be viewed as a process with two main steps; representation of the source text and a mechanism for selecting sentences from the source representation.

2. **Abstractive Summarization**

   In abstractive summarization, the content of the text is learned the approach is to identify the important sections, interpret the context and reproduce in a new way and the sentences in summary are generated, not just extracted from original text.

1.3 Algorithm description

In this section, we will describe algorithms for automatic summarization.

1.3.1 TextRank

TextRank is an extractive summarization technique. It is based on the concept that words that occur more frequently are significant. Hence, the sentences containing highly frequent words are important. The algorithm can be described as a three-step process including sentence representation, sentence ranking, and sentence selection.

1. **Sentence representation**

   The input text is represented as a graph, where each sentence is converted to a node where an edge between two nodes represents similarity between the two sentences. There are multiple different ways of measuring the similarity between sentences.

2. **Sentence ranking**

   When the sentences have been converted to a graph, the next step is to rank each node (sentence) in the graph, which is done with the PageRank algorithm. PageRank is the basis of the Google search engine.

   PageRank determines the relative importance of web pages using the link structure of the web. The result is a ranking where web pages with high centrality on the web are given preference and are considered more important.

   The core algorithm in PageRank is a graph-based scoring or ranking algorithm, where pages are scored or ranked based on their importance. Web sites and pages contain further links embedded in them, which link to more pages with more links, and this continues across the Internet and nodes(web pages) connected to higher scored node always have an importance score.

   The web is modeled as a graph where web pages are represented by nodes with directed edges if there is a link from a page to another. Even if the model was initially developed for web pages, it can be used In the context of graph-based summarization, the nodes represent sentences, and the edges represent similarity between them, but the underlying mechanics works just the same.

3. **Sentence selection**

   When each sentence has been scored, the last step is to select which sentences to output as the summary given the desired length of the summary. One usual approach is to simply pick the highest scoring sentences until the desired length is obtained.

1.3.2 LexRank

LexRank is an unsupervised graph based approach for automatic text summarization. The scoring of sentences is done using the graph method. LexRank is used for computing sentence importance based on the concept of eigenvector centrality in a graph representation of sentences. LexRank is a calculation basically indistinguishable to TextRank, and both utilize this methodology for record synopsis. In both LexRank and TextRank, a diagram is developed by making a vertex for every sentence in the report. The edges between sentences depend on some type of semantic likeness or substance cover. While LexRank utilizes cosine comparability of TF-IDF vectors, TextRank utilizes a fundamentally the same measure taking into account the quantity of words two sentences has in like manner. In both calculations, the sentences are positioned by applying PageRank to the subsequent diagram.

LEXRANK	TEXTRANK
In addition to PageRank approach, it uses similarity metrics	Uses typical PageRank approach
Considers position and length of sentences	Does not consider any such parameters
Use for Multi-document summarization	Used for Single document summarization

TF-IDF

TF-IDF (Term Frequency-Inverse Document Frequency) is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. Is the most popular scheme of weighting words in the area of information retrieval but requires the references to the entire text collection which is called corpus. The word weights which are computed by the TF-IDF scheme are proportional to the occurrences in the given text, but reversely proportional to that in other texts. The TF-IDF weight, w_{ij} , of the word, t_i, in the text, d_i, is computed by

$$w_{ij} = log\frac{N}{DF_i}(1 + logTF_i) \tag{1.3.1}$$

N : the total number of texts in the collection.
DF_i : the number of texts which include the word, t_i, in the collection.
TF_i : the occurrence of the word, t_i, in the text, d_i.

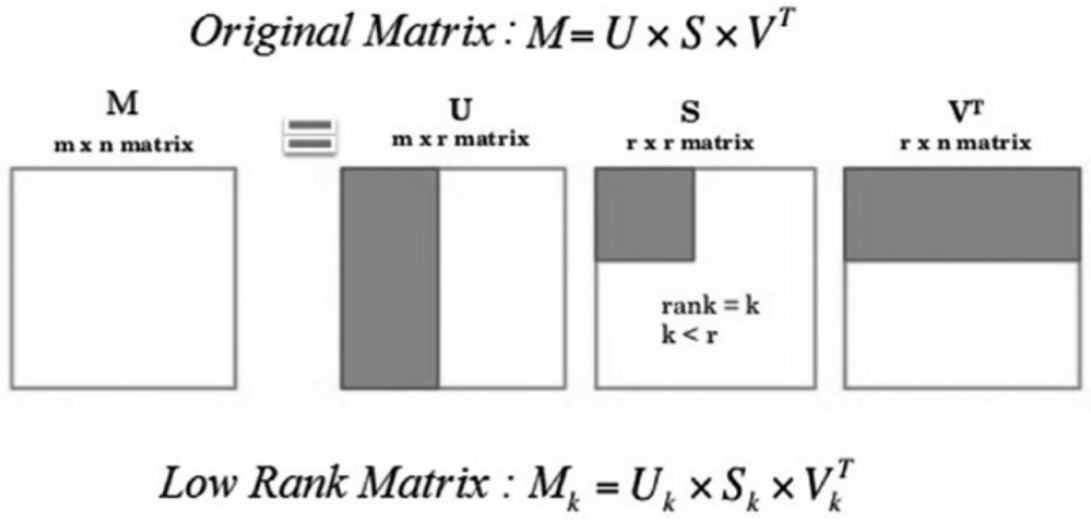
1.3.3 Latent Semantic Analysis

The core principle behind latent semantic analysis (LSA) is that in any document, there exists a latent structure among terms, which are related contextually and hence should be correlated in the same singular space.

Here, we will use summarizing text documents by utilizing document sentences, the terms in each sentence of the document and using some sort of feature weights like Bag of Words or TF-IDF weights. and with this methods our terms get their weight. However, when it occurs to corpus (group of documents) there is a subtle drawback, we cannot infer anything by observing, sometimes too large to even compute for further processes. In this case, we perform a Low-Rank Approximation using a Dimensional reduction technique using a Singular Value Decomposition (SVD).

Singular value decomposition is a technique in linear algebra that factorizes any matrix M into the product of 3 separate matrices: M=U*S*V, such that U and V are the orthogonal matrices and S is a diagonal matrix of the singular values of M.

The original matrix can be represented as a term-document matrix, where the rows will be terms and each column will be a document, that is, a sentence from our document in this case. The values can be any type of weighting, we use Bag of Words model-based frequencies.



we perform the following computations :

- Getting the sentence vectors from the matrix V (k rows).
- Getting the top k singular values from S.
- Applying a threshold-based approach to remove singular values that are less than half of the largest singular value if any exist.
- Multiplying each term sentence column from V squared with its corresponding singular value from S also squared, to get sentence weights per topic.
- Computing the sum of the sentence weights across the topics and take the square root of the final score to get the salience scores for each sentence in the document.

1.3.4 SumBasic

SumBasic is an algorithm that uses a sentence selection component based on the frequency with a component to re-weight the word probabilities in order to minimize redundancy. it calculates the probability distribution of the words that appear in the input and, then it computes its average and assigns it a weight equal to this value in each sentence. After that, the best scoring phrase that contains in the most likely word is considered. Finally, the probability of each word in the sentence that has been chosen in the previous step is updated. The algorithm can be applied iteratively to reduce further the length of the summary.

1.4 Software prerequisites and their installation, configuration, cmd and codes

1.4.1 Jupyter Notebook

The Jupyter Notebook is an open-source web application that allows to create and share documents that contain live code, equations, visualizations and narrative text. Uses

include: data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.

Jupyter Notebooks Requirements

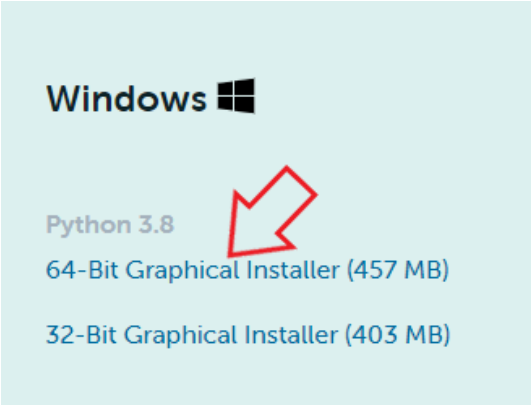
Memory and disk space required per user: 512MB RAM + 1GB of disk + .5 CPU core.

installing Jupyter Notebook

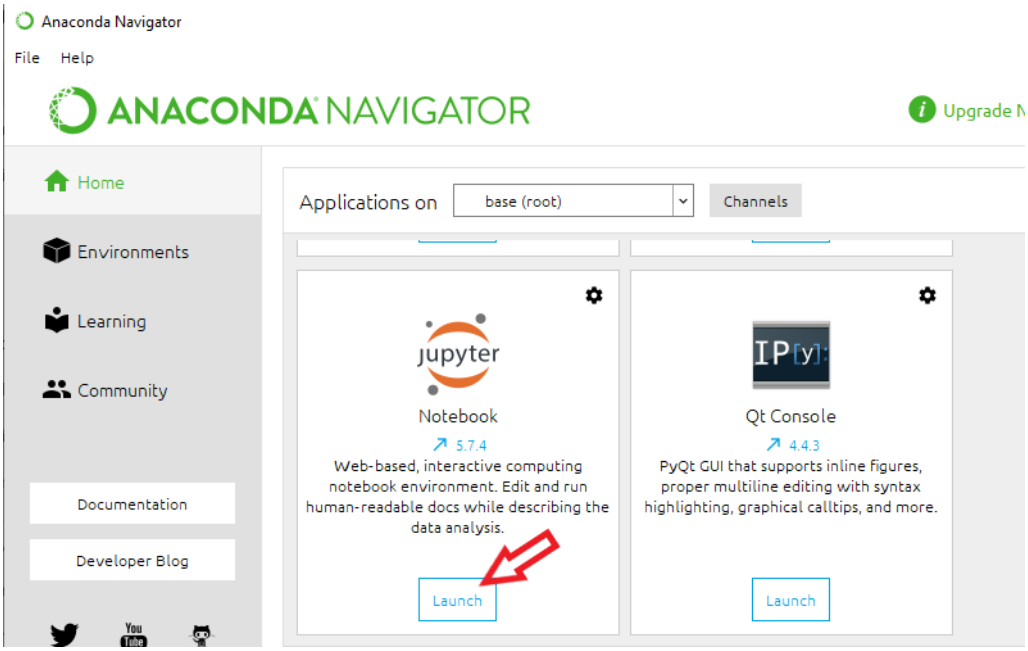
First of all we download and install Anaconda Navigator . It is a desktop graphical user interface (GUI) included in Anaconda distribution that allows us to launch applications and easily manage conda packages, environments, and channels.

Link for download : https://www.anaconda.com/products/individual

We choose the correct version :



After the installing we run Anaconda Navigator in the appearing window we click install button for Jupyter Notebook then launch button

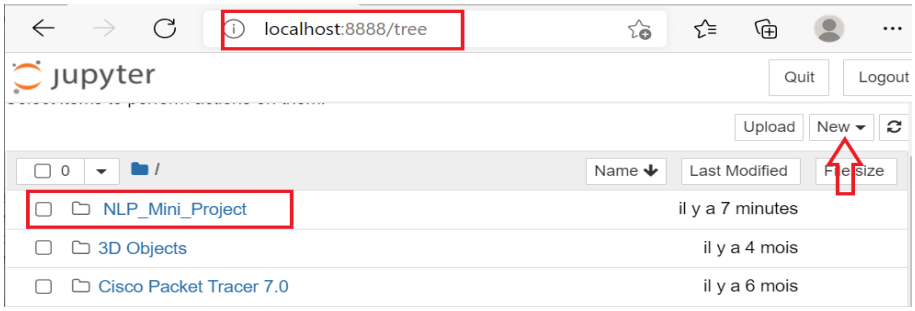


Jupyter Notebook work perfectly

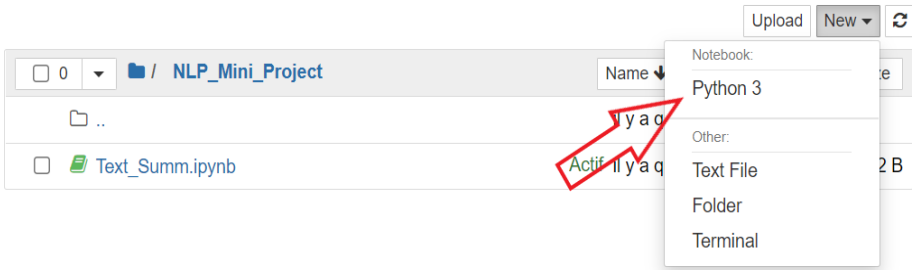
```
(base) C:\Users\HP>jupyter notebook
[I 2021-09-01 16:29:08.183 LabApp] JupyterLab extension loaded from C:\ProgramData\Anaconda3\lib\site-packages\jupyterlab
[I 2021-09-01 16:29:08.187 LabApp] JupyterLab application directory is C:\ProgramData\Anaconda3\share\jupyter\lab
[I 16:29:08.195 NotebookApp] Serving notebooks from local directory: C:\Users\HP
[I 16:29:08.195 NotebookApp] Jupyter Notebook 6.3.0 is running at:
[I 16:29:08.196 NotebookApp] http://localhost:8888/?token=2a2e5ff9cc0647a9e65c61d602d6eb009d121ac6cf3e8f67
[I 16:29:08.197 NotebookApp]  or http://127.0.0.1:8888/?token=2a2e5ff9cc0647a9e65c61d602d6eb009d121ac6cf3e8f67
[I 16:29:08.198 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 16:29:08.335 NotebookApp]

    To access the notebook, open this file in a browser:
        file:///C:/Users/HP/AppData/Roaming/jupyter/runtime/nbserver-4264-open.html
    Or copy and paste one of these URLs:
        http://localhost:8888/?token=2a2e5ff9cc0647a9e65c61d602d6eb009d121ac6cf3e8f67
     or http://127.0.0.1:8888/?token=2a2e5ff9cc0647a9e65c61d602d6eb009d121ac6cf3e8f67
```

By using http://localhost:8888/tree we can access to the web interface, we click new and folder to create new folder for our project



Inside the folder we can create "Python 3" files for our project.



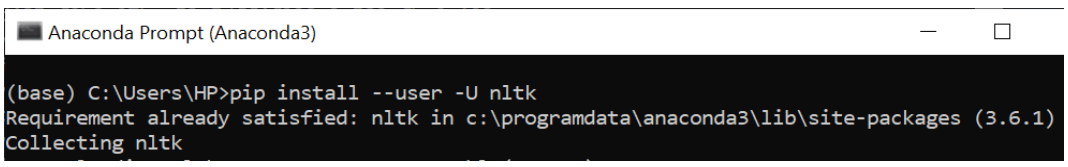
1.4.2 NLTK Library

NLTK is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries, and an active discussion forum.

Installing NLTK

NLTK requires Python versions 3.5, 3.6, 3.7, or 3.8.

In Anaconda Prompt we enter nltk installation command:



1.4.3 NumPy Library

NumPy is a Python library open source project used for working with arrays. It also has functions for working in domain of linear algebra, fourier transform, and matrices.

Installing NumPy

We use the pip command: !pip install numpy

```
Entrée [4]: !pip install numpy
            Requirement already satisfied:
```

1.4.4 Pandas Library

Pandas is a Python library help work with datasets in Python.

Installing Pandas

We use the pip command: !pip install pandas

```
Entrée [3]: !pip install pandas
            Requirement already satisfied: pandas in c:\programdata\anaconda3\lib\site-pack
            ages (1.2.4)
            Requirement already satisfied: numpy>=1.16.5 in c:\programdata\anaconda3\lib\si
            te-packages (from pandas) (1.20.1)
```

1.4.5 Scipy Library

The SciPy library provides many user-friendly and efficient numerical routines, such as routines for numerical integration, interpolation, optimization, linear algebra, and statistics.

Installing Scipy

We use the pip command: !pip install scipy

```
Entrée [2]: !pip install scipy
            Requirement already satisfied: scipy in c:\programdata\anaconda3\lib\site-packa
            ges (1.6.2)
```

1.4.6 Sklearn Library

Sklearn is a free machine learning library for Python. It features various algorithms like support vector machine, random forests, and k-neighbours, and it also supports Python numerical and scientific libraries like NumPy and SciPy.

Installing Sklearn

We use the pip command: !pip install sklearn

```
Entrée [1]: !pip install sklearn

            Requirement already satisfied: sklearn in c:\programdata\anaconda3\lib\site-pac
            kages (0.0)
```

1.5 Linguistic features

When creating a Text Mining system (Text Summarization, Sentiment analysis...), some preprocessing is needed based on some linguistic features of the input text as the follow:

1.5.1 Tokenization

Tokenization is as the process of segmenting a text or texts into tokens by the white space or punctuation marks. The tokens could be both individual words or entire sentences.

1.5.2 Text Normalization

The process of cleaning, normalizing, and standardizing textual data with techniques like removing special symbols and characters, removing extraneous HTML tags, removing stopwords, correcting spellings, stemming, and lemmatization.

1.5.3 Stop-Word Removal

Stop-word removal refers to the process of removing stop words from the list of tokens. Stop words are the grammatical words, which are irrelevant to text contents, so they need to be removed because they do not have any meaningful semantics.

1.5.4 PoS tagging

A PoS tagger classify words. An example of this would be to tag a word as a noun, adjective or verb etc.

1.5.5 Stemming

Word will occur in a text in different forms stemming is a heuristic approach that chops of the ends of words such that these similar words look the same fishing, fished, and fisher to the stem fish.

1.5.6 Lemmatization

The process of lemmatization is very similar to stemming we remove word affixes to get to a base form of the word. But in this case lemmatization uses vocabulary and morphological analysis of words to return the base or dictionary form of a word For

example, "better" is reduced to "good" and "walking" is reduced "walk" (stemming and lemmatization results in the same base in this case).

1.6 Programs and Results

Here, we will discuss our programs steps in detail:

The first thing we need to do is the process of cleaning, normalizing, and standardizing textual data with techniques like removing special symbols and characters, removing extraneous HTML tags, removing stopwords, correcting spellings, stemming, and lemmatization. This steps of cleaning we do to our document.

we remove special symbols and characters and we get the number of sentence in the document.

Entrée [8]:
```
import re

DOCUMENT = re.sub(r'\n|\r', ' ', DOCUMENT)
DOCUMENT = re.sub(r' +', ' ', DOCUMENT)
DOCUMENT = DOCUMENT.strip()
```

Entrée [10]:
```
import nltk

sentences = nltk.sent_tokenize(DOCUMENT)
len(sentences)
```

Out[10]: 53

Function of cleaning

Entrée [48]:
```
import numpy as np
import re

stop_words = nltk.corpus.stopwords.words('english')

def normalize_document(doc):
    # lower case and remove special characters\whitespaces
    doc = re.sub(r'[^a-zA-Z\s]', '', doc, re.I|re.A)
    doc = doc.lower()
    doc = doc.strip()
    # tokenize document
    tokens = nltk.word_tokenize(doc)
    # filter stopwords out of document
    filtered_tokens = [token for token in tokens if token not in stop_words]
    # re-create document from filtered tokens
    doc = ' '.join(filtered_tokens)
    return doc

normalize_corpus = np.vectorize(normalize_document)
```

print the 5 clean sentence .

```
Entrée [15]: norm_sentences = normalize_corpus(sentences)
             norm_sentences[:5]

Out[15]: array(['morocco growing popular tourist destination incredibly accessible way experience north africa
       n culture budget',
              'located northwestern edge massive african continent morocco quick flight europebudget airline
       s like ryanair fly morocco less super easy fit morocco european travel adventure',
              'youre coming elsewhere world marrakech casablanca fez major international airports reliable a
       irlines flying daily',
              'easy flights available reason visit morocco',
              'country desert mountains beaches small villages big cities little something everyone'],
             dtype='<U173')
```

Then we calculate the weight of words in sentence using TfidfVectorizer the result is a matrix of words and their weights.

```
Entrée [16]: from sklearn.feature_extraction.text import TfidfVectorizer
             import pandas as pd

             tv = TfidfVectorizer(min_df=0., max_df=1., use_idf=True)
             dt_matrix = tv.fit_transform(norm_sentences)
             dt_matrix = dt_matrix.toarray()

             vocab = tv.get_feature_names()
             td_matrix = dt_matrix.T
             print(td_matrix.shape)
             pd.DataFrame(np.round(td_matrix, 2), index=vocab).head(10)

             (353, 53)
```

Out[16]:

	0	1	2	3	4	5	6	7	8	9	...	43	44	45	46	47	48	49	50	51	52
able	0.00	0.0	0.0	0.0	0.0	0.00	0.0	0.0	0.0	0.0	...	0.00	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
abundance	0.00	0.0	0.0	0.0	0.0	0.00	0.0	0.0	0.0	0.0	...	0.00	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
accessible	0.31	0.0	0.0	0.0	0.0	0.00	0.0	0.0	0.0	0.0	...	0.00	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
accommodations	0.00	0.0	0.0	0.0	0.0	0.00	0.0	0.0	0.0	0.0	...	0.32	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
adding	0.00	0.0	0.0	0.0	0.0	0.00	0.0	0.0	0.0	0.0	...	0.00	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
adventure	0.00	0.2	0.0	0.0	0.0	0.00	0.0	0.0	0.0	0.0	...	0.00	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
adventures	0.00	0.0	0.0	0.0	0.0	0.00	0.0	0.0	0.0	0.0	...	0.00	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
affordable	0.00	0.0	0.0	0.0	0.0	0.43	0.0	0.0	0.0	0.0	...	0.00	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
africa	0.00	0.0	0.0	0.0	0.0	0.00	0.0	0.0	0.0	0.0	...	0.00	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
african	0.28	0.2	0.0	0.0	0.0	0.00	0.0	0.0	0.0	0.0	...	0.00	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

10 rows × 53 columns

1.6.1 Latent Semantic Analysis

We applicate the steps of Latent Semantic Analysis (LSA) algorithm:

get low rank SVD components, we choose number of sentences = 8 and number of topics = 3 and remove singular values below threshold .

Entrée [42]:
```
###Latent Semantic Analysis (LSA)
```

Entrée [43]:
```
from scipy.sparse.linalg import svds
```

Entrée [44]:
```
# low rank SVD
def low_rank_svd(matrix, singular_count=2):
    u, s, vt = svds(matrix, k=singular_count)
    return u, s, vt
```

Entrée [45]:
```
num_sentences = 8
num_topics = 3
# get low rank SVD components
u, s, vt = low_rank_svd(td_matrix, singular_count=num_topics)
print(u.shape, s.shape, vt.shape)
term_topic_mat, singular_values, topic_document_mat = u, s, vt
```

```
(353, 3) (3,) (3, 50)
```

we compute salience scores for all sentences in document.

Entrée [46]:
```
# remove singular values below threshold
sv_threshold = 0.5
min_sigma_value = max(singular_values) * sv_threshold
singular_values[singular_values < min_sigma_value] = 0
```

Entrée [47]:
```
# compute salience scores for all sentences in document
salience_scores = np.sqrt(np.dot(np.square(singular_values),
                                 np.square(topic_document_mat)))
salience_scores
```

Out[47]:
```
array([0.36068191, 0.52457065, 0.10764799, 0.35665745, 0.36367493,
       0.06644566, 0.38297003, 0.34570092, 0.50153681, 0.27013283,
       0.28672518, 0.12066918, 0.13614824, 0.29533541, 0.29979231,
       0.15387121, 0.20541654, 0.04082031, 0.24483424, 0.49822149,
       0.27656639, 0.53315594, 0.14628311, 0.4426064 , 0.44032635,
       0.16681535, 0.28606176, 0.35817741, 0.37708389, 0.22889358,
       0.48288288, 0.39795767, 0.06553104, 0.37399227, 0.10845917,
       0.36740725, 0.48553991, 0.2504293 , 0.46713609, 0.28005087,
       0.2803087 , 0.29174462, 0.070966  , 0.43988483, 0.15055325,
       0.35283103, 0.12704831, 0.14408735, 0.16750811, 0.09457124])
```

Rank the sentences based on their scores, and we sort them, print result.

Entrée [48]:
```
# rank sentences based on their salience scores
top_sentence_indices = (-salience_scores).argsort()[:num_sentences]
# sort top sentence
top_sentence_indices.sort()
```

Entrée [49]:
```
print('\n'.join(np.array(sentences)[top_sentence_indices]))
```

```
Located on the north-western edge of the massive African continent, Morocco is just a quick flight fr
om Europe.Budget airlines like Ryanair fly to Morocco for less than $100, so it's super easy to fit M
orocco into a European travel adventure.
here is a list of reasons to travel to Morocco.
Morocco’s Atlas Mountains have three separate ranges: the High Atlas, the Middle Atlas, and the Anti
Atlas.
Jbel Toubkal is the country’s highest mountain, located in the High Atlas Mountains.
The Rif Mountains, in the north of the country and close to the blue city of Chefchaouen, are also gr
eat for outdoor adventures.
Morocco is known for its tasty cuisine.
The most well-known Moroccan dishes include couscous and tagine.
You’ll also find numerous sandwiches and pizzas, as well as a wide array of Moroccan pastries.
```

1.7 Discussion

The creation of a summary, or a headline, to correctly represent the meaning of a document it is achievable with several methods. Some of them rely on information retrieval techniques, while others are more advanced. We have two strategies: extracting sentences or parts thereof from the original text, generating abstract summaries.

First, for this to work we have to exclude what are called stopwords. These are common words present in most documents, such as the or is. Otherwise, we might find meaningless sentences that include lots of them. We could also perform tokenization and stemming and lemmatization before applying these algorithms to improve the results.

We see the SumBasic algorithm this technique is quite simple. It does not require having a database of documents to build a general probability of a word appearing in any document. We just need to calculate the probabilities in each input document, and finally we calculate the score of a sentence according to the frequencies of the word by using TF-IDF. SumBasic is good enough that is frequently used as a baseline in the literature.

In addition, we see TextRank algorithm. it take the inspiration from PageRank, for extracting phrases TextRank uses as a unit whole sentences, and as a similarity measure the number of words in common between them, the similarity is normalized based on the length of the phrases. To avoid the issue of having longer phrases having higher similarity than shorter ones.

The two algorithm TextRank and SumBasic we have seen so far have a weakness: they do not take into account semantics. This weakness is evident when we consider that there are words that have similar meanings (synonyms) and that most words can have different meaning depending on the context.

Latent Semantic Analysis attempt to overcome these issues. The expression Latent Semantic Analysis describes a technique more than a specific method. A technique that could be useful whenever we need to represent the meaning of words. The founding idea is quite simple: words with similar meaning will appear in similar parts of a text. Therefore, we start with a normal TF-IDF matrix. Such matrix contains nothing else than the frequencies of individual words, both inside a specific document and in all the documents evaluated. This technique will transform the original matrix from one linking each term with its frequency, into one with a (weighted) combination of terms linked to each document. So we could find the most relevant phrase and then find the phrases with are most close to it.

1.8 Conclusion

In the big data era, there has been an explosion in the amount of text data from a variety of sources. This volume of text is an inestimable source of information and knowledge which needs to be effectively summarized to be useful. Automatic text summarization come to solve this problem. It is one of the hot areas of research and attracts many attentions from different fields. We have seen that text summarization systems can be categorized in a various groups based on different approaches. We have seen different text summarization algorithms TextRank, LexRank, Latent Semantic Analysis (LSA), and SumBasic . In addition, we discuss after getting our programs results that Abstractive Summarization (LSA) is characterized by taking into account semantics; However Extractive Summarization (TextRank, LexRank and Sumbasic) is based on word weight calculation.