

MINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH
UNIVERSITY OF FERHAT ABBAS 1

FACULTY OF SCIENCES
COMPUTER SCIENCE DEPARTEMENT

Information Retrieval Practical sessions report

Report by :

BERRIMI Mohamed ,
Gueliani Sliman Nedjm Eddine
Master 2 IDTW , F3I
Module :RIW

Supervised by :

Dr. Fozi Harrag .

December , 2018



Université Ferhat Abbas Sétif 1

Contents

1	Introduction	2
2	Textual data manipulation	2
3	Code , Version 1	6
3.1	Vocabulary construction	6
3.1.1	Vocabulary of each file	6
3.1.2	All Vocabulary	8
3.2	Boolean Matrix	8
3.2.1	Queries on boolean matrix	10
3.3	Incidence Matrix	11
3.3.1	Vecorial Model	11
3.3.2	Queries in Vecorial modal	13
3.3.3	Cosine similarity	16
4	Code , Version 2	17
4.1	Read the files and store in one list	17
4.2	Preprocessing with nltk	17
4.3	Normalize by stemming	17
4.4	Turn text into vectors of term frequency	17
4.5	Print the TF matrix	18
4.6	Calculate idf and turn tf matrix to tf-idf matrix	19
4.7	Cosin similarity matrix	20
5	Code guide	21
6	conclusion	22

1 Introduction

Information retrieval is one of the most important applications in text mining field , during the course sessions of IRW we have introduced many techniques and concepts used in IR, and in the practical sessions we have developed an IR based system . in this report we are interested in detailing the approachs and the process that we have followed to impliment this system .

2 Textual data manipulation

In this section we are interested in getting familiar with textual data manipulation and processing using python .

To implement the functions in this practical work , we have used Python 3 as a programming language , it was challenging because we are not familiar with it . In the filed of text mining and Information retrieval , Python provides many pre-defined functions and packages that allows the user to deal with textual data, but most of the functions in this work are implimented from scratch, we will discuss the packages in the last section

Notes : We haven't followed the structure of the quesions in the TP sheet .

We have used a corpus that contain six files collected from a book [2] , we have claculated the number of words of each file .

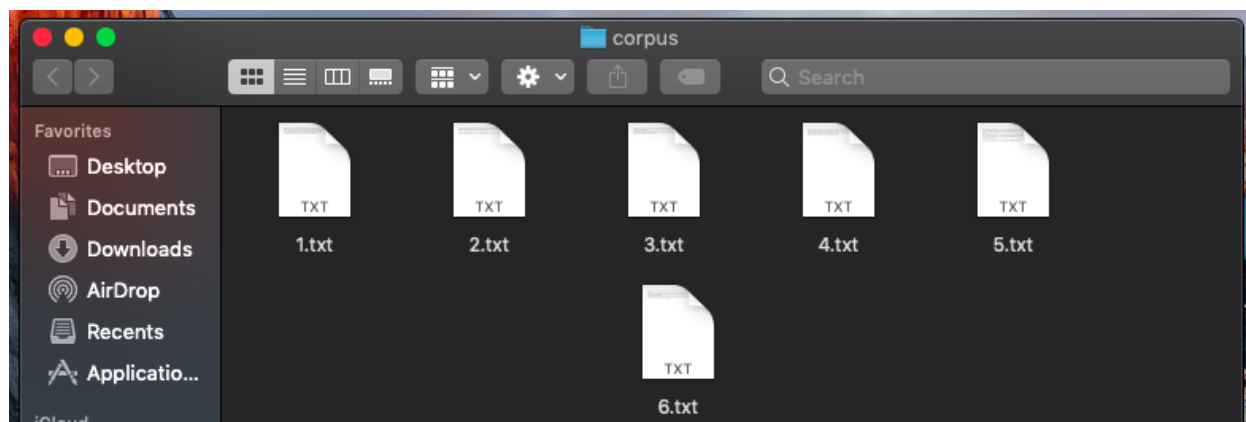


Figure 1: Text files in the corpus

```

1111         print ( "The total word count in file : ", name , " is : ", len(words))
The total word count in file : /Users/macbookair/Desktop/corpus/5.txt is : 211
The total word count in file : /Users/macbookair/Desktop/corpus/4.txt is : 124
The total word count in file : /Users/macbookair/Desktop/corpus/6.txt is : 105
The total word count in file : /Users/macbookair/Desktop/corpus/3.txt is : 104
The total word count in file : /Users/macbookair/Desktop/corpus/2.txt is : 84
The total word count in file : /Users/macbookair/Desktop/corpus/1.txt is : 135
1112
1113

```

Figure 2: Information about the files inside the corpus

After getting the size of each file (number of words) we applied a Stop-word elimination function on each file , we obtained these results .

```

1114         print ( "-----")
The total word count in file : /Users/macbookair/Desktop/corpus/5.txt is : 211

Number of words after stop words elimination =====> : 139
The total word count in file : /Users/macbookair/Desktop/corpus/4.txt is : 124

Number of words after stop words elimination =====> : 85
The total word count in file : /Users/macbookair/Desktop/corpus/6.txt is : 105

Number of words after stop words elimination =====> : 57
The total word count in file : /Users/macbookair/Desktop/corpus/3.txt is : 104

Number of words after stop words elimination =====> : 64
The total word count in file : /Users/macbookair/Desktop/corpus/2.txt is : 84

Number of words after stop words elimination =====> : 47
The total word count in file : /Users/macbookair/Desktop/corpus/1.txt is : 135

Number of words after stop words elimination =====> : 83

```

Figure 3: execution of the function "StopWordElem ".

It's also asked that we look for the existence of a specific word and the number of its occurrence in each line , file of the corpus

```

Please enter the word you are looking for : Father
You are looking in File : /Users/macbookair/Desktop/corpus/5.txt
----The word doesnt exist in this line-----
----found-----
----The word doesnt exist in this line-----
----The word doesnt exist in this line-----
----The word doesnt exist in this line-----
----The word doesnt exist in this line-----
The word Father exist in this file : 1 times
Number of lines of this file ***** 6
You are looking in File : /Users/macbookair/Desktop/corpus/4.txt
----The word doesnt exist in this line-----
----found-----
----The word doesnt exist in this line-----
The word Father exist in this file : 1 times
Number of lines of this file ***** 3
You are looking in File : /Users/macbookair/Desktop/corpus/6.txt
----The word doesnt exist in this line-----
----The word doesnt exist in this line-----
----The word doesnt exist in this line-----
----The word doesnt exist in this line-----
The word Father exist in this file : 0 times
Number of lines of this file ***** 4
You are looking in File : /Users/macbookair/Desktop/corpus/3.txt
----The word doesnt exist in this line-----
----The word doesnt exist in this line-----
----The word doesnt exist in this line-----
The word Father exist in this file : 0 times
Number of lines of this file ***** 3
You are looking in File : /Users/macbookair/Desktop/corpus/2.txt
----The word doesnt exist in this line-----
----The word doesnt exist in this line-----
----found-----
The word Father exist in this file : 1 times
Number of lines of this file ***** 3
You are looking in File : /Users/macbookair/Desktop/corpus/1.txt
----found-----
----The word doesnt exist in this line-----
----found-----
The word Father exist in this file : 2 times
Number of lines of this file ***** 3

```

Information about files in the corpus after and befor stop word elimination

To obatin these two results we have used this function :

```

def Word_Search(path):

    path = '/Users/macbookair/Desktop/corpus/*.txt'

    corpus = glob.glob(path)
    string1 = input("Please enter the word you are looking for : ")
    for name in corpus:
        try:
            with open(name) as f:
                print('You are looking in File :', name)
                count = 0
                nblines=0
                for line in f:
                    nblines +=1

                    if string1 in line:
                        count+=1
                        print('----found-----')

                    else:
                        print('-----The word doesnt exist in this line----- ')
                print("The word ",string1,"exist in this file :",count ,"times")
                print("Number of lines of this file *****",nblines)

        except IOError as exc:
            if exc.errno != errno.EISDIR:
                print('Files not found ')

```

This function takes the path of corpus as a parameters , and ask for the word that user wants to look for and save the input in string1 variable . it calculate the number of lines and words and check if the word exists in the corpus lines , if it's the case , how many times .

The same thing we can apply when we look for a sentence in a text , in python , we simply type this commend :

```

Sentence = 'he felt ashamed of havin lost his temper'

```

```

if Sentence in data:
    print('Sentence Exists in this Text !')
else:
    print('Sentence doesn/t exist')

```

Indeed ! this sentence exist in the document .

3 Code , Version 1

In this section we are going to programme all the functions from scratch , and construct the matrices from scratch as well.

3.1 Vocabulary construction

This section is very important , because we are going to extract the vocabulary -terms- from our corpus .

To construct the vocabulary we will follow this approche :

3.1.1 Vocabulary of each file

construct a vocabular of each textual data in the corpus and store it in a vectore . To construct the vocabular of the data we need to :

- Remove Digits and punctuations from the text using Regular expressions :

```
new_data=re.sub(r'\w*\d\w*', '', file)
#remove digits
tokenizer = RegexpTokenizer(r'\w+')

```

- Word tokenization : split the textual data into words in order to faciliate the processing .

```
tokenizer = RegexpTokenizer(r'\w+')
file=tokenizer.tokenize(new_data)
```

- Apply stop-word elimination :

There are some stop-words that hasn't removed by the python function (from nltk.corpus import stopwords) , we have added an extra stop-words list to remove them from our data .

```

stopWords = set(stopwords.words('english'))
other = ['And', 'I', 'of', 'In', 'in', 'a', 'was',
        'two', 'and', 'the', 'your', 'her', 'his', 'has',
        'to', 'he ', 'that',
        'in', 'The', 'Not', 'He',
        'We', 'But', 'one', 'tooo']

for word in list(file):
    if word in other:
        file.remove(word)

for word in list(file):
    if word in stopWords:
        file.remove(word)

```

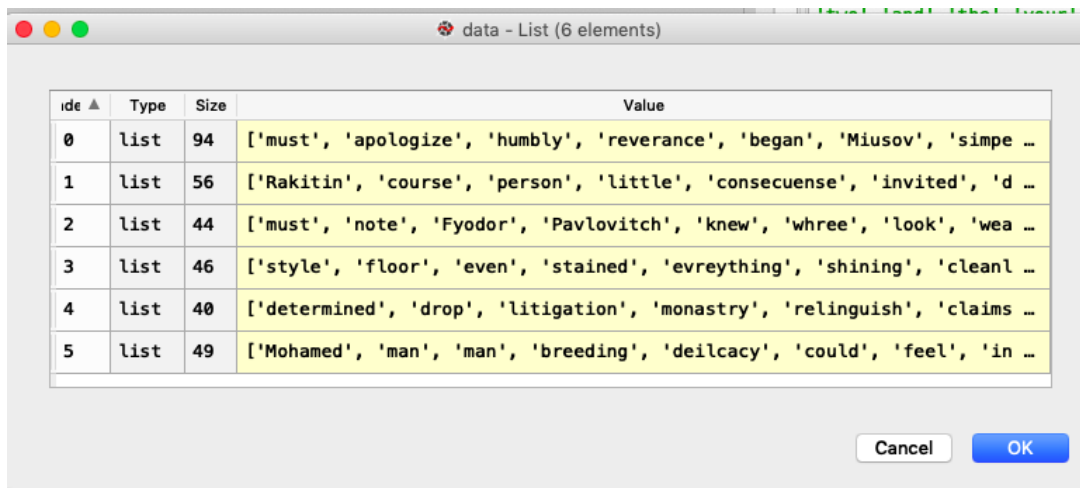
- Lemmatization : Lemmatization usually refers to doing things properly with the use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the lemma . [1]

```

from nltk.stem.wordnet import WordNetLemmatizer
lmtzr = WordNetLemmatizer()

for name in data:
    data[y]=[lmtzr.lemmatize(word) for word in name]
    y=y+1

```



ide ▲	Type	Size	Value
0	list	94	['must', 'apologize', 'humbly', 'reverance', 'began', 'Miusov', 'simpe ...
1	list	56	['Rakitin', 'course', 'person', 'little', 'consecuense', 'invited', 'd ...
2	list	44	['must', 'note', 'Fyodor', 'Pavlovitch', 'knew', 'whree', 'look', 'wea ...
3	list	46	['style', 'floor', 'even', 'stained', 'evreything', 'shining', 'cleanl ...
4	list	40	['determined', 'drop', 'litigation', 'monastery', 'relinguish', 'claims ...
5	list	49	['Mohamed', 'man', 'man', 'breeding', 'deilcacy', 'could', 'feel', 'in ...

The complete version of the code is available in the function `vocFile`.

3.1.2 All Vocabulary

In this section we are going to construct the vocabulary from all the texts available in the corpus . we made two different versions of this function :

- **Version 1 :** Construct the vocabulary from all the corpus, like the previous section , but this time the vocabulary is stored in one hashmap (array list) . ref.code (Fuction : def AllVoab)
- **Version 2:** The vocabulary of the corpus is the sum of the vocabularies of all the text files in the corpus , we just need to remove the repeated words .

```
Complete = []
for name in data:
    for word in name:
        Complete.append(word)
len(Complete)
from collections import OrderedDict

All=[]
All = list(OrderedDict.fromkeys(Complete))
len(All)
```

len(Complete) gives : 329

len(All) gives : 263

3.2 Boolean Matrix

In this section we are going to construct the boolean matrix . Python provides a hashmap structure to store the elements properly , we initialized a the hashmap with the vocabulary terms and with 0 count as values at first ,the we browsed the text files and see if the word from the vocabulary exists in the files or not , if it exists at least once , the value of the corresponding word in the hashmap is equal to 1 , else 0 .

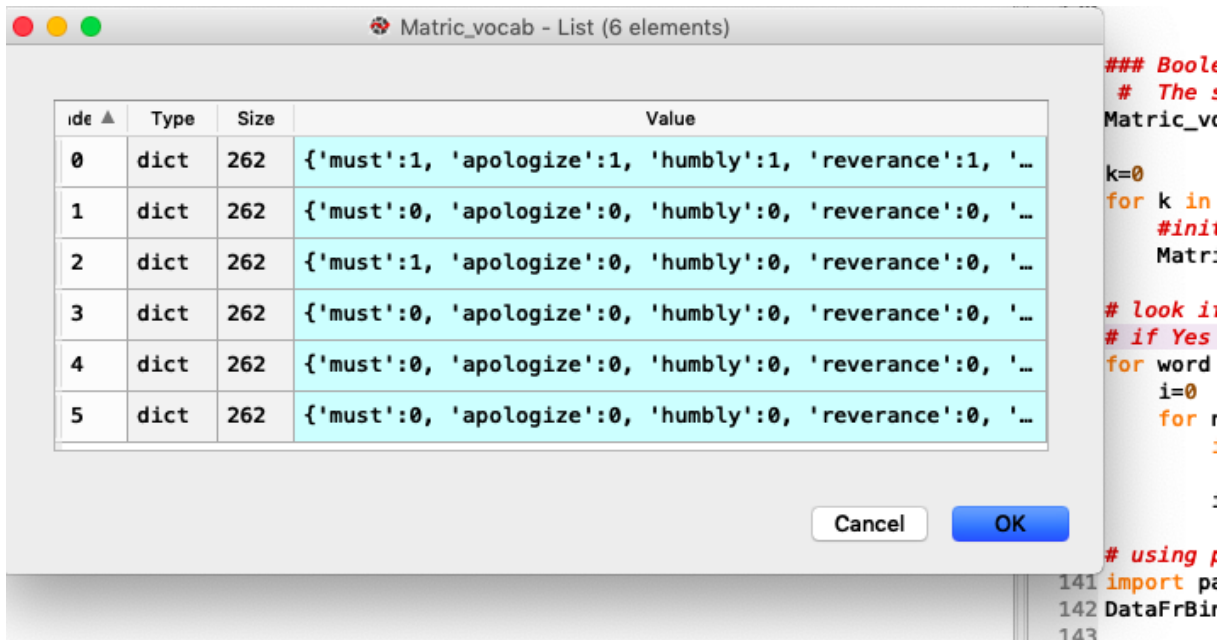
```
### Boolean Matrix
# The size of matrix == number of files in the corpus
Matric_vocab=[[]]*len(data)

k=0
for k in range(len(data)):
    #initialize the matrix of the file X with a AllVocStemed as a header and values with 0
    Matric_vocab[k] = dict.fromkeys(AllVocStemed,0)

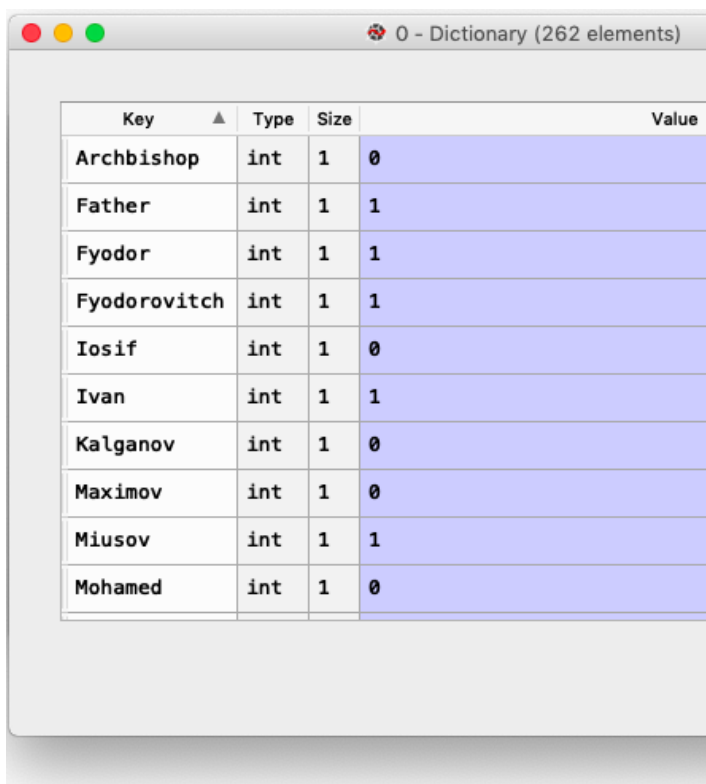
# look if the word in Vocabulary exists in file
# if Yes , put the value of that word =1 in the matrix
for word in AllVocStemed:
    i=0
    for name in data:
        if word in name:
            Matric_vocab[i][word]=1
        i=i+1

# using pandas to create the dataframe
import pandas as pd
DataFrBinaire =pd.DataFrame(Matric_vocab, index=['file1','file2','file3','file4','file5','file6'])
```

The Matric-Vocab looks like this in the result :



Let's take a closer look at one row of the matric-Vocab :



In the end , the boolean matrix data frame is created using Pandas python library .

DataFrBinaire - DataFrame												
Index	Archbishop	Father	Fyodor	Fyodorovitch	Iosif	Ivan	Kalaganov	Maximov	Miusov	Mohamed	Paissy	Pardon
file1	0	1	1	1	0	1	0	0	1	0	0	1
file2	0	1	0	0	1	1	1	1	1	0	1	0
file3	1	0	1	0	0	0	0	0	0	0	0	0
file4	0	0	0	0	0	0	0	0	0	0	0	0
file5	0	1	0	0	0	0	0	0	0	0	0	0
file6	0	1	1	0	0	1	0	0	0	1	0	0

3.2.1 Queries on boolean matrix

we wil see how to receive the query from the user in the next section .

Unfortunately we weren't able to accomplish the functions **AND** , **OR** , **NOT** of this section .

3.3 Incidence Matrix

As we have seen earlier in previous sections when we constructed the boolean matrix , this time we are going to create the incidence matrix , such that , the word box in the matrix will contain the number of the occurrences of this in the corresponding file .

```
### matrice Multiple

Matric_vocabMultp=[[]]*len(data)
k=0
for k in range(len(data)):
    # initilize an empty matrix with AllVocStemmed vocabulary as its header
    Matric_vocabMultp[k] = dict.fromkeys(AllVocStemed,0)

for word in AllVocStemed:
    z=0
    for name in data: # for each file in corpus (vocabulary of file )
        for w in name: # for each word in vocabulary file
            if w==word :
                # if the word exsits , increment its count by 1 .
                Matric_vocabMultp[z][word]+=1
            z=z+1

import pandas as pd
# use pandas to create the data frame | 
DataFrameMult =pd.DataFrame(Matric_vocabMultp)

# Store the dataframe into a csv file .
DataFrameMult.to_csv('Matrix.csv',index=False , encoding='utf-8')
```

3.3.1 Vecorial Model

Here we are going to use the previous matrix and calculate the TF , IDF , TF-IDF .
Lets now code TF-IDF in Python from scratch.

The function computeTF computes the TF score for each word in the corpus, by document.

```
def computeTF (wordDict , file):

    tfDict ={}
    fileCount = len(file )
    for word , count in wordDict.items():
        tfDict[word] = count/float(fileCount)
    return tfDict
```

Let's calculate the TF of the first row in our matrix :

```
In [366]: tf1 = computeTF(Matric_vocabMultp[0],data[0])
```

```
In [367]: tf1
```

```
Out[367]:
```

```
{'Archbishop': 0.0,  
'As': 0.010638297872340425,  
'Father': 0.010638297872340425,  
'Fyodor': 0.010638297872340425,  
'Fyodorovitch': 0.010638297872340425,  
'Iosif': 0.0,  
'Ivan': 0.010638297872340425,  
'Kalganov': 0.0,  
'Maximov': 0.0,  
'Miusov': 0.02127659574468085,
```

The function computeIDF computes the IDF score of every word in the corpus.

```
def computeIDF(Matric_vocaQuery):  
    import math  
    idfDict={}  
    N=len(AllVocStemed)  
  
    idfDict = dict.fromkeys(Matric_vocaQuery[0],0)  
    for doc in Matric_vocaQuery:  
        for word, val in doc.items():  
            if val > 0 :  
                idfDict[word] +=1  
    for word , val in idfDict.items():  
        if val > 0:  
            idfDict[word]=math.log10(N/float(val))  
  
    return idfDict
```

The function computeTFIDF below computes the TF-IDF score for each word, by multiplying the TF and IDF scores.

```
def computeTFIDF (TF , idfs ):  
    tfidfs={}  
    for word , val in TF.items():  
        tfidfs[word]= val * idfs[word]  
    return tfidfs
```

The output produced by the above code for the set of documents :

Index	Archbishop	Father	Fyodor	Fyodorovitch	Iosif	Ivan	Kalganov	Maximov	Miusov	Mohamed
5	0	0.111198	0.0396159	0	0	0.0396159	0	0	0	0.0493...
0	0	0.0193217	0.0206509	0.0257266	0	0.0206509	0	0	0.0450483	0
1	0	0.0972986	0	0	0.043184	0.0346639	0.043184	0.043184	0.0378084	0
2	0.0549614	0	0.0882355	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0
4	0	0.0908121	0	0	0	0	0	0	0	0

3.3.2 Queries in Vecorial modal

In this section we are going to take a user query and calculate its score in each file of the corpus . The query need to be tokenized , lemmatized and without stop words .

The code bellow explain the process from receiving the query from the user till the query is normalized .

```
Clean_query =[]
# Receive the user input .
query = input("Please enter your Query : ")
# Tokenize the query
tokenizer = RegexpTokenizer(r'\w+')
query=tokenizer.tokenize(query)
# Stop words elimination .
stopWords = set(stopwords.words('english'))

for word in query:
    if word in stopWords:
        query.remove(word)
# Lemmatization phasse
for word in query :
    q = lmtzr.lemmatize(word)
    Clean_query.append(q)
```

Now let's move the important phase , where we are going to create a vector initilized by zeros , and browse the query , if the word from the vocabulary exists in the query increment its value in the vector by 1 .

```
# Initilize a vectore with Vocabulary as its header
Quer= dict.fromkeys(AllVocStemed,0)

# Check if the words of the vocabulary exists in the query .

for word in AllVocStemed:
    for w in Clean_query:
        # If The word exists , increment the occurence in the vector by 1 .
        if word == w:
            Quer[w]+=1
```

Now we take the incidence matrix that we have created in the previous section and concatenate the query vector with it .

```
# Now we are going to concatenate the query(vector ) with the Indicence Matrix

Matric_vocaQuery=[[]]*len(data)

k=0
for k in range(len(data)):
    Matric_vocaQuery[k] = dict.fromkeys(AllVocStemed,0)

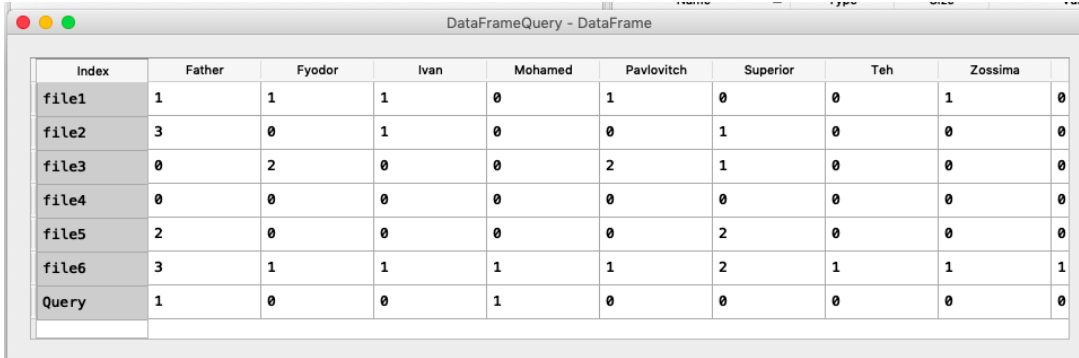
for word in AllVocStemed:
    z=0
    for name in data:
        for w in name:
            if w==word :
                # if the word exsits , increment its count by 1 .
                Matric_vocaQuery[z][word]+=1
            z=z+1

Matric_vocaQuery.append(Quer)

import pandas as pd

DataFrameQuery =pd.DataFrame(Matric_vocaQuery)
```

The resumed dataframe will contain the incidence matrix with the query as its last row .

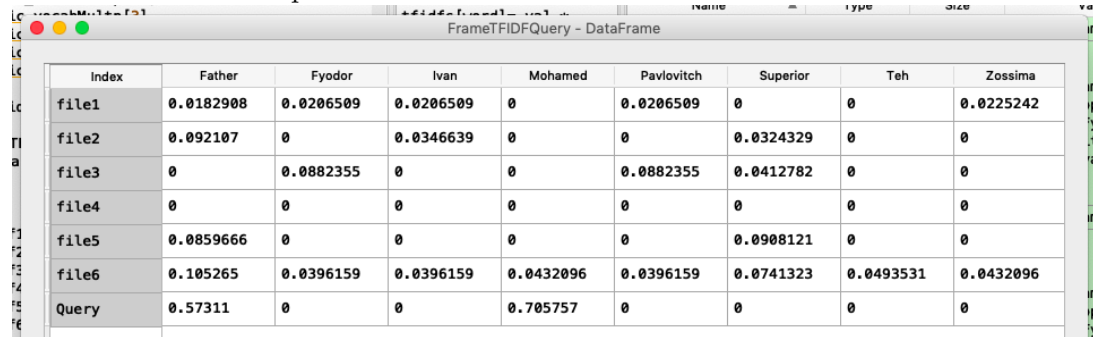


Index	Father	Fyodor	Ivan	Mohamed	Pavlovitch	Superior	Teh	Zossima	
file1	1	1	1	0	1	0	0	1	0
file2	3	0	1	0	0	1	0	0	0
file3	0	2	0	0	2	1	0	0	0
file4	0	0	0	0	0	0	0	0	0
file5	2	0	0	0	0	2	0	0	0
file6	3	1	1	1	1	2	1	1	1
Query	1	0	0	1	0	0	0	0	0

Calculate the TF-IDF of the query We have already developed the functions in the previous section ,now we are just applying them in the new matrix .

Query = " Father and Mohamed ".

The TF-IDF coresponded to the new matrix is :



Index	Father	Fyodor	Ivan	Mohamed	Pavlovitch	Superior	Teh	Zossima	
file1	0.0182908	0.0206509	0.0206509	0	0.0206509	0	0	0.0225242	
file2	0.092107	0	0.0346639	0	0	0.0324329	0	0	
file3	0	0.0882355	0	0	0.0882355	0.0412782	0	0	
file4	0	0	0	0	0	0	0	0	
file5	0.0859666	0	0	0	0	0.0908121	0	0	
file6	0.105265	0.0396159	0.0396159	0.0432096	0.0396159	0.0741323	0.0493531	0.0432096	
Query	0.57311	0	0	0.705757	0	0	0	0	

3.3.3 Cosine similarity

gmnrpigggeththththteh

```
vec = TfidfVectorizer()
X = vec.fit_transform(dataAll) # 'X' will now be a TF-IDF representation of the data,
                               # the first row of 'X' corresponds to the first sentence in 'dataAll'

# Calculate the pairwise cosine similarities
#(depending on the amount of data that you are going to have this could take a while)

CS = cosine_similarity(X)

dataAll = dataAll.append(Query)
X = vec.fit_transform(dataAll)
CSQuery = cosine_similarity(X)
```

The resulted Cosine similarity matrix is :

```
'File 1 ' [[1. , 0.29877665, 0.23224166, 0.21611863, 0.35185987, 0.39261077, 0.14478037],
'File 1 ' [0.29877665, 1. , 0.28473521, 0.30038877, 0.33269883, 0.30107891, 0.20481128],
'File 1 ' [0.23224166, 0.28473521, 1. , 0.34646199, 0.27787227, 0.25476568, 0.27541781],
'File 1 ' [0.21611863, 0.30038877, 0.34646199, 1. , 0.28191748, 0.16762841, 0.27995128],
'File 1 ' [0.35185987, 0.33269883, 0.27787227, 0.28191748, 1. , 0.30639993, 0.1979079 ],
'File 1 ' [0.39261077, 0.30107891, 0.25476568, 0.16762841, 0.30639993, 1. , 0.11314225],
'Query' [0.14478037, 0.20481128, 0.27541781, 0.27995128, 0.1979079 , 0.11314225, 1. ]]
```

If we sort the values of the last line in the matrix (the query) we notice that 0.279 that corresponds to file4.txt is the highest value , wich mean that this file is more likely to have this query .

4 Code , Version 2

in this section we are going to use the predefined python libraries and packages in Text manipulations ;

4.1 Read the files and store in one list

```
path = '/Users/macbookair/Desktop/corpus/*.txt'

corpus = glob.glob(path)

dataAll=[]
for name in corpus:

    dataAll.append(open(name).read().replace('\n', ''))
```

4.2 Preprocessing with nltk

The default functions of CountVectorizer and TfidfVectorizer in scikit-learn detect word boundary and remove punctuations automatically. However, if we want to do stemming or lemmatization, we need to customize certain parameters in CountVectorizer and TfidfVectorizer. Doing this overrides the default tokenization setting, which means that we have to customize tokenization, punctuation removal, and turning terms to lower case altogether.

4.3 Normalize by stemming

```
import nltk, string, numpy
nltk.download('punkt') # first-time use only
stemmer = nltk.stem.porter.PorterStemmer()
def StemTokens(tokens):
    return [stemmer.stem(token) for token in tokens]
remove_punct_dict = dict((ord(punct), None) for punct in string.punctuation)
def StemNormalize(text):
    return StemTokens(nltk.word_tokenize(text.lower().translate(remove_punct_dict)))
```

4.4 Turn text into vectors of term frequency

```
from sklearn.feature_extraction.text import CountVectorizer
LemVectorizer = CountVectorizer(tokenizer=LemNormalize, stop_words='english')
LemVectorizer.fit_transform(dataAll)
```

Print the vocabulary lemmatized :

```
In [30]: print( LemVectorizer.vocabulary_)
{'apologize': 7, 'humbly': 105, 'reverance': 175, 'began': 22, 'miusov': 138, 'simpering': 189, 'affably': 3,
'speakin': 192, 'dignified': 59, 'respectful': 173, 'tone': 214, 'pardonus': 153, 'having': 100, 'come': 43,
'gentleman': 91, 'invited': 111, 'fyodor': 89, 'pavlovitch': 154, 'felt': 80, 'obliged': 147, 'decline': 51,
'honor': 101, 'hospitalty': 103, 'wihtout': 232, 'reason': 164, 'reverand': 176, 'father': 78, 'zossimas': 241,
'cell': 36, 'wa': 226, 'carried': 33, 'away': 18, 'unhappy': 218, 'dissention': 65, 'son': 191, 'let': 125,
'fall': 76, 'word': 239, 'quite': 160, 'keeping': 118, 'fact': 75, 'unseamly': 219, 'glanced': 92, 'monk': 143,
'doubt': 66, 'aware': 17, 'recognising': 166, 'blame': 26, 'regret': 170, 'shame': 185, 'begged': 23, 'ivan':
115, 'fyodorovitch': 90, 'convey': 47, 'apologees': 6, 'brief': 32, 'hope': 102, 'desire': 54, 'make': 132,
'amends': 5, 'later': 123, 'asks': 15, 'blessinq': 27, 'begs': 24, 'forget': 85, 'ha': 97, 'takn': 206,
'placeas': 157, 'utterred': 221, 'terade': 210, 'completely': 44, 'recovered': 167, 'selfcomplecency': 183,
'trace': 216, 'irritation': 114, 'disappaered': 63, 'fuly': 88, 'sincerelly': 190, 'loved': 131, 'humanity': 104,
'rakitin': 161, 'course': 48, 'person': 156, 'tooo': 215, 'little': 127, 'consecuense': 46, 'dinner': 62,
'iosif': 113, 'paissy': 152, 'othr': 149, 'inmate': 108, 'monastery': 141, 'alraedy': 4, 'waiting': 227,
'kalganov': 117, 'arrived': 11, 'guest': 96, 'maximov': 135, 'stood': 198, 'aside': 14, 'superior': 204,
'stepped': 197, 'middle': 137, 'room': 180, 'receive': 165, 'tall': 207, 'vigorous': 224, 'old': 148, 'man':
134, 'black': 25, 'hair': 98, 'streakd': 199, 'grey': 95, 'long': 128, 'grave': 94, 'ascetic': 12, 'face': 74,
'bowed': 29, 'silence': 188, 'time': 213, 'approaced': 8, 'note': 146, 'knew': 120, 'whree': 231, 'look': 129,
'weak': 228, 'spot': 194, 'malicius': 133, 'rumor': 181, 'reached': 162, 'archbishop': 9, 'regarding': 169,
'institution': 109, 'elder': 68, 'existed': 73, 'respect': 174, 'paid': 151, 'detrement': 57, 'aughtority': 16,
'abused': 2, 'sacrament': 182, 'confession': 45, 'absurd': 1, 'charge': 37, 'died': 58, 'spirit': 193, 'folly':
84, 'caught': 35, 'style': 202, '1820': 0, 'floor': 82, 'stained': 195, 'evreything': 70, 'shining': 186,
'cleanlyness': 41, 'chioce': 38, 'flower': 83, 'window': 233, 'sumptuous': 203, 'thing': 212, 'moment': 140,
'beatifully': 19, 'decorated': 52, 'table': 205, 'cloth': 42, 'clean': 40, 'service': 184, 'shone': 187, 'kind':
119, 'wellbaked': 229, 'bread': 30, 'bottle': 28, 'wine': 234, 'excellent': 72, 'mead': 136, 'large': 122,
'glass': 93, 'jug': 116, 'kvas': 121, 'famous': 77, 'neighborhood': 144, 'vodka': 225, 'related': 171,
'determined': 56, 'drop': 67, 'litigation': 126, 'monastery': 142, 'relinquish': 172, 'claim': 39, 'woodcuting':
238, 'fishery': 81, 'rihghts': 178, 'ready': 163, 'becuase': 21, 'right': 177, 'becom': 20, 'le': 124,
'valuable': 223, 'vaguest': 222, 'idea': 106, 'wood': 237, 'river': 179, 'quedtion': 159, 'werethese': 230,
'excellent': 71, 'intention': 110, 'strengthened': 200, 'entered': 69, 'diniingroom': 60, 'stricttly': 201,
'diningroom': 61, 'mohamed': 139, 'breeding': 31, 'deilcacy': 53, 'feel': 79, 'inwrd': 112, 'qualm': 158,
'ashamed': 13, 'havin': 99, 'lost': 130, 'temper': 209, 'ought': 150, 'disdained': 64, 'despicable': 55,
'wretch': 240, 'upset': 220, 'forgotten': 86, 'teh': 208, 'case': 34, 'reflceted': 168, 'step': 196, 'theyre':
211, 'decent': 50, 'people': 155, 'understand': 217, 'nobleman': 145, 'friendly': 87, 'courteous': 49,
'withthem': 235, 'wont': 236, 'argue': 10, 'ill': 107}
```

4.5 Print the TF matrix

tf_matrix - NumPy array

	0	1	2	3	4	5	6	7	
0	0	0	0	1	0	1	1	1	
1	0	0	0	0	1	0	0	0	
2	0	1	1	0	0	0	0	0	
3	1	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	
5	0	0	0	0	0	0	0	0	

Format Resize ☐ Background color

```
from sklearn.feature_extraction.text import TfidfTransformer
tfidfTran = TfidfTransformer(norm="l2")
tfidfTran.fit(tf_matrix)
print (tfidfTran.idf_)
```

[illegible]

```
tfidf_matrix = tfidfTran.transform(tf_matrix)
print (tfidf_matrix.toarray())
```

```
In [37]: print (tfidf_matrix.toarray())
```

[0.	0.	0.	...	0.20781762	0.	0.08520671]
[0.	0.	0.	...	0.	0.]
[0.	0.1562779	0.1562779	...	0.	0.]
[0.1506555	0.	0.	...	0.	0.]
[0.	0.	0.	...	0.	0.]
[0.	0.	0.	...	0.	0.14706817	0.120598]]

4.7 Cosin similarity matrix

```
Out[50]:  
array([[1.          , 0.06972252, 0.04631406, 0.02054477, 0.0331402 ,  
        0.12763934],  
       [0.06972252, 1.          , 0.04068121, 0.0990062 , 0.08320689,  
        0.11653208],  
       [0.04631406, 0.04068121, 1.          , 0.04218377, 0.03221493,  
        0.07569669],  
       [0.02054477, 0.0990062 , 0.04218377, 1.          , 0.06633345,  
        0.          ],  
       [0.0331402 , 0.08320689, 0.03221493, 0.06633345, 1.          ,  
        0.08682827],  
       [0.12763934, 0.11653208, 0.07569669, 0.          , 0.08682827,  
        1.          ]])
```

Using Python pre-implemented functions and packages helps a lot in speeding the workflow and facilitate the processing phase .

We have implimented the functions from scratch in the first version because we wanted to challenge our selves and get to know the text processing tools better.

5 Code guide

The code provided is written in python , to test it , is it recommended to use Spyder 3 as running environment .

The full version of the code is provided in the file "codes.py" , The first lines contains the used packages and libraries :

From sklearn import * Sickit learn is a free software machine learning library for the Python programming language. it includes all the machine learning algorithms implimented inside its corp .

From nltk import * nltk refers to : Natural Language Tool kit . The Natural Language Toolkit (NLTK) is a platform used for building Python programs that work with human language data for applying in statistical natural language processing (NLP). It contains text processing libraries for tokenization, parsing, classification, stemming, tagging and semantic reasoning. [3]

from nltk.stem.porter import PorterStemmer import Porter stemming predefined package to stem the textual data .

from nltk.tokenize import RegexpTokenizerer This library helps in removing digits , punctuations and in word-tokenzing the textual data .

from nltk.stem.wordnet import WordNetLemmatizer Predifined package that helps in retrieve the lemm from the text .

import pandas as pd library written for the Python programming language for data manipulation and analysis. we used it here to create the dataFrames .

There are other implimented functions that we didn't mention here , they are all availble and commented in the code .

6 conclusion

In this report we have detailed the process that we have followed to develop an IR system , during the practical sessions in IRW module with Dr. Harrag Fozi .

We have faced many challenges while working on it , especially when we started working with python because we are not familiar with it , but in the end of this practical work , we have learned a lot about it .

The most exciting part is that we took a textual data as an input and we had an incidence matrix as an output , which we can use to apply some machine learning algorithms on it .

References

.

- [1] Stemming and lemmatization - Stanford NLP , Retrieved 15 dec , 2018 from <https://nlp.stanford.edu/IR-book/html/.../stemming-and-lemmatization-1.html>.
- [2] Fyodor Dostoyevsk , 2011 , The Works of Fyodor Dostoyevsky , Galgotha press .
- [3] NLTK 3.4 documentation , Retrieved 15 december , 2018 from <https://www.nltk.org/>