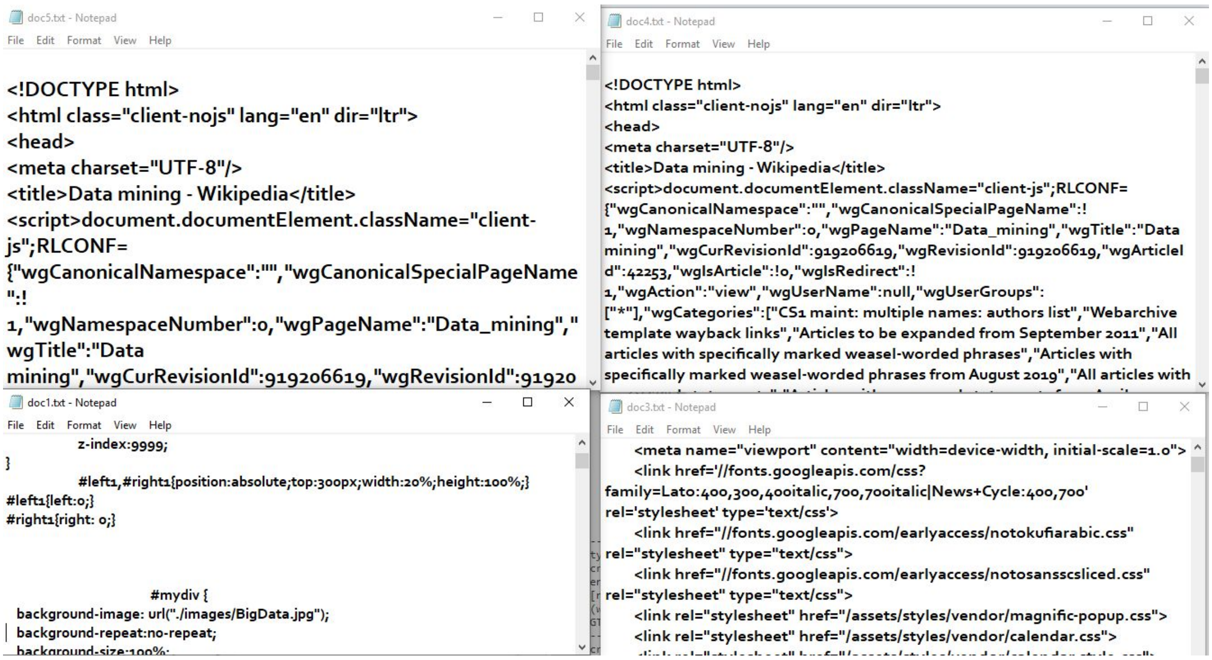
Part 1: Preprocessing to Build Document Vectors for Web Page Content Analysis

Step 1: Analysing Dataset

The first step in any of the Machine Learning tasks is to analyse the data. So if we look at the dataset, the data is of html source code.

Now one of the important tasks is to identify the body(important text), if we analyse the documents, there are different patterns of alignment. Now we need to figure out a way to extract the Content of the page.



doc5.txt - Notepad

```
<!DOCTYPE html>
<html class="client-nojs" lang="en" dir="ltr">
<head>
<meta charset="UTF-8"/>
<title>Data mining - Wikipedia</title>
<script>document.documentElement.className="client-
js";RLCONF=
{"wgCanonicalNamespace":"","wgCanonicalSpecialPageName
":!
1,"wgNamespaceNumber":0,"wgPageName":"Data_mining","
wgTitle":"Data
mining","wgCurRevisionId":919206619,"wgRevisionId":91920
```

doc4.txt - Notepad

```
<!DOCTYPE html>
<html class="client-nojs" lang="en" dir="ltr">
<head>
<meta charset="UTF-8"/>
<title>Data mining - Wikipedia</title>
<script>document.documentElement.className="client-js";RLCONF=
{"wgCanonicalNamespace":"","wgCanonicalSpecialPageName":!
1,"wgNamespaceNumber":0,"wgPageName":"Data_mining","wgTitle":"Data
mining","wgCurRevisionId":919206619,"wgRevisionId":919206619,"wgArticleI
d":42253,"wgIsArticle":!0,"wgIsRedirect":!
1,"wgAction":"view","wgUserName":null,"wgUserGroups":
["*"],"wgCategories":["CS1 maint: multiple names: authors list","Webarchive
template wayback links","Articles to be expanded from September 2011","All
articles with specifically marked weasel-worded phrases","Articles with
specifically marked weasel-worded phrases from August 2019","All articles with
```

doc1.txt - Notepad

```
            z-index:9999;
}
            #left1,#right1{position:absolute;top:300px;width:20%;height:100%;}
#left1{left:0;}
#right1{right: 0;}


                        #mydiv {
  background-image: url("./images/BigData.jpg");
  background-repeat:no-repeat;
```

doc3.txt - Notepad

```
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link href='//fonts.googleapis.com/css?
family=Lato:400,300,400italic,700,700italic|News+Cycle:400,700'
rel='stylesheet' type='text/css'>
    <link href="//fonts.googleapis.com/earlyaccess/notokufiarabic.css"
rel="stylesheet" type="text/css">
    <link href="//fonts.googleapis.com/earlyaccess/notosansscsliced.css"
rel="stylesheet" type="text/css">
    <link rel="stylesheet" href="/assets/styles/vendor/magnific-popup.css">
    <link rel="stylesheet" href="/assets/styles/vendor/calendar.css">
```

Step 2: Preprocessing

Preprocessing is one of the major steps when we are dealing with any kind of text models. During this stage we have to look at the distribution of our data, what techniques are needed and how deep we should clean.

This step totally depends on the problem statement. Few mandatory preprocessing are converting to lowercase, removing punctuation, removing stop words and lemmatization/stemming. In this problem removing <...> tags is also our main preprocessing step.

Tag Removal

Removing all the tags from the text file which are between < >, and storing the content back to textfiles.

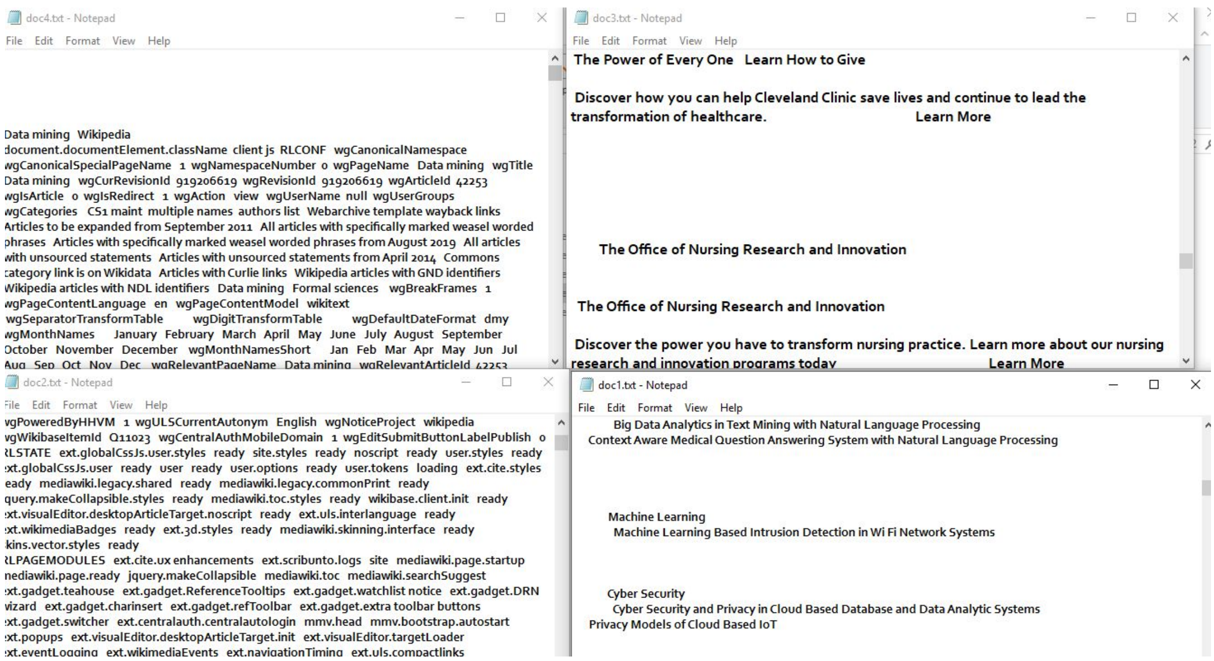
```
tag_removal = re.compile(r'<[^>]+>')

def remove_tags(text):
```

```
        return tag_removal.sub('',text)
'''The process of removing html tags from the text by replacing with null'''
for i in range(1,6):
        string = open('doc'+str(i)+'.txt').read()
        open('doc'+str(i)+'.txt','w').write(remove_tags(string))
```

Special Symbols

```
''' #Removing all the symbols like punctuation marks ,question marks etc...
        and replacing by '  '(space)'''
for i in range(1,6):
  string1 = open('doc'+str(i)+'.txt').read()
  new_str = re.sub('[^a-zA-Z0-9\n\.]',' ',string1)
  string1 = re.sub("\S*\d\S*", "", new_str).strip()
  open('doc'+str(i)+'.txt','w').write(string1)
```

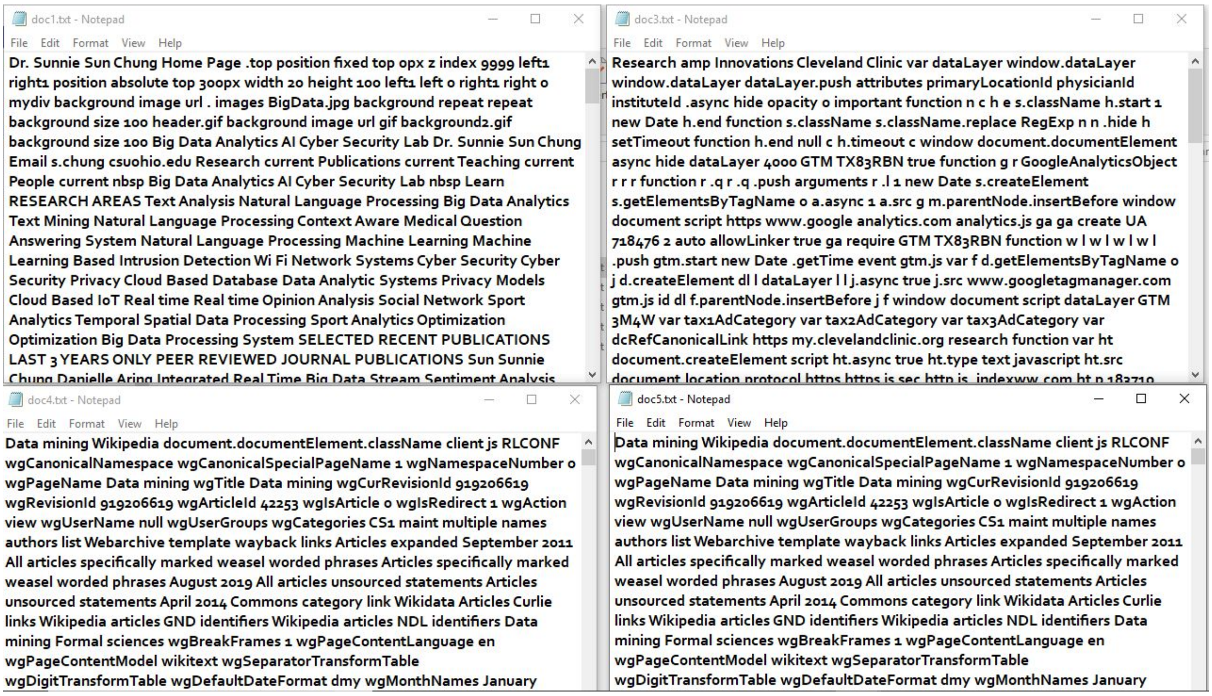
After removing all the tags and special symbols from the text files.



Stop Words

Stop words are the most commonly occurring words which don’t give any additional value to the document vector. in-fact removing these will increase computation and space efficiency. nltk library has a method to download the stopwords, so instead of explicitly mentioning all the stopwords ourselves we can just use the nltk library and iterate over all the words and remove the stop words.

```
'''Removing stop words using tokenize and stop_words in python '''
stop_words = stopwords.words("english")
for i in range(1,6):
        string2= open('doc'+str(i)+'.txt').read()
        open('doc'+str(i)+'.txt','w').write(' '.join([word for word in string2.split() if word not in stop_words]).lower()+' ')
```



doc1.txt - Notepad

File Edit Format View Help

Dr. Sunnie Sun Chung Home Page .top position fixed top opx z index 9999 left1 right1 position absolute top 300px width 20 height 100 left1 left o right1 right o mydiv background image url . images BigData.jpg background repeat repeat background size 100 header.gif background image url gif background2.gif background size 100 Big Data Analytics AI Cyber Security Lab Dr. Sunnie Sun Chung Email s.chung csuohio.edu Research current Publications current Teaching current People current nbsp Big Data Analytics AI Cyber Security Lab nbsp Learn RESEARCH AREAS Text Analysis Natural Language Processing Big Data Analytics Text Mining Natural Language Processing Context Aware Medical Question Answering System Natural Language Processing Machine Learning Machine Learning Based Intrusion Detection Wi Fi Network Systems Cyber Security Cyber Security Privacy Cloud Based Database Data Analytic Systems Privacy Models Cloud Based IoT Real time Real time Opinion Analysis Social Network Sport Analytics Temporal Spatial Data Processing Sport Analytics Optimization Optimization Big Data Processing System SELECTED RECENT PUBLICATIONS LAST 3 YEARS ONLY PEER REVIEWED JOURNAL PUBLICATIONS Sun Sunnie Chung Danielle Aring Integrated Real Time Big Data Stream Sentiment Analysis

doc3.txt - Notepad

File Edit Format View Help

Research amp Innovations Cleveland Clinic var dataLayer window.dataLayer window.dataLayer dataLayer.push attributes primaryLocationId physicianId instituteId .async hide opacity o important function n c h e s.className h.start 1 new Date h.end function s.className s.className.replace RegExp n n .hide h setTimeout function h.end null c h.timeout c window document.documentElement async hide dataLayer 4000 GTM TX83RBN true function g r GoogleAnalyticsObject r r r function r .q r .q .push arguments r .l 1 new Date s.createElement s.getElementsByTagName o a.async 1 a.src g m.parentNode.insertBefore window document script https www.google analytics.com analytics.js ga ga create UA 718476 2 auto allowLinker true ga require GTM TX83RBN function w l w l w l w l .push gtm.start new Date .getTime event gtm.js var f d.getElementsByTagName o j d.createElement dl l dataLayer l l j.async true j.src www.googletagmanager.com gtm.js id dl f.parentNode.insertBefore j f window document script dataLayer GTM 3M4W var tax1AdCategory var tax2AdCategory var tax3AdCategory var dcRefCanonicalLink https my.clevelandclinic.org research function var ht document.createElement script ht.async true ht.type text javascript ht.src document.location.protocol https https is sec http is. indexww.com ht p 183710

doc4.txt - Notepad

File Edit Format View Help

Data mining Wikipedia document.documentElement.className client js RLCONF wgCanonicalNamespace wgCanonicalSpecialPageName 1 wgNamespaceNumber o wgPageName Data mining wgTitle Data mining wgCurRevisionId 919206619 wgRevisionId 919206619 wgArticleId 42253 wgIsArticle o wgIsRedirect 1 wgAction view wgUserName null wgUserGroups wgCategories CS1 maint multiple names authors list Webarchive template wayback links Articles expanded September 2011 All articles specifically marked weasel worded phrases Articles specifically marked weasel worded phrases August 2019 All articles unsourced statements Articles unsourced statements April 2014 Commons category link Wikidata Articles Curlie links Wikipedia articles GND identifiers Wikipedia articles NDL identifiers Data mining Formal sciences wgBreakFrames 1 wgPageContentLanguage en wgPageContentModel wikitext wgSeparatorTransformTable wgDigitTransformTable wgDefaultDateFormat dmy wgMonthNames January

doc5.txt - Notepad

File Edit Format View Help

Data mining Wikipedia document.documentElement.className client js RLCONF wgCanonicalNamespace wgCanonicalSpecialPageName 1 wgNamespaceNumber o wgPageName Data mining wgTitle Data mining wgCurRevisionId 919206619 wgRevisionId 919206619 wgArticleId 42253 wgIsArticle o wgIsRedirect 1 wgAction view wgUserName null wgUserGroups wgCategories CS1 maint multiple names authors list Webarchive template wayback links Articles expanded September 2011 All articles specifically marked weasel worded phrases Articles specifically marked weasel worded phrases August 2019 All articles unsourced statements Articles unsourced statements April 2014 Commons category link Wikidata Articles Curlie links Wikipedia articles GND identifiers Wikipedia articles NDL identifiers Data mining Formal sciences wgBreakFrames 1 wgPageContentLanguage en wgPageContentModel wikitext wgSeparatorTransformTable wgDigitTransformTable wgDefaultDateFormat dmy wgMonthNames January

```
''' #Stemming data
stemmer  = PorterStemmer()
temp_string = ""
for i in range(1,6):
        string = open('doc'+str(i)+'.txt').read()
        tokens = word_tokenize(string)
        for w in tokens:
        temp_string = temp_string+" "+stemmer.stem(w)
        open('doc'+str(i)+'.txt','w').write(temp_string)

'''
```

Creating Tokens

We have the preprocessed text files with only content after removing all the stop words,html tags,punctuation and converting to lowercase .We create a List in the python file which saves the tokens that is each word of the text.

```
'''Creating tokens '''
processed_text = []
for i in range(1,6):
    string3 = open('doc'+str(i)+'.txt').read()
    processed_text.append(word_tokenize(string3))
```

Creating Bigrams

A bigram or digram is a sequence of two adjacent elements from a string of tokens, which are typically letters, syllables, or words. A bigram is an *n*-gram for *n*=2. The frequency distribution of every bigram in a string is commonly used for simple statistical analysis of text in many applications, including in computational linguistics, cryptography, speech recognition, and so on.

```
'''function to create ngrams from a list of words '''
def generate_ngrams(words_list,n):
    ngrams_list = []
    for num in range(0,len(words_list)):
    ngram = ' '.join(words_list[num:num + n])
    ngrams_list.append(ngram)
    return ngrams_list
'''Replacing data mining and machine learning'''
bigrams=[]
for i in range(0,5):
    bigrams.append(generate_ngrams(processed_text[i],2))
data=[]
'''Total N '''
for i in range(0,5):
    processed_text[i] = processed_text[i] + bigrams[i]
    data.extend(processed_text[i])
    #data.extend(bigrams[i])
```

N = len(data)

After creating bigrams we append that to the already existing unigram list of lists.

Extra Credit: Inverted Index (Term Dictionary) Construction with TF-IDF

TF-IDF = Term Frequency (TF) * Inverse Document Frequency (IDF)

Creating tables for storing tf,df and tf_idf :

With coulumns as DOC#,Term and Freq(for tf,tf_idf)

'''DataBase connection'''

Connect

db = pymysql.connect(host="localhost", user="root",passwd="abc5s3",db="mydb")

cursor = db.cursor()

cursor.execute("DELETE FROM tf")

cursor.execute("DELETE FROM df")

cursor.execute("DELETE FROM tf_idf")

Terminology

- t — term (word)
- d — document (set of words)
- N — count of corpus
- corpus — the total document set

TF-IDF stands for **"Term Frequency — Inverse Document Frequency"**. This is a technique to quantify a word in documents, we generally compute a weight to each word which signifies the importance of the word in the document and corpus. This method is a widely used technique in Information Retrieval and Text Mining.

Term Frequency

This measures the frequency of a word in a document. This highly depends on the length of the document and the generality of word , for example a very common word such as "was" can appear multiple times in a document.

TF is individual to each document and word, hence we can formulate TF as follows.

tf(t,d) = count of t in d / number of words in d

Document Frequency

This measures the importance of document in whole set of corpus, this is very similar to TF. The only difference is that TF is frequency counter for a term t in document d, where as DF is the count of **occurrences** of term t in the document set N. In other words, DF is the number of documents in which the word is present. We consider one occurrence if the term consists in the document at least once, we do not need to know the number of times the term is present.

df(t) = occurrence of t in documents

To keep this also in a range, we normalize by dividing with the total number of documents. Our main goal is to know the informativeness of a term, and DF is the exact inverse of it. that is why we inverse the DF

Inverse Document Frequency

IDF is the inverse of the document frequency which measures the informativeness of term t. When we calculate IDF, it will be very low for the most occurring words such as stop words (because stop words such as "is" is present in almost all of the documents, and N/df will give a very low value to that word). This finally gives what we want, a relative weightage.

idf(t) = N/df

During the query time, when a word which is not in vocab occurs, the df will be 0. As we cannot divide by 0, we smoothen the value by adding 1 to the denominator.

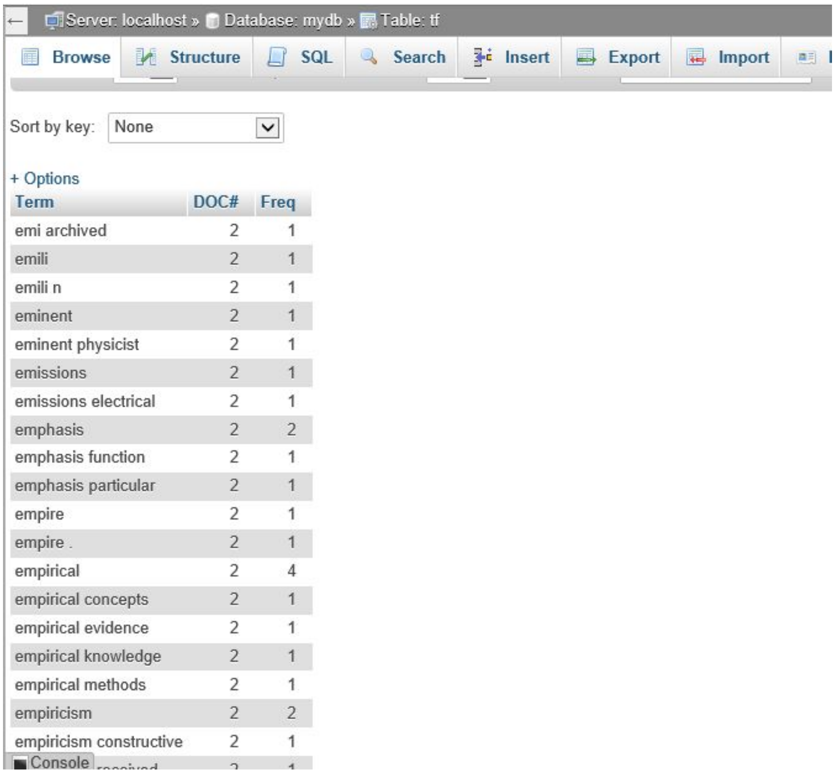
tf-idf(t, d) = tf(t, d) * log(N/(df + 1))

```
'''Calculating DF for all words'''

DF = {}

for j in range(0,5):

    tokens = processed_text[j]

    for w in tokens:

    try:

    DF[w].add(j)

    except:

    DF[w] = {j}

for i in DF:

    DF[i] = len(DF[i])

total_vocab_size = len(DF)


total_vocab = [x for x in DF]


def doc_freq(word):

    c =0
```

```
        try:
        c = DF[word]
        except:
        pass
        return c
'''Calculating TF-IDF '''
doc = 1
tf_idf = {}
vtf = {}
for j in processed_text:
        tokens = j
        counter = Counter(tokens)
words_count = len(tokens)
        for token in np.unique(tokens):
        tf = counter[token]/words_count
        df = doc_freq(token)
        idf = np.log((N+1)/(df+1))
        L = [str(token),int(doc),int(counter[token])]
        cursor.execute("INSERT INTO tf VALUES(%s,%s,%s)",L)
        #vtf[doc,token]=counter[token]
        L1 = [str(token),doc_freq(token)]
        cursor.execute("INSERT INTO df VALUES(%s,%s)",L1)
        tf_idf[doc,token] = tf*idf
        L2=[int(doc),str(token),float(tf_idf[doc,token])]
        cursor.execute("INSERT INTO TF_IDF VALUES(%s,%s,%s)",L2)
        doc +=1
```

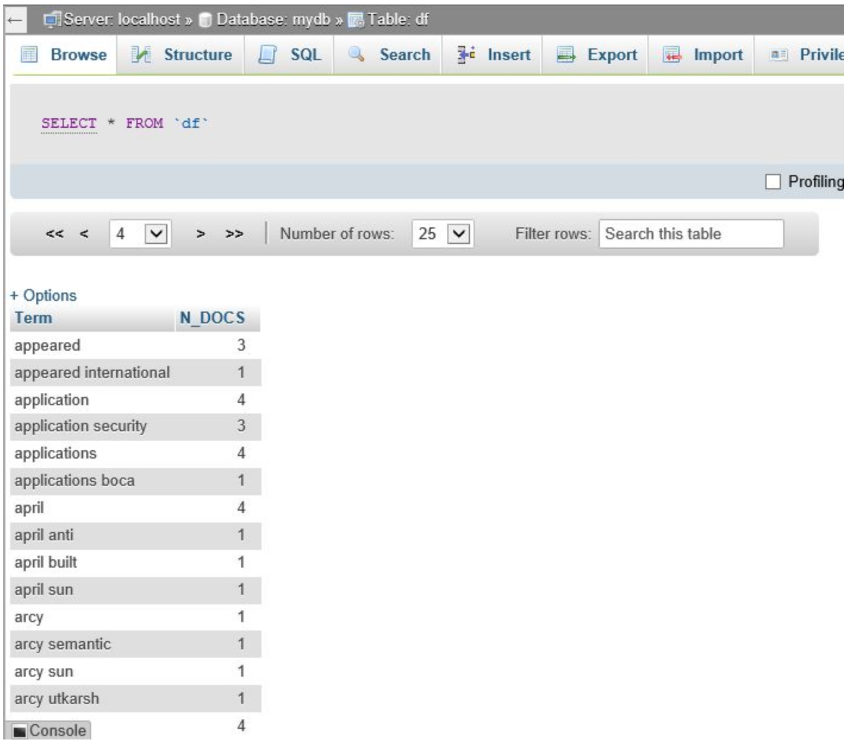
After storing tf,df,tf_idf for all the Terms

We have tf as



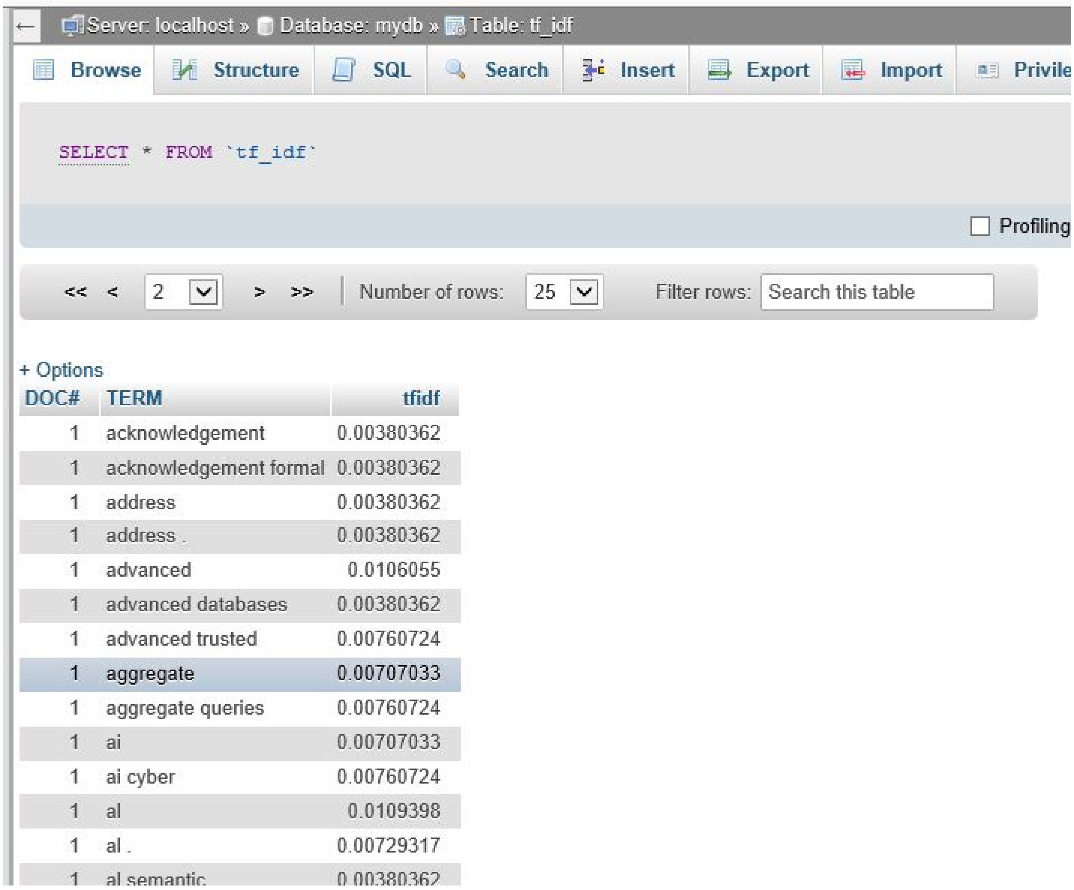
Term	DOC#	Freq
emi archived	2	1
emili	2	1
emili n	2	1
eminent	2	1
eminent physicist	2	1
emissions	2	1
emissions electrical	2	1
emphasis	2	2
emphasis function	2	1
emphasis particular	2	1
empire	2	1
empire .	2	1
empirical	2	4
empirical concepts	2	1
empirical evidence	2	1
empirical knowledge	2	1
empirical methods	2	1
empiricism	2	2
empiricism constructive	2	1

We have df as :



Term	N_DOCS
appeared	3
appeared international	1
application	4
application security	3
applications	4
applications boca	1
april	4
april anti	1
april built	1
april sun	1
arcy	1
arcy semantic	1
arcy sun	1
arcy utkarsh	1
	4

We have tf_idf as :



DOC#	TERM	tfidf
1	acknowledgement	0.00380362
1	acknowledgement formal	0.00380362
1	address	0.00380362
1	address .	0.00380362
1	advanced	0.0106055
1	advanced databases	0.00380362
1	advanced trusted	0.00760724
1	aggregate	0.00707033
1	aggregate queries	0.00760724
1	ai	0.00707033
1	ai cyber	0.00760724
1	al	0.0109398
1	al .	0.00729317
1	al semantic	0.00380362

Part 2: Data Transformation for Topic Analysis of Documents (Webpages)

Building document vectors using if_idf because it already uses weight of each term in the document and will be in normalized way .

```
d_f=pd.DataFrame(index=[1,2,3,4,5],columns=['engineering','research','data','mi
ning','data mining','machine learning'])
for i in d_f.index:
    for j in d_f.columns:
    te = [str(j),i]
    cursor.execute("SELECT tfidf FROM tf_idf WHERE `TERM` =%s AND
`DOC#` =%s",te)
    _tfidf = cursor.fetchall()
    if len(_tfidf) > 0:
    d_f.at[i,j] = float(_tfidf[0][0])
    else :
    d_f.at[i,j] = 0
```

Finding cosine similarity of the vectors built.

```
print('-------------------------TF-IDF-----------------------------')
print(d_f)
cosinesimilarity = pd.DataFrame(index=[0,1,2,3,4],columns=[0,1,2,3,4])
a =cosine_similarity(d_f)
print('------------------Cosine Similarity-------------------')
cosinesimilarity=pd.DataFrame(data=a,index=['doc1','doc2','doc3','doc4','doc5'],
columns=['doc1','doc2','doc3','doc4','doc5'])
print(cosinesimilarity)
```

Final Output:

```
==== RESTART: C:\Users\BHUVAN\Desktop\study zone\DataMinig\Lab 2\lab2.py ====
--------------------------TF-IDF-----------------------------
  engineering   research        data      mining data mining machine learning
1   0.0379362  0.0675626    0.151745  0.00689749  0.00353517        0.0353517
2    0.195894  0.0113659  0.00204768  0.00409535           0                0
3           0   0.115418           0           0           0                0
4   0.0053046   0.011258    0.236939    0.152949    0.132313        0.0290002
5   0.0053046   0.011258    0.236939    0.152949    0.132313        0.0290002
------------------Cosine Similarity------------------
          doc1      doc2      doc3      doc4      doc5
doc1  1.000000  0.249744  0.387880  0.723602  0.723602
doc2  0.249744  1.000000  0.057907  0.037078  0.037078
doc3  0.387880  0.057907  1.000000  0.035956  0.035956
doc4  0.723602  0.037078  0.035956  1.000000  1.000000
doc5  0.723602  0.037078  0.035956  1.000000  1.000000
```

Analysis and Discussion Discuss briefly about your topic analysis from your cosine similarity focusing on whether the indications by the values of your Cosine Sim are all correct?

The Topics of Doc1 is similar to the Topics of Doc 4 and 5? Explain Why or Why Not in terms of 6 TFs? If not, what are the reasons?

- The documents 4 and 5 are similar

Doc4: https://en.wikipedia.org/wiki/Data_mining

Doc5: https://en.wikipedia.org/wiki/Data_mining#Data_mining

So we can see the cosine similarity matrix that they both are similar.

- Document 1 is a website of prof. SS Chung which is related to data mining and machine learning because of her content in the webpage.And related document or close to document 4 and 5 because they are about data mining.
- Document 3 was about research and cleveland clinic which has nothing to do with data mining and machine learning so we can see the results in both the matrices .