Introduction:

Tf-idf represents “term frequency – Inverse Document Frequency” which is a numeric statistic that intended to reflect the importance of a word in the collection or corpus and is calculated by the equation: Term Frequency (TF) * Inverse Document Frequency (IDF). This method is a widely used technique all over the world in information retrieval and text mining.

Design & Implementation:

A represents a collection of documents which are represented by a document-by-word matrix.
A = (aik)
N = the number of documents in the dataset
fik represents the frequency of word k in document i.
fik = number of words appears in document I / the length of document i
nk represents the total number of times word k occurs in the dataset.
nk = whether the word appear in the document, if appeared, the count plus one / the number of all the documents in the dataset(N)
aik = fik * log(N/nk)
Aik = aik/sqrt(aij^2)

Test

1.1 read the text files from 5 sub directions in dataset

Using os.listdir function to read the paths from the documents.

```
##1
##read the path
path = 'C:/Users/Administrator/PycharmProjects/pythonProject3/dataset'
file = []
##use file to store the file path and present
for i in os.listdir(path):
    file.append(i)
```

And the output is:

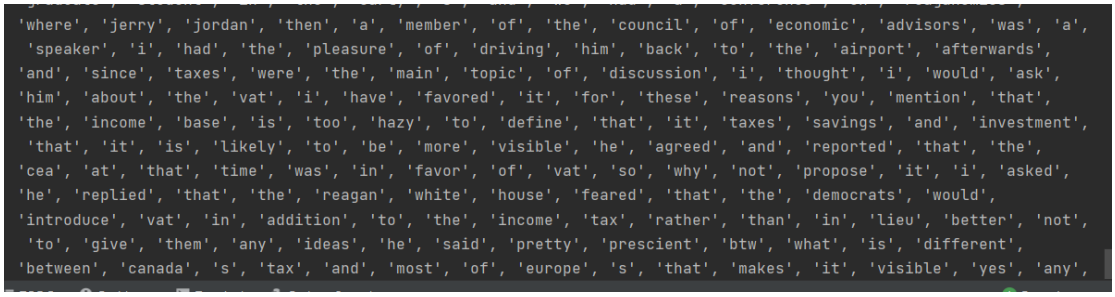
```
['alt.atheism', 'comp.graphics', 'rec.motorcycles', 'soc.religion.christian', 'talk.politics.misc']
```

1.2 split the document text into words

Secondly, I use the subpath to output all the txt document and gathered them into the list subfile. The length of this list is the same number of all the txts which is 2726.

```
subfile=[]
for a in range(0, 5):
    subpath = 'C:/Users/Administrator/PycharmProjects/pythonProject3/dataset/'+file[a]
    for txt in os.listdir(subpath):
        position = subpath+'/'+txt
        f = codecs.open(position, 'r', encoding='Latin1')
        data = f.read()
        subfile.append(data)
```

The outcome turns to be



2. remove the stopwords & convert words into their lower cases form & delete all non-alphabet characters from the text

First, I input the stop words from the path and store them in the 'stopwords'

```
##2
##store the stopwords
stopword = codecs.open('stopwords.txt','r','Latin1')
stopwords = [''.join(c for c in s if c.isalpha()) for s in stopword.read().split()]
print(stopwords)
```

The output of the stop words is:

Then I choose to transfer the file in to lower cases and remove the non-alphabet words in advance.

```
#remove the non-alphabet things from the list
subfile2 = []
for i in range(0,len(subfile)):
    ww = re.sub(r'[^a-z]', ' ', subfile[i].lower()).strip().split()
    subfile2.append(ww)
```

After that I create the nostopwords list to store the out come of the removed file.
By using the if and for function to choose the word that not in the stopwords file and gather.

```
nostopwords = []
for i in range(len(subfile2)):
    no = [wd for wd in subfile2[i] if wd not in stopwords]
    nostopwords.append(no)
```

The output turns out to be:

```
['mathew', 'mantis', 'uk', 'subject', 'alt', 'atheism', 'faq', 'atheist', 'resources',
```

3. Stemmer

Then come to the third process, using the nltk package and store the file result in the singles list.

```
singles = []
stemmer = PorterStemmer()
for i in range(len(nostopwords)):
    token = nostopwords[i]
    stem = [stemmer.stem(m) for m in token]
    singles.append(stem)
print(len(singles))
```

The outcome is:

```
['mathew', 'manti', 'uk', 'subject', 'alt', 'atheism', 'faq', 'atheist', 'resourc', 'summari', 'book', 'address', '

Process finished with exit code 0
```

4.1 the outcomes of N

To calculate the TFIDF, there is a few things should be calculated:
Firstly, is the N, which is the # of all the documents, this can be figured out to be

```
N = len(singles)
```

and the result is

```
2726
```

4.2 calculate idf/nk: the total number of times word K occurs in the dataset

Then, calculating the DF which is also called Nk in the task sheet, and is the the total number of times word k occurs in the dataset called the document frequency.

```
DF = {}
for i in range(len(singles)):
    tokens = singles[i]
    for w in tokens:
        try:
            DF[w].add(i)
        except:
            DF[w] = {i}
for i in DF:
    DF[i] = len(DF[i])
uniqueword = [x for x in DF]
print(len(uniqueword))
```

Create dic DF. For I in 0-2726, if the word does exist in the txt then the count will add 1, otherwise the count will not change. So the final result is:

```
{'mathew': 37, 'manti': 36, 'uk': 260, 'subject': 2726, 'alt': 83, 'atheism':
105, 'faq': 108, 'atheist': 240, 'resourc': 46, 'summari': 123, 'book': 261,
'address': 148, 'music': 26, 'keyword': 199, 'fiction': 18, 'contact': 89,
'expir': 23, 'thu': 23, 'apr': 828, 'gmt': 102, 'distribut': 447, 'organ': 2597,
'consult': 43, 'cambridg': 39, 'supersed': 11, 'line': 2725, 'archiv': 26,
'modifi': 23, 'decemb': 22, 'version': 228, 'usa': 257, 'freedom': 71,
'religion': 224, 'foundat': 59, 'darwin': 4, 'fish': 14, 'bumper': 9, 'sticker':
16, 'assort': 5, 'paraphernalia': 2, 'write': 1729, 'ffrf': 1, 'box': 67,
```

And to out put the exact number of the DF/nk I use another for loop:

```
for i in DF:
    z = len(DF[i])
    count.append(z)
print(count)
```

Then the outcome is:

```
[37, 36, 260, 2726, 83, 105, 108, 240, 46, 123, 261, 148, 26, 199, 18, 89, 23, 23, 828, 102, 447,
```

Moreover, through this DF we can also get the outcome that there are 23130 unique words in the txt file.

```
uniqueword = [x for x in DF]
print(len(uniqueword))
```

Then I calculate the idf in this step, by the equation that idf = log(N/nk)

```
idf = []
for i in range(0,23130):
    tokens = numpy.log(N/(count[i]))
    idf.append(tokens)
##print(idf)
```

The outcome is:

```
[4.299672699612254, 4.327071673800368, 2.3499089812409504, 0.0, 3.49175000445988, 3.256630262
 .0984062568840605, 2.3460702049337847, 2.913378338492363, 4.652494074234996, 2.6172857875319
 1.1915774578712182, 3.285617798972207, 1.808032017642909, 0.04847840059372956, 4.14939049656
 .652494074234996, 4.775096396327328, 4.819548158898162, 2.4812449833020374, 2.36151452736125
 5.27153328264122, 5.713366034920258, 5.138001890016697, 6.301152699822378, 7.217443431696533
 5.831149070576642, 4.966151633090037, 4.5093932305943225, 3.219242730027334, 3.5411427597894
 4.0187703141458515, 2.6023229148552733, 3.0664035257978868, 5.831149070576642, 7.91059061225
 3.5931024987201674, 5.964680463201165, 1.1771887204191187, 3.5285639775825963, 4.50939323059
```

4.3 calculate tf/fik: the frequency of word k in document l

By using the uniquewords, we can quickly get through all the files. And do the calculation that the number of K which appeared in one document / the length of the document to get the frequency.

```
#use listcount to store all the countings
listcount = []
for i in range(len(singles)):
    rowcount=[]
    for j in range(len(uniqueword)):
        count = singles[i].count(uniqueword[j])/len(singles[i])
        rowcount.append(count)
    listcount.append(rowcount)
print(listcount)
```

The outcomes are:

```
[0.003194888178913738, 0.005324813631522897, 0.005324813631522897, 0.002129925452609159,
 0.003194888178913738, 0.013844515441959531, 0.002129925452609159, 0.011714589989350373,
 0.005324813631522897, 0.0010649627263045794, 0.01810436634717785, 0.005324813631522897,
 0.003194888178913738, 0.0010649627263045794, 0.003194888178913738, 0.0010649627263045794,
 0.0010649627263045794, 0.0010649627263045794, 0.0010649627263045794, 0.0010649627263045794,
 0.0010649627263045794, 0.003194888178913738, 0.0010649627263045794, 0.0010649627263045794,
 0.0010649627263045794, 0.002129925452609159, 0.005324813631522897, 0.0010649627263045794,
 0.0010649627263045794, 0.004259850905218318, 0.004259850905218318, 0.002129925452609159,
```

4.4 Calculating the aik(A)

For the equation that aik = idf * tf. The code is at below. The two For loop is to take the number from each sublist in the tf out to do the calculation.

```
aik = []
for i in range(len(tf)):
    for j in range(len(tf[i])):
        step = idf[j] * tf[i][j]
        aik.append(step)
print(aik)
```

And the outcomes are:

```
[0.013736973481189308, 0.023040850233228796, 0.012512827376149895, 0.0, 0.011155750812970863,
```

The length

```
print(len(aik))
```

are:

```
2726
```

4.5 Calculating the Aik(the final result)

Firstly, the total sum is calculated and then it is used to do the final calculation.By taking out the number from the 2726 sublist and do the calculation.

```
total = []
A=[]
B=[]
Aik = []
for i in range(len(aik)):
    for j in range(len(aik[i])):
        total = total + aik[i][j]
        A = aik[i][j]/ numpy.sqrt(total)
        B.append(A)
    Aik.append(B)
print(Aik[0])
```

By giving out 4 list and fill in the blankest, the result for the first sublist comes out to be:

Conclusion

In conclusion, there is no denying that stemmer is a useful package to arrange the documents and td-idf is a powerful method to see the word frequency. By rearranging these tests, we can get better understanding on the weight to each word in the document and corpus.

0.060085