2466883 JiaxinLiu

November 7, 2024

Contents

INT401: Fundamentals of Machine Learning **Fall Semester**

Lab 1: Feature Selection

Student: Jiaxin Liu *ID: 2466883*

1.1 Read Text Files and Split the Document Texts into Words

1.1.1 Read Text Files from 5 Subdirectories

In order to process the data, the first step is to read the data. The data in this task is stored in five folders. Therefore, you need to read the data in the five folders separately and write the file path to a list for easy access. To obtain the pathnames in bulk, you need to construct a function that concatenates the upper-layer path of the file with the filename obtained using the **os.listdir()** function in the os package. In the report is the **write_path(top path,name)** custom function. Each file path name is generated and written into a list to read at one time.

[1]:
```
#Read Files
import os
###os.os.listdir()
files1= os.listdir('E:/mlhomework/dataset-temp/documents-export-2023-10-23/dataset/alt.
 ↪atheism/')
files2= os.listdir('E:/mlhomework/dataset-temp/documents-export-2023-10-23/dataset/
 ↪comp.graphics/')
files3= os.listdir('E:/mlhomework/dataset-temp/documents-export-2023-10-23/dataset/rec.
 ↪motorcycles/')
files4= os.listdir('E:/mlhomework/dataset-temp/documents-export-2023-10-23/dataset/soc.
 ↪religion.christian/')
files5= os.listdir('E:/mlhomework/dataset-temp/documents-export-2023-10-23/dataset/
 ↪talk.politics.misc/')
files=[]

def write_path(top_path,name):     ###Construct a function that gets the pathname
    path=[]
    for i in name:
        path.append(top_path+str(i))
    return path

p1=write_path('E:/mlhomework/dataset-temp/documents-export-2023-10-23/dataset/alt.
 ↪atheism/',files1)
p2=write_path('E:/mlhomework/dataset-temp/documents-export-2023-10-23/dataset/comp.
 ↪graphics/',files2)
p3=write_path('E:/mlhomework/dataset-temp/documents-export-2023-10-23/dataset/rec.
 ↪motorcycles/',files3)
p4=write_path('E:/mlhomework/dataset-temp/documents-export-2023-10-23/dataset/soc.
 ↪religion.christian/',files4)
p5=write_path('E:/mlhomework/dataset-temp/documents-export-2023-10-23/dataset/talk.
 ↪politics.misc/',files5)

files_paths=p1+p2+p3+p4+p5 ##Write five paths to a list

#Build Text Database
```

```
textbase=[]
for i in files_paths:
    with open(i,'r',encoding='Latin1') as file:
        content=file.read()
        textbase.append(content)
print(textbase[0][0:500]) #See if the first 500 characters of the first article were read successfully
```

```
From: mathew <mathew@mantis.co.uk>
Subject: Alt.Atheism FAQ: Atheist Resources
Summary: Books, addresses, music -- anything related to atheism
Keywords: FAQ, atheism, books, music, fiction, addresses, contacts
Expires: Thu, 29 Apr 1993 11:57:19 GMT
Distribution: world
Organization: Mantis Consultants, Cambridge. UK.
Supersedes: <19930301143317@mantis.co.uk>
Lines: 290

Archive-name: atheism/resources
Alt-atheism-archive-name: resources
Last-modified: 11 December 1992
Version: 1.0
```

[19]:
```
len(files_paths) ### check the number of files
```

[19]:
```
2726
```

1.1.2 Split the Document Text into Words

In order to extract stem, calculate word frequency, and so on, we need to break down the whole text into individual words. Since the subsequent calculation of a_ik and n_k is different, it is easier to construct a list of all file terms and a list of file-specific terms here. It is recorded as a list of terms for each file using list nesting, i.e. $[[words_{doc1}], [words_{doc2}]...]$.

In order to better calculate word frequency, we need to remove all non-word characters in the text, including numbers, punctuation, and null values. This report uses the **re.sub()** function in the re package to eliminate all non-word characters using regular expressions. We use **str.lower()** to convert all uppercase letters to lowercase, and then we use the **split()** function to split words using Spaces as delimiters.

[20]:
```
import re
wordsbase=[]
total_doc_words=[]
words_by_text=[]
for i in textbase:  ###Build a list of all file's words
    total_doc_words.extend(re.sub(r'[^a-z]',' ' ,i.lower()).strip().split(' '))###Remove all punctuation marks, convert them to lower case and add them to the list
total_doc_words=list(filter(None, total_doc_words)) ### Revmove Null
print(total_doc_words[0:50])

#important
for i in textbase: ###Build a list of words by article
    wordsbase.append(re.sub(r'[^a-z]',' ' ,i.lower()).strip().split(' '))
```

```
for k in wordsbase:
    c=list(filter(None, k))
    words_by_text.append(c)
print(words_by_text[0][0:10])
```

```
['from', 'mathew', 'mathew', 'mantis', 'co', 'uk', 'subject', 'alt', 'atheism',
'faq', 'atheist', 'resources', 'summary', 'books', 'addresses', 'music',
'anything', 'related', 'to', 'atheism', 'keywords', 'faq', 'atheism', 'books',
'music', 'fiction', 'addresses', 'contacts', 'expires', 'thu', 'apr', 'gmt',
'distribution', 'world', 'organization', 'mantis', 'consultants', 'cambridge',
'uk', 'supersedes', 'mantis', 'co', 'uk', 'lines', 'archive', 'name', 'atheism',
'resources', 'alt', 'atheism']
['from', 'mathew', 'mathew', 'mantis', 'co', 'uk', 'subject', 'alt', 'atheism',
'faq']
```

1.2 Remove the stopwords

1.2.1 Read stopwords

In order to get rid of the words that we don't need for research, we need to load the stop vocabulary to remove the words that we don't need, which are frequent words that carry no informatio,n from the list and prevent them from affecting the subsequent research of other words.

[21]:
```
with open("E:/mlhomework/dataset-temp/documents-export-2023-10-23/stopwords.txt", "r")␣
 ↪as f:  # open the stopwords file
    stopwords = f.read().split('\n') #read the file
print(stopwords[1:10])
```

```
['about', 'above', 'abroad', 'according', 'accordingly', 'across', 'actually',
'adj', 'after']
```

1.2.2 Remove the stopwords from the text collections

Use the *for* loop statement to remove words from the stop words.

[22]:
```
###The total words list removes stopwords
filtered_total_doc_words = [i for i in total_doc_words if not i in stopwords]
filtered_words_by_text=[]
###The words by text list removes stopwords
for i in words_by_text:
    filtered_text = [w for w in i if not w in stopwords]
    filtered_words_by_text.append(filtered_text)
print(filtered_words_by_text[0][0:50])
print(filtered_total_doc_words[0:50])
```

```
['mathew', 'mathew', 'mantis', 'uk', 'subject', 'alt', 'atheism', 'faq',
'atheist', 'resources', 'summary', 'books', 'addresses', 'music', 'atheism',
'keywords', 'faq', 'atheism', 'books', 'music', 'fiction', 'addresses',
'contacts', 'expires', 'thu', 'apr', 'gmt', 'distribution', 'organization',
'mantis', 'consultants', 'cambridge', 'uk', 'supersedes', 'mantis', 'uk',
'lines', 'archive', 'atheism', 'resources', 'alt', 'atheism', 'archive',
'resources', 'modified', 'december', 'version', 'atheist', 'resources',
'addresses']
['mathew', 'mathew', 'mantis', 'uk', 'subject', 'alt', 'atheism', 'faq',
'atheist', 'resources', 'summary', 'books', 'addresses', 'music', 'atheism',
'keywords', 'faq', 'atheism', 'books', 'music', 'fiction', 'addresses',
'contacts', 'expires', 'thu', 'apr', 'gmt', 'distribution', 'organization',
'mantis', 'consultants', 'cambridge', 'uk', 'supersedes', 'mantis', 'uk',
'lines', 'archive', 'atheism', 'resources', 'alt', 'atheism', 'archive',
'resources', 'modified', 'december', 'version', 'atheist', 'resources',
'addresses']
```

1.3 Perform word stemming to remove the word suffix

In English, one word is often a variant of another. Word has, plurality, temporal difference, lead to such as "cat, cats/watch, watching/happy, happiness" can be as different word word frequency calculation, so we will need stemming to better access to the actual frequency words. In information retrieval system, one thing we often do is to extract stemming, that is, to remove the end of English word segmentation transformation form, in the process of Term normalization. For exmaple,happy is called the stem of happiness. To extract the stem, we used the nLTk.stem.porter section of the nltk package to extract the stem. Using *PorterStemmer*(), a stem-extraction algorithm based on suffix stripping, the Porter stemmer algorithm, also known as the Porter stemmer. This method is moderately complex and widely used in stem extraction in natural language processing.

[23]:

```
from nltk.stem.porter import *     ###Stem extraction
### Porter's stem
stemmer=PorterStemmer()
### Total words list after stem extraction
singles= [stemmer.stem(filtered_total_doc_words) for filtered_total_doc_words in␣
 ↪filtered_total_doc_words]
print(singles[0:50])

###The words by text list after stem extraction
stem_words_by_text=[]
for i in filtered_words_by_text:
    stem_words= [stemmer.stem(i) for i in i]
    stem_words_by_text.append(stem_words)
print(stem_words_by_text[0][0:50])  #View the word list of first text
```

```
['mathew', 'mathew', 'manti', 'uk', 'subject', 'alt', 'atheism', 'faq',
'atheist', 'resourc', 'summari', 'book', 'address', 'music', 'atheism',
'keyword', 'faq', 'atheism', 'book', 'music', 'fiction', 'address', 'contact',
'expir', 'thu', 'apr', 'gmt', 'distribut', 'organ', 'manti', 'consult',
'cambridg', 'uk', 'supersed', 'manti', 'uk', 'line', 'archiv', 'atheism',
'resourc', 'alt', 'atheism', 'archiv', 'resourc', 'modifi', 'decemb', 'version',
'atheist', 'resourc', 'address']
['mathew', 'mathew', 'manti', 'uk', 'subject', 'alt', 'atheism', 'faq',
'atheist', 'resourc', 'summari', 'book', 'address', 'music', 'atheism',
'keyword', 'faq', 'atheism', 'book', 'music', 'fiction', 'address', 'contact',
'expir', 'thu', 'apr', 'gmt', 'distribut', 'organ', 'manti', 'consult',
'cambridg', 'uk', 'supersed', 'manti', 'uk', 'line', 'archiv', 'atheism',
'resourc', 'alt', 'atheism', 'archiv', 'resourc', 'modifi', 'decemb', 'version',
'atheist', 'resourc', 'address']
```

1.4 TFIDF Representation

1.4.1 Compute f_{ik}: the Frequency of Word k in Document i

In order to calculate the word frequency, we need to define a word frequency count function $count(input - text)$, calculate the word frequency, the word in each list and the word corresponding word frequency input into the dict to save. In this way, with the word as the key, its corresponding value is the word frequency. The function **count(input-text)** will enter the dict word for the first time as the key, and update its value count each time after the loop, and finally stored in dict form. In the case of **words_freq_by_text** , because we need to annotate the text file from which the text comes, this report uses nested dict to store the data. For example,

{doc1: {word1 : 32, word2 : 41… },doc2:{word1 :12,word3 : 16.. },doc3:{… }… }}

[24]:

```
###Customize a word frequency counting function
def count(input_text):  #Count word frequency
    word_dict = {}
    for i in input_text:
        if i in word_dict.keys():
            word_dict[i] += 1
        else:
            word_dict.update({i:1})
    new_word_dict = word_dict.copy()
    return new_word_dict

text_name=files1+files2+files3+files4+files5

def words_freq(text_name,text): ## Record word frequency in dict format
    words_freq={}
    for i in range(len(text)):
        words_freq.update({text_name[i]:count(text[i])})
    return words_freq

words_freq_by_text=words_freq(text_name,stem_words_by_text)
### check this dict
print(list(words_freq_by_text.items())[0])
```

```
('49960', {'mathew': 3, 'manti': 5, 'uk': 5, 'subject': 2, 'alt': 3, 'atheism':
13, 'faq': 2, 'atheist': 11, 'resourc': 5, 'summari': 1, 'book': 17, 'address':
5, 'music': 3, 'keyword': 1, 'fiction': 3, 'contact': 1, 'expir': 1, 'thu': 1,
'apr': 1, 'gmt': 1, 'distribut': 1, 'organ': 3, 'consult': 1, 'cambridg': 1,
'supersed': 1, 'line': 2, 'archiv': 5, 'modifi': 1, 'decemb': 1, 'version': 4,
'usa': 4, 'freedom': 2, 'religion': 6, 'foundat': 2, 'darwin': 4, 'fish': 6,
'bumper': 1, 'sticker': 1, 'assort': 3, 'paraphernalia': 1, 'write': 8, 'ffrf':
1, 'box': 3, 'madison': 1, 'wi': 1, 'telephon': 4, 'evolut': 3, 'design': 3,
'sell': 2, 'symbol': 1, 'christian': 10, 'stick': 1, 'car': 1, 'feet': 1,
'word': 1, 'written': 2, 'delux': 1, 'mould': 1, 'plastic': 1, 'postpaid': 1,
'laurel': 1, 'canyon': 1, 'north': 1, 'hollywood': 1, 'peopl': 6, 'san': 1,
'francisco': 1, 'bay': 1, 'area': 2, 'lynn': 2, 'gold': 1, 'mail': 4, 'figmo':
1, 'netcom': 1, 'net': 2, 'price': 1, 'american': 8, 'press': 8, 'aap': 2,
'publish': 4, 'critiqu': 2, 'bibl': 8, 'list': 2, 'biblic': 1, 'contradict': 3,
'handbook': 1, 'ball': 2, 'foot': 2, 'isbn': 5, 'edit': 3, 'absurd': 1, 'atroc':
1,'repli': 1})
```

As you can see from the above results, for example, the stem 'mathew' in the file '49960' appears 3 times in the whole text, and the word 'subject' appears twice.

1.4.2 Compute N: the number of documents in the dataset

N is the total number of texts, and you just use the **len()** function.

```
[25]: len(text_name)
```

```
[25]: 2726
```

So there are 2726 files in the entire database.

1.4.3 Coumpute n_k: the total number of documents that word k occurs in the dataset called the document frequency

n_k is the total number of documents that word k occurs in the dataset called the document frequency. The method of calculating $n_k$and the previous calculation The method of f_{ik} is the same, and can be calculated using the previously written word frequency counting function **count(input_text)**. The difference is that the storage of $n_k$does not require nested dict, only one layer of dict.

n_k is the total number of documents that word k occurs in the dataset called the document frequency.

```
[26]: n_k=count(singles)   #Use the previous custom function count to compute n_k. Singles␣
       ↪is the total file after summary
      print(list(n_k.items())[0:50])
```

```
[('mathew', 102), ('manti', 85), ('uk', 567), ('subject', 3005), ('alt', 149),
('atheism', 327), ('faq', 216), ('atheist', 611), ('resourc', 104), ('summari',
133), ('book', 547), ('address', 231), ('music', 37), ('keyword', 203),
('fiction', 29), ('contact', 176), ('expir', 24), ('thu', 25), ('apr', 1073),
('gmt', 109), ('distribut', 491), ('organ', 2807), ('consult', 59), ('cambridg',
43), ('supersed', 11), ('line', 2854), ('archiv', 115), ('modifi', 30),
('decemb', 29), ('version', 415), ('usa', 296), ('freedom', 124), ('religion',
522), ('foundat', 80), ('darwin', 7), ('fish', 28), ('bumper', 9), ('sticker',
27), ('assort', 7), ('paraphernalia', 2), ('write', 2608), ('ffrf', 1), ('box',
116), ('madison', 19), ('wi', 8), ('telephon', 26), ('evolut', 46), ('design',
170), ('sell', 125), ('symbol', 24)]
```

As you can see from the above results, the root word 'mathew' appears 102 times in the entire database, that is, 2726 files.(To view the top 50 key-value pairs in the dict, use item() to take the key-value pairs from the dictionary, place them in a list, and then print them.)

1.4.4 Generating a Dict Containing f_{ik} and a_{ik}

In order to calculate a_{ik} more easily and intuitively, this report writes the corresponding fik and nk into the same dict for easy calculation. First, construct a custom function **get_value(nk,fik)** to fetch the values of $f_{ik}$and n_k from two dictionaries (words_freq_by_text: f_{ik} and nk :n_k) and write them to a double-layer dict store. Loop through fik first, then nk, combining $f_{ik}$and n_k for each word into a list, then building dict with the word as key and the list as value. Its format

$$\{\{doc1 : \{word_1 : [32, 2], word_2 : [41, 7]...\}, doc2 : \{word_1 : [12, 1], word_3 : [16, 2]...\}, doc3 : \{...\}...\}\}$$

Note: Because the resulting dict will not contain words other than those in this text, no f_{ik} value will equal 0, so no 0 will appear when a_{ik} is computed later.

```
[28]: ###Write the corresponding fik and nk into the same dict for easy calculation and␣
       ↪storage
      ###words_freq_by_text --> fik {'409960':{'a':1,'b':2,'c':3},'409961':{'d':5,'e':2}}
      ### nk {'a':10,'b':11,'c':12,'d':10,'e':15}
```

```
def get_value(nk,fik):
    dic2={}
    for i1 in fik:
        dic1 = {}  #IMPORTRANT Empty the dictionary in the outer layer and initialize the loop
        for i2 in nk:
            if i2 in fik.get(i1):
                dic1.update({i2:[nk.get(i2),fik.get(i1).get(i2)]}) #The fik and nk of each word are included in the key-value pair as a list
        dic2.update({i1:dic1}) #Enter the document name and form the data format {file number: {word: [fik,nk]}}
    return dic2
nk_fik_dic=get_value(n_k,words_freq_by_text)
```

Since the actual data is too large to show, use the following example to view the dict after it has been collated by the **get_value()** function.

[29]:

```
### example
f_ik1={'409960':{'a':1,'b':2,'c':3},'409961':{'d':5,'e':2}}
n_k1={'a':10,'b':11,'c':12,'d':10,'e':15}
example=get_value(n_k1,f_ik1)
print(example)
```

```
{'409960': {'a': [10, 1], 'b': [11, 2], 'c': [12, 3]}, '409961': {'d': [10, 5],
'e': [15, 2]}}
```

The above results show that for this hypothetical example, word 'a' in the '40990' doc corresponds to a nik value of 10 and a fik value of 1 .

1.4.5 Compute a_{ik}

After generating a dict containing f_{ik} and n_k , we can calculate a_{ik} much more easily. We only need to define a function to compute the value of a_{ik}, store a_{ik} in the same format in the file-separated dict, and then proceed to the next step. The formula for calculating a_{ik} is as follows:

$a_{ik} = log(f_{ik} + 1.0)log(N/n_k)$

After the calculation is written to the dict in the following format:

$\{\{doc1 : \{word_1 : a_{11}, word_2 : a_{12}...\}, doc2 : \{word_1 : a_{21}, word_3 : a_{22}...\}, doc3 : \{...\}...\}\}$

This format is very convenient for us to query the a_{ik} value of a word in a document.

[30]:

```
####Compute aik
import math
def aik(fik,nk,n):
    a_ik=math.log(fik+1)*math.log(n/nk) #Use the log() function in the math library to calculate the logarithm
    return a_ik

###the entry aik is 0 if the word k is not included in the document i
###To increase the speed of operation the word k not included in the document i is not in the dict
###So you don't need to worry about whether aik is 0 or not
```

```
def aik_dic(dic,n):
    dic2={}
    for i1 in dic:
        dic1={}
        for i2 in dic.get(i1):
            k=dic.get(i1).get(i2)
            v=aik(k[1],k[0],n)
            dic1.update({i2:v})
        dic2.update({i1:dic1})
    return dic2
```

[31]:

```
compute_aik=aik_dic(nk_fik_dic,len(n_k))
print(compute_aik['49960']) ###check the aik dict of 49960 doc
```

```
{'mathew': 7.519858958979399, 'manti': 10.045953085666858, 'uk':
6.645716722198903, 'subject': 2.2426759040388595, 'alt': 6.994490143089284,
'atheism': 11.24091758530694, 'faq': 5.135052284083036, 'atheist':
9.030914795265115, 'resourc': 9.684484170628146, 'summari': 3.5759846350033957,
'book': 10.824313973115716, 'address': 8.254612074639406, 'music':
8.925637549602687, 'keyword': 3.282882601077032, 'fiction': 9.26336946903541,
'contact': 3.381809941893074, 'expir': 4.762857293010677, 'thu':
4.734561642604129, 'apr': 2.1287871641349585, 'gmt': 3.7139217875910897,
'distribut': 2.6706685685060894, 'organ': 2.9244334960416065, 'consult':
4.139382762313235, 'cambridg': 4.358651089469357, 'supersed': 5.303621997565879,
'line': 2.2993160129513073, 'archiv': 9.504338471122017,'repli': 2.572137731163589}
```

1.4.6 Compute A_{ik}

Calculating $a_{ik}$is much easier when we calculate A_{ik}separated by files. $A_{ik}$is calculated as follows:

$$A_{ik} = \frac{a_{ik}}{\sqrt{\sum_{j=1}^{D} a_{ij}^2}}$$

As you can see from the formula, A_{ik} is acturally the value that taking the length of different documents into account to normalize the representation of the documen. s$A_{ik}$is also computed using a custom function, **normalize_aik()**, which takes a_{ik} from the dict and puts it into the dict in the same format as a_{ik}. Finally, we have completed the task of this report by calculating the IDF value of each word in each text and saving it to A_{N*D} where D is the number of the unique words in the document collection.

[34]:

```
def sq(i):
    return i**2

def normalize_aik(dic):
    dic1={}
    dic2={}
    for i in dic:
        dic1={}
        l=list(dic.get(i).values())    ##Get the aij of the full file and write it to a␣
 ↪list
        sigma=pow(sum(map(sq,l)),1/2) ##Use map() to compute the square of each␣
 ↪element in the list separately and to compute the root of the sum of squares of the␣
 ↪aij for the full word in the document
        for w in dic.get(i):
            n_aik=dic.get(i).get(w)/sigma
            dic1.update({w:n_aik})
        dic2.update({i:dic1})
    return dic2
A_nd=normalize_aik(compute_aik)
print(A_nd['49960'])  ###check the values in 49960 doc
```

```
{'mathew': 0.05244944657509002, 'manti': 0.0700684258224514, 'uk':
0.046352486938326984, 'subject': 0.015642196303917423, 'alt':
0.048785108747537644, 'atheism': 0.0784030537755677, 'faq':
0.035815917812221966, 'atheist': 0.06298874562173093, 'resourc':
0.06754725559156029, 'summari': 0.024941746393127694, 'book':
0.07549733054062305, 'address': 0.057574196187537646, 'music':
0.062254458808365803, 'keyword': 0.02289742088737945, 'fiction':
0.06461006844966596, 'contact': 0.023587418439893506, 'expir':
0.03321993543991225, 'thu': 0.03302257918464186, 'apr': 0.014847846115745186,
'gmt': 0.02590382924000468, 'distribut': 0.018627355801184253, 'organ':
0.020397313200918796, 'consult': 0.02887133072975383, 'cambridg':
0.030400681542517648, 'supersed': 0.03699165637723045, 'line':
0.016037249240763184, 'archiv': 0.06629077694036212, 'modifi':
0.03214113524426631, 'decemb': 0.03230503422483298, 'version':
0.04513912270150432, 'usa': 0.048932430976633044, 'freedom':
0.040068631837891944, 'religion': 0.05146265477857459, 'foundat':
0.04342680029991036, 'darwin': 0.09096572228373041, 'fish': 0.09116796735856016,
'bumper': 0.037961810272559224,'bay': 0.03321993543991225, 'area': 0.03516990690383026,
'repli': 0.017940123777633675}
```

1.4.7 Export the Data

Finally, the dataset is save into .npz file, where A is a matrix represented with the numpy array.

[1]:
```
import numpy as np    ###Save data to a compressed file as .npz
np.savez('train-20ng.npz',X=A_nd)
data = np.load('train-20ng.npz',allow_pickle=True)
print(data['X']) ###Check whether the file is saved successfully
```

1.5 Results and interpretation

TF-IDF is a statistical method used to assess the importance of a word to a document in a corpus or a corpus. The importance of a word increases directly with the number of times it appears in a document, but decreases inversely with the frequency of its appearance in a corpus. The main idea of TF-IDF is that if a word has a high frequency of TF in one article and rarely appears in other articles, it is considered that this word or phrase has a good classification ability and is suitable for classification.

The Aik matrix calculated in this report is actually an TFIDF value matrix. Inverse file frequency (IDF) : The IDF of a particular word can be obtained by dividing the total number of files by the number of files containing the word, and then taking the logarithm of the resulting quotient.

If the fewer documents containing the term t, the larger the IDF, then the term has good categorization ability.

So as mentioned above, we can look at the TF*IDF value to see which words in which article are most representative and best represent the article.

For example, if you look at 49960 doc in the dict running results, the TFIDF value of the word ’atheism’ is 0.078, which is higher than the TFIDF value of the word ’subject’ is 0.015, indicating that the word ’athesim’ is a better representation of the article. It appears more frequently in this article but less frequently in other articles, which is easier to be used as a distinguishing mark different from other articles.

For subsequent analysis, because TF-IDF is proportional to the number of occurrences of a word in a document and inversely proportional to the number of occurrences of that word in the entire language. Therefore, the automatic keyword extraction algorithm is very clear, that is, calculate the TF-IDF value of each word in the document, and then arrange it in descending order, take the first few words, you can extract the most representative words of the article.