山东大学 计算机科学与技术 学院

信息检索与数据挖掘 课程实验报告

学号:201600150109	姓名：沈棋韬	班级： 16 人工智能班

实验题目：Inverted index and Boolean Retrieval Model

实验内容：

Use 30548 tweets to create inverted index and realize Boolean Retrieval Model.

实验过程中遇到和解决的问题：

（记录实验过程中遇到的问题，以及解决过程和实验结果。可以适当配以关键代码辅助说明，但不要大段贴代码。）

First we need to load these tweets into memory. Because it is stored with json, I use

```
for line in file:
    tweets.append(json.loads(line))
```

to generate a list named tweets which contains a dictionary of every tweet.

Then, the texts of these tweets need to be pre-processd to get rid of meaningless symbols and correct some type errors. After this, I create a list (doc_word) to store the words of every tweets.

When the preparation is done, I can start to count the frequency of every word in these tweets.

```
for tweet in doc_word:
    tweet_num+=1
    for word in tweet:
        if word in word_dic.keys():
            #列表的第一个值为频数
            word_dic[word][0]+=1
            word_dic[word].append(tweet_num)
        else:
            word_dic[word]=[]
            word_dic[word].append(1)
            word_dic[word].append(tweet_num)
```

For every tweet in the list, I traverse every word in it and add 1 to the frequency and append the number of tweet if it already exists in the word_dic or create a Key with this word and append the number

of the tweet and set its frequency to 1 if it is not in the word_dic.

The structure of doc_word:

```
['how',
 'to',
 'make',
 'money',
 'from',
 'horse',
 'racing',
 'with',
 'betting',
 'systems',
 'http',
 'bitlyhtpzz3'],
['sales',
 'rise',
 'lifting',
 'mcdonald',
 'profit',
 'new',
 'menu',
 'items',
 'helped',
 'earnings',
 'grow',
 'and',
 'overcome',
 'slow',
 'decembers',
 'http',
 'bitlyfip3ot'],
```

The structure of word_dic:

```
 'between': [132,
  127,
  190,
  248,
  493,
  2212,
  2622,
  2743,
  2843,
  2955,
  2972,
  2994,
  3035,
  3442,
  3579,
  3583,
  3888,
  4192,
  5214,
  5224,
  5643,
  6524,
  6580,
  7343,
  7420,
  7608,
```

Every word in these tweets is a key of the dictionary. The first number of the list is the frequency of the word and the other numbers are the tweets in which the word appears.

Merge function: analyze the query and then invoke one of three functions(*mergeAnd*, *mergeOr* and *mergeNot* respectively). For example, if the query looks like "between and home", the merge function will invoke *mergeAnd* and "between" and "home" will be the two parameters of the function.

The definition of these functions are as follows:

```
def merge(query):
    words=nltk.word_tokenize(query)
    words=process(words)
    if 'and' in query:
#        w1=process(words[0])
        w1=words[0]
#        w2=process(words[2])
        w2=words[2]
        return mergeAnd(w1,w2)
    elif 'or' in query:
#        w1=process(words[0])
        w1=words[0]
#        w2=process(words[2])
        w2=words[2]
        return mergeOr(w1,w2)
    elif 'not' in query:
#        w1=process(words[1])
        w1=words[1]
        return mergeNot(w1)
    else:
        print('wrong query')


def mergeAnd(w1,w2):
    if w1 not in word_bag or w2 not in word_bag:
        print('No result')
        return
    lst=[]
    p1=1
    p2=1
    while True:
        if p1==len(word_dic[w1]):
            break
        if p2==len(word_dic[w2]):
            break
        if word_dic[w1][p1]<word_dic[w2][p2]:
            p1+=1
        elif word_dic[w1][p1]>word_dic[w2][p2]:
            p2+=1
        elif word_dic[w1][p1]==word_dic[w2][p2]:
            lst.append(word_dic[w1][p1])
            p1+=1
            p2+=1
        else:
            pass
    return lst
```

```
def mergeOr(w1,w2):
    if w1 not in word_bag and w2 not in word_bag:
        print('No result')
        return
    lst=[]
    p1=1
    p2=1
    while True:
        if word_dic[w1][p1]<word_dic[w2][p2]:
            lst.append(word_dic[w1][p1])
            p1+=1
            if p1==len(word_dic[w1]):
                while p2<len(word_dic[w2]):
                    lst.append(word_dic[w2][p2])
                    p2+=1
                return lst
        if word_dic[w1][p1]>word_dic[w2][p2]:
            lst.append(word_dic[w2][p2])
            p2+=1
            if p2==len(word_dic[w2]):
                while p1<len(word_dic[w1]):
                    lst.append(word_dic[w1][p1])
                    p1+=1
                return lst
        if word_dic[w1][p1]==word_dic[w2][p2]:
            lst.append(word_dic[w1][p1])
            p1+=1
            p2+=1
            if p1==len(word_dic[w1]):
                while p2<len(word_dic[w2]):
                    lst.append(word_dic[w2][p2])
                    p2+=1
                return lst
            if p2==len(word_dic[w2]):
                while p1<len(word_dic[w1]):
                    lst.append(word_dic[w1][p1])
                    p1+=1
                return lst

def mergeNot(w1):
    lst=[]
    if w1 not in word_bag:
        for i in range(1,len(doc_word)):
            lst.append(i)
        return lst
    newlst=[]
    for i in range(1,len(word_dic[w1])):
        newlst.append(word_dic[w1][i])
    for i in range(1,len(doc_word)+1):
        if i not in word_dic[w1]:
            lst.append(i)
    return lst

```

Examples:

```
In [21]: word_dic['harry']
Out[21]:
[23,
 6926,
 9607,
 12101,
 12175,
 12441,
 12663,
 12700,
 12717,
 12740,
 12788,
 12794,
 12940,
 13602,
 16887,
 16963,
 17034,
 17172,
 18010,
 19124,
 19220,
 19450,
 19511,
 21469]

In [22]: word_dic['potter']
Out[22]: [4, 16887, 16963, 17172, 21469]


In [23]: merge('potter and harry')
Out[23]: [16887, 16963, 17172, 21469]
```

```
In [24]: merge('potter or harry')
Out[24]:
[6926,
 9607,
 12101,
 12175,
 12441,
 12663,
 12700,
 12717,
 12740,
 12788,
 12794,
 12940,
 13602,
 16887,
 16963,
 17034,
 17172,
 18010,
 19124,
 19220,
 19450,
 19511,
 21469]
```

结论分析与体会：

Using inverted index can help to reduce the space used to store the words, their frequency and the document in which they appear for the matrix is sparse.

Boolean retrieval model can analyze the combination of several logical Operators and return the documents that fit the query.