# 智能信息获取的大作业报告的说明：

本次大作业报告分为两个部分：

1是信息检索系统，包括建立倒排索引，输入查询关键字，并且实现基于tf-idf的打分函数，根据打分函数得出的分数，输出基于输入查询关键字的tfidf打分排名中的前10名，并且输出分数。

2是基于SCIKIT-LEARN工具实现文本情感分类程序，使用nb，svm，lr这3种模型进行机器学习和测试，测试后结果的主要参数为：Precision，Recall，F1和Accuracy。本程序将会对大量对于京东商城的评价（正面/负面）进行学习并且测试模型。

## 信息检索系统

### 信息检索系统的各个模块

#### 字典的建立

通过读取教师所给的“dictionary.txt”来实现

具体实现请见下方

def creat\_dictionary():  
 dictionary = {}  
 with open('dictionary.txt') as fread:  
 for line in fread:  
 word = line.strip()  
 dictionary[word] = 0  
 return dictionary

#### 按照字典给所给的文档分词

文档的数据结构list【list【】，list【】，……list【】】

分词的方法是对总文档的每一个list进行遍历，然后对每一个list进行分词

遍历如下

def fmm\_segs(docs,dictionary):  
 result=[]  
 for doc in docs:  
 newresult=fmm\_seg(doc,dictionary)  
 result.append(newresult)  
 return result

分词如下

def fmm\_seg(sentence, dictionary):*#result为字典结果* max\_len = 7  
 start = 0  
 result=[]  
 while start < len(sentence):  
 end = min(start + max\_len, len(sentence)) *# 边界* while end > start:  
 candidate = sentence[start: end]  
 *# print(candidate)* if candidate in dictionary or end == start + 1:  
 result.append(candidate)  
 start = end  
 break  
 else:  
 end -= 1  
 return result

#### collection类以及类内的函数

首先说明一下建立的倒排索引的结构，建立的倒排索引的结构如下所示

Dictionary{key=词a，value=dictionary{key=文档编号，value=词a在该文档的出现次数}}

下面是该类中的各个变量：

def \_\_init\_\_(self , strdocuments):  
 self.documents = strdocuments  
 self.n\_docs = len(self.documents)  
 self.dic = self.build\_dic(self.documents)  
 self.idf = self.compute\_idf(self.documents)  
 self.tf\_vecs = self.build\_tf\_vecs(self.documents)

引用这个类的时候要使用构造函数，传进来的参数为一个已经分好了词的list

self.documents：传进来的，已经分好词的文档集，数据结构如下

list【list【】，list【】，……list【】】

self.n\_docs:：文档集有多少篇文档

self.dic:：一个属于documents的字典，比老师给的分词字典少很多词，大大减少倒排索引的规模

self.idf:：每个dic里词的idf值

self.tf\_vecs: 建立的倒排索引

每个变量都有与之对应的函数来生成这个变量，这些函数的实现如下所示：

def build\_dic(self,documents):  
 dic={}  
 for document in documents:  
 for word in document:  
 if word in dic:  
 dic[word]+=1  
 else:  
 dic[word]=0  
 return dic  
  
  
def compute\_idf(self, documents):  
 idf={}  
 df={}  
 for doc in documents:  
 for word in doc:  
 if word in idf and word not in df :  
 idf[word]+=1  
 df[word]=1  
 elif word not in idf and word not in df:  
 idf[word]=1  
 df[word]=1  
 df.clear()  
 *#print(len(documents))* for word in idf :  
 *#print(len(documents)/(idf[word]+1))* idf[word]=math.log(len(documents)/(idf[word]+1))  
 *#print(idf)* return idf  
  
def build\_tf\_vecs(self, documents):  
 vecs = self.dic  
 start = time.clock()  
 for word in vecs:  
 vecs[word]={}  
 for j in range(len(self.documents)):  
 vecs[word][j]=0  
 i=0  
 for doc in documents:  
 self.build\_tf\_vec(doc,vecs,i)  
 i+=1  
 end = time.clock()  
 print('Time in seconds:', end - start)  
 return vecs  
  
def build\_tf\_vec(self, document,inverted\_index:dict,i):  
  
 for word in document :  
 inverted\_index[word][i]+=1  
 return

#### TFIDFModel类以及其中的函数

这个类是针对上一个类（collection）来做的，本质上是为了从上一个类的对象中获取一些信息，算tfidf的分并且排序，可以看到从构造函数传进来的是一个collection的对象。

def \_\_init\_\_(self, collection):  
 self.documents = collection.documents  
 self.idf = collection.idf  
 self.tf\_vecs = collection.tf\_vecs

函数如下

def ranking (self,score):  
 *"""doc\_scores = []  
 for docid, doc in enumerate(self.tf\_vecs):  
 s\_doc\_scores = sorted(doc\_scores, key=lambda item: -item[1])  
 result=[]  
 for docid, score in s\_doc\_scores:  
 result.append((documents[docid], score))"""* rankres = sorted(score.items(), key=lambda d: d[1], reverse=True)  
 return rankres  
  
  
def doc\_score(self, query):  
 *#score(q, d) = sum\_{query中所有词} count(w, d) \* idf(w)* score={}  
 *#print(self.tf\_vecs)* print()  
 *#print(self.idf['北京'])* i=0  
 j=0  
 for word in query:  
 if word not in self.tf\_vecs:  
 prstr='%s关键词无效，不做考察'  
 continue  
 for docnum in self.tf\_vecs[word] :  
 str='%s%d'%('doc',i)  
 if j==0:  
 score[str] = self.tf\_vecs[word][i] \* self.idf[word].real  
 else :  
 score[str] += self.tf\_vecs[word][i] \* self.idf[word].real  
 i+=1  
 i=0  
 j+=1  
 return score

可以看到，score是按照输入的关键词，计算每一个文档对于每个关键词的tfidf值，并且将汇总，返回如下的数据：

Dic{key=文档号，value=该文档对于所有输入关键词的tfidf值的总和}

Ranking函数是将score函数的返回结果进行排序，返回按照value值从高到低排列的数据，其结构和上面的一样

### 测试程序

代码如下

if \_\_name\_\_ == '\_\_main\_\_':  
 doc\_strings = []  
 with open('corpus\_1000.txt','r',encoding='utf-8') as fread:  
 for line in fread:  
 doc\_strings.append(line.strip())  
 dictionary=creat\_dictionary()  
 *#print(dictionary)* documents=fmm\_segs(doc\_strings,dictionary)  
 collect=collection(documents)在这一步构建的倒排索引  
 print(len(collect.dic))  
 *# inverted\_index\_rank(collect.tf\_vecs)* tfidf\_model = TFIDFModel(collect)  
 score = tfidf\_model.doc\_score(['北京', '旅游'])  
 result = tfidf\_model.ranking(score)  
 presult = result[0:9]  
 for word in presult:  
 print(word)  
 print()

输出结果：

