

# Homework1 实验报告

姓名:	程倩楠	学号:	201834858
实验课程:	Data Mining	实验名称:	VSM and KNN
指导老师:	尹建华	完成时间:	2018. 11. 03

## 一、实验目的

1. 理解向量空间模型的基本原理，能够为文档构造向量空间表示。
2. 理解 K 近邻的基本原理，能够实现 K 近邻分类器。

## 二、实验环境

python3.6、Java 1.8

## 三、实验内容

1. 预处理新闻文本数据集，并且得到每个文本的 VSM 表示。
2. 实现 KNN 分类器，测试其在 20Newsgroups 上的效果。

## 四、实验过程

### （一）下载数据集

1. 20 个新闻组数据集是大约 20,000 个新闻文档的集合，这些文档在 20 个不同的新闻组中几乎均匀分布。

名称	修改日期	类型
alt.atheism	2018/10/14 9:43	文件夹
comp.graphics	2018/10/14 9:43	文件夹
comp.os.ms-windows.misc	2018/10/14 9:44	文件夹
comp.sys.ibm.pc.hardware	2018/10/14 9:44	文件夹
comp.sys.mac.hardware	2018/10/14 9:44	文件夹
comp.windows.x	2018/10/14 9:44	文件夹
misc.forsale	2018/10/14 9:44	文件夹
rec.autos	2018/10/14 9:44	文件夹
rec.motorcycles	2018/10/14 9:44	文件夹
rec.sport.baseball	2018/10/14 9:44	文件夹
rec.sport.hockey	2018/10/14 9:44	文件夹
sci.crypt	2018/10/14 9:44	文件夹
sci.electronics	2018/10/14 9:44	文件夹
sci.med	2018/10/14 9:44	文件夹
sci.space	2018/10/14 9:44	文件夹
soc.religion.christian	2018/10/14 9:44	文件夹
talk.politics.guns	2018/10/14 9:44	文件夹
talk.politics.mideast	2018/10/14 9:44	文件夹
talk.politics.misc	2018/10/14 9:44	文件夹
talk.religion.misc	2018/10/14 9:44	文件夹

## 2. [20news-18828.tar.gz](#) (18828 个新闻文档；20 个新闻类)

comp.graphics comp.os.ms-windows.misc comp.sys.ibm.pc.hardware comp.sys.mac.hardware comp.windows.x	rec.autos rec.motorcycles rec.sport.baseball rec.sport.hockey	sci.crypt sci.electronics sci.med sci.space
misc.forsale	talk.politics.misc talk.politics.guns talk.politics.mideast	talk.religion.misc alt.atheism soc.religion.christian

### (二) 新闻文档预处理

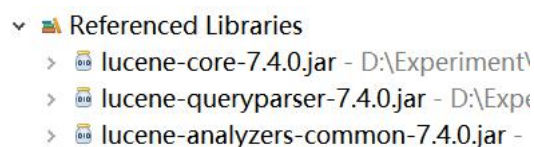
#### 1. 预处理方法

Tokenization	将文本分割为单个词条
Stemming(poter stemmer)	将变形或派生的单词缩减为根形式
Nomalization(lower case)	将单词的不同形式转换为词汇表中的规范化形式(单词小写化)
Remove stopwords	去除停用词，构建可控的词汇表
Remove punctuations	去除标点符号
Remove digital	去除数字

#### 2. 编程实现

借助 Lucene 提供的类编写 java 程序实现新闻文档的预处理, 主要包含以下过程:

##### (1) 导入 Lucene 提供的以下三个 jar 包



##### (2) 编写两个函数实现新闻文本的预处理

<code>String batch_process(String path)</code>	对 path 路径下的新闻文档进行遍历
<code>String pre_process(String doc_path)</code>	对一个文档进行以下处理: 获取文档中的全部文本内容 Remove digital

	Tokenization Remove punctuations Lower case Remove stopwords Porter stemmer
--	---

\* 有关去除低频次和高频词的文本预处理过程在向量空间模型的构建过程中实现。

### (三) 向量空间模型 (Vector Space Model, VSM)

#### 1. 获取新闻语料

遍历预处理后的新闻文档，获得记录元组(news\_class, news\_id)的顺序列表 news\_info，以及记录新闻文本内容的顺序列表 news，将 news\_class (tab) news\_id (tab) text 作为一行存入文件 news\_corpus.txt，得到新闻语料文件。

```

102 #得到news_corpus.txt
103 #格式: news_class news_id text
104 print("get_news_corpus...")
105 news_info=[] #记录(news_class,news_id)
106 news=[] #记录新闻文本['text1','text2','text3',...]
107 corpus_file=open('news_corpus.txt','w')
108 for root,dirs,files in os.walk("../Data/preprocessed_news"):
109     for file in files:
110         news_path=os.path.join(root,file) #news文件路径
111         news_class=news_path.strip().split('\\')[-2] #news所属的类
112         news_id=file #news编号
113         news_file=open(news_path,'r')
114         content=news_file.read().strip() #news文本内容
115         news_file.close()
116         news_info.append((news_class,news_id))
117         news.append(content)
118         corpus_line=news_class+'\t'+news_id+'\t'+content+'\n'
119         corpus_file.write(corpus_line)
120 corpus_file.close()

```

#### 2. 分词

通过一个 word\_seg(news)函数实现对 news 列表中的所有新闻文本进行分词的过程。

```

16 #分词: news=[['word1','word2',...],[...],[...],...]
17 def word_seg(news):
18     print("word_seg...")
19     news=[word_tokenize(sent) for sent in tqdm(news)] #show the progress of word
20     return news

```

#### 3. 统计词条的文档频率(Document Frequency, DF)和逆文档频率(Inverse Document Frequency, IDF)

通过 `cal_words_stat(news)` 函数实现对所有单词 `df` 与 `idf` 的统计。

```
22 #获取单词统计: word_stats={'word1':{'df':int,'idf':float},...}
23 def cal_words_stat(news):
24     print("cal_words_stat...")
25     words_stats={}
26     news_num=len(news) #news的数目
27     for ws in news:
28         for w in set(ws): #遍历一个news中的单词集合
29             if w not in words_stats:
30                 words_stats[w]={}
31                 words_stats[w]['df']=0
32                 words_stats[w]['idf']=0
33                 words_stats[w]['df']+=1 #统计每个单词的df
34     for w,winfo in words_stats.items(): #将dict转为list的形式[(word,{ 'df':int,'idf
35         words_stats[w]['idf']=np.log((1.+news_num)/(1.+winfo['df'])) #计算每个单词
36     return words_stats
```

#### 4. 单词过滤

除去 `df > six.MAXSIZE` 高频单词和 `df < 5` 的低频单词，进一步减少词汇表中单词的数目，从而降低向量空间表示的维度。以上方法通过 `word_filter(news,words_stats)` 函数实现。

```
38 #过滤无用单词 construct controlled vocabulary
39 def word_filter(news,words_stats):
40     print("word_filter...")
41     words_useless=set()
42     min_freq=5
43     max_freq=six.MAXSIZE
44     for w,winfo in words_stats.items():
45         #filter too frequent words and rare words
46         if winfo['df']<min_freq or winfo['df']>max_freq:
47             words_useless.add(w)
48     #filter with useless words
49     news=[w for w in ws if w not in words_useless for ws in tqdm(news)]
50     for wu in words_useless:
51         words_stats.pop(wu) #在words_stats字典中删除元素
52     return news,words_stats,words_useless
```

#### 5. 构建与保存顺序词典

为新闻语料构建一个包含全部词汇的顺序词典，作为向量空间表示的每一个维度，并将 `word (tab) df (tab) idf` 作为每一行存入 `word_dict.txt` 文件中进行保存，此过程通过以下两个函数实现。

```
64 #save word_dict.txt
65 #form: word df idf
66 def save_word_dict(word_dict,words_stats):
67     print("save_word_dict...")
68     savef=open('word_dict.txt','w')
69     for w in word_dict:
70         line=w+'\t'+str(words_stats[w]['df'])+'\t'+str(words_stats[w]['idf'])
71         savef.write(line+'\n')
72     savef.close()
```



```

74 #计算tf + tf normalization(sub-linear tf scaling)
75 def cal_tf_norm(news):
76     print("cal_tf_norm...")
77     news_tf=[] #格式: [{'word1':tf1,'word2':tf2,...},{...},{...},...]
78     for ws in news:
79         d=dict() #格式: {'word1':tf1,'word2':tf2,...}
80         for w in ws:
81             if w not in d:
82                 d[w]=0
83                 d[w]+=1
84             #tf normalization
85             for w,tf in d.items():
86                 tf_norm=1+np.log(tf)
87                 d[w]=tf_norm
88             news_tf.append(d)
89     return news_tf

```

## 6. 统计词频(Term Frequency, TF)和词频正规化(TF-Normalization)

首先对一个文档中每个单词出现的频率进行统计。由于文档长度是不同的，并且语义信息不会与术语出现的次数成比例的增加，导致原始的统计词频是不精确的，需要进行词频正规化。

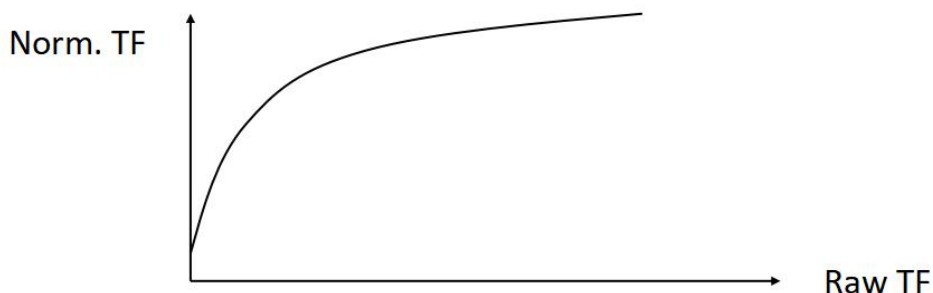
### TF normalization

- Two views of document length
  - A doc is long because it is verbose (冗长的)
  - A doc is long because it has more content
- Raw TF is inaccurate
  - Document length variation
  - “Repeated occurrences” are less informative than the “first occurrence”
  - Information about semantic does not increase proportionally with number of term occurrence
- Generally penalize long document, but avoid over-penalizing
  - Length normalization

本次实验采用 Sub-linear TF scaling 的词频正规化方法。

- Sub-linear TF scaling

$$- tf(t, d) = \begin{cases} 1 + \log c(t, d), & \text{if } c(t, d) > 0 \\ 0, & \text{otherwise} \end{cases}$$



以上过程通过 `cal_tf_norm(news)` 函数实现。

```

74 #计算tf + tf normalization(sub-linear tf scaling)
75 def cal_tf_norm(news):
76     print("cal_tf_norm...")
77     news_tf=[] #格式: [{'word1':tf1,'word2':tf2,...},{...},{...},...]
78     for ws in news:
79         d=dict() #格式: {'word1':tf1,'word2':tf2,...}
80         for w in ws:
81             if w not in d:
82                 d[w]=0
83                 d[w]+=1
84             #tf normalization
85             for w,tf in d.items():
86                 tf_norm=1+np.log(tf)
87                 d[w]=tf_norm
88             news_tf.append(d)
89     return news_tf

```

## 7. 计算 TF-IDF 权重，保存新闻文本的表示向量

以词典中单词的数目作为维度构建表示向量，每个元素为单词在文档中对应的 TF-IDF 权重，将 `news_class` (tab) `news_id` (tab) `vector` 作为每一行写入文件 `news_vector.txt` 进行保存。

```

133 #计算tf-idf权重/得到news_vector/保存到news_vector.txt
134 vdim=len(word_dict) #向量维数
135 print("cal_tf-idf_weights...")
136 print("get_news_vector("+str(vdim)+"d)...")
137 vec_file=open('news_vector.txt','w')
138 for idx,n in enumerate(news_info): #idx:序号 n:值(news_class,news_id)
139     vec=[]
140     for w in word_dict:
141         if w in news_tf[idx]:
142             vec.append(news_tf[idx][w]*words_stats[w]['idf']) #计算tf-idf
143         else:
144             vec.append(0)
145     #norm_vec=vector_unitization(vec) #得到单位向量
146     line=n[0]+'\\t'+n[1]+'\\t'+ ' '.join(['%f' % k for k in vec])
147     vec_file.write(line+'\\n')
148 vec_file.close()
149 print("vsm finished!")

```

## 8. vsm 运行结果

```

get_news_corpus...
word_seg...
100%|██████████| 18828/18828 [00:27<00:00, 695.83it/s]
cal_words_stat...
word_filter...
100%|██████████| 18828/18828 [00:00<00:00, 36444.82it/s]
build_word_dict...
save_word_dict...
cal_tf_norm...
cal_tf-idf_weights...
get_news_vector(23676d)...
vsm finished!

```

输出文件	行格式
news_corpus.txt	news_class (tab) news_id (tab) text
word_dict.txt	word (tab) df (tab) idf
news_vector.txt	news_class (tab) news_id (tab) vector

#### (四) K 近邻 (K Nearest Neighbors, KNN)

##### 1. 分数据

需要将 18828 个新闻文本数据分为 20% 的测试数据和 80% 的训练数据。因为新闻文本数据在 20 个类中均匀分布，所以本次实验采用的方法是在每个类中随机抽取 20% 的数据，合并作为 KNN 分类器的测试数据，其余数据作为 KNN 分类器的训练数据。

首先，建立一个 dict 记录每个类中所包含的新闻数目，其格式为 {'class1':count1, 'class2':count2, ...}，将 dict 每个 key 对应的 value 乘 0.20 作为每个新闻类分为测试数据的数目。

```

10 #构建classDict: {'class1':count, 'class2':count, ...}
11 classDict={}
12 datadir='../Data/preprocessed_news'
13 classList=os.listdir(datadir)
14 for nclass in classList:
15     newList=os.listdir(datadir+'\\'+nclass)
16     classDict[nclass]=len(newList)
17 print(classDict)
18
19 #classDict每个元素的value*20%: 每个类划分为testdata的数据
20 for cl,co in classDict.items():
21     classDict[cl]=int(co*0.20) #hold out 20%
22 print(classDict)

```

在 news\_vector.txt 文件中，分别为每个类抽取相应数目的数据，记录在 testData.txt 文件中，合并作为最终 KNN 分类器的测试数据。

```

24 #分别对每个新闻类随机取出20%的数据，合并
25 #得到文件: testData.txt
26 testNewsID=[] #记录作为测试数据的news_id:(class,id)
27 testf=open('testData.txt','w')
28 for cl,co in classDict.items():
29     f=open('..\VSM\news_vector.txt','r')
30     for line in f:
31         r=line.strip().split('\t')
32         if r[0]==cl:
33             testf.write(line) #写入测试数据
34             testNewsID.append((r[0],r[1]))
35             co=co-1
36         if co==0:
37             break
38     f.close()
39 testf.close()
40 print(len(testNewsID))

```

将除去测试数据的其余数据作为 KNN 分类器的训练数据。

```

42 #将除了测试数据的其他数据作为训练数据
43 #得到文件: trainData.txtA
44 num=0
45 trainf=open('trainData.txt','w')
46 f=open('..\VSM\news_vector.txt','r')
47 for line in f:
48     r=line.strip().split('\t')
49     if (r[0],r[1]) not in testNewsID:
50         trainf.write(line) #写入训练数据
51         num+=1
52 f.close()
53 trainf.close()
54 print(num)
55 print("Dividing Data Finished!")

```

## 2. 实现 KNN 分类器

KNN 的实现主要依靠以下几个核心函数：

函数名称	file2matrix(filename)
函数功能	将 file 文件读取为一个矩阵,矩阵的行向量代表一个新闻文本的表示向量。
<pre> 23 def file2matrix(filename): 24     f=open(filename,'r') 25     lines=f.readlines() #将文件读取为一个list, 其中一个line作为一个元素 26     row=len(lines) #行数 27     vector=lines[0].strip().split('\t')[2] 28     col=len(vector.split()) #列数 29     print("Matrix_dim: ["+str(row)+","+str(col)+"]") #输出矩阵维数 30     returnMat=np.zeros((row,col)) #创建特征矩阵[row,col] 31     classLabel=[] #创建类标签列表 32     index=0 33     for line in lines: 34         vec=line.strip().split('\t')[2].split() #字符串列表 35         vec=[float(k) for k in vec] #将list元素转为float类型 36         returnMat[index,:]=vec[0:col] 37         label=line.strip().split('\t')[0] 38         classLabel.append(label) 39         index+=1 40     f.close() 41     return returnMat,classLabel </pre>	



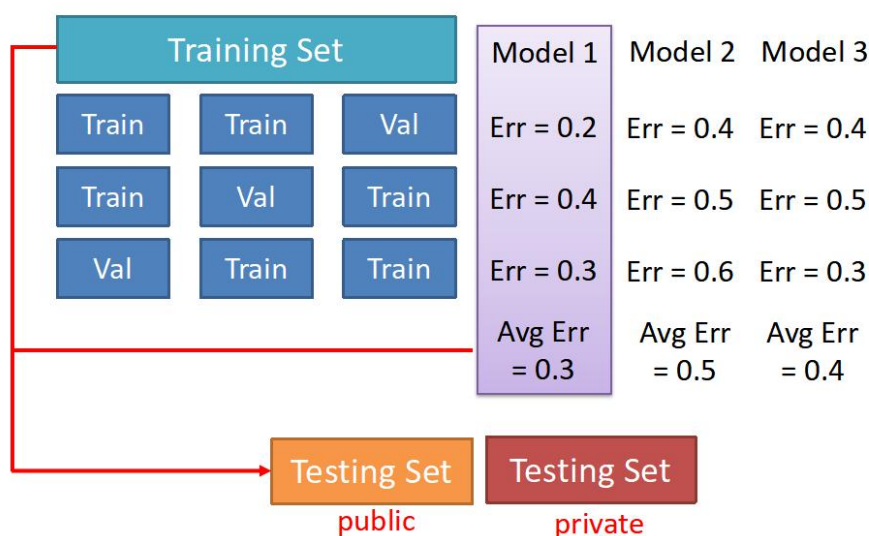
函数名称	autoNorm(dataMat)
函数功能	此函数对矩阵进行特征值归一化: $\text{newValue}=(\text{oldValue}-\text{min})/(\text{max}-\text{min})$ , 即对特征进行缩放, 以防止相似性度量被其中一个特征所支配。
<pre> 10 #特征值归一化: newValue=(oldValue-min)/(max-min) 11 def autoNorm(dataMat): 12     minVals=dataMat.min(0) #每一列的最小值 13     maxVals=dataMat.max(0) #每一列的最大值 14     ranges=maxVals-minVals #特征值的范围 15     ranges=np.where(ranges&gt;0,ranges,1.0) 16     normDataMat=np.zeros(np.shape(dataMat)) #构建一个空矩阵作为归一化后的特征值矩阵 17     m=dataMat.shape[0] #矩阵行数 18     normDataMat=dataMat-np.tile(minVals,(m,1)) #分子 19     normDataMat=normDataMat/np.tile(ranges,(m,1)) #newValue 20     return normDataMat,ranges,minVals </pre>	
函数名称	CosSimilarity(vec,Mat)
函数功能	<p>计算一个向量与矩阵所有行向量之间的余弦相似度。</p> <p>两向量 <math>d_i, d_j</math> 之间的余弦相似度计算公式如下:</p> $- \cosine(d_i, d_j) = \frac{v_{d_i}^T v_{d_j}}{ v_{d_i} _2 \times  v_{d_j} _2}$ <p style="text-align: right;">Unit vector</p>
<pre> 43 #计算一个vector与一个matrix每一行的CosSimilarity 44 def CosSimilarity(vec,Mat): 45     fenzi=np.dot(vec,Mat.T) #分子: vec与矩阵每一行的内积 46     vecNorm=np.linalg.norm(vec) #向量的模 47     MatNorm=np.linalg.norm(Mat,axis=1) #矩阵行向量的模 48     fenmu=vecNorm*MatNorm #分母: vec的模与矩阵行向量模的乘积 49     cos=fenzi/fenmu #CosSimilarity: 内积/模的乘积 50     return cos </pre>	
函数名称	knnClassify(vecX,dataMat,labels,k)
函数功能	计算与训练数据之间的余弦相似度, 识别 k 个近邻, 使用近邻的标签决定未知记录的标签, 其中设置权重因子为余弦相似度。
<pre> 52 #knn分类器 53 def knnClassify(vecX,dataMat,labels,k): 54     cos=CosSimilarity(vecX,dataMat) #计算与train_data之间的余弦相似度 55     sortedIndex=np.argsort(-cos) #返回value从大到小的index 56     #识别k个近邻 57     classCount={} 58     for i in range(k): 59         voteClass=labels[sortedIndex[i]] #排序第i个近邻的类标签 60         c=cos[sortedIndex[i]] #排序第i个近邻的余弦相似度 61         classCount[voteClass]=classCount.get(voteClass,0)+c 62     sortedClassCount=sorted(classCount.items(),key=operator.itemgetter(1), \ 63                             reverse=True) #排序vote 64     return sortedClassCount[0][0] #返回得分最高的类 </pre>	

函数名称	errorRate(testMat,testLabels,trainMat,trainLabels,k)
函数功能	计算对测试数据进行分类的错误率： 在整个测试集中，分类错误的样本占全部测试样本的比例。
<pre> 66 #Evaluation: Classifier error rate 67 def errorRate(testMat,testLabels,trainMat,trainLabels,k): 68     trainNormMat,ranges,minVals=autoNorm(trainMat) 69     m=testMat.shape[0] #test样本数目 70     errorCount=0.0 71     for i in range(m): 72         testNormVec=(testMat[i,:]-minVals)/ranges #test样本特征值归一 73         classifyResult=knnClassify(testNormVec,trainNormMat,trainLabels,k) 74         print("kNN class: %s, real class : %s"%(classifyResult, testLabels[i])) 75         if(classifyResult!=testLabels[i]): 76             errorCount+=1.0 77     ErrorRate=errorCount/float(m) 78     return ErrorRate </pre>	

### 3. N-折交叉验证

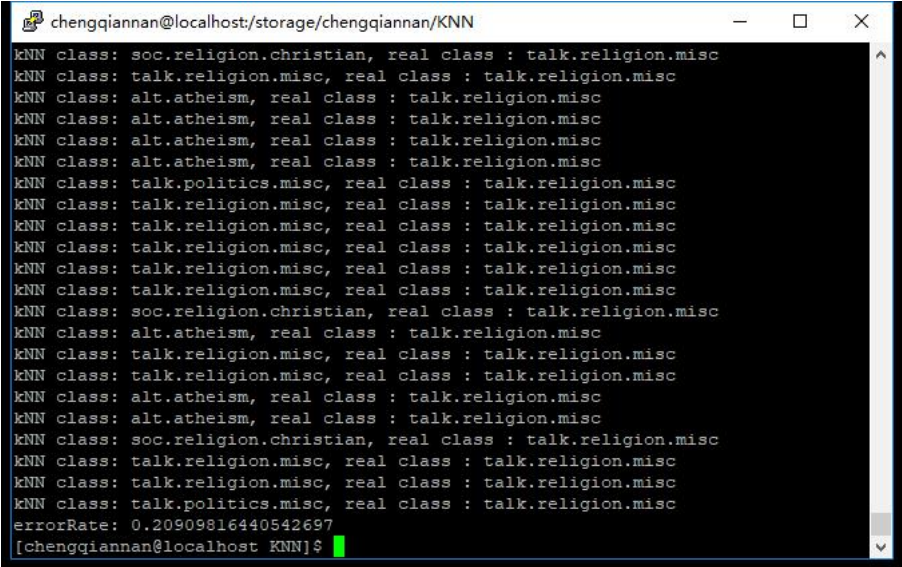
本次实验采用五折交叉验证的方法，对 KNN 参数  $k$  进行调整。将训练数据随机分为五份，在一定范围内循环取  $k$  值，选取一份校验数据，其余四份合并作为训练数据，计算 KNN 分类器的错误率。将每份数据作为测试数据得到的五个错误率取平均，得到某  $k$  值下的平均错误率。将所有  $k$  值对应的平均错误率进行比较，最终得出使 KNN 分类器错误率最小的最优  $k$  值。将最优  $k$  值在在测试数据上进行测试，计算在测试数据上的错误率，基本思想如下图所示。

## N-fold Cross Validation



五、实验结果

1. 任意选择一个 k 值，测试 KNN 分类器的错误率

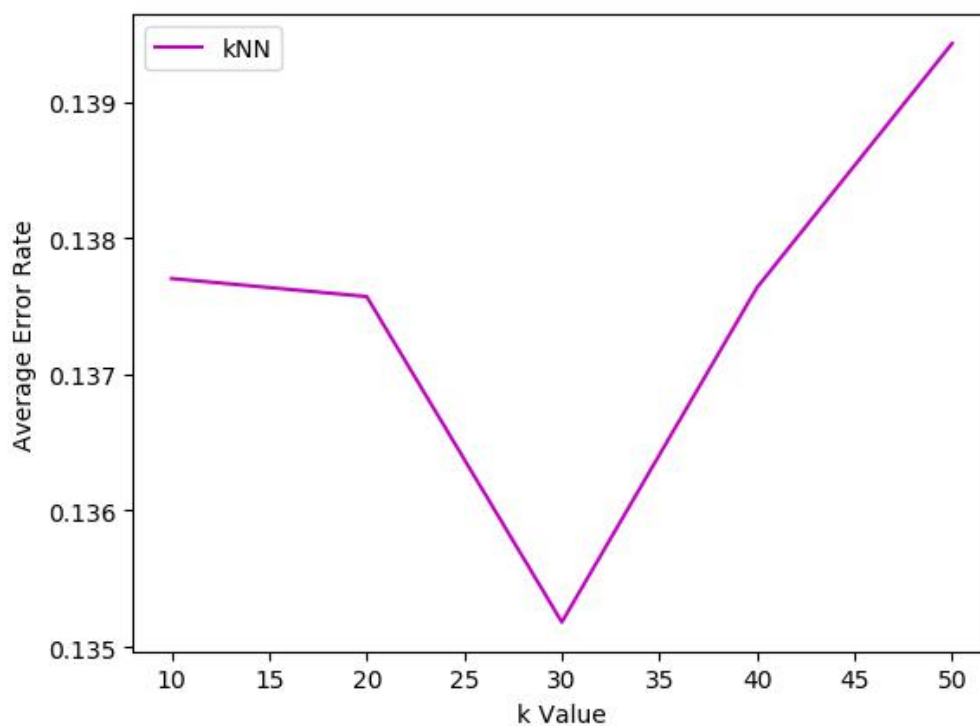
k=50	Error Rate=0.2091
	

2. 五折交叉验证

程序运行耗时太长，所以只循环了 5 个 k 值，将中间结果写入一个 record.txt 记录文件。

K value	Error Rate	Average Error Rate
k=10	0.14205111184865582	0.13770232868925852
	0.14238300696979755	
	0.14238300696979755	
	0.1330899435778294	
	0.12860457408021214	
k=20	0.15167607036176567	0.13756983466078548
	0.1364088947892466	
	0.14072353136408894	
	0.13143046797212082	
	0.12761020881670534	

k=30	0.14603385330235646	0.13518045380854876
	0.13574510454696315	
	0.13773647527381347	
	0.1297709923664122	
	0.12661584355319855	
k=40	0.14470627281778958	0.1376357736517077
	0.134749419183538	
	0.1433786923332227	
	0.13607699966810488	
	0.12926748425588333	
k=50	0.14570195818121473	0.1394276552792281
	0.13939595087952208	
	0.1433786923332227	
	0.1380683703949552	
	0.1305933046072257	



可以清晰地看到在  $k=30$  处，平均错误率达到最小值，将  $k=30$  代入测试数据进行测试，得到在测试集上的分类错误率为 0.1372。