

《Data Mining》

实 验 报 告

学 院： 计算机科学与技术学院

专 业： 计算机技术

学 号： 201834858

姓 名： 程倩楠

实验时间： 2018 年第 1 学期

指导教师： 尹建华

目录

Homework1: VSM and KNN.....	1
一、 实验目的.....	1
二、 实验环境.....	1
三、 实验内容.....	1
四、 实验过程.....	1
(一) 下载数据集.....	1
(二) 新闻文档预处理.....	2
(三) 向量空间模型 (Vector Space Model, VSM).....	3
(四) K 近邻 (K Nearest Neighbors, KNN).....	7
五、 实验结果.....	11
Homework2: NBC.....	13
一、 实验目的.....	13
二、 实验环境.....	13
三、 实验内容.....	13
四、 实验过程.....	13
(一) 数据预处理.....	13
(二) 划分数据.....	13
(三) 朴素贝叶斯分类器.....	14
(四) 5 折交叉验证.....	16
五、 实验结果.....	17
(一) 随机划分数据 (20% test, 80% train)	17
(二) 五折交叉验证.....	17
Homework3: Clustering with sklearn.....	19
一、 实验目的.....	19
二、 实验环境.....	19
三、 实验内容.....	19
四、 实验过程.....	19
(一) 数据处理.....	19
(二) 聚类算法调用与评估.....	20
(三) 比较 8 种聚类算法.....	23
五、 实验结果.....	23
(一) 单独调用和评估聚类算法.....	23
(二) 循环调用和比较聚类算法.....	23

Homework1: VSM and KNN

完成时间：2018 年 11 月 3 日

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

```

v Referenced Libraries
> lucene-core-7.4.0.jar - D:\Experiment\
> lucene-queryparser-7.4.0.jar - D:\Exp
> lucene-analyzers-common-7.4.0.jar -

```

(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':float})]
35         words_stats[w]['idf']=np.log((1.+news_num)/(1.+winfo['df'])) #计算每个单词的idf
36     return words_stats

```

4. 单词过滤

除去 $df > \text{six.MAXSIZE}$ 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。

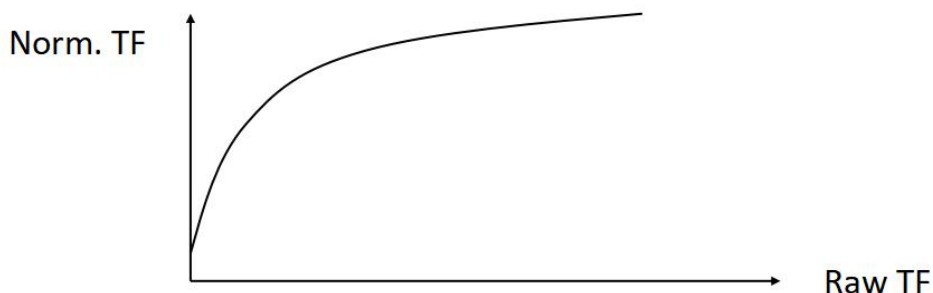
TF normalization

- 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>	

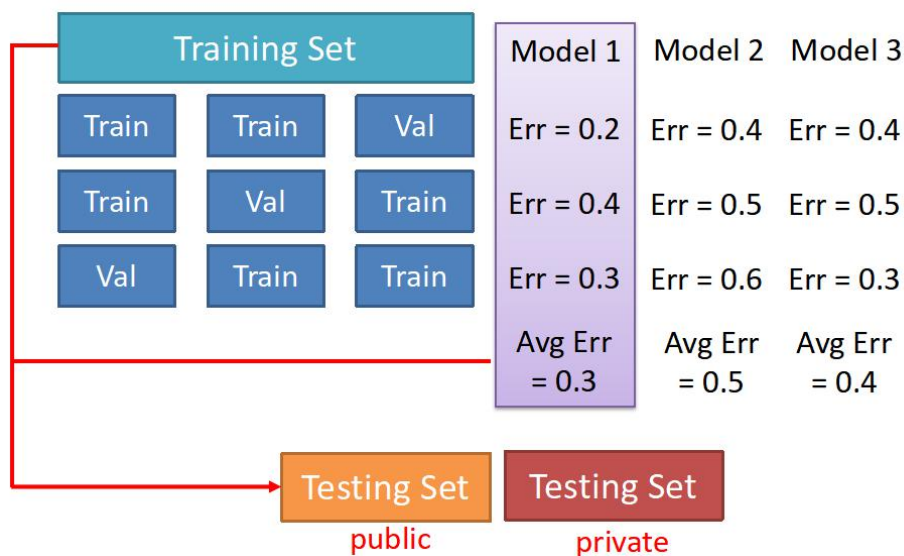
函数名称	autoNorm(dataMat)
函数功能	此函数对矩阵进行特征值归一化: $\text{newValue}=(\text{oldValue}-\text{min})/(\text{max}-\text{min})$, 即对特征进行缩放, 以防止相似性度量被其中一个特征所支配。
<pre> 10 #特征值归一化: newValue=(oldValue-min)/(max-min) 11 def autoNorm(dataMat): 12 minVals=dataMat.min(0) #每一列的最小值 13 maxVals=dataMat.max(0) #每一列的最大值 14 ranges=maxVals-minVals #特征值的范围 15 ranges=np.where(ranges>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 d_i, d_j 之间的余弦相似度计算公式如下:</p> $- \cos(d_i, d_j) = \frac{v_{d_i}^T v_{d_j}}{ v_{d_i} _2 \times v_{d_j} _2}$ <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

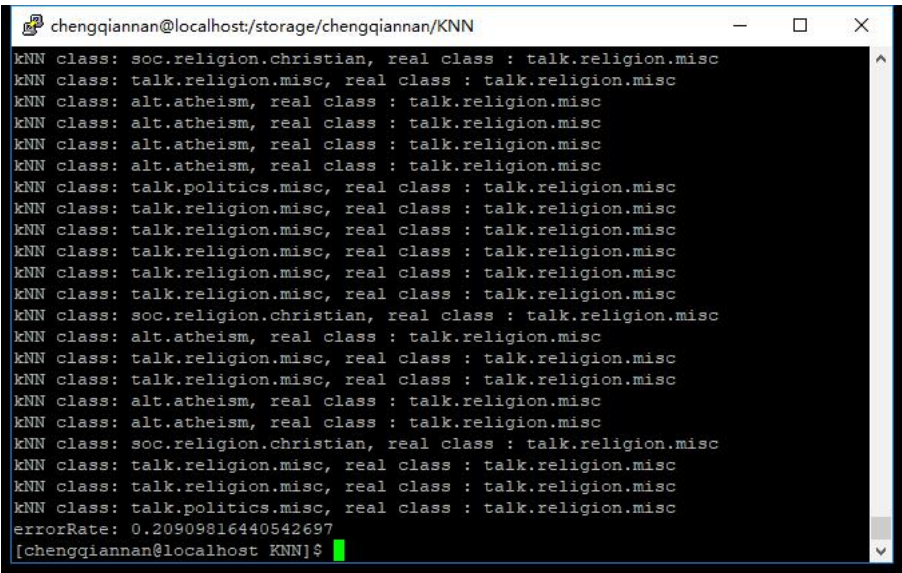
本次实验采用五折交叉验证的方法，对 KNN 参数 k 进行调整。将训练数据随机分为五份，在一定范围内循环取 k 值，选取一份校验数据，其余四份合并作为训练数据，计算 KNN 分类器的错误率。将每份数据作为测试数据得到的五个错误率取平均，得到某 k 值下的平均错误率。将所有 k 值对应的平均错误率进行比较，最终得出使 KNN 分类器错误率最小的最优 k 值。将最优 k 值在在测试数据上进行测试，计算在测试数据上的错误率，基本思想如下图所示。

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

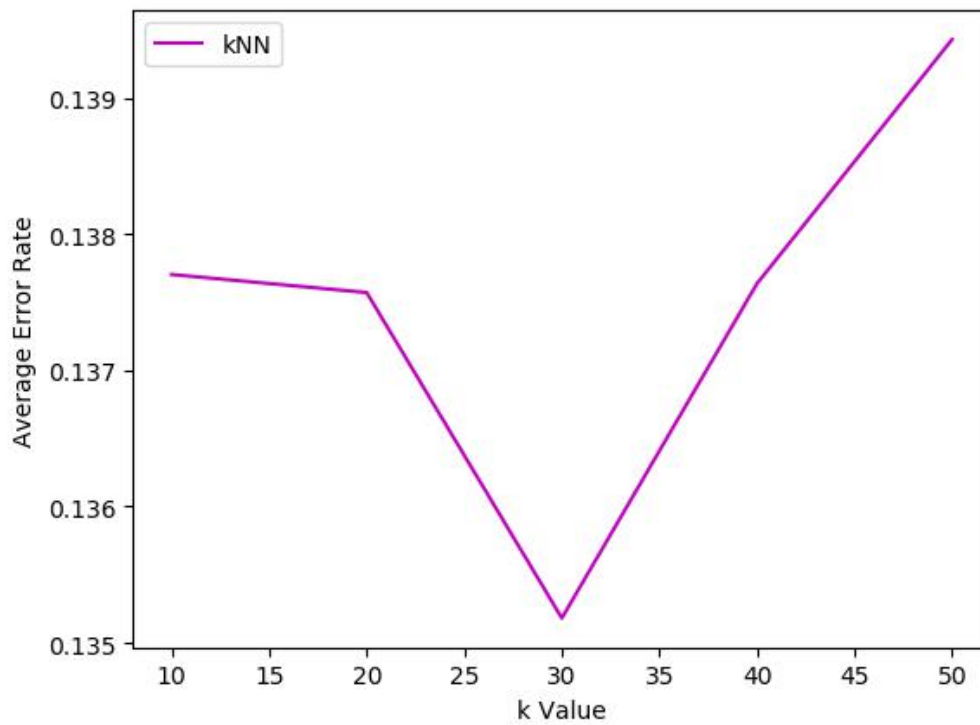
k=50	Error Rate=0.2091
 <pre> chengqiannan@localhost:/storage/chengqiannan/KNN kNN class: soc.religion.christian, real class : talk.religion.misc kNN class: talk.religion.misc, real class : talk.religion.misc kNN class: alt.atheism, real class : talk.religion.misc kNN class: alt.atheism, real class : talk.religion.misc kNN class: alt.atheism, real class : talk.religion.misc kNN class: alt.atheism, real class : talk.religion.misc kNN class: talk.politics.misc, real class : talk.religion.misc kNN class: talk.religion.misc, real class : talk.religion.misc kNN class: talk.religion.misc, real class : talk.religion.misc kNN class: talk.religion.misc, real class : talk.religion.misc kNN class: talk.religion.misc, real class : talk.religion.misc kNN class: talk.religion.misc, real class : talk.religion.misc kNN class: soc.religion.christian, real class : talk.religion.misc kNN class: alt.atheism, real class : talk.religion.misc kNN class: talk.religion.misc, real class : talk.religion.misc kNN class: talk.religion.misc, real class : talk.religion.misc kNN class: alt.atheism, real class : talk.religion.misc kNN class: alt.atheism, real class : talk.religion.misc kNN class: soc.religion.christian, real class : talk.religion.misc kNN class: talk.religion.misc, real class : talk.religion.misc kNN class: talk.religion.misc, real class : talk.religion.misc kNN class: talk.politics.misc, real class : talk.religion.misc errorRate: 0.20909816440542697 [chengqiannan@localhost KNN]\$ </pre>	

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。

Homework2: NBC

完成时间：2018 年 11 月 16 日

一、实验目的

理解朴素贝叶斯的基本原理，能够编程实现朴素贝叶斯分类器。

二、实验环境

python3.6

三、实验内容

实现朴素贝叶斯分类器，测试在 20 Newsgroups 数据集上的效果。

四、实验过程

（一）数据预处理

在 Homework1 中借助 Lucene 工具用 java 程序实现新闻文档的预处理，这里借助 nltk 模块用 python 程序实现预处理，处理步骤如下：

1. 词条化（利用非字母的字符对文本进行分割）
2. 去除停用词
3. 小写化
4. 词干化（Porter Stemmer）
5. 统计出每个单词的集合频率（Collection Frequency, CF），即单词在所有新闻文档中出现的次数。去除文档中 $CF < 5$ 的所有低频单词。

（二）划分数据

划分数据依靠函数 `dataSeg(fold, rightCateFile, trainDataProp=0.8)` 来实现，函数的主要功能为划分出一定比例的测试数据和训练数据，并将标注的测试数据记录在特定文件中。

参数	解释
fold	将第几份数据作为测试数据 (用于 n-Fold Cross Validation)
rightCateFile	记录标注测试数据的文件

	<newsName_newsClass newsClass>
trainDataProp	划分为训练数据的比例
<pre> 15 def dataSeg(fold, rightCateFile, trainDataProp=0.8): 16 fw=open(rightCateFile, 'w') 17 srcDir='used_news' 18 srcClassList=os.listdir(srcDir) 19 for i in range(len(srcClassList)): 20 srcClassDir=srcDir+'/'+srcClassList[i] 21 srcNewsList=os.listdir(srcClassDir) 22 m=len(srcNewsList) #新闻类中所包含的新闻文档的数目 23 testBeginIndex=fold*(m*(1-trainDataProp)) #测试数据的起始索引 24 testEndIndex=(fold+1)*(m*(1-trainDataProp)) #测试数据的结束索引 25 for j in range(m): #遍历新闻类下的所有新闻文档 26 if (j>=testBeginIndex) and (j<testEndIndex): 27 #记录标注: 文档id(文档名_所属类) 所属类 28 fw.write('%s %s\n' % (srcNewsList[j]+'_'+srcClassList[i], \ 29 srcClassList[i])) 30 targetClassDir='TestData'+str(fold)+'/'+srcClassList[i] 31 else: 32 targetClassDir='TrainData'+str(fold)+'/'+srcClassList[i] 33 if os.path.exists(targetClassDir)==False: 34 os.makedirs(targetClassDir) 35 nf=open(targetClassDir+'/'+srcNewsList[j], 'w') 36 lineList=open(srcClassDir+'/'+srcNewsList[j], 'rb').readlines() 37 for line in lineList: 38 line=line.decode('utf-8', 'ignore') 39 nf.write('%s\n' % line.strip('\n')) 40 nf.close() 41 fw.close() </pre>	
<p>由于 20news-18828 中含有重名文档, 这里利用 newsName_newsClass 作为文档标识符</p>	

(三) 朴素贝叶斯分类器

朴素贝叶斯的实现主要包括以下几个函数:

1. 函数 trainNB
<p>训练朴素贝叶斯分类器: 对训练样本数据进行统计, 得到类 cate 下单词 word 出现的次数、每个新闻类中单词总数、训练数据中不重复的单词总数。</p>
<pre> 15 def trainNB(trainDir): 16 cateWordCount={} 17 cateWordNum={} 18 vocab=set() 19 classList=os.listdir(trainDir) 20 for i in range(len(classList)): 21 count=0 #记录每个新闻类中单词总数 22 classDir=trainDir+'/'+classList[i] #类目录 23 newsList=os.listdir(classDir) 24 for j in range(len(newsList)): 25 newsDir=classDir+'/'+newsList[j] 26 lines=open(newsDir, 'rb').readlines() 27 for line in lines: </pre>

```

28         line=line.decode('utf-8','ignore')
29         count+=1
30         word=line.strip('\n')
31         vocab.add(word) #记录训练数据中不重复词汇
32         key=classList[i]+'_'+word
33         cateWordCount[key]=cateWordCount.get(key,0)+1 #记录每个类中每个单词
34         cateWordNum[classList[i]]=count
35     vocabNum=len(vocab)
36     return cateWordCount,cateWordNum,vocabNum

```

2. 函数 calCateProb

计算测试文档属于某个类的概率：

$$p(\text{cate}|\text{doc})=p(w_1,w_2,\dots|\text{cate})*p(\text{cate})$$

多项式模型+平滑技术+取对数（防止下溢）：

$$p(\text{word}|\text{cate})=(\text{类 cate 下单词 word 出现的次数}+1)/(\text{类 cate 下单词总数}+\text{训练数据中不重复的单词总数})$$

$$p(\text{cate})=\text{类 cate 下单词总数}/\text{训练数据中单词总数}$$

$$p(\text{cate}|\text{doc})=\sum_{i=1}^n \log(p(w_i|\text{cate}))+\log(p(\text{cate}))$$

```

42 def calCateProb(k,testNewsWords,cateWordCount,cateWordNum,totalNum,vocabNum):
43     prob=0
44     wordNumInCate=cateWordNum[k] #新闻类k中单词总数
45     for i in range(len(testNewsWords)):
46         key=k+'_'+testNewsWords[i]
47         if key in cateWordCount:
48             wordCountInCate=cateWordCount[key]
49         else:
50             wordCountInCate=0.0
51         xcProb=np.log((wordCountInCate+1)/(wordNumInCate+vocabNum))
52         prob=prob+xcProb
53     res=prob+np.log(wordNumInCate)-np.log(totalNum)
54     return res

```

3. 函数 classifyNB

朴素贝叶斯分类器通过比较 $p(\text{cate}_i|\text{doc})$, $i=1,2,\dots,20$ 对每个测试文档进行分类，将分类结果记录在特定文件中。

```

56 #朴素贝叶斯对测试文档进行分类
57 def classifyNB(trainDir,testDir,resultCateFile):
58     fw=open(resultCateFile,'w')
59     #训练分类器
60     cateWordCount,cateWordNum,vocabNum=trainNB(trainDir)
61     #得到训练数据单词总数
62     totalNum=sum(cateWordNum.values())
63     #对测试文档做分类
64     testClassList=os.listdir(testDir)
65     for i in range(len(testClassList)):
66         testClassDir=testDir+'/'+testClassList[i]
67         testNewsList=os.listdir(testClassDir)

```

```

68     for j in range(len(testNewsList)):
69         testNewsWords=[] #测试文档的单词列表
70         testNewsDir=testClassDir+'/'+testNewsList[j]
71         lines=open(testNewsDir,'rb').readlines()
72         for line in lines:
73             line=line.decode('utf-8','ignore')
74             word=line.strip('\n')
75             testNewsWords.append(word)
76         maxP=0.0
77         trainClassList=os.listdir(trainDir)
78         for k in range(len(trainClassList)):
79             p=calCateProb(trainClassList[k],testNewsWords,cateWordCount, \
80                           cateWordNum,totalNum,vocabNum)
81             if k==0:
82                 maxP=p
83                 bestCate=trainClassList[k]
84                 continue
85             if p>maxP:
86                 maxP=p
87                 bestCate=trainClassList[k]
88             fw.write('%s %s\n' % (testNewsList[j]+'_'+testClassList[i], \
89                                   bestCate))
90     fw.close()

```

4. 函数 errorRate

依据标注测试样本文件与朴素贝叶斯分类结果文件，计算机朴素贝叶斯分类器的错误率。

```

93 def errorRate(rightCateFile,resultCateFile):
94     rightCateDict={}
95     resultCateDict={}
96     errorCount=0.0
97     for line in open(rightCateFile,'rb').readlines():
98         line=line.decode('utf-8','ignore')
99         (newsID,cate)=line.strip('\n').split()
100        rightCateDict[newsID]=cate
101    for line in open(resultCateFile,'rb').readlines():
102        line=line.decode('utf-8','ignore')
103        (newsID,cate)=line.strip('\n').split()
104        resultCateDict[newsID]=cate
105    for key in rightCateDict.keys():
106        #输出分类结果
107        print('新闻ID: '+key)
108        print('朴素贝叶斯分类: '+resultCateDict[key])
109        print('新闻真实所属类: '+rightCateDict[key])
110        if rightCateDict[key]!=resultCateDict[key]:
111            errorCount+=1.0
112    errorRate=errorCount/len(rightCateDict)
113    print('error rate: %f' % (errorRate))
114    return errorRate

```

(四) 5 折交叉验证

将数据分为五折，选择其中一折作为测试数据，其余作为训练数据计算错误率，取五次错误率的平均值，作为最终朴素贝叶斯分类器的错误率。包括以下几个过程：

1. 生成五次实验的测试数据、训练数据、标注文件；
2. 朴素贝斯分类器对五次实验的测试数据进行分类；
3. 计算并记录五次实验的错误率；
4. 绘制条形图：可视化每次实验的错误率；
5. 计算五次实验的平均错误率；

五、实验结果

（一）随机划分数据（20% test, 80% train）

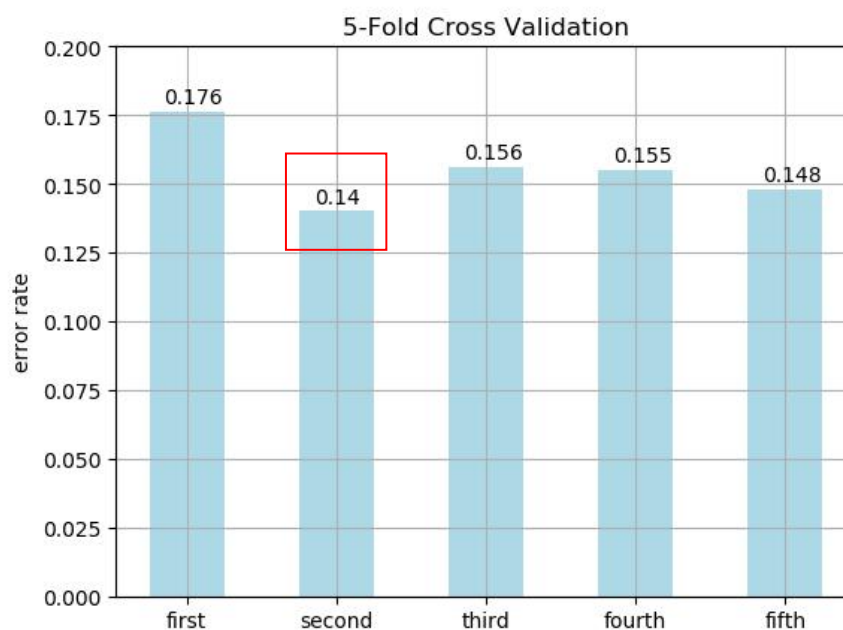
```

新闻ID: 83528_talk.religion.misc
朴素贝叶斯分类: talk.religion.misc
新闻真实所属类: talk.religion.misc
新闻ID: 83529_talk.religion.misc
朴素贝叶斯分类: soc.religion.christian
新闻真实所属类: talk.religion.misc
新闻ID: 83535_talk.religion.misc
朴素贝叶斯分类: talk.religion.misc
新闻真实所属类: talk.religion.misc
新闻ID: 83544_talk.religion.misc
朴素贝叶斯分类: talk.religion.misc
新闻真实所属类: talk.religion.misc
新闻ID: 83547_talk.religion.misc
朴素贝叶斯分类: soc.religion.christian
新闻真实所属类: talk.religion.misc
新闻ID: 83558_talk.religion.misc
朴素贝叶斯分类: talk.politics.mideast
新闻真实所属类: talk.religion.misc
error rate: 0.175504
Naive Bayes Finished!

```

（二）五折交叉验证

1. 条形图可视化五次实验的错误率



2. 输出五次实验的错误率和平均错误率

```
errorRate0 = 0.175504  
errorRate1 = 0.140239  
errorRate2 = 0.156358  
errorRate3 = 0.155113  
errorRate4 = 0.147912  
averageErrorRate = 0.155025  
5-Fold Cross Validation Finished!
```


Homework3: Clustering with sklearn

完成时间：2018 年 12 月 08 日

一、实验目的

1. 理解各种聚类算法的原理。
2. 学会调用 sklearn 模块中的各种聚类函数和评估函数。

二、实验环境

python3.7

三、实验内容

1. 测试 sklearn 中以下聚类算法在 Tweets 数据集上的聚类效果。

Method name	Parameters	Scalability	Usecase	Geometry (metric used)
K-Means	number of clusters	Very large <code>n_samples</code> , medium <code>n_clusters</code> with MiniBatch code	General-purpose, even cluster size, flat geometry, not too many clusters	Distances between points
Affinity propagation	damping, sample preference	Not scalable with <code>n_samples</code>	Many clusters, uneven cluster size, non-flat geometry	Graph distance (e.g. nearest-neighbor graph)
Mean-shift	bandwidth	Not scalable with <code>n_samples</code>	Many clusters, uneven cluster size, non-flat geometry	Distances between points
Spectral clustering	number of clusters	Medium <code>n_samples</code> , small <code>n_clusters</code>	Few clusters, even cluster size, non-flat geometry	Graph distance (e.g. nearest-neighbor graph)
Ward hierarchical clustering	number of clusters	Large <code>n_samples</code> and <code>n_clusters</code>	Many clusters, possibly connectivity constraints	Distances between points
Agglomerative clustering	number of clusters, linkage type, distance	Large <code>n_samples</code> and <code>n_clusters</code>	Many clusters, possibly connectivity constraints, non Euclidean distances	Any pairwise distance
DBSCAN	neighborhood size	Very large <code>n_samples</code> , medium <code>n_clusters</code>	Non-flat geometry, uneven cluster sizes	Distances between nearest points
Gaussian mixtures	many	Not scalable	Flat geometry, good for density estimation	Mahalanobis distances to centers

2. 使用 NMI(Normalized Mutual Information)作为评价指标,对各种聚类算法进行比较。

四、实验过程

(一) 数据处理

定义一个数据处理函数,读取 json 文件,返回 sklearn 聚类函数及评估函数可直接调用的数据形式,包括以下几个步骤:

1. 读取 json 数据文件，得到文本列表['text1','text2','text3',...]和标签数组[label1 label2 label3 ...]。
2. 调用 sklearn 模块中的 TfidfVectorizer 函数从文本 list 中提取特征，获取 tweets 文本的表示向量，得到一个[n_samples,n_features]维度的稀疏矩阵，每一行对应一个文本表示向量。

TfidfVectorizer 函数实参

```
26 #文本向量化
27 #Extracting features from the dataset, using a sparse vectorizer
28 #Vectorizer results are normalized
29 vectorizer=TfidfVectorizer(max_df=0.5, #过滤掉df>(0.5*doc_num)的单词
30                             max_features=3000, #构建词汇表仅考虑max_features
31                             min_df=2, #过滤掉df<2的单词
32                             stop_words='english', #过滤掉英文停用词
33                             use_idf=True) #启动idf重新计算权重
34 X=vectorizer.fit_transform(data) #稀疏矩阵
```

(二) 聚类算法调用与评估

定义函数，分别调用 sklearn 中的八种聚类算法对 Tweets 数据进行聚类并返回聚类标签。利用 sklearn 自带的 NMI 评估函数对每个聚类算法聚类效果进行单独评价，得到评估分数。

1. K-Means

```
13 #定义函数实现KMeans聚类算法
14 # @param X 数据
15 # @param k 簇的数目
16 # @param minibatch 布尔型(True:MiniBatchKMeans False:KMeans)
17 # @return y_pred,km
18 def KMeansAlgorithm(X, k, minibatch):
19     if minibatch:
20         km=MiniBatchKMeans(n_clusters=k, #形成的簇数以及生成的中心数
21                             init='k-means++', #用智能的方式选择初始聚类中心以加速收敛
22                             n_init=3, #随机初始化的次数
23                             batch_size=100, #Size of the mini batches
24                             )
25     else:
26         km=KMeans(n_clusters=k,
27                    init='k-means++',
28                    max_iter=300, #一次单独运行的最大迭代次数
29                    n_init=10, #使用不同的聚类中心进行初始化的次数
30                    )
31     km.fit(X)
32     y_pred=km.labels_
33     return y_pred,km
```

2. Affinity Propagation

```

11 #定义函数实现AffinityPropagation聚类算法
12 # @param X 样本特征数据
13 # @return y_pred
14 def AffinityPropagationAlgorithm(X):
15     '''
16     1. 参数damping: float, optional, default: 0.5
17         Damping factor (between 0.5 and 1) is the extent to which the current
18         value is maintained relative to incoming values (weighted 1 - damping).
19         This in order to avoid numerical oscillations when updating these values
20         (messages).
21     2. 参数preference: array-like, shape (n_samples,) or float, optional
22         The number of exemplars, ie of clusters, is influenced by the input
23         preferences value.
24         If the preferences are not passed as arguments, they will be set to the
25         median of the input similarities.
26     '''
27     ap=AffinityPropagation(damping=0.5, preference=None)
28     ap.fit(X)
29     y_pred=ap.labels_
30     return y_pred

```

3. Mean-shift

```

11 #定义函数执行AffinityPropagation聚类算法
12 # @param X 样本特征数据
13 # @return y_pred
14 def MeanShiftAlgorithm(X):
15     '''
16     参数bandwidth: float, optional
17     Bandwidth used in the RBF kernel.
18     '''
19     ms=MeanShift(bandwidth=0.9)
20     ms.fit(X)
21     y_pred=ms.labels_
22     return y_pred

```

4. Spectral Clustering

```

12 #定义函数执行SpectralClustering聚类算法
13 # @param X 样本特征数据
14 # @param k 簇的数目
15 # @return y_pred
16 def SpectralClusteringAlgorithm(X,k):
17     #参数n_clusters: integer, optional
18     #The dimension of the projection subspace.
19     sc=SpectralClustering(n_clusters=k)
20     sc.fit(X)
21     y_pred=sc.labels_
22     return y_pred

```

5. Ward Hierarchical Clustering

```

12 #定义函数执行WardHierarchicalClustering聚类算法
13 # @param X 样本特征数据
14 # @param k 簇的数目

```

```

15 # @return y_pred
16 def WardHierarchicalClusteringAlgorithm(X,k):
17     #参数linkage: optional (default="ward")
18     #ward minimizes the variance of the clusters being merged.
19     #参数n_clusters: int, default=2
20     #The number of clusters to find.
21     whc=AgglomerativeClustering(linkage='ward',
22                                 n_clusters=k)
23     whc.fit(X)
24     y_pred=whc.labels_
25     return y_pred

```

6. Agglomerative Clustering

```

12 #定义函数执行AgglomerativeClustering聚类算法
13 # @param X 样本特征数据
14 # @param k 簇的数目
15 # @return y_pred
16 def AgglomerativeClusteringAlgorithm(X,k):
17     '''
18     1. 参数linkage: {"ward", "complete", "average", "single"}, optional
19         average uses the average of the distances of each observation of the
20         two sets.
21     2. 参数n_clusters: int, default=2
22         The number of clusters to find.
23     '''
24     ac=AgglomerativeClustering(linkage='average',
25                               n_clusters=k)
26     ac.fit(X)
27     y_pred=ac.labels_
28     return y_pred

```

7. DBSCAN

```

11 #定义函数执行DBSCAN聚类算法
12 # @param X 样本特征数据
13 # @return y_pred
14 def DBSCANAlgorithm(X):
15     '''
16     1. 参数eps: float, optional
17         The maximum distance between two samples for them to be considered as in
18         the same neighborhood.
19     2. 参数min_samples: int, optional
20         The number of samples (or total weight) in a neighborhood for a point to
21         be considered as a core point.
22         This includes the point itself.
23     '''
24     ds=DBSCAN(eps=0.5,
25              min_samples=1,
26              metric='cosine')
27     ds.fit(X)
28     y_pred=ds.labels_
29     return y_pred

```

8. Gaussian Mixture


```

12 #定义函数执行GaussianMixture聚类算法
13 # @param X 样本特征数据
14 # @param k 簇的数目
15 # @return y_pred
16 def GaussianMixtureAlgorithm(X,k):
17     #参数n_components: int, defaults to 1
18     #The number of mixture components.
19     gm=GaussianMixture(n_components=k)
20     gm.fit(X)
21     y_pred=gm.predict(X)
22     return y_pred

```

（三）比较 8 种聚类算法

主文件 main.py 循环调用 8 种聚类算法，得到在 Tweets 数据集上的聚类标签，对 NMI 评估分数进行直观的比较。主要包括以下过程：

1. 调用数据处理函数，得到 sklearn 函数可以直接调用的数据格式。
2. 定义聚类算法列表，循环调用 8 种聚类算法对 Tweets 文本进行聚类，得到聚类标签，将 NMI 评估分数保存到评估列表。
3. 绘制条形图，直观比较 8 种聚类算法的效果。
4. 排序并输出聚类算法及其对应的评估分数。

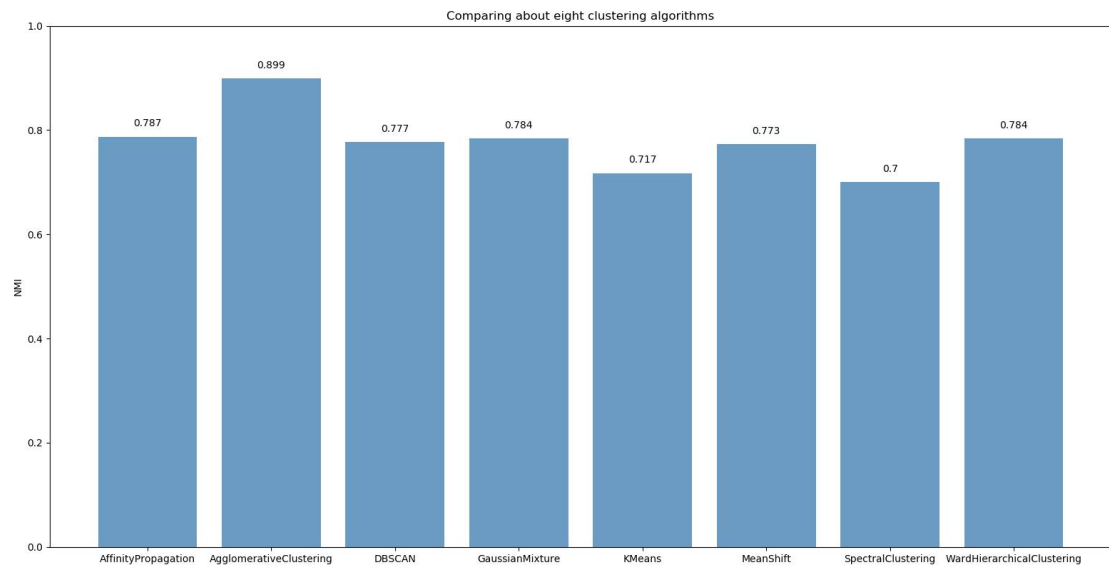
五、实验结果

（一）单独调用和评估聚类算法

聚类算法	NMI
K-Means	0.708
Affinity Propagation	0.787
Mean-shift	0.773
Spectral Clustering	0.708
Ward Hierarchical Clustering	0.784
Agglomerative Clustering	0.899
DBSCAN	0.777
Gaussian Mixture	0.789

（二）循环调用和比较聚类算法

1. 绘制条形图，直观比较 8 种聚类算法的效果。其中，横坐标代表 8 种聚类算法，纵坐标代表 NMI 评估分数。



2. 依据 NMI 评估分数对算法进行排序

```
Clustering algorithm and corresponding NMI score:  
1. AgglomerativeClustering: 0.899  
2. AffinityPropagation: 0.787  
3. GaussianMixture: 0.784  
4. WardHierarchicalClustering: 0.784  
5. DBSCAN: 0.777  
6. MeanShift: 0.773  
7. KMeans: 0.717  
8. SpectralClustering: 0.7
```