

**实验报告**



**文本数据的分类与分析**

**——人工智能原理课程实验**

**学 号2018211290，2018211507，2018211251**

**姓 名： 崔思颖、陈姝仪、黄芝遵**

**学 院： 计算机学院**

**2020年12月31日**

目录

[实验目的 3](#_Toc60507844)

[实验类型 3](#_Toc60507845)

[实验要求 3](#_Toc60507846)

[实验内容 3](#_Toc60507847)

[实验步骤 4](#_Toc60507848)

[数据收集 4](#_Toc60507849)

[数据处理 5](#_Toc60507850)

[朴素贝叶斯分类器 15](#_Toc60507851)

[SVM分类器 19](#_Toc60507852)

[实验结果 26](#_Toc60507853)

[朴素贝叶斯分类器 26](#_Toc60507854)

[SVM分类器 27](#_Toc60507855)

[实验总结 29](#_Toc60507856)

# 实验目的

1.   掌握数据预处理的方法，对训练集数据进行预处理；

2.   掌握文本建模的方法，对语料库的文档进行建模；

3.   掌握分类算法的原理，基于有监督的机器学习方法，训练文本分类器；

4.   利用学习的文本分类器，对未知文本进行分类判别；

5.   掌握评价分类器性能的评估方法。

# 实验类型

数据挖掘算法的设计与编程实现。

# 实验要求

1.       文本类别数：>=10类；

2.       训练集文档数：>=50000篇；每类平均5000篇。

3.       测试集文档数：>=50000篇；每类平均5000篇。

# 实验内容

利用分类算法实现对文本的数据挖掘，主要包括：

1.       语料库的构建，主要包括利用爬虫收集Web文档等；

2.       语料库的数据预处理，包括文档建模，如去噪，分词，建立数据字典，使用词袋模型或主题模型表达文档等；

3.       选择分类算法（朴素贝叶斯（必做）、SVM/其他等），训练文本分类器，理解所选的分类算法的建模原理、实现过程和相关参数的含义；

4.       对测试集的文本进行分类

5.       对测试集的分类结果利用正确率和召回率进行分析评价：计算每类正确率、召回率，计算总体正确率和召回率。

# 实验步骤

## 数据收集

本次实验的文本数据来自github上的“大规模中文自然语言处理语料 Large Scale Chinese Corpus for NLP”项目：

<https://github.com/brightmart/nlp_chinese_corpus>

我们使用了其中的“百科类问答json版”语料库，它含有150万个预先过滤过的、高质量问题和答案，每个问题属于一个类别。总共有492个类别，其中频率达到或超过10次的类别有434个。原始数据文件概览如下：



其中，qid是问题编号，category是问题类型，title是问题标题，desc是问题描述，可以为空或与标题内容一致。

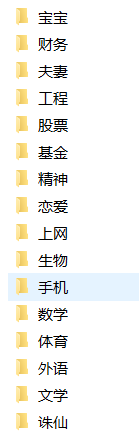
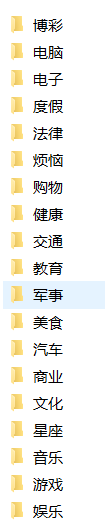
## 数据处理

#### 1.json文件转txt文件

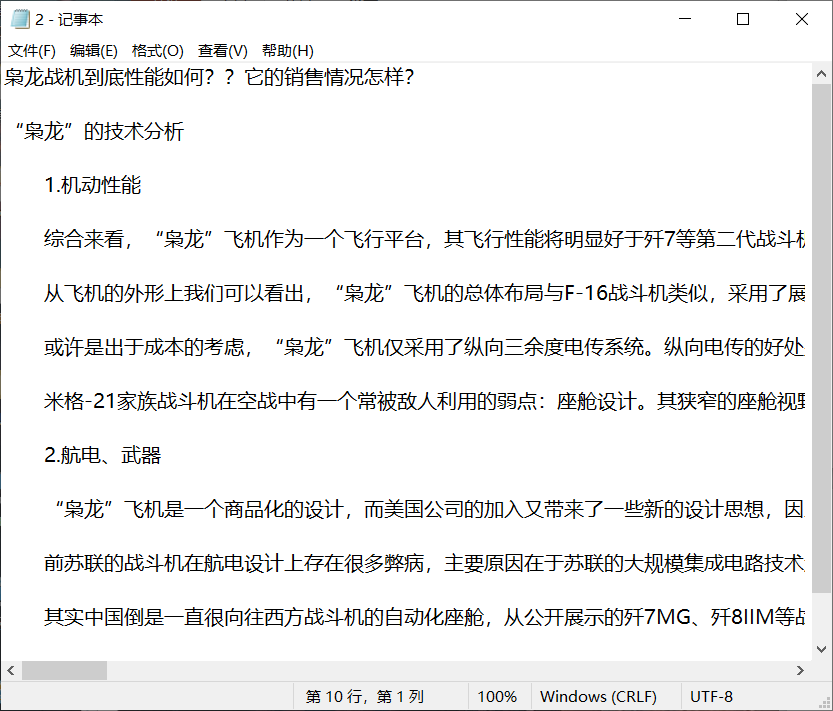
经过对json文件初步观察后，我们筛选出了希望用来训练与测试的部分类别。我们利用json库把下载的json文件以字典形式读入，并将每条问答数据转化为字符串，写入对应类别文件夹中的txt下。每类文件夹中的txt以从1到10000的标号命名。

1. **import** json
2. **import** os
3. '''''返回类别修正后的类别名'''
4. **def** rename(s):
5. '''''对类别名进行切片，将多个大类拆分为小类，或将多个小类合并为大类'''
6. type0=s[0:2]
7. type1=s[5:7]
8. type2=s[3:5]
9. type3=s[8:10]
10. type4=s[6:8]
11. type5=s[11:13]
12. type6=s[10:12]
13. **if** type5 **in** {'生物','数学','工程'}:
14. s=type5
15. **elif** type6=='上网':
16. s=type6
17. **elif** type3=='诛仙':
18. s=type3
19. **elif** type4 **in** {'财务','股票','基金','文学','外语'}:
20. s=type4
21. **elif** type1 =='手机':#{'军事','法律'}:
22. s=type1
23. **elif** type2 **in** {'精神','恋爱','夫妻','财务','宝宝'}:#{'博彩','度假','星座','音乐','购物','交通','美食'} :
24. s=type2
25. **elif** type0=='体育':#{'电脑','电子','烦恼','汽车','商业','文化','游戏','教育','健康'}:
26. s=type0
27. **else**:
28. **return** ""
29. **return** s
31. fr=open("baike\_qa\_train.json","r",encoding='utf-8')
32. data=[]
33. typenum={'体育':0,'精神':0,'恋爱':0,'夫妻':0,'财务':0,'宝宝':0,'手机':0,'财务':0,'股票':0,'基金':0,'文学':0,'外语':0,'诛仙':0,'上网':0,'生物':0,'数学':0,'工程':0}#设置词典标识每个类的文本数量
34. #{'娱乐':0,'健康':0,'美食':0,'教育':0,'军事':0,'法律':0,'博彩':0,'度假':0,'星座':0,'音乐':0,'购物':0,'交通':0,'电脑':0,'电子':0,'烦恼':0,'汽车':0,'商业':0,'文化':0,'游戏':0}
35. '''''将json文件转化成不同类别文件夹下的txt'''
36. **for** line **in** fr.readlines():
37. '''''以字典形式读取json文件'''
38. ls=json.loads(line)
39. data.append(ls["title"])
40. data.append(ls["desc"])
41. data.append(ls["answer"])
42. s=""
43. '''''转化为字符串'''
44. **for** each **in** data:
45. s=s+each+'\n'
46. '''''过滤长度小于60字的文本'''
47. **if** s.\_\_len\_\_()>60:
48. '''''转化为对应的类别'''
49. type=rename(ls["category"])
50. '''''是所选择的类别，并且文档数未达到10000篇'''
51. **if** type!="" **and** typenum[type]<=9999:
52. typenum[type] += 1
53. '''''新建文件夹'''
54. **if** **not** os.path.exists('QAsamples2/' + type):
55. os.makedirs('QAsamples2/' + type)
56. '''''写txt'''
57. file = 'QAsamples2/' + type+'/'+ str(typenum[type]) + '.txt'
58. fw=open(file,'w',encoding='utf-8')
59. **print**(type+'/'+ str(typenum[type]))
60. fw.write(s)
61. fw.close()
62. data.clear()

处理后的文件目录如图：



处理过的示例文本如图：



**2.分词并去停用词**

将所有类别的txt依次进行分词、去停用词、去英文词、取名词和惯用语处理，并将处理过的txt整理入新的目录中。

1. **import** os
2. **import** jieba.posseg as pseg

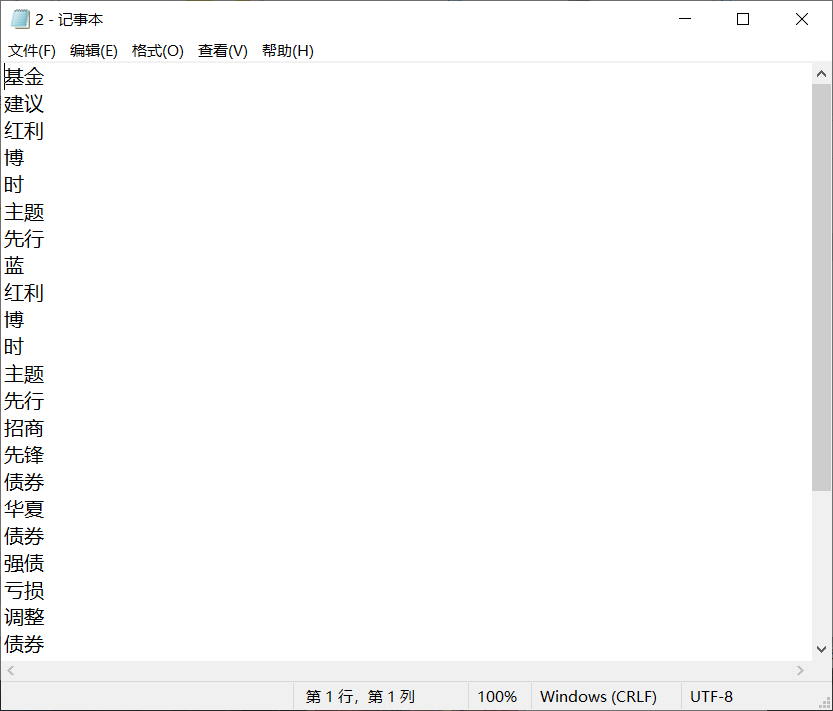
5. '''''读取停用词文件'''
6. f\_stop=open('stop\_words.txt','rb')
7. stopwords=f\_stop.readlines()

10. '''''判断是否为中文'''
11. **def** is\_Chinese(word):
12. **for** ch **in** word:
13. **if** '\u4e00' <= ch <= '\u9fff':
14. **return** True
15. **return** False

18. **def** wordSplit(path):
19. '''''生成指定文件夹下文件列表'''
20. **for** file **in** os.listdir(path):
21. type=file
22. file\_path = os.path.join(path, file)
23. **if** os.path.isdir(file\_path):
24. **for** each\_file **in** os.listdir(file\_path):
25. f = open(file\_path+'/'+each\_file, 'rb')
26. p = f.read().decode('utf-8')
27. '''''去空格'''
28. p = p.replace(" ","")
29. **print**(type+'\t'+each\_file)
30. '''''分词'''
31. words = pseg.cut(p)
32. '''''创建结果目录文件夹与txt'''
33. **if** **not** os.path.exists('result7/' + type):
34. os.makedirs('result7/' + type)
35. r = open('result7/' + type + '/'+each\_file, 'w',encoding='utf-8')
37. '''''过滤停用词与非中文词，取名词和惯用语'''
38. **for** w **in** words:
39. **if** ('n' **or** 'l') **in** w.flag **and** **not** w.word **in** stopwords **and** is\_Chinese(w.word):
40. r.write(w.word+'\n')
41. r.close()

44. '''''获取所有文件夹名称'''
45. wordSplit('E:/Documents/PPT/人工智能原理/文本实验/result6')

处理过的示例文本如图：



**3.划分测试集与训练集**

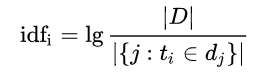
在每个类的文件夹下创建train、test文件夹，并将每个类前5000个文本放入/train中，后五千放入/test中。

1. **import** os
2. **import** shutil
3. path='E:/Documents/PPT/人工智能原理/文本实验/result7'

6. '''''生成每个类的文件目录'''
7. **for** file **in** os.listdir(path):
8. type=file
9. file\_path = os.path.join(path, file)
10. **if** os.path.isdir(file\_path):
11. i = 1
13. '''''创建train、test文件夹'''
14. **if** **not** os.path.exists(file\_path + '/train'):
15. os.makedirs(file\_path + '/train')
16. **if** **not** os.path.exists(file\_path + '/test'):
17. os.makedirs(file\_path + '/test')
18. '''''将txt前5000移动到train文件夹中，后50000移动到test文件夹中'''
19. **for** each\_file **in** os.listdir(file\_path):
20. ori\_path = file\_path + '/' + each\_file
21. des\_path = ''
22. **if** **not** os.path.isdir(ori\_path):
23. **if** i<=5000:
24. des\_path=file\_path + '/train/'+str(i)+'.txt'
25. **else**:
26. des\_path = file\_path + '/test/' + str(i-5000)+'.txt'
27. **print**(type + '\t' + each\_file)
28. shutil.move(ori\_path,des\_path)
29. i+=1

**4.计算idf**

IDF为逆向文件频率，公式如下：



* |D|：语料库中的文件总数
* j：包含词语t的文件数目

其主要思想是：如果包含词条t的文档越少，IDF越大，则说明词条t具有很好的类别区分能力。如果某一类文档C中包含词条t的文档数为m，而其它类包含t的文档总数为k，显然所有包含t的文档数n=m+k，当m大的时候，n也大，按照IDF公式得到的IDF的值会小，就说明该词条t类别区分能力不强。

此处|D|作加一处理，防止某个词在所有文档中都出现，导致idf为0。

1. **import** os
2. **import** re
3. **import** operator
4. **import** math

7. '''''path这里改成clean\_data文件夹的位置'''
8. path='E:/Documents/PPT/人工智能原理/文本实验/result3'
9. tf={}
10. idf={}
11. '''''索引每一类路径'''
12. **for** file **in** os.listdir(path):
13. type=file
14. file\_path = path+'/'+file
15. tf\_sum = 0
16. tf={}
17. **if** os.path.isdir(file\_path):
18. i = 1
19. sp=[]
20. file\_path+='/train'
21. '''''进入每一类的train文件夹'''
22. **for** each\_file **in** os.listdir(file\_path):
23. each\_path=file\_path+'/'+each\_file
24. f=open(each\_path,'r', encoding='utf-8')
25. **print**(type+'\t'+each\_file)
26. rf=f.readlines()
27. '''''将所有词加入列表sp中'''
28. **for** each **in** rf:
29. sp+=each.split()
30. f.close()
31. **for** each **in** set(sp):
32. **if** each **in** idf.keys():
33. idf[each] += 1
34. **else**:
35. idf[each] = 1
36. '''''输出'''
37. f=open('idf.txt','w')
38. **for** (word, count) **in** idf.items():
39. s = '%s %7lf' % (word, math.log(50001/count,10))#50000为总文档数，+1防止log1导致idf=0
40. f.write(s + '\n')
41. f.close()

**5.生成词袋**

计算文档中读取到的每一个词的TF，TF表示词条在文档d中出现的频率，公式如下：

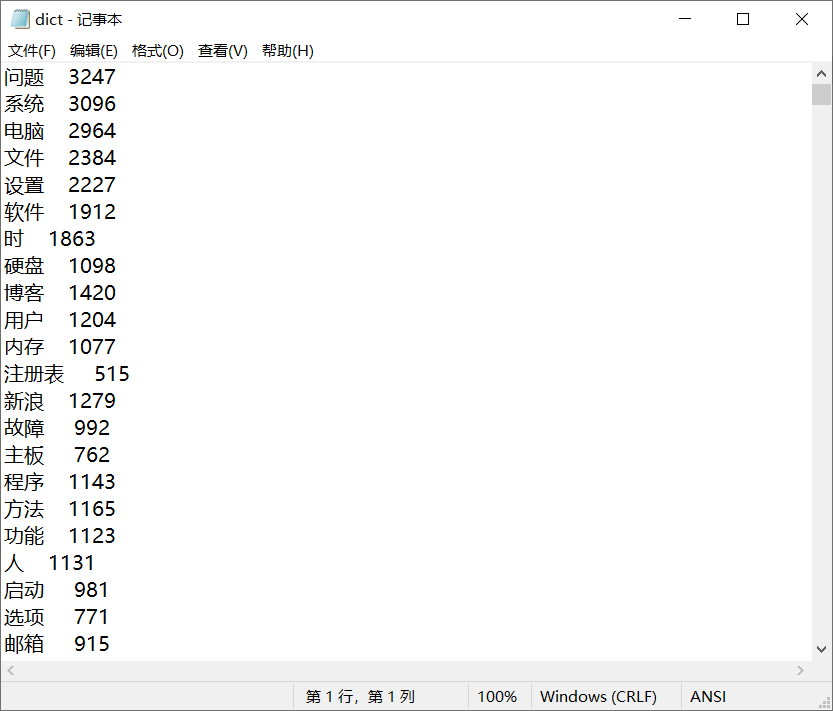


每个类取TF\*IDF前两千的词生成词典，将单词和词频写入词典中。

1. **import** os

4. '''''此处改成clean\_data的位置'''
5. path='E:/Documents/PPT/人工智能原理/文本实验/result3'
6. idf={}
7. '''''读取idf值'''
8. f=open('idf.txt','r')
9. p = f.readlines()
10. **for** each **in** p:
11. idf[each.split()[0]]=float(each.split()[1])
12. f.close()
13. '''''索引每一类路径'''
14. **for** file **in** os.listdir(path):
15. type=file
16. file\_path = path+'/'+file
17. tf={}
18. **if** os.path.isdir(file\_path):
19. i = 1
20. sp=[]
21. f\_dict = open(file\_path + '/dict.txt', 'w')
22. file\_path+='/train'
23. '''''进入每一类的train文件夹'''
24. **for** each\_file **in** os.listdir(file\_path):
25. each\_path=file\_path+'/'+each\_file
26. f=open(each\_path,'r', encoding='utf-8')
27. **print**(type+'\t'+each\_file)
28. rf=f.read()
29. '''''将所有词加入列表sp中'''
30. sp = [one **for** one **in** rf.split()]
31. f.close()
32. **for** each **in** sp:
33. **if** each **in** tf.keys():
34. tf[each] += 1
35. **else**:
36. tf[each] = 1
37. '''''找出前2000,根据tfidf降维'''
38. tf=dict((sorted(tf.items(), key = **lambda** kv:((kv[1]\*idf[kv[0]]), kv[0]),reverse=True))[0:1999])
39. '''''写单类dict，输出单词 词频'''
40. **for** (word,fre) **in** tf.items():
41. s='%s %7d'%(word,fre)
42. f\_dict.write(s+'\n')
43. f\_dict.close()

生成词典如下所示：



再将每个类的词典中的词合并，生成总词典

1. **import** os
3. '''''path这里改成data文件夹的位置'''
4. path='E:/Documents/PPT/人工智能原理/文本实验/result3'
5. total\_dict=set()
6. '''''索引每一类路径'''
7. **for** file **in** os.listdir(path):
8. type=file
9. file\_path = path+'/'+file
10. **if** os.path.isdir(file\_path):
11. i = 1
12. sp=[]
13. f=open(file\_path+'/dict.txt','r')
14. **print**(type+'\tdict.txt')
15. p = f.readlines()
16. **for** each **in** p:
17. total\_dict.add(each.split()[0])
18. f.close()
19. '''''将每一类的dict合并为total\_dict'''
20. f=open(path+'/total\_dict.txt','w')
21. **for** word **in** total\_dict:
22. f.write(word + '\n')
23. f.close()

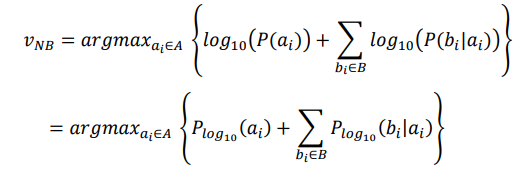
## 朴素贝叶斯分类器

* **原理**

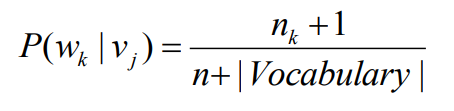
朴素贝叶斯分类的公式如下：



本次实验，我们使用TF-IDF进行文本特征提取。并且为了防止乘法造成数值过小，我们将公式改进如下：



采纳m-估计方法，即有统一的先验概率并且m等于词汇表的大小，因此



* **代码**

1. **import** os
2. **import** math
3. **import** numpy as np
4. ROOTPATH='C:\\Users\92994\Desktop\sy2\clean\_data2.1'
5. **global** categories#=  ['健康', '商业', '娱乐', '教育', '文化', '游戏', '烦恼', '生活', '电脑', '社会']#注意不可改变顺序！！！！
6. **global** cateCount
7. **global** CATENUM#类别总数
8. **global** VOCABULARYNUM
9. **global** bigDic
10. **global** gmatrix
11. **global** idfBook;
12. **def** form\_big\_dic():
13. # 构造所有类别的词典
14. **global** bigDic,categories
15. bigDic = dict()
16. contents = os.listdir(ROOTPATH)  # 电脑、烦恼、健康。。。。
17. categories=[]
18. **for** each **in** contents:  # each是电脑、烦恼、健康等某一类
19. **if** os.path.isdir(ROOTPATH + '\\' + each):  # 判断是文件夹，打开
20. categories.append(each)
21. bigDic[each] = read\_file(ROOTPATH + '\\' + each + '\\' + 'dict.txt')
22. # print(bigDic)
23. # print(len(bigDic['电脑']))

26. #读一个字典向量文件，返回一个字典
27. **def** read\_file(filepath):
28. with open(filepath) as fp:
29. content=fp.read();
30. book=content.split('\n')
31. d=dict();
32. **for** each **in** book:
33. **if** each:#each不为空
34. temp=each.split()
35. # print(temp)
36. **if** len(temp)==2:
37. d[temp[0]]=float(temp[1])
38. **else**:
39. d[temp[0]] =0
40. **return** d

43. **def** P(word,vj):
44. # P(wk|vj)=(nk+1) / (n+|Vocabulary|)
45. nk=bigDic[vj].get(word,0)#单词wk出现在Textj中的次数，若没有出现，则为0
46. n=cateCount[vj]
47. ans=(nk+1) / (n+VOCABULARYNUM)
48. **return** ans
49. # V为所有类的向量，text为待分类文本string，返回值为分类结果
50. **def** Vnb(text,V):
51. max=-9999999999999999999999999999999999999
52. tans=0
53. retu=''
54. l=text.split()
55. **for** j **in** V:#对于每一类
56. **for** word **in** l:#对弈一篇文本中的每一个单词
57. idf=idfBook.get(word,math.log(50000,10))
58. # print(idf)
59. tans=tans+math.log(P(word,j)\*idf,10)
60. # print("tans=",tans)
61. **if** tans>max:
62. max=tans
63. retu=j
64. tans=0
65. # print("j=",retu,"max=",max)
66. **return** retu
67. **def** cal\_cateCount(categories):
68. n=0
69. **for** vj **in** categories:#each为健康等类别
70. **for** key **in** bigDic[vj]:
71. # print("key=",key)
72. n = n + bigDic[vj][key]
73. cateCount[vj]=n
74. n=0
75. # 打印混淆矩阵
76. **def** print\_matrix(matrix):
77. **print**('{:>8}'.format(''), end='')
78. **for** label **in** range(len(categories)):
79. **print**('{:>7}'.format(categories[label]), end='')
80. **print**('\n')
81. **for** row **in** range(len(categories)):
82. **print**('{:>8}'.format(categories[row]), end='')
83. **for** col **in** range(len(categories)):
84. **print**('{:>8}'.format(matrix[row][col][0]), end='')
85. **print**('\n')
86. # def print\_matrix(matrix):
87. #     print(categories)
88. #     for i in range(len(categories)):
89. #         print(categories[i],matrix[i])
90. **def** classify\_all\_texts(rootpath,matrix):
91. contents = os.listdir(rootpath)  # 电脑、烦恼、健康。。。。
92. **print**(contents)#注意顺序！！！
93. **for** each **in** contents:  # each是电脑、烦恼、健康等某一类
94. **if** os.path.isdir(rootpath + '\\' + each):  # 判断是文件夹，打开
95. texts = os.listdir(rootpath + '\\' + each + '\\' + 'test')
96. **for** text **in** texts:
97. with open(rootpath + '\\' + each + '\\' + 'test' + '\\' + text, encoding='utf-8') as fp:
98. string = fp.read()
99. vj = Vnb(string, categories)
100. i = categories.index(each)  # 实际值
101. j = categories.index(vj)  # 预测值
102. matrix[i][j][0] += 1
103. # print(matrix)
104. print\_matrix(matrix)
105. **def** cal\_precision\_and\_recall(matrix):
106. precisionList = []
107. recallList = []
108. **for** j **in** range(CATENUM):  # 先对列进行遍历
109. sum = 0
110. **for** i **in** range(CATENUM):
111. sum = sum + matrix[i][j][0]
112. a = matrix[j][j][0]
113. recall = a / 5000
114. precision = a / sum
115. precisionList.append(precision)
116. recallList.append(recall)
117. **print**("类别：", categories[j])
118. **print**("a:", a)
119. **print**("sum:", sum)
120. **print**("precision={} , recall={}".format(precision, recall))
121. total\_precision = np.mean(precisionList)
122. total\_recall = np.mean(recallList)
123. **print**("total\_precision={} , total\_recall={}".format(total\_precision, total\_recall))

126. **if** \_\_name\_\_ == '\_\_main\_\_':
127. # 计算十万篇文本的单词总数
128. book=read\_file(ROOTPATH+'\\'+"total\_dict.txt")
129. VOCABULARYNUM=len(book)
130. # print(book)
131. **print**("VOCABULARY=",VOCABULARYNUM)
132. idfBook=read\_file(ROOTPATH+'\\'+"idf.txt")
133. #构造所有类别的词典
134. form\_big\_dic()
135. #计算每一类的位置总数
136. cateCount=dict()
137. cal\_cateCount(categories)
138. CATENUM=len(cateCount)
139. **print**(cateCount)
140. #测试一篇文章
141. # with open("D:\机器学习数据\sy发的数据\clean\_data \_newdict\电脑\\test\\test20.txt", encoding='utf-8') as fp:
142. #     string = fp.read()
143. #     vj=Vnb(string,categories)
144. #对所有文章进行分类
145. gmatrix = [[[0] **for** j **in** range(CATENUM)] **for** i **in** range(CATENUM)]
146. classify\_all\_texts(ROOTPATH,gmatrix)
147. #计算准确率和召回率
148. cal\_precision\_and\_recall(gmatrix)

## SVM分类器

* **原理**

SVM方法是建立在统计学习理论的VC维理论和结构风险最小原理基础上的，根据有限的样本信息在模型的复杂性和学习能力之间寻求最佳折衷，以期获得最好的推广能力（或称泛化能力）。

* **实现过程**

本次实验使用libsvm工具包和调用python的libsvm库完成svm分类器的设计与实现。

(1) 首先编写代码create\_train\_test\_file.py将训练集和测试集转化为libsvm所支持的数据格式，如下所示：

<label1> <index1>:<value1> <index2>:<value2> ..... <index L>:<valueL>

label即各个类别的标识，为整数；index即词典中各个特征词的序列号；value即特征值。其中特征值采用TFIDF进行构建，其计算公式如下：

tfidf=tf \* idf

tf(w,d) = count(w, d) / size(d)

idf = log(n / docs(w, D))

count(w,d)为词w在文档d中出现的次数，size(d)为文档d总词数，n为文档总数，docs(w,D)为包含词w的文档数。

(2) 接着运行create\_train\_test\_file.py会生成训练集文件train\_tfidf\_File.txt和测试集文件test\_tfidf\_File.txt，由于原始数据的范围可能过大或者过小，我们在libsvm工具包的tools目录下使用libsvm工具包中的svm-scale工具将数据调整到适当范围（0-1），以提高训练和测试的速度，以及预测的精度，得到文件train\_tfidf\_scale.txt和test\_tfidf\_scale.txt。如下图所示：



(3) 然后编写训练代码train.py，调用libsvm库并输入适当的参数进行训练，训练完成后保存训练好的模型。

(4) 最后编写测试代码使用测试集对训练好的模型进行测试，并输出测试结果，包括混淆矩阵和每一类的准确率、召回率、F测度，以及总的准确率、召回率、F测度。

(5) 使用不同的核函数和参数进行训练和测试并记录测试结果，从中选择分类效果最好的核函数和参数，根据实验结果选择了RBF核作为最终模型，参数C=35.0，参数g=0.0079125。

* **代码**

(1) create\_train\_test\_file.py

1. **import** codecs
2. file\_path = r'E:\大三\课件和code\人工智能原理\实验\小组实验资料'  # 存放数据文件夹的父目录
3. label\_list = ['育儿', '博彩', '财务', '法律', '基金', '军事', '汽车', '星座', '音乐', '游戏']

6. # 获取特征向量（此处以文件夹中total\_dict中的词作特征向量）
7. **def** get\_feature\_list():
8. feature\_list = []
9. f = open(file\_path+r'\clean\_data\_test7\total\_dict.txt', 'r')
10. word = f.readlines()
11. f.close()
12. **for** each **in** word:
13. feature\_list.append(each.split()[0])
14. **return** feature\_list

17. # 获取每个文档的分词结果并以字典形式存储，参数为路径（file\_path下每个文档的路径）,返回字典和文档中单词总数
18. **def** get\_each\_word(each\_path):
19. word\_dict = {}
20. word\_num = 0
21. f = open(file\_path + each\_path, 'r', encoding='utf-8')
22. word = f.readlines()  # 读取所有行
23. f.close()
24. word\_list = []
25. **for** each **in** word:
26. word\_list.append(each.split()[0])
27. **for** each **in** word\_list:
28. word\_num = word\_num + 1
29. **if** each **not** **in** word\_dict:
30. word\_dict[each] = 1
31. **else**:
32. word\_dict[each] = word\_dict[each] + 1
33. **return** word\_dict, word\_num

36. # 读取idf存入字典并返回
37. **def** read\_idf():
38. idf\_dict = {}
39. f = codecs.open(file\_path + r'\idf.txt', 'r', encoding='gbk')
40. word = f.readlines()
41. f.close()
42. **for** each **in** word:
43. pos = each.find(' ')  # 根据空格位置分隔中文词和浮点数
44. idf\_dict[each[0:pos]] = float(each[pos+1:])
45. **print**(idf\_dict)
46. **return** idf\_dict

49. # 训练集转化成libsvm要求的数据格式，参数为特征向量和各单词idf值
50. **def** create\_train\_file(feature\_list, idf\_dict):  # 传入特征向量维数
51. featureNUM = len(feature\_list)
52. with open(file\_path+r'\train\_tfidf\_File.txt', 'w') as f:
53. **for** label **in** range(1, 11):  # 类别
54. **for** file\_num **in** range(1, 5001):  # 每类文档数
55. # 获取当前文档的分词后的字典和该文档单词总数
56. word\_dict, word\_num = get\_each\_word('\\clean\_data\_test7\\' + label\_list[label-1] + '\\train\\' + str(file\_num) + '.txt')
57. f.write(str(label) + " ")  # 写入类别
58. **for** feature\_num **in** range(1, featureNUM+1):  # 每个特征值（特征向量维数）
59. feature\_word = feature\_list[feature\_num - 1]   # 特征值对应的单词
60. **if** feature\_word **in** word\_dict:  # 当前特征值下该文档存在该特征值
61. tf = word\_dict[feature\_word] / word\_num
62. tfidf = tf \* idf\_dict[feature\_word]
63. **if** feature\_num != featureNUM:
64. f.write(str(feature\_num) + ":" + str(tfidf)+" ")
65. **else**:
66. f.write(str(feature\_num) + ":" + str(tfidf))
67. f.write("\n")  # 每个文档一行

70. # 测试集转化成libsvm要求的数据格式，参数为特征向量和各单词idf值
71. **def** create\_test\_file(feature\_list, idf\_dict):  # 传入特征向量维数
72. featureNUM = len(feature\_list)
73. with open(file\_path+r'\test\_tfidf\_File.txt', 'w') as f:
74. **for** label **in** range(1, 11):  # 类别
75. **for** file\_num **in** range(1, 5001):  # 每类文档数
76. # 获取当前文档的分词后的字典和该文档单词总数，此处填写file\_path下路径即可
77. word\_dict, word\_num = get\_each\_word('\\clean\_data\_test7\\' + label\_list[label-1] + '\\test\\' + str(file\_num) + '.txt')
78. f.write(str(label) + " ")  # 写入类别
79. **for** feature\_num **in** range(1, featureNUM+1):  # 每个特征值（特征向量维数）
80. feature\_word = feature\_list[feature\_num - 1]   # 特征值对应的单词
81. **if** feature\_word **in** word\_dict:  # 当前特征值下该文档存在该特征值
82. tf = word\_dict[feature\_word] / word\_num
83. tfidf = tf \* idf\_dict[feature\_word]
84. **if** feature\_num != featureNUM:
85. f.write(str(feature\_num) + ":" + str(tfidf)+" ")
86. **else**:
87. f.write(str(feature\_num) + ":" + str(tfidf))
88. f.write("\n")  # 每个文档一行

91. **def** main():
92. feature\_list = get\_feature\_list()  # 特征向量
93. idf\_dict = read\_idf()  # 各单词idf值
94. create\_train\_file(feature\_list, idf\_dict)  # 测试集格式转化
95. create\_test\_file(feature\_list, idf\_dict)  # 训练集格式转化

98. **if** \_\_name\_\_ == '\_\_main\_\_':
99. main()

(2) train.py

1. **from** libsvm.commonutil **import** \*
2. **from** libsvm.svm **import** \*
3. **from** libsvm.svmutil **import** \*
4. file\_path = r'E:\大三\课件和code\人工智能原理\实验\小组实验资料'  # 存放数据文件夹的父目录
6. **def** main():
7. y, x = svm\_read\_problem(file\_path+r'\train\_tfidf\_scale.txt')
8. # -s: svm类型c-svc   -t: 核函数类型 0线性核函 2RBF核
9. m = svm\_train(y, x, '-s 0 -t 2 -c 35.0 -g 0.0079125')  # 训练分类器
10. svm\_save\_model('libsvm.model', m)  # 保存训练生成的模型
12. **if** \_\_name\_\_ == '\_\_main\_\_':
13. main()

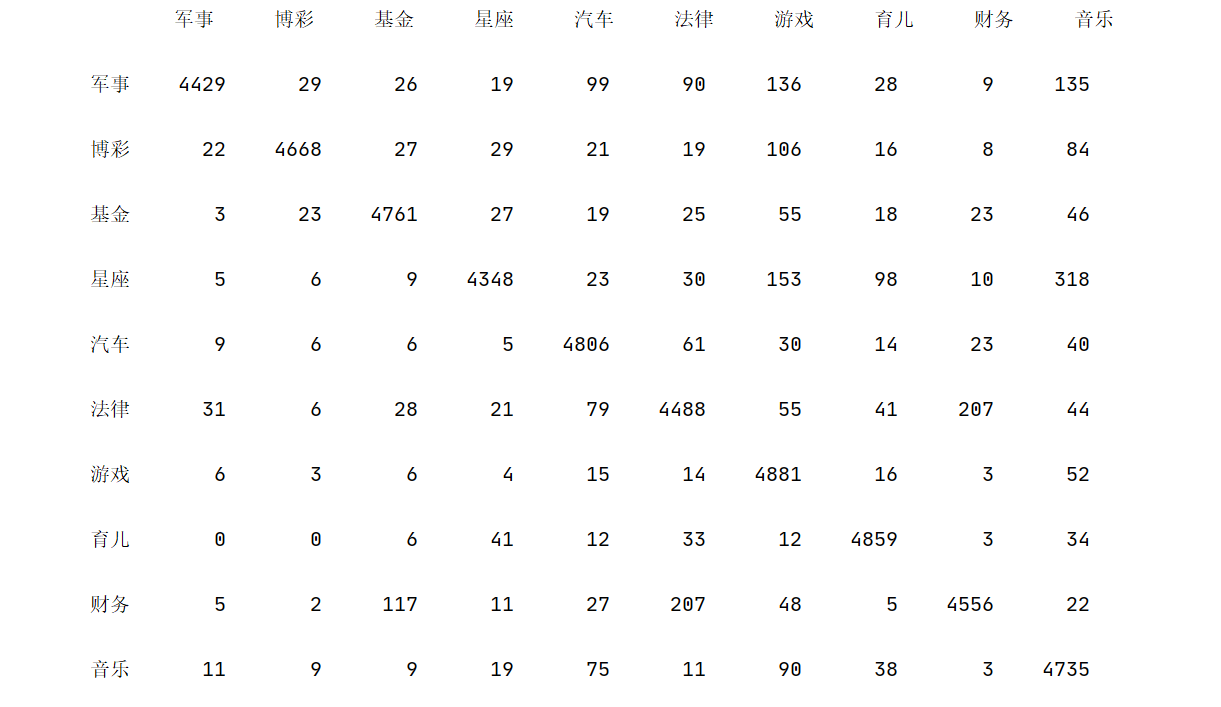
(3) test.py

1. **from** libsvm.commonutil **import** \*
2. **from** libsvm.svm **import** \*
3. **from** libsvm.svmutil **import** \*
4. **import** time
5. file\_path = r'E:\大三\课件和code\人工智能原理\实验\小组实验资料'  # 存放数据文件夹的父目录
6. label\_list = ['育儿', '博彩', '财务', '法律', '基金', '军事', '汽车', '星座', '音乐', '游戏']
8. # 打印混淆矩阵
9. **def** print\_matrix(matrix):
10. **print**('{:>8}'.format(''), end='')
11. **for** label **in** range(10):
12. **print**('{:>7}'.format(label\_list[label]), end='')
13. **print**('\n')
14. **for** row **in** range(10):
15. **print**('{:>8}'.format(label\_list[row]), end='')
16. **for** col **in** range(10):
17. **print**('{:>8}'.format(matrix[row][col]), end='')
18. **print**('\n')
20. # 输出预测结果，每类和总的的准去率、召回率，f-score
21. **def** test\_result(label):   # label为列表
22. matrix = [[0 **for** i **in** range(10)] **for** i **in** range(10)]  # 10\*10二维矩阵
23. predict\_list = []
24. **for** i **in** range(10):
25. predict\_list.append(label[i\*5000:i\*5000+5000])
26. **for** col **in** range(10):
27. matrix[0][col] = predict\_list[col].count(1.0)
28. matrix[1][col] = predict\_list[col].count(2.0)
29. matrix[2][col] = predict\_list[col].count(3.0)
30. matrix[3][col] = predict\_list[col].count(4.0)
31. matrix[4][col] = predict\_list[col].count(5.0)
32. matrix[5][col] = predict\_list[col].count(6.0)
33. matrix[6][col] = predict\_list[col].count(7.0)
34. matrix[7][col] = predict\_list[col].count(8.0)
35. matrix[8][col] = predict\_list[col].count(9.0)
36. matrix[9][col] = predict\_list[col].count(10.0)
37. print\_matrix(matrix)
38. # 每一行每一列的和
39. row\_sum = []
40. col\_sum = []
41. **for** i **in** range(10):
42. row\_sum\_temp = 0
43. col\_sum\_temp = 0
44. **for** j **in** range(10):
45. row\_sum\_temp += matrix[i][j]
46. col\_sum\_temp += matrix[j][i]
47. row\_sum.append(row\_sum\_temp)
48. col\_sum.append(col\_sum\_temp)
49. # 计算每一类以及总的的准确率precision和召回率recall
50. precision\_list = []
51. recall\_list = []
52. total\_precision = 0
53. total\_recall = 0
54. **for** kind **in** range(10):
55. precision\_list.append(matrix[kind][kind] / row\_sum[kind])
56. recall\_list.append(matrix[kind][kind] / col\_sum[kind])
57. total\_precision += precision\_list[kind]
58. total\_recall += recall\_list[kind]
59. total\_precision /= 10
60. total\_recall /= 10
61. # 输出准确率precision和召回率recall
62. **for** kind **in** range(10):
63. **print**(label\_list[kind]+":")
64. **print**("precision={:.6f},  recall={:.6f},  f-score={:.6f}".format(precision\_list[kind], recall\_list[kind], 2\*precision\_list[kind]\*recall\_list[kind] / (precision\_list[kind]+recall\_list[kind])))
65. **print**("total\_precision={:.6f},  total\_recall={:.6f},  total\_f-score={:.6f}".format(total\_precision, total\_recall, 2\*total\_precision\*total\_recall / (total\_precision+total\_recall)))
67. # 测试分类器并输出分类结果
68. **def** main():
69. t1 = time.time()
70. y, x = svm\_read\_problem(file\_path+r'\test\_tfidf\_scale.txt')
71. m = svm\_load\_model('libsvm.model')  # model为train.py生成的文件，放在当前python文件同录下
72. [lable, acc, val] = svm\_predict(y, x, m)   # lable返回一个列表，包含预测的类别
73. test\_result(lable)
74. t2 = time.time()
75. **print**("测试时间：{:.6f}s".format(t2-t1))
76. **if** \_\_name\_\_ == '\_\_main\_\_':
77. main()

# 实验结果

## 朴素贝叶斯分类器

* **混淆矩阵**

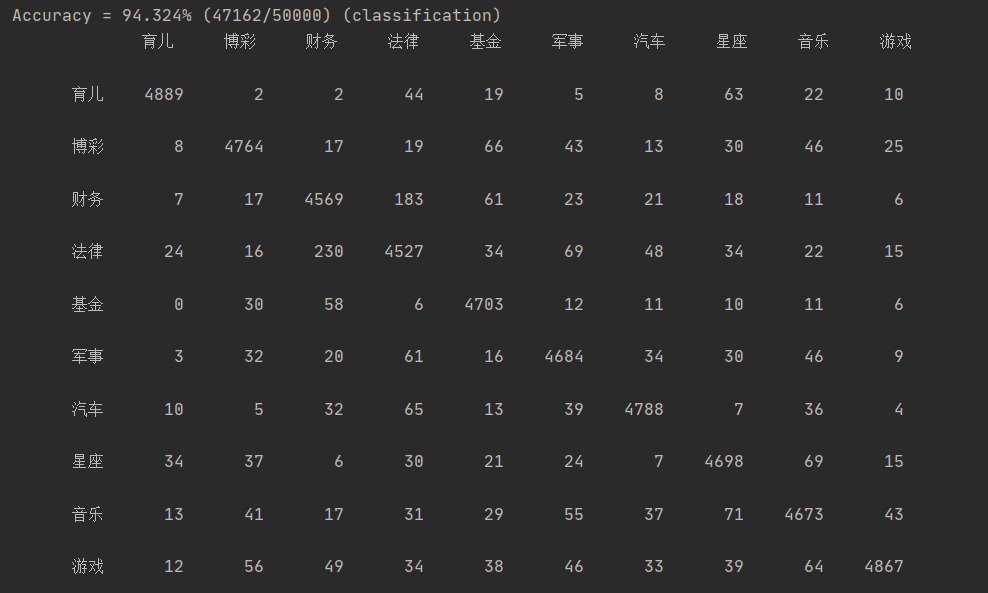


* **准确率、召回率**

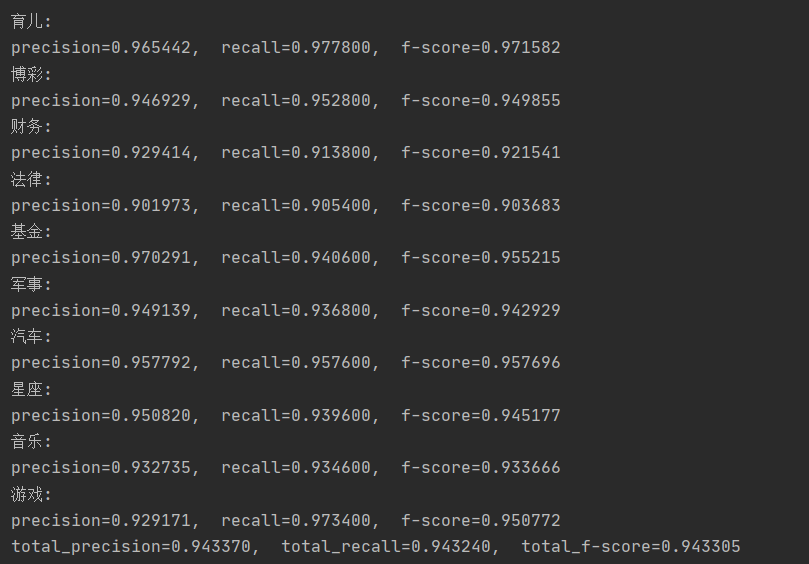


## SVM分类器

* **混淆矩阵**



* **准确率、召回率、F测度**



# 实验总结

通过这次实验，我们了解并学习到了文本分类的整体流程，从实验数据的搜集、文本格式的转换处理，到分词并去除停用词，生成词典，然后到朴素贝叶斯分类器的实现和SVM分类器的实现，对于各个流程的实现有了一定的认识与掌握。在实验的过程中，我们掌握了一些数据预处理的方法和文本建模的方法，对于分类算法的原理也有了一定的认识与了解，熟悉了机器学习中的文本分类器，包括朴素贝叶斯分类器和SVM分类器，同时还学习到了通过输出混淆矩阵来观察分类器的分类结果和通过计算准确率和召回率来评估分类器性能的评估方法。