

# 北京邮电大学

## 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
实验类型 .....	3
实验要求 .....	3
实验内容 .....	3
实验步骤 .....	4
数据收集 .....	4
数据处理 .....	5
朴素贝叶斯分类器 .....	15
SVM 分类器 .....	19
实验结果 .....	26
朴素贝叶斯分类器 .....	26
SVM 分类器 .....	27
实验总结 .....	29

## 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： $\geq 10$  类；
2. 训练集文档数： $\geq 50000$  篇；每类平均 5000 篇。
3. 测试集文档数： $\geq 50000$  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](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：

```
{
  "qid": "qid_5982723620932473219",
  "category": "教育/科学-理工学科-地球科学",
  "title": "人站在地球上为什么没有头朝下的感觉",
  "desc": "",
  "answer": "地球上重力作用一直是指向球心的，因此只要头远离球心，人们就回感到头朝上。"}
{
  "qid": "qid_5679706523376347837",
  "category": "娱乐-宠物",
  "title": "我的小baby",
  "desc": "我的小baby-辛巴。温顺可爱的，两个月大的小家伙，第一次养狗，该注意什么呢？求指教~[爱你]",
  "answer": "勤洗澡，养成好的卫生习惯"}
{
  "qid": "qid_6610724023825624555",
  "category": "娱乐-度假旅游",
  "title": "请问这起交通事故是谁的责任居多？小车和摩托车发生事故，在无红绿灯的十字路口，小停车看看左右，在觉得安全的情况下刹车慢慢以时速10公里左右的速度靠右行驶过路口，好没有出到十字路口正中时，被左边突然快速行驶过来的摩托车撞在车头前，摩托车主摔到膝盖和擦伤脸部，请问这起交通事故是谁的责任居多。如果双方都有责任的话，大概各占几成？~\r",
  "answer": "通过没有信号控制的十字路口，应该减速慢行，让右边的车先行，按你说的，摩托车好像在汽车的左边，所以严格来说可能摩托车全责。当然还要看汽车是否证照齐全，是否饮酒等。具体由交警调查后认定。"}
{
  "qid": "qid_8948366474478096617",
  "category": "电脑/网络-互联网",
  "title": "松本面板可以配什么品牌的超五类模块？",
  "desc": "",
  "answer": "AMP的试试吧，还有普天的。"}
{
  "qid": "qid_3057155567155332897",
  "category": "生活-美食/烹饪",
  "title": "请教怎么能很快能洗干净猪肠？有什么方法",
  "desc": "有什么方法",
  "answer": "先用清水冲一下，在用食盐反复的搓揉，直到肠的黏液全去掉为止，在用清水反复冲洗干净，最好把肠上的油去掉一些再下锅。"}
}
```

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
30.
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:

```

```

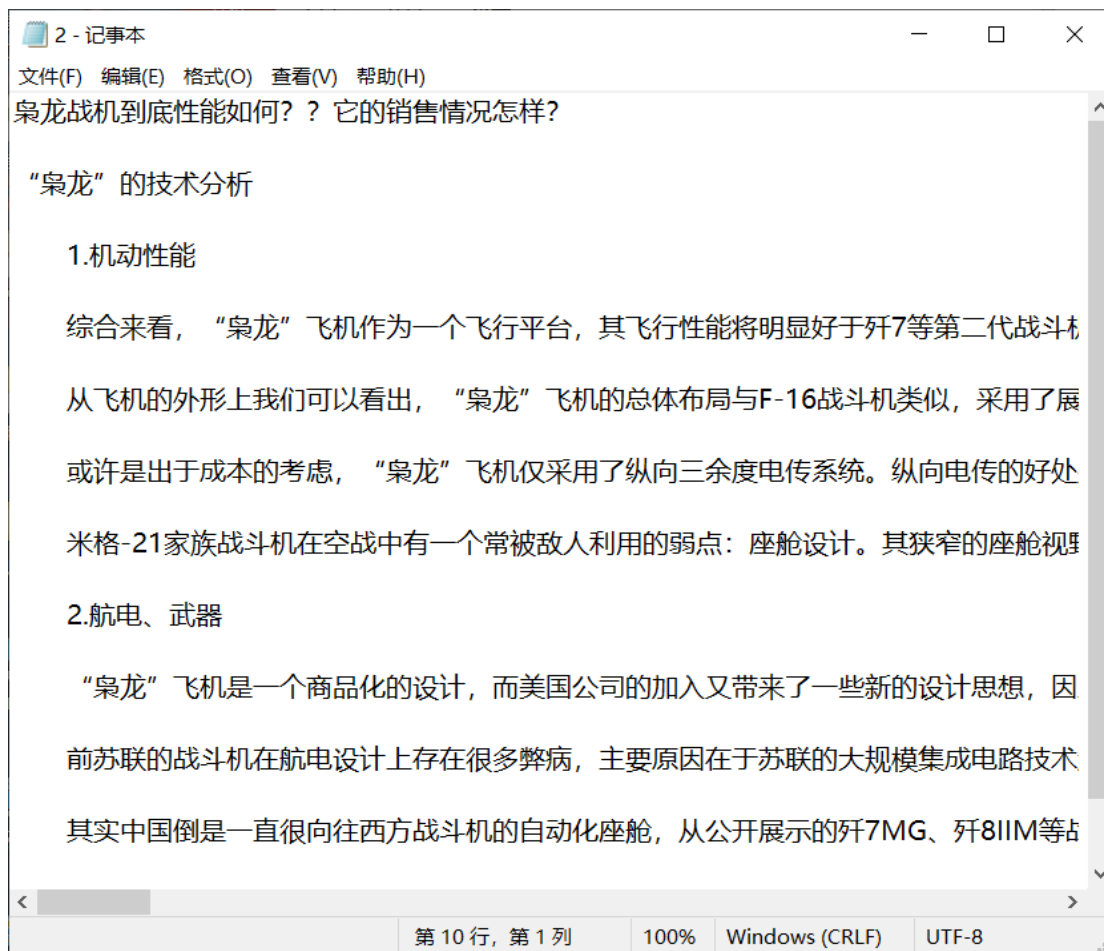
52.         typenum[type] += 1
53.         '''新建文件夹'''
54.         if not os.path.exists('QAsamples2/' + type):
55.             os.makedirs('QAsamples2/' + type)
56.         '''写txt'''
57.         file = 'QAsamples2/' + type + '/' + str(typenum[type]) + '.txt'
58.         fw=open(file, 'w', encoding='utf-8')
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：

■ 博彩	■ 宝宝
■ 电脑	■ 财务
■ 电子	■ 夫妻
■ 度假	■ 工程
■ 法律	■ 股票
■ 烦恼	■ 基金
■ 购物	■ 精神
■ 健康	■ 恋爱
■ 交通	■ 上网
■ 教育	■ 生物
■ 军事	■ 手机
■ 美食	■ 数学
■ 汽车	■ 体育
■ 商业	■ 外语
■ 文化	■ 文学
■ 星座	■ 诛仙
■ 音乐	
■ 游戏	
■ 娱乐	

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
3.
4.
5. ''' 读取停用词文件'''
6. f_stop=open('stop_words.txt','rb')
7. stopwords=f_stop.readlines()
8.
9.
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
```

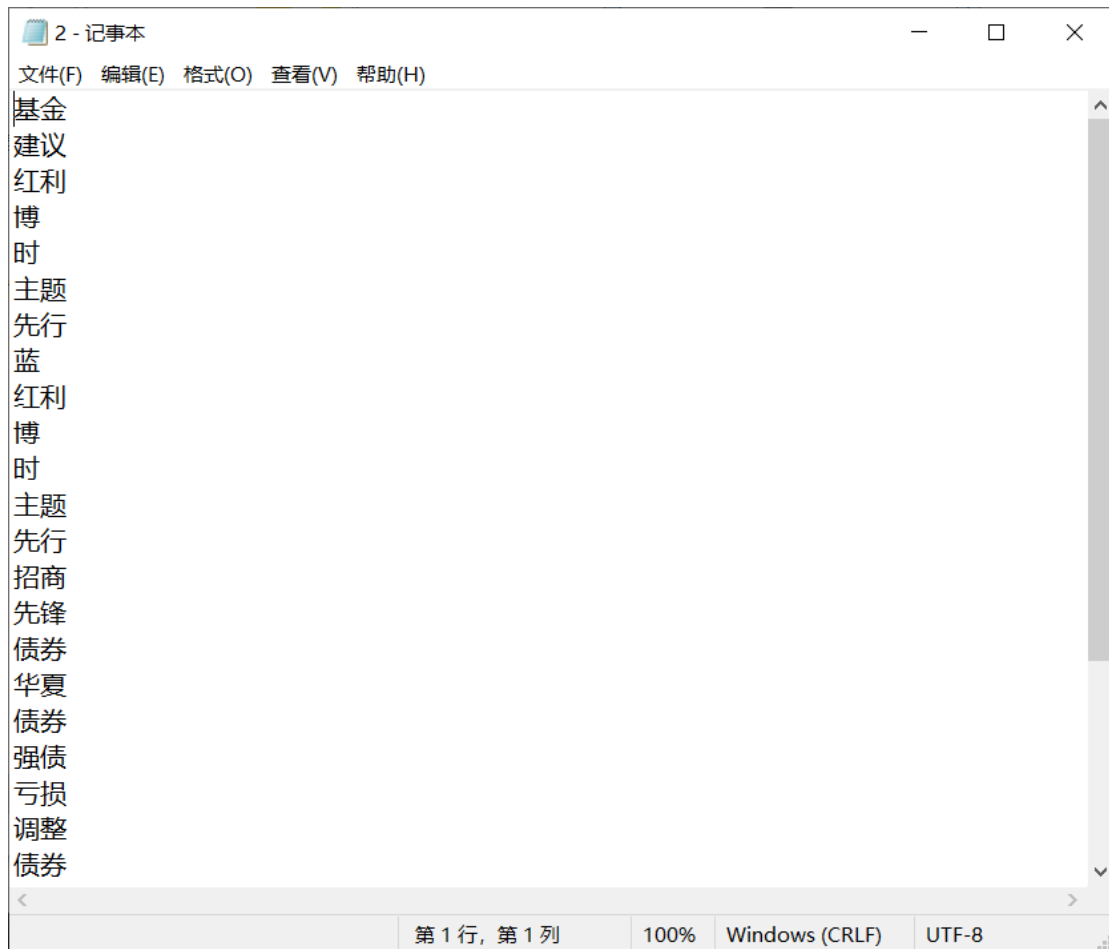


```

15.     return False
16.
17.
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')
36.
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()
42.
43.
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'
4.
5.
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
12.
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：

$$\text{idf}_i = \lg \frac{|D|}{|\{j : t_i \in d_j\}|}$$

- $|D|$ ：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
5.
6.
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：

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}}$$

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。

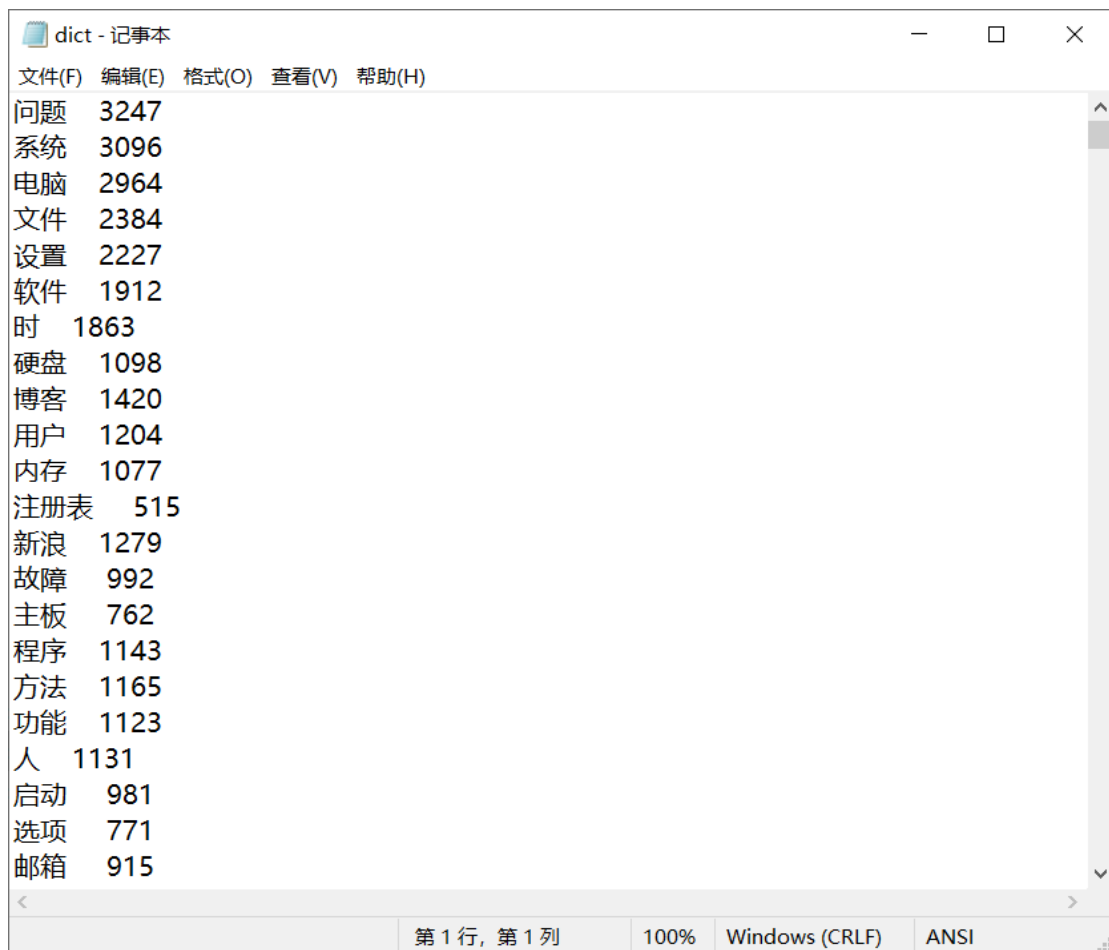
```
1. import os
2.
3.
4. '''此处改成 clean_data 的位置'''
5. path='E:/Documents/PPT/人工智能原理/文本实验/result3'
6. idf={}
7. '''读取 idf 值'''
8. f=open('idf.txt','r')
9. p=f.readlines()
10. for each in p:
11.     idf[each.split()[0]]=float(each.split()[1])
12. f.close()
13. '''索引每一类路径'''
14. for file in os.listdir(path):
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:



```

dict - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
问题 3247
系统 3096
电脑 2964
文件 2384
设置 2227
软件 1912
时 1863
硬盘 1098
博客 1420
用户 1204
内存 1077
注册表 515
新浪 1279
故障 992
主板 762
程序 1143
方法 1165
功能 1123
人 1131
启动 981
选项 771
邮箱 915
第 1 行, 第 1 列 100% Windows (CRLF) ANSI

```

再将每个类的词典中的词合并, 生成总词典

```

1. import os
2.
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：

$$v_{NB} = \operatorname{argmax}_{a_i \in A} \{P(a_i)P(b_1|a_i)P(b_2|a_i) \dots P(b_n|a_i)\}$$

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：

$$\begin{aligned}
 v_{NB} &= \operatorname{argmax}_{a_i \in A} \left\{ \log_{10}(P(a_i)) + \sum_{b_i \in B} \log_{10}(P(b_i|a_i)) \right\} \\
 &= \operatorname{argmax}_{a_i \in A} \left\{ P_{\log_{10}}(a_i) + \sum_{b_i \in B} P_{\log_{10}}(b_i|a_i) \right\}
 \end{aligned}$$

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

$$P(w_k | v_j) = \frac{n_k + 1}{n + |Vocabulary|}$$

#### ◆ 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']))
24.
25.
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')
```



[illegible]

```

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))
124.
125.
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：

$$\text{tfidf} = \text{tf} * \text{idf}$$

$$\text{tf}(w, d) = \text{count}(w, d) / \text{size}(d)$$

$$\text{idf} = \log(n / \text{docs}(w, D))$$

$\text{count}(w, d)$  为词  $w$  在文档  $d$  中出现的次数， $\text{size}(d)$  为文档  $d$  总词数， $n$  为文档总数， $\text{docs}(w, D)$  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：

```
svm-scale -l 0 -u 1 train_tfidf_File.txt>train_tfidf_scale.txt
```

```
svm-scale -l 0 -u 1 test_tfidf_File.txt>test_tfidf_scale.txt
```

(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']
4.
5.
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
15.
```

```

16.
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
34.
35.
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
47.
48.
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
68.
69.
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
89.
90.
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
96.
97.

```

```
98. if __name__ == '__main__':
99.     main()
```

## (2) train.py

```
1. from libsvm.commonutil import *
2. from libsvm.svm import *
3. from libsvm.svutil 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
   件的父目录
5.
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
11.
12. if __name__ == '__main__':
13.     main()
```

## (3) test.py

```
1. from libsvm.commonutil import *
2. from libsvm.svm import *
3. from libsvm.svutil 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']
7.
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')
19.
```



```

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_li
            st[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)))
66.
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 文件同目录下
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

	军事	博彩	基金	星座	汽车	法律	游戏	育儿	财务	音乐
军事	4429	29	26	19	99	90	136	28	9	135
博彩	22	4668	27	29	21	19	106	16	8	84
基金	3	23	4761	27	19	25	55	18	23	46
星座	5	6	9	4348	23	30	153	98	10	318
汽车	9	6	6	5	4806	61	30	14	23	40
法律	31	6	28	21	79	4488	55	41	207	44
游戏	6	3	6	4	15	14	4881	16	3	52
育儿	0	0	6	41	12	33	12	4859	3	34
财务	5	2	117	11	27	207	48	5	4556	22
音乐	11	9	9	19	75	11	90	38	3	4735

### ◆ 准确率、召回率

类别: 军事

precision=0.979651 , recall=0.885800

类别: 博彩

precision=0.982323 , recall=0.933600

类别: 基金

precision=0.953153 , recall=0.952200

类别: 星座

precision=0.961096 , recall=0.869600

类别: 汽车

precision=0.928516 , recall=0.961200

类别: 法律

precision=0.901567 , recall=0.897600

类别: 游戏

precision=0.876931 , recall=0.976200

类别: 育儿

precision=0.946620 , recall=0.971800

类别: 财务

precision=0.940351 , recall=0.911200

类别: 音乐

precision=0.859347 , recall=0.947000

total\_precision=0.932956 , total\_recall=0.930620

## SVM 分类器

### ◆ 混淆矩阵

Accuracy = 94.324% (47162/50000) (classification)										
	育儿	博彩	财务	法律	基金	军事	汽车	星座	音乐	游戏
育儿	4889	2	2	44	19	5	8	63	22	10
博彩	8	4764	17	19	66	43	13	30	46	25
财务	7	17	4569	183	61	23	21	18	11	6
法律	24	16	230	4527	34	69	48	34	22	15
基金	0	30	58	6	4703	12	11	10	11	6
军事	3	32	20	61	16	4684	34	30	46	9
汽车	10	5	32	65	13	39	4788	7	36	4
星座	34	37	6	30	21	24	7	4698	69	15
音乐	13	41	17	31	29	55	37	71	4673	43
游戏	12	56	49	34	38	46	33	39	64	4867

### ◆ 准确率、召回率、F 测度

```

育儿:
precision=0.965442, recall=0.977800, f-score=0.971582
博彩:
precision=0.946929, recall=0.952800, f-score=0.949855
财务:
precision=0.929414, recall=0.913800, f-score=0.921541
法律:
precision=0.901973, recall=0.905400, f-score=0.903683
基金:
precision=0.970291, recall=0.940600, f-score=0.955215
军事:
precision=0.949139, recall=0.936800, f-score=0.942929
汽车:
precision=0.957792, recall=0.957600, f-score=0.957696
星座:
precision=0.950820, recall=0.939600, f-score=0.945177
音乐:
precision=0.932735, recall=0.934600, f-score=0.933666
游戏:
precision=0.929171, recall=0.973400, f-score=0.950772
total_precision=0.943370, total_recall=0.943240, total_f-score=0.943305

```

# 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。