
密级： 保密期限：

北京邮电大学

人工智能原理-文本分类实验报告



题目：基于朴素贝叶斯/SVM 的文本分类实验

学 号：2018210547, 2018210074

班 级：2018211302

姓 名：胡天翼，熊宇

专 业：计算机科学与技术

导 师：王晓茹

学 院：计算机学院

2021 年 1 月 1 日

人工智能实验：文本数据的分类与分析

目录

- 一、实验目的..... 2
- 二、实验类型..... 2
- 三、实验要求..... 2
- 四、实验内容..... 3
 - 4.1 实验分工..... 3
- 五、实验步骤..... 4
 - 5.1 数据收集..... 4
 - 5.2 数据预处理 7
 - 5.3 朴素贝叶斯分类器 14
 - 5.4 SVM 分类器 18
- 六、思考与体会 22
- 附录 实验代码 23
 - GetData_1.py..... 23
 - GetData_2.py..... 24
 - SelectNews.py..... 26
 - DataPro_1.py 27
 - DataPro_2.py 29
 - DataPro_Test.py 29
 - 朴素贝叶斯.py 30
 - SVC.py 34

一、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。

二、实验类型

数据挖掘算法的设计与编程实现。

三、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。
4. 分组完成实验，组员数量 ≤ 3 ，个人实现可以获得实验加分。

四、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；

注：使用主题模型，如 LDA 可以获得实验加分；

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。

4.1 实验分工

胡天翼：语料库构建、数据预处理

熊宇：朴素贝叶斯实现及 SVM

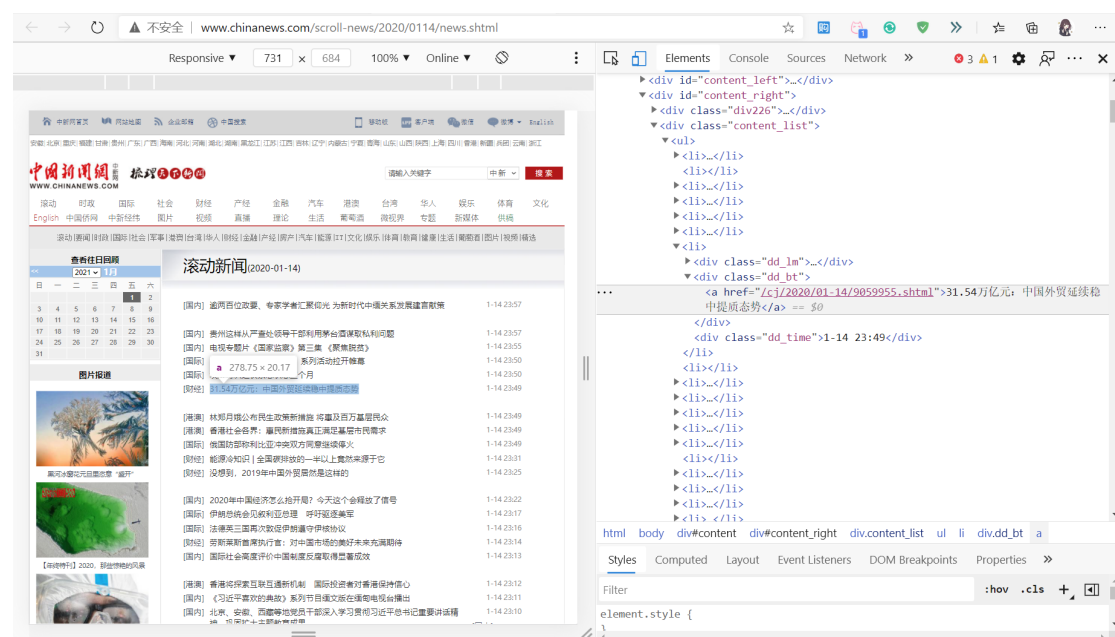
五、实验步骤

5.1 数据收集

在本次实验中，我通过自编爬虫，获取中国新闻网的新闻文本。

第一步，在中国新闻网的“滚动新闻（scroll-news）”页面，爬取所有新闻的 url。

其中，yyyy 年 mm 月 dd 日的滚动新闻网页格式为 <http://www.chinanews.com/scroll-news/yyyy/mmdd/news.shtml>。使用 BeautifulSoup 分析网页，提取出类别为‘dd_bt’的 div 块，其中的内容为每条新闻网页 url。



```

def getUrIs(url):
    req = requests.get(url).text
    if (req == None):
        return
    bf = BeautifulSoup(req, 'html.parser')
    div_bf = bf.find('div', attrs={'class': 'content_list'})
    if isinstance(div_bf, bs4.element.Tag):
        div_a = div_bf.find_all('div', attrs={'class': 'dd_bt'})
        urltxt = open(b'./data/url.txt', 'a', encoding='UTF-8')
        for div in div_a:
            link = div.find('a').get("href")
            urltxt.write(link+'\n')
        urltxt.close()

years = ['2011', '2012', '2013', '2014', '2015', '2016', '2017', '2018', '2019', '2020']
months = ["01", "02", "03", "04", "05", "06", "07", "08", "09", "10", "11", "12"]
days = ["01", "02", "03", "04", "05", "06", "07", "08", "09", "10", "11", "12", "13", "14", "15", "16", "17", "18", "19", "20", "21", "22", "23", "24", "25", "26", "27", "28", "29", "30", "31"]
for year in range(10):
    for month in range(12):
        print(years[year] + " " + months[month])
        for day in range(31):
            url = "http://www.chinanews.com/scroll-news/" + years[year] + "/" + months[month] + days[day] + "/news.shtml"
            print(url)
            getUrIs(url)
            head = requests.head(url)
            req = requests.get(url)
            req.encoding = 'GB2312'
            bf = BeautifulSoup(req.text, 'html.parser')
            div = bf.find('div', attrs={'class': 'content'})
            print(div)
            if isinstance(div, bs4.element.Tag):
                h1 = div.find('h1')

```

提取结果 url.txt

```

/auto/2013/09-06/5255532.shtml
/edu/2013/09-06/5255531.shtml
/ny/2013/09-06/5255530.shtml
/life/2013/09-06/5255529.shtml
/gj/2013/09-06/5255528.shtml
/fortune/2013/09-06/5255527.shtml
/auto/2013/09-06/5255414.shtml
/it/2013/09-06/5255526.shtml
/cj/2013/09-06/5255408.shtml
/ny/2013/09-06/5255525.shtml
/it/2013/09-06/5255524.shtml
/tw/2013/09-06/5255416.shtml
/it/2013/09-06/5255523.shtml
/auto/2013/09-06/5255522.shtml
/fz/2013/09-06/5255521.shtml
/it/2013/09-06/5255520.shtml
/fz/2013/09-06/5255519.shtml
/mil/2013/09-06/5255516.shtml
/edu/2013/09-06/5255324.shtml
/sh/2013/09-06/5255515.shtml
/ga/2013/09-06/5255514.shtml
/auto/2013/09-06/5255513.shtml
/sh/2013/09-06/5255346.shtml
/cj/2013/09-06/5255512.shtml
/sh/2013/09-06/5255511.shtml
/gj/2013/09-06/5255510.shtml
/gj/2013/09-06/5255509.shtml
/fz/2013/09-06/5255508.shtml
/gj/2013/09-06/5255507.shtml
/fz/2013/09-06/5255506.shtml
/ty/2013/09-06/5255505.shtml
/sh/2013/09-06/5255504.shtml
/fz/2013/09-06/5255503.shtml
/auto/2013/09-06/5255502.shtml
/gj/2013/09-06/5255501.shtml

```

第二步，使用收集到的新闻页面 url，获取新闻文本，提取其中的'left_zw'div 块。

特别要注意的是，2019 年以前的网页使用 GBK 编码，而 2019 年后的网页使用 utf-8 编码。

```
for url in urls:
    l = url.split('/')
    if (url[0] == '/' and len(l) > 2 and l[1].isalpha() and l[1] == 'it' and l[2].isdigit()):

        # print(url)
        # print(l[1], l[2])
        Type = l[1];
        Year = int(l[2])
        if (Year >= 2012):
            cnt = cnt + 1
            url = 'http://www.chinanews.com' + url
            # url = 'http://www.chinanews.com/cul/2020/11-23/9345057.shtml'
            print(url)
            res = requests.get(url)
            if (Year < 2019):
                res.encoding = 'GBK' # htm1: ISO-8859-1 (2012)
            else:
                res.encoding = 'utf-8' # res.encoding = 'utf-8' # (2019)

            soup = BeautifulSoup(res.text, 'html.parser')
            title = soup.find('h1')
            if isinstance(title, bs4.element.Tag):
                news_contents = title.text.strip()
                temp = soup.find('div', 'left_zw')
                if isinstance(temp, bs4.element.Tag):
                    contents = temp.find_all('p')
                    for content in contents:
                        if 'function' in content.text:
                            continue
                    news_contents = news_contents + content.text.strip()
```

共提取 200 万条新闻。



5.2 数据预处理

首先在爬取到的新闻文本中，选择有代表性的 10 类，这里我选择['IT', '股票', '犯罪', '教育', '军事', '能源', '汽车', '体育', '娱乐', '住房']。在每一类中，筛选长度为 2000~3000 字的文章进行处理，从而使得数据集文档词数接近。

```
for t in types:
    dirpath = './data/news/' + t
    print(t, ' ', end='')
    cnt = 0
    for root, dirs, files in os.walk(dirpath):
        # root 表示当前正在访问的文件夹路径
        # dirs 表示该文件夹下的子目录名list
        # files 表示该文件夹下的文件list
        # 遍历文件
        for f in files:
            Path = os.path.join(root, f)
            # print(Path)
            file_obj = open(Path, 'rb')
            doc = file_obj.read()
            file_obj.close()
            l = len(doc)
            if (l >= R[t][0] and l <= R[t][1]):
                cnt += 1
                words = pseg.cut(doc)
                word_str = ''
                first = True
                for word in words:
                    now_word = str(word.word)
                    if (('n' or 1) in word.flag and word.flag != 'nr' and is_all_chinese(now_word)):
                        if (first):
                            first = False
                        else:
                            word_str += ' '
                    word_str += word.word

                Path = os.path.join('./data/Data_Set/' + t, f)
                out = open(Path, 'w')
                out.write(word_str)
                out.close()
```

然后利用 Jieba 对筛选本文进行分词，分词后得到一个 List，每个元素包含 flag 与 word 两个元素。其中 flag 表明词语的属性，如下表所示：

| 标签 | 含义 | 标签 | 含义 | 标签 | 含义 | 标签 | 含义 |
|-----|------|-----|------|-----|------|------|------|
| n | 普通名词 | f | 方位名词 | s | 处所名词 | t | 时间 |
| nr | 人名 | ns | 地名 | nt | 机构名 | nw | 作品名 |
| nz | 其他专名 | v | 普通动词 | vd | 动副词 | vn | 名动词 |
| a | 形容词 | ad | 副形词 | an | 名形词 | d | 副词 |
| m | 数量词 | q | 量词 | r | 代词 | p | 介词 |
| c | 连词 | u | 助词 | xc | 其他虚词 | w | 标点符号 |
| PER | 人名 | LOC | 地名 | ORG | 机构名 | TIME | 时间 |

为了准确提取文章特征，我们只选择名词，且排除人名。

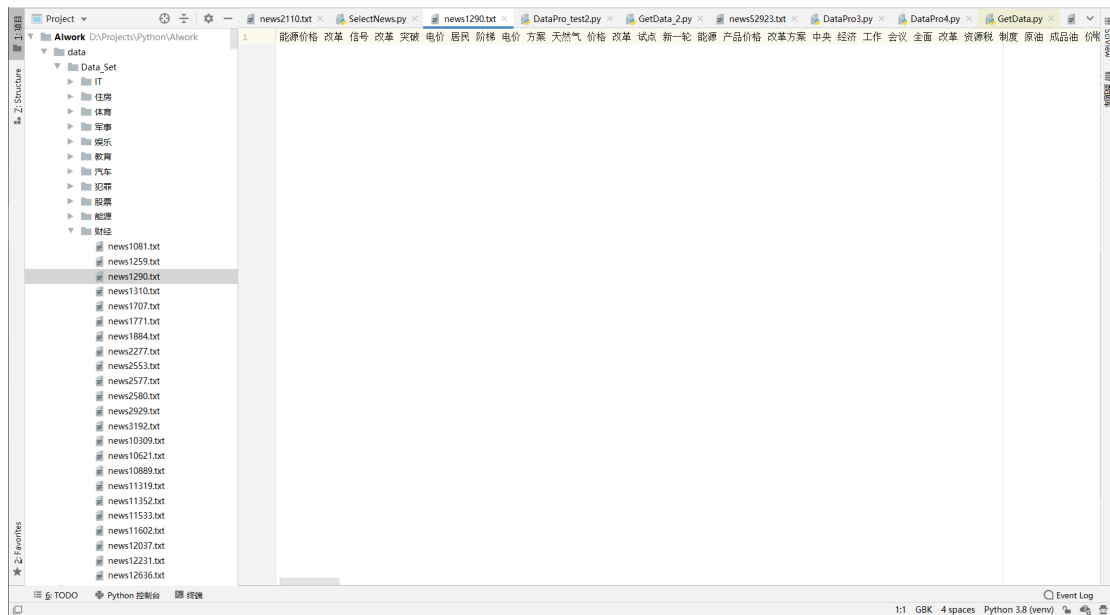
```

for t in types:
    dirpath = './data/news/' + t
    print(t, ' ', end='')
    cnt = 0
    for root, dirs, files in os.walk(dirpath):
        # root 表示当前正在访问的文件夹路径
        # dirs 表示该文件夹下的子目录名list
        # files 表示该文件夹下的文件list
        # 遍历文件
        for f in files:
            Path = os.path.join(root, f)
            # print(Path)
            file_obj = open(Path, 'rb')
            doc = file_obj.read()
            file_obj.close()
            l = len(doc)
            if (l >= R[t][0] and l <= R[t][1]):
                cnt += 1
                words = pseg.cut(doc)
                word_str = ''
                first = True
                for word in words:
                    now_word = str(word.word)
                    if (('n' or 1) in word.flag and word.flag != 'nr' and is_all_chinese(now_word)):
                        if (first):
                            first = False
                        else:
                            word_str += ' '
                            word_str += word.word

                Path = os.path.join('./data/Data_Set/' + t, f)
                out = open(Path, 'w')
                out.write(word_str)

```

得到完成分词的 10 类文本，每类一万篇。



将每类的 1 万篇文档交替存入两个词袋模型，用于划分相等数量的训练集与测试集。

```
for t in types:
    dirpath = './data/Data_Set/' + t
    print(dirpath)
    Train = True
    for root, dirs, files in os.walk(dirpath):
        # root 表示当前正在访问的文件夹路径
        # dirs 表示该文件夹下的子目录名list
        # files 表示该文件夹下的文件list
        # 遍历文件
        for f in files:
            Path = os.path.join(root, f)
            print(Path)
            file_obj = open(Path, 'r')
            doc = file_obj.read()
            file_obj.close()
            print(Train)
            if (Train):
                Bunch_train.filesnames.append(str(f))
                Bunch_train.label.append(str(t))
                Bunch_train.contents.append(str(doc))
            else:
                Bunch_test.filesnames.append(str(f))
                Bunch_test.label.append(str(t))
                Bunch_test.contents.append(str(doc))
            Train = not Train

with open(word_bag_filepath_train, 'wb') as file_obj1:
    pickle.dump(Bunch_train, file_obj1)

with open(word_bag_filepath_test, 'wb') as file_obj2:
    pickle.dump(Bunch_test, file_obj2)
```

结果得到 raw_Bunch_test 与 raw_Bunch_train 两个原始词袋文件

| | | | |
|---|----------------|----|------------|
|  raw_Bunch_test | 20/12/30 15:53 | 文件 | 134,824 KB |
|  raw_Bunch_train | 20/12/30 15:53 | 文件 | 135,038 KB |

使用 TF-IDF 模型提取训练集中的特征词。TFIDF 的主要思想是：如果某个词或短语在一篇文章中出现的频率 TF 高，并且在其他文章中很少出现，则认为此词或者短语具有很好的类别区分能力，适合用来分类。计算数据集中词频（TF）和逆文档频率（IDF），两者相乘得到 TF-IDF。

$$\text{词频(TF)} = \frac{\text{某个词在文章中的出现次数}}{\text{文章的总词数}}$$

$$\text{逆文档频率(IDF)} = \log\left(\frac{\text{语料库的文档总数}}{\text{包含该词的文档数} + 1}\right)$$

$$\text{TF-IDF} = \text{词频(TF)} \times \text{逆文档频率 (IDF)}$$

处理为 TF-IDF 模型后，利用 TfidfVectorizer 方法得到指定维度的字典为 2000，得到数据字典，并生成训练集、测试集词袋模型。其中在生成 TF-IDF 的 TfidfVectorizer 指定去除在少于 0.1%的文档中出现的词，在高于 15%的文档中出现的词。本实验共选择 10 类文本，每类占比 10%，如果一个词在 15%以上的文档中都出现，说明它在不只一个类别中多次，那么该词语不具有好的代表性，应该去掉。对于少于 0.1%的词，同样不具有代表性。

```

bunch_path = './data/raw_Bunch_train'

with open(bunch_path, 'rb') as file_obj:
    bunch = pickle.load(file_obj)
begin_time = datetime.datetime.now()
vectorizer = TfidfVectorizer(sublinear_tf=True, max_df=0.15, min_df=0.001, max_features=2000)

tfidf_space = Bunch(filenamees = bunch.filenamees, # 所有文章的文件名
                    tfidf_weight_matrices = [], # TF-IDF 权重矩阵
                    vocabulary = {}) # 词典

tfidf_space.tfidf_weight_matrices = vectorizer.fit_transform(bunch.contents)

print(vectorizer.get_feature_names())
print(tfidf_space.tfidf_weight_matrices)

end_time = datetime.datetime.now()
print("执行完毕, 用时为: " + str((end_time - begin_time).seconds) + "秒")

tfidf_space.vocabulary = vectorizer.vocabulary_

with open('./data/Bunch_train', 'wb') as file_obj1:
    pickle.dump(tfidf_space, file_obj1)

```

数据字典:



TF-IDF 权重矩阵

```
(0, 321) 0.15826739135191856
(0, 377) 0.13056084960846473
(0, 246) 0.06493592668817769
(0, 1057) 0.05310287027208342
(0, 1815) 0.2522189600853962
(0, 1749) 0.053814228554817586
(0, 1048) 0.06458291959421159
(0, 1097) 0.06060674511974396
(0, 551) 0.06890257358257013
(0, 499) 0.22710753881654488
(0, 1352) 0.13259260148164034
(0, 1800) 0.039671551355072235
: :
(49999, 1290) 0.12800947414402564
(49999, 1179) 0.11681596377779152
(49999, 1234) 0.1496513412847749
(49999, 1906) 0.16134337675692398
(49999, 947) 0.1114420043086834
(49999, 1305) 0.10873841969389322
(49999, 1875) 0.14621764517433541
(49999, 1809) 0.08925994233169261
(49999, 961) 0.08985539888909592
(49999, 564) 0.12924800018892316
(49999, 980) 0.0752689147819013
(49999, 1079) 0.088759061039017
(49999, 16) 0.11811966630363836
(49999, 1216) 0.12071970283693718
(49999, 419) 0.1057459233304072
(49999, 712) 0.13191405757523816
(49999, 1011) 0.10083028923401535
```

(用 numpy 稀疏矩阵表示, (i,j)代表第 i 篇测试文档在第 j 维向量上的特征值)

用训练集得到的字典生成测试集的特征向量。

```

from sklearn.feature_extraction.text import TfidfVectorizer
import pickle
from sklearn.utils import Bunch

bunch_path = './data/raw_Bunch_test'
train_tfidf_path = './data/Bunch_train'

with open(bunch_path, 'rb') as file_obj:
    bunch = pickle.load(file_obj)

with open(train_tfidf_path, 'rb') as file_obj1: # 加载训练集 需要使用字典
    trainbunch = pickle.load(file_obj1)

tfidf_space = Bunch(
    filenames = bunch.filenames, # 所有文章的文件名
    tfidf_weight_matrices = [], # TF-IDF 权重矩阵
    vocabulary = trainbunch.vocabulary)


vectorizer = TfidfVectorizer(sublinear_tf=True, max_df=0.13, min_df=0.001, vocabulary=trainbunch.vocabulary)
tfidf_space.tfidf_weight_matrices = vectorizer.fit_transform(bunch.contents)

print(vectorizer.get_feature_names())
print(tfidf_space.tfidf_weight_matrices)

with open('./data/Bunch_test', 'wb') as file_obj2:
    pickle.dump(tfidf_space, file_obj2)

```

得到训练集与测试集的词袋文件

| | | | |
|---|----------------|----|-----------|
|  Bunch_test | 20/12/30 15:53 | 文件 | 28,578 KB |
|  Bunch_train | 21/1/2 15:36 | 文件 | 28,612 KB |

5.3 朴素贝叶斯分类器

利用贝叶斯公式 $P(B|A) = \frac{P(A|B)P(B)}{P(A)}$ ，在给定描述实例的属性值

$\langle a_1, a_2, \dots, a_n \rangle$ ，得到最可能的目标值 V_{MAP} 。

$$v_{MAP} = \arg \max_{v_j} P(v_j | a_1, \dots, a_n)$$

$$v_{MAP} = \arg \max_{v_j \in V} \frac{P(a_1, a_2 \dots a_n | v_j)P(v_j)}{P(a_1, a_2 \dots a_n)} = \arg \max_{v_j \in V} P(a_1, a_2 \dots a_n | v_j)P(v_j)$$

在此基础上，我们利用 TF-IDF 进行改进，使用 TF-IDF 将测试集中单词出现的次数转为一个 0~1 的 TF-IDF 权值，它在原贝叶斯方法的基础上额外考虑的单词的出现频率（重要程度）。与此同时取对数减少乘法运算次数，并在对数内加 1 以处理零概率。因此，计算后验概率式子变为

$$\max(\ln(n_1 * P(x_1 | y_i) + 1) + \ln(n_2 * P(x_2 | y_i) + 1) + \dots + \ln(n_n * P(x_n | y_i) + 1)) * P(y_i)$$

（ n_i 为当前测试集第 i 个词的 TF-IDF 值）。

对于正确率、召回率、F1-score 的计算。

正确率（precision）指的是预测值为 1 且真实值也为 1 的样本在预测值为 1 的所有样本中所占的比例。

$$p = \frac{TP}{TP + FP}$$

召回率（recall）指的是预测值为 1 且真实值也为 1 的样本在真实值为 1 的所有样本中所占的比例。

$$r = \frac{TP}{TP + FN}$$

F1-score，又称为平衡 F 分数，是正确率和召回率的调和平均。

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

本实验的朴素贝叶斯分类器主要有三个步骤：

第一步，通过前面预处理后的文本数据，训练得到在各个类别中出现各个特征词汇的概率，同时得到各类文章的统计概率；

```
def init():
    for j in range(1000): #遍历词汇
        tot_sum = 0.0
        type_sum = np.zeros(10)
        for i in range(50000): #遍历文章
            now_type = int(i/5000)
            now_val = m[i][j]
            type_sum[now_type] += now_val
            tot_sum += now_val
        for t in range(10):
            print("第" + str(j) + "类词在第" + str(t) + "类文本中出现的概率为" , type_sum[t] / tot_sum)
            P[j][t] = type_sum[t] / tot_sum
```

第二步，对于测试集的每一篇文章遍历，然后对每一篇文章的每一个词遍历，去找这个词在训练集中的概率，也就是对训练集中所有该列，分 10 类遍历，求最大。


```

outp=np.zeros((50000,2))
outs=np.zeros((((((((((10,10))))))))))

threadLock = threading.Lock()
threads = []

class myThread(threading.Thread):
    def __init__(self, beginPos):
        threading.Thread.__init__(self)
        self.beginPos = beginPos
    def run(self):
        # 对于每一篇文章遍历，然后对每一个词遍历，去找这个词在训练集中的概率
        # 也就是对训练集中所有该列，分10类遍历，求最大
        for w in range(self.beginPos, self.beginPos + 5000):
            print("正在预测第" + str(w) + "篇")
            s = np.zeros(10)
            for i in range(0, 1000):
                for j in range(0, 10):
                    s[j] += np.log(n[w][i] * P[i][j] + 1)
            pos = GetPos(s, (np.max(s)))
            threadLock.acquire()
            outp[w][0] = int(w / 5000)
            outp[w][1] = int(pos)
            outs[int(w / 5000)][int(pos)] += 1
            threadLock.release()
            print("预测类别:" + str(int(pos)) + ', 实际类别:' + str(int(w / 5000)))

```

第三步，打印混淆矩阵，计算各类和总体正确率、召回率、F1-score。

```

total=0
totalRecall=0
f1=0

for op1 in range(0,10):
    recall=outs[op1][op1]/5000
    totalRecall+=recall
    sum=0
    for i in range(0,10):
        sum+=outs[op1][i]
    sp=outs[op1][op1]/sum
    total+=sp
    f=(2.0*recall*sp)/(sp+recall)
    f1+=f
    print(types[op1] + "类准确率为: " + str(sp)+", 召回率为: "+str(recall)+", f1-score为: "+str(f))

totalRecall=(totalRecall*1.0)/10
total=(total*1.0)/10
f1=(f1*1.0)/10

print("总体正确率为: "+str(total))
print("召回率为: "+str(totalRecall))
print("F1 Score = "+ str(f1))

```

实验结果：平均正确率达到 89.03%，其中最高准确率可达 97.31%。

```
Project
news2110.txt SelectNews.py news1290.txt DataPro_test2.py GetData_2.py news52923.txt DataPro3.py DataPro4.py GetData.py
Run: 补数据时断
正在预测第14998篇
预测类别:2, 实际类别:2
正在预测第14999篇
预测类别:2, 实际类别:2
执行完毕, 用时为: 1937秒
混淆矩阵:
IT 住房 体育 军事 娱乐 教育 汽车 犯罪 能源 股票
IT 4655 66 52 71 16 11 28 26 76 7
住房 153 4629 42 7 4 187 8 3 14 33
体育 67 21 4552 121 38 88 13 1 36 79
军事 105 6 271 4481 26 33 2 32 98 26
娱乐 108 3 96 62 4631 12 4 14 65 5
教育 210 513 282 52 61 3728 188 6 18 110
汽车 242 74 178 23 3 193 4177 21 28 61
犯罪 46 6 40 57 32 24 2 4694 93 6
能源 58 15 130 47 16 0 1 20 4787 6
股票 146 261 250 73 7 135 28 7 13 4888
44254.0
IT类准确率为: 0.883972366148532, 召回率为: 0.931, f1-score为: 0.8628359592215815
住房类准确率为: 0.827493743296389, 召回率为: 0.9258, f1-score为: 0.8738988816511118
体育类准确率为: 0.783872423877516, 召回率为: 0.9184, f1-score为: 0.8419495852251919
军事类准确率为: 0.8956843956843956, 召回率为: 0.8882, f1-score为: 0.8878353843858233
娱乐类准确率为: 0.9595938665561542, 召回率为: 0.9262, f1-score为: 0.9426812619588705
教育类准确率为: 0.8623640998853284, 召回率为: 0.7456, f1-score为: 0.7997425721334335
汽车类准确率为: 0.9573687829475132, 召回率为: 0.8354, f1-score为: 0.8922353946384707
犯罪类准确率为: 0.9738514096185738, 召回率为: 0.9388, f1-score为: 0.9556188925881434
能源类准确率为: 0.9157587548638132, 召回率为: 0.9414, f1-score为: 0.9284823668639852
股票类准确率为: 0.9245411284848245, 召回率为: 0.816, f1-score为: 0.8668862211834697
总体正确率为: 0.8982828978482232
召回率为: 0.8858888888888881
F1 Score = 0.8851998439668319

Process finished with exit code 0
```

5.4 SVM 分类器

这里本实验采用的是 libsvm 的 LinearSVC。

首先介绍一下 svm。支持向量机 (Support Vector Machine, SVM) 是一类按监督学习 (supervised learning) 方式对数据进行二元分类的广义线性分类器 (generalized linear classifier)，其决策边界是对学习样本求解的最大边距超平面 (maximum-margin hyperplane)。SVM 使用铰链损失函数 (hinge loss) 计算经验风险 (empirical risk) 并在求解系统中加入了正则化项以优化结构风险 (structural risk)，是一个具有稀疏性和稳健性的分类器。SVM 可以通过核方法 (kernel method) 进行非线性分类，是常见的核学习 (kernel learning) 方法之一。

接着介绍一下 libsvm。libsvm 是台湾大学林智仁 (Lin Chih-Jen) 教授等开发设计的一个简单、易于使用和快速有效的 SVM 模式识别与回归的软件包，他不但提供了编译好的可在 Windows 系列系统的执行文件，还提供了源代码，方便改进、修改以及在其它操作系统上应用；该软件对 SVM 所涉及的参数调节相对比较少，提供了很多的默认参数，利用这些默认参数可以解决很多问题；并提供了交互检验 (Cross Validation) 的功能。该软件可以解决 C-SVM、 ν -SVM、 ϵ -SVR 和 ν -SVR 等问题，包括基于一对一算法的多类模式识别问题。

而支持向量分类的 SVC (C-Support Vector Classification) 就是基于 libsvm 实现的。其数据拟合的时间复杂度是数据样本的二次方，这使得他很难扩展到 10000 个数据集，当输入是多类别时 (SVM 最初是处理二分类问题的)，通过一对一的方案解决。

LinearSVC (Linear Support Vector Classification)：线性支持向量分类，类似于 SVC，但是其使用的核函数是 "linear" 上边介绍的两种是按照 brf (径向基函数计算的，

其实现也不是基于 LIBSVM，所以它具有更大的灵活性在选择处罚和损失函数时，而且可以适应更大的数据集，他支持密集和稀疏的输入是通过一对一的方式解决的。

LinearSVC 参数解释

C: 目标函数的惩罚系数 C，用来平衡分类间隔 margin 和错分样本的，default C = 1.0;

loss : 指定损失函数

penalty :

dual : 选择算法来解决对偶或原始优化问题。当 $n_samples > n_features$ 时 dual=false。

tol : (default = $1e - 3$) : svm 结束标准的精度;

multi_class: 如果 y 输出类别包含多类，用来确定多类策略， ovr 表示一对多，“crammer_singer”优化所有类别的一个共同的目标

如果选择 “crammer_singer”，损失、惩罚和优化将会被被忽略。

fit_intercept :

intercept_scaling :

class_weight : 对于每一个类别 i 设置惩罚系数 $C = class_weight[i]*C$,如果不给出，权重自动调整为 $n_samples / (n_classes * np.bincount(y))$

verbose: 跟多线程有关

在整个实验中，所有的训练、测试、打印混淆矩阵和正确率等参数的过程都可以通过调用 sklearn.svm 中的函数实现。

```

# 从本地加载数据到内存
with open('G:\AI\Bunch_1w\Bunch_train', 'rb') as file_obj: # word_bag_filepath为本地文件路径
    bunch_train = pickle.load(file_obj)
with open('G:\AI\Bunch_1w\Bunch_test', 'rb') as file_obj_2: # word_bag_filepath为本地文件路径
    bunch_test = pickle.load(file_obj_2)

def metrics_result(actual, predict):
    print("精度: {0:.3f}".format(metrics.precision_score(actual, predict, average='micro')))
    print("召回: {0:.3f}".format(metrics.recall_score(actual, predict, average='micro')))
    print("f1-score: {0:.3f}".format(metrics.f1_score(actual, predict, average='micro')))

X_train = bunch_train.label
y_train = bunch_train.tfidf_weight_matrices
clf = LinearSVC(C=1, tol=1e-5)
begin_time_train = datetime.datetime.now()
clf.fit(y_train, X_train)
end_time_train = datetime.datetime.now()
print("训练完毕, 训练时长为: " + str((end_time_train - begin_time_train).seconds) + "秒")

begin_time_test = datetime.datetime.now()
predicted = clf.predict(bunch_test.tfidf_weight_matrices)
metrics_result(bunch_test.label, predicted)
end_time_test = datetime.datetime.now()
print("预测完毕, 预测时长为: " + str((end_time_test - begin_time_test).seconds) + "秒")

print(classification_report(bunch_test.label, predicted)) # 打印结果

types = ['IT', '住房', '体育', '军事', '娱乐', '教育', '汽车', '犯罪', '能源', '股票']

pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
confusion_matrix = pd.DataFrame(confusion_matrix(bunch_test.label, predicted), columns=types, index=types)
print(confusion_matrix)

```

实验结果：平均正确率达到 92.7%，其中最高准确率可达 98%。

精度: 0.927
召回: 0.927
f1-score:0.927
预测完毕, 预测时长为: 0秒

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| IT | 0.94 | 0.93 | 0.94 | 5000 |
| 住房 | 0.92 | 0.91 | 0.91 | 5000 |
| 体育 | 0.98 | 0.97 | 0.98 | 5000 |
| 军事 | 0.98 | 0.96 | 0.97 | 5000 |
| 娱乐 | 0.95 | 0.96 | 0.96 | 5000 |
| 教育 | 0.91 | 0.94 | 0.93 | 5000 |
| 汽车 | 0.95 | 0.94 | 0.94 | 5000 |
| 犯罪 | 0.87 | 0.90 | 0.88 | 5000 |
| 股票 | 0.90 | 0.90 | 0.90 | 5000 |
| 能源 | 0.86 | 0.86 | 0.86 | 5000 |
| accuracy | | | 0.93 | 50000 |
| macro avg | 0.93 | 0.93 | 0.93 | 50000 |
| weighted avg | 0.93 | 0.93 | 0.93 | 50000 |

| | IT | 住房 | 体育 | 军事 | 娱乐 | 教育 | 汽车 | 犯罪 | 能源 | 股票 |
|----|------|------|------|------|------|------|------|------|------|------|
| IT | 4661 | 19 | 6 | 13 | 55 | 71 | 37 | 42 | 45 | 51 |
| 住房 | 16 | 4538 | 13 | 3 | 9 | 53 | 12 | 137 | 110 | 109 |
| 体育 | 10 | 3 | 4862 | 8 | 31 | 44 | 10 | 20 | 4 | 8 |
| 军事 | 24 | 5 | 4 | 4820 | 50 | 31 | 1 | 37 | 2 | 26 |
| 娱乐 | 32 | 10 | 19 | 15 | 4816 | 43 | 2 | 51 | 6 | 6 |
| 教育 | 25 | 29 | 20 | 12 | 29 | 4691 | 6 | 154 | 1 | 33 |
| 汽车 | 27 | 16 | 13 | 2 | 13 | 9 | 4691 | 77 | 36 | 116 |
| 犯罪 | 34 | 127 | 4 | 20 | 24 | 138 | 43 | 4479 | 12 | 119 |
| 能源 | 79 | 95 | 5 | 5 | 15 | 5 | 29 | 26 | 4509 | 232 |
| 股票 | 31 | 97 | 5 | 28 | 7 | 46 | 119 | 132 | 259 | 4276 |

六、思考与体会

胡天翼：在本次实验中，我通过爬虫获取到了原始文本，并进行文本去噪，使用 Jieba 进行分词，TF-IDF 模型进行文档建模。在这个过程中，我首先进一步的掌握的爬虫脚本的编写过程，懂得如何分析网页，提取所需信息，避免失败等操作。在数据预处理的过程中，我通过反复调节长度筛选范围，TF-IDF 参数，明白了不同维度、满足不同条件的数据与处理方式会带来不同的训练效果。

熊宇：在本次文本分类实验中，我对于朴素贝叶斯的原理及实现过程有了更加深刻的理解，在编程实现朴素贝叶斯的过程中，通过对其进行相关改进的尝试，体会到了数据预处理对于实验繁杂程度及实验结果的损失程度的影响，认识到了人工智能对于数据的依赖能力；在调包实现 SVM 的过程中，通过对比朴素贝叶斯和 libsvm，了解了不同分类器的不同思路，培养了对人工智能、数据科学的兴趣，提高了学习、汲取、进步的能力，受益匪浅。

附录 实验代码

GetData_1.py

```
1. # GetData_1.py
2. # 获取滚动新闻页面 url 合集
3. import requests
4. from bs4 import BeautifulSoup
5. import bs4
6. import re
7.
8. def getUrls(url):
9.     req = requests.get(url).text
10.    if (req == None):
11.        return
12.    bf = BeautifulSoup(req, 'html.parser')
13.    div_bf = bf.find('div', attrs={'class': 'content_list'})
14.    if isinstance(div_bf, bs4.element.Tag):
15.        div_a = div_bf.find_all('div', attrs={'class': 'dd_bt'})
16.        urltxt = open(b'./data/url.txt', 'a', encoding='UTF-8')
17.        for div in div_a:
18.            link = div.find('a').get("href")
19.            urltxt.write(link+'\n')
20.        urltxt.close()
21.
22. years = ['2011', '2012', '2013', '2014', '2015', '2016', '2017', '2018', '20
    19', '2020']
23. months = ["01", "02", "03", "04", "05", "06", "07", "08", "09", "10", "11", "12"]
24. days = ["01", "02", "03", "04", "05", "06", "07", "08", "09", "10", "11", "12", "13", "14
    ", "15", "16", "17", "18", "19", "20", "21", "22", "23", "24", "25", "26", "27", "28", "29"
    , "30", "31"]
25. for year in range(10):
26.     for month in range(12):
27.         print(years[year] + " " + months[month])
28.         for day in range(31):
29.             url = "http://www.chinanews.com/scroll-
                news/" + years[year] + "/" + months[month] + days[day] + "/news.shtml"
30.             print(url)
31.             getUrls(url)
32.             head = requests.head(url)
33.             req = requests.get(url)
34.             req.encoding = 'GB2312'
```



```

35.         bf = BeautifulSoup(req.text, 'html.parser')
36.         div = bf.find('div', attrs={'class': 'content'})
37.         print(div)
38.         if isinstance(div, bs4.element.Tag):
39.             h1 = div.find('h1')
40.             head = re.sub(r'\s+', '', h1.get_text())
41.             out = open('./data/urldata', 'w', encoding='GB2312', errors=
                'ignore')
42.             out.write(head + '\n')
43.             timediv = div.find('div', attrs={'class': 'left-t'})
44.             time = timediv.get_text().replace(" ", "")[0:16]
45.             out.write(time)
46.             p = div.find('div', attrs={'class': 'left_zw'}).find_all('p'
                , text=True)
47.             for ptext in p:
48.                 out.write('\n' + ptext.text);
49.             out.close()

```

GetData_2.py

```

1. # GetData_2.py
2. # 根据 GetData_1.py 获取到的 url.txt 提取新闻文本
3. import requests
4. from bs4 import BeautifulSoup
5. import bs4
6. import re
7. import os
8.
9.
10. urls = open('./data/url.txt').read().split('\n')
11. cnt = 0;
12.
13. for url in urls:
14.     l = url.split('/')
15.     if (url[0] == '/' and len(l) > 2 and l[1].isalpha() and l[1] == 'it' and
        l[2].isdigit()):
16.
17.         # print(url)
18.         # print(l[1], l[2])
19.         Type = l[1];
20.         Year = int(l[2])

```

```
21.         if (Year >= 2012):
22.             cnt = cnt + 1
23.             url = 'http://www.chinanews.com' + url
24.             # url = 'http://www.chinanews.com/cul/2020/11-
25.             23/9345057.shtml'
26.             print(url)
27.             res = requests.get(url)
28.             if (Year < 2019):
29.                 res.encoding = 'GBK' # html: ISO-8859-1 (2012)
30.             else:
31.                 res.encoding = 'utf-8' # res.encoding = 'utf-8' # (2019)
32.             soup = BeautifulSoup(res.text, 'html.parser')
33.             title = soup.find('h1')
34.             if isinstance(title, bs4.element.Tag):
35.                 news_contents = title.text.strip()
36.                 temp = soup.find('div', 'left_zw')
37.                 if isinstance(temp, bs4.element.Tag):
38.                     contents = temp.find_all('p')
39.                     for content in contents:
40.                         if 'function' in content.text:
41.                             continue
42.                         news_contents = news_contents + content.text.strip()
43.
44.
45.             filepath = './data/news/' + Type + '/'
46.             filename = './data/news/' + Type + '/news' + str(cnt) +
47.             '.txt'
48.             if not os.path.exists(filepath):
49.                 os.makedirs(filepath)
50.
51.             if not os.path.exists(filename):
52.                 with open(filename, "w", errors='ignore') as f:
53.                     f.write(news_contents)
54.
55.                 with open(filename, "a", errors='ignore') as f:
56.                     f.write(news_contents)
57.
58.                 f.close()
59.
60.             if (cnt % 10 == 0):
61.                 print(cnt)
```

SelectNews.py

```
1. # SelectNews.py
2. # 筛选合适长度的新闻文本
3. import jieba
4. import jieba.posseg as pseg
5. import os
6. from sklearn.utils import Bunch
7. import pickle
8.
9.
10. types = ['it', '犯罪', '教育', '军事', '能源', '汽车', '体育', '娱乐', '住房',
11.          '']
11. R = {'it': (1900, 2400),
12.       '犯罪': (1800, 2800),
13.       '教育': (1400, 3000),
14.       '军事': (1700, 2800),
15.       '能源': (1600, 2800),
16.       '汽车': (1800, 2400),
17.       '体育': (1900, 2300),
18.       '娱乐': (1800, 2400),
19.       '住房': (1700, 2500)}
20.
21. def is_all_chinese(strs):
22.     for _char in strs:
23.         if (not '\u4e00' <= _char <= '\u9fa5'):
24.             return False
25.     return True
26.
27. for t in types:
28.     dirpath = './data/news/' + t
29.     print(t, ' ', end = '')
30.     cnt = 0
31.     for root, dirs, files in os.walk(dirpath):
32.         # root 表示当前正在访问的文件夹路径
33.         # dirs 表示该文件夹下的子目录名 list
34.         # files 表示该文件夹下的文件 list
35.         # 遍历文件
36.         for f in files:
37.             Path = os.path.join(root, f)
38.             # print(Path)
39.             file_obj = open(Path, 'rb')
40.             doc = file_obj.read()
41.             file_obj.close()
```

```

42.         l = len(doc)
43.         if (l >= R[t][0] and l <= R[t][1]):
44.             cnt += 1
45.             words = pseg.cut(doc)
46.             word_str = ''
47.             first = True
48.             for word in words:
49.                 now_word = str(word.word)
50.                 if (('n' or 1) in word.flag and word.flag != 'nr' and is
                    _all_chinese(now_word)):
51.                     if (first):
52.                         first = False
53.                     else:
54.                         word_str += ' '
55.                         word_str += word.word
56.
57.             Path = os.path.join('./data/Data_Set/' + t, f)
58.             out = open(Path, 'w')
59.             out.write(word_str)
60.             out.close()
61.             print(cnt)
62.         # print(cnt)

```

DataPro_1.py

```

1. # DataPro_1.py
2. # 分词+去噪
3. import sys
4.
5. import jieba
6. import jieba.posseg as pseg
7. import os
8. from sklearn.utils import Bunch
9. from sklearn.feature_extraction.text import TfidfVectorizer
10. import pickle
11.
12. word_bag_filepath_train = './data/raw_Bunch_train'
13. word_bag_filepath_test = './data/raw_Bunch_test'
14. Bunch_train = Bunch(label=[], filenames=[], contents=[])
15. Bunch_test = Bunch(label=[], filenames=[], contents=[])
16.

```

```
17. types = ['IT', '股票', '犯罪', '教育', '军事', '能源', '汽车', '体育', '娱乐', '住房']
18.
19.
20. def is_all_chinese(strs):
21.     for _char in strs:
22.         if (not '\u4e00' <= _char <= '\u9fa5'):
23.             return False
24.     return True
25.
26. for t in types:
27.     dirpath = './data/Data_Set/' + t
28.     print(dirpath)
29.     Train = True
30.     for root, dirs, files in os.walk(dirpath):
31.         # root 表示当前正在访问的文件夹路径
32.         # dirs 表示该文件夹下的子目录名 list
33.         # files 表示该文件夹下的文件 list
34.         # 遍历文件
35.         for f in files:
36.             Path = os.path.join(root, f)
37.             print(Path)
38.             file_obj = open(Path, 'r')
39.             doc = file_obj.read()
40.             file_obj.close()
41.             print(Train)
42.             if (Train):
43.                 Bunch_train.filesnames.append(str(f))
44.                 Bunch_train.label.append(str(t))
45.                 Bunch_train.contents.append(str(doc))
46.             else:
47.                 Bunch_test.filesnames.append(str(f))
48.                 Bunch_test.label.append(str(t))
49.                 Bunch_test.contents.append(str(doc))
50.             Train = not Train
51.
52. with open(word_bag_filepath_train, 'wb') as file_obj1:
53.     pickle.dump(Bunch_train, file_obj1)
54.
55. with open(word_bag_filepath_test, 'wb') as file_obj2:
56.     pickle.dump(Bunch_test, file_obj2)
```

DataPro_2.py

```
1. # DataPro_2.py
2. # TF-IDF 建模
3. import datetime
4.
5. from sklearn.feature_extraction.text import TfidfVectorizer
6. import pickle
7. from sklearn.utils import Bunch
8.
9. bunch_path = './data/raw_Bunch_train'
10.
11. with open(bunch_path, 'rb') as file_obj:
12.     bunch = pickle.load(file_obj)
13. begintime = datetime.datetime.now()
14. vectorizer = TfidfVectorizer(sublinear_tf=True, max_df=0.15, min_df=0.001,ma
    x_features=2000)
15.
16. tfidf_space = Bunch(filenamees = bunch.filenamees,      # 所有文章的文件名
17.                     tfidf_weight_matrices = [],        # TF-IDF 权重矩阵
18.                     vocabulary = {})                   # 词典
19.
20.
21. tfidf_space.tfidf_weight_matrices = vectorizer.fit_transform(bunch.contents)
22.
23. print (vectorizer.get_feature_names())
24. print(tfidf_space.tfidf_weight_matrices)
25.
26. endtime = datetime.datetime.now()
27. print("执行完毕, 用时为: " + str((endtime - begintime).seconds)+ "秒")
28.
29. tfidf_space.vocabulary = vectorizer.vocabulary_
30.
31. with open('./data/Bunch_train', 'wb') as file_obj1:
32.     pickle.dump(tfidf_space, file_obj1)
```

DataPro_Test.py

```
1. # DataPro_Test.py
```

```

2. # 生成测试集字典
3. from sklearn.feature_extraction.text import TfidfVectorizer
4. import pickle
5. from sklearn.utils import Bunch
6.
7. bunch_path = './data/raw_Bunch_test'
8. train_tfidf_path = './data/Bunch_train'
9.
10. with open(bunch_path, 'rb') as file_obj:
11.     bunch = pickle.load(file_obj)
12.
13. with open(train_tfidf_path, 'rb') as file_obj1: # 加载训练集 需要使用字典
14.     trainbunch = pickle.load(file_obj1)
15.
16. tfidf_space = Bunch(    filenames = bunch.filenames,    # 所有文章的文件名
17.                        tfidf_weight_matrices = [],      # TF-IDF 权重矩阵
18.                        vocabulary = trainbunch.vocabulary)
19.
20. vectorizer = TfidfVectorizer(sublinear_tf=True, max_df=0.13, min_df=0.001, v
    ocabulary=trainbunch.vocabulary)
21. tfidf_space.tfidf_weight_matrices = vectorizer.fit_transform(bunch.contents)
22.
23. print (vectorizer.get_feature_names())
24. print(tfidf_space.tfidf_weight_matrices)
25.
26. with open('./data/Bunch_test', 'wb') as file_obj2:
27.     pickle.dump(tfidf_space, file_obj2)
28.
29.

```

朴素贝叶斯.py

```

1. import datetime
2. import pickle
3. import os
4. import sys
5. import pandas as pd
6. import numpy as np
7. from sklearn.utils import Bunch
8. from sklearn.naive_bayes import MultinomialNB

```

```

9. from sklearn.feature_extraction.text import TfidfVectorizer
10. from sklearn import metrics
11. from sklearn.metrics import classification_report, confusion_matrix
12. from sklearn.svm import LinearSVC
13. from sklearn.preprocessing import MinMaxScaler
14. from sklearn.naive_bayes import MultinomialNB
15. from sklearn.model_selection import train_test_split
16. from sklearn.datasets import make_blobs
17. from sklearn.metrics import brier_score_loss
18. import threading
19.
20. bunch_train = Bunch()
21. bunch_test = Bunch()
22.
23. # 从本地加载数据到内存
24. with open('./data/Bunch_train', 'rb') as file_obj: # word_bag_filepath 为本
    地文件路径
25.     bunch_train = pickle.load(file_obj)
26. with open('./data/Bunch_test', 'rb') as file_obj_2: # word_bag_filepath 为本
    地文件路径
27.     bunch_test = pickle.load(file_obj_2)
28.
29.
30. #Bunch(label = bunch.label, # 所有文章的标签
31. #          filenames, # 所有文章的文件名
32. #          tfidf_weight_matrices = [],# TF-IDF 权重矩阵
33. #          vocabulary = {})
34. #
35.
36. #矩阵的格式为 (0,5458,0.14573526)指的是, 第 0+1=1 篇文章, 在词典第 5458 个词上有权
    值, 权值为 0.145..
37.
38. #各类在训练集中的概率都是 0.1, 所以朴素贝叶斯中不予计算
39.
40. #求这个
    max{ (ln(n1 * P(x1|yi) + 1) + ln(n2 * P(x2|yi) + 1) + ..... + ln(nn * P(xn|yi)
    + 1)) * P(yi) }
41. #对于第 j 个词: 第 j 个词出现次数*P (xj|第 i 类) 然后把每类文本所有这些 乘起来
42. #为了避免之前说的 可能为 0, 每个 ln+1, 测试集的矩阵 val, 就等价于出现次数
43.
44. D = 2000 #维度
45.
46. m=bunch_train.tfidf_weight_matrices.toarray();
47. n=bunch_test.tfidf_weight_matrices.toarray();

```



```

48. P = np.zeros((D,10)) #P[j][i] = P(xj|第i类)
49.
50.
51. outp=np.zeros((50000,2))
52. outs=np.zeros((((((((((10,10))))))))))
53.
54. threadLock = threading.Lock()
55. threads = []
56.
57. class myThread (threading.Thread):
58.     def __init__(self, beginPos):
59.         threading.Thread.__init__(self)
60.         self.beginPos = beginPos
61.     def run(self):
62.         # 对于每一篇文章遍历, 然后对每一个词遍历, 去找这个词在训练集中的概率
63.         # 也就是对训练集中所有该列, 分 10 类遍历, 求最大
64.         for w in range(self.beginPos, self.beginPos + 5000):
65.             print("正在预测第" + str(w) + "篇")
66.             s = np.zeros(10)
67.             for i in range(0, D):
68.                 for j in range(0, 10):
69.                     s[j] += np.log(n[w][i] * P[i][j] + 1)
70.             pos = GetPos(s, (np.max(s)))
71.             threadLock.acquire()
72.             outp[w][0] = int(w / 5000)
73.             outp[w][1] = int(pos)
74.             outs[int(w / 5000)][int(pos)] += 1
75.             threadLock.release()
76.             print("预测类别:" + str(int(pos)) + ', 实际类别:' + str(int(w / 5000)))
77.
78.
79. def GetPos(shuzu,zuida):
80.     i=0
81.     objx=tuple(shuzu)
82.     for i in range(len(objx)):
83.         if(objx[i]==zuida):
84.             return i
85.
86. def init():
87.     for j in range(D): #遍历词汇
88.         tot_sum = 0.0
89.         type_sum = np.zeros(10)
90.         for i in range(50000): #遍历文章

```

```

91.         now_type = int(i/5000)
92.         now_val = m[i][j]
93.         type_sum[now_type] += now_val
94.         tot_sum += now_val
95.         for t in range(10):
96.             print("第" + str(j) + "类词在第" + str(t) + "类文本中出现的概率为"
97.                 " ,type_sum[t] / tot_sum)
98.             P[j][t] = type_sum[t] / tot_sum
99.
100. begintime = datetime.datetime.now()
101. init()
102. for i in range(10):
103.     threads.append(myThread(5000 * i))
104.     threads[i].start()
105. for i in range(10):
106.     threads[i].join()
107.
108. endtime = datetime.datetime.now()
109. print("执行完毕, 用时为: " + str((endtime - begintime).seconds)+ "秒")
110. print("混淆矩阵:")
111. types = ['IT', '住房', '体育', '军事', '娱乐', '教育', '汽车', '犯罪', '能源', '股票'
112.          '']
113. for op1 in range(0,10):
114.     print(types[op1]+" ",end="")
115. print(" ")
116.
117. for op1 in range(0,10):
118.     print(types[op1]+" ",end="")
119.     for op2 in range(0,10):
120.         print(str(int(outs[op1][op2]))+" ",end="")
121.     print(" ")
122.
123. success=0
124. for op1 in range(0,10):
125.     success+=outs[op1][op1]
126.
127. print(success)
128. total=0
129. totalRecall=0
130. f1=0
131.
132. for op1 in range(0,10):

```

```

133.     recall=outs[op1][op1]/5000
134.     totalRecall+=recall
135.     sum=0
136.     for i in range(0,10):
137.         sum+=outs[i][op1]
138.     sp=outs[op1][op1]/sum
139.     total+=sp
140.     f=(2.0*recall*sp)/(sp+recall)
141.     f1+=f
142.     print(types[op1] + "类准确率为: " + str(sp)+"，召回率为:
        "+str(recall)+"， f1-score 为: "+str(f))
143.
144. totalRecall=(totalRecall*1.0)/10
145. total=(total*1.0)/10
146. f1=(f1*1.0)/10
147.
148. print("总体正确率为: "+str(total))
149. print("召回率为: "+str(totalRecall))
150. print("F1 Score = "+ str(f1))

```

SVC.py

```

1.  #SVC
2.  import numpy as np
3.  import os
4.  import time
5.  import codecs
6.  import re
7.  from sklearn.feature_extraction.text import CountVectorizer
8.  from sklearn.feature_extraction.text import TfidfTransformer
9.  from sklearn.naive_bayes import MultinomialNB
10. from sklearn.svm import SVC
11. from sklearn.pipeline import Pipeline
12. from sklearn import metrics
13. import datetime
14. import pickle
15. import os
16. import sys
17. import pandas as pd
18. import numpy as np
19. from sklearn.utils import Bunch
20. from sklearn.naive_bayes import MultinomialNB
21. from sklearn.feature_extraction.text import TfidfVectorizer
22. from sklearn import metrics

```

```

23. from sklearn.metrics import classification_report, confusion_matrix
24. from sklearn.svm import LinearSVC
25. from sklearn.preprocessing import MinMaxScaler
26. from sklearn.naive_bayes import MultinomialNB
27. from sklearn.model_selection import train_test_split
28. from sklearn.datasets import make_blobs
29. from sklearn.metrics import brier_score_loss
30.
31. bunch_train = Bunch()
32. bunch_test = Bunch()
33.
34. # 从本地加载数据到内存
35. with open('G:\AI\Bunch_1w\Bunch_train', 'rb') as file_obj:
36.     bunch_train = pickle.load(file_obj)
37. with open('G:\AI\Bunch_1w\Bunch_test', 'rb') as file_obj_2:
38.     bunch_test = pickle.load(file_obj_2)
39.
40. def metrics_result(actual, predict):
41.     print("精度:
42.         {0:.3f}".format(metrics.precision_score(actual, predict, average='micro')))
43.     print("召回:
44.         {0:.3f}".format(metrics.recall_score(actual, predict, average='micro')))
45.     print("f1-
46.         score:{0:.3f}".format(metrics.f1_score(actual, predict, average='micro')))
47.
48. X_train = bunch_train.label
49. y_train = bunch_train.tfidf_weight_matrices
50. clf = LinearSVC(C=1, tol=1e-5)
51. begintime_train = datetime.datetime.now()
52. clf.fit(y_train, X_train)
53. endtime_train = datetime.datetime.now()
54. print("训练完毕, 训练时长为:
55.     " + str((endtime_train - begintime_train).seconds)+ "秒")
56.
57. begintime_test = datetime.datetime.now()
58. predicted = clf.predict(bunch_test.tfidf_weight_matrices)
59. metrics_result(bunch_test.label, predicted)
60. endtime_test = datetime.datetime.now()
61. print("预测完毕, 预测时长为:
62.     " + str((endtime_test - begintime_test).seconds)+ "秒")
63.
64. print(classification_report(bunch_test.label, predicted)) # 打印结果

```

```
60.  
61. types = ['IT', '住房', '体育', '军事', '娱乐', '教育', '汽车', '犯罪', '能源', '股票'  
    '']  
62.  
63. pd.set_option('display.max_columns', None)  
64. pd.set_option('display.max_rows', None)  
65. confusion_matrix = pd.DataFrame(confusion_matrix(bunch_test.label, predicted  
    ), columns=types, index=types)  
66. print(confusion_matrix)
```