

# 贝叶斯网络编程作业

## 贝叶斯网络编程作业

### TASK1:使用朴素贝叶斯过滤垃圾邮件

#### 一、软件设计

##### 1. 设计目标

###### ① 目标分析

##### 2. 算法设计分析

###### ① 朴素贝叶斯

###### ② 代码实现

```
===== >> 数据预处理 <<=====
===== >> 贝叶斯算法实现 <<=====
===== >> main函数 <<=====
```

#### 二、运行结果

截图一\_文件目录

截图二\_运行结果，少数循环

截图三\_运行结果，多次循环

#### 三、总结体会

程序设计心得:

### TASK2: 使用朴素贝叶斯对搜狗新闻语料库进行分类

#### 一、软件设计

##### 1. 设计目标

###### ① 数据集讲解:

###### ② 设计提示思路

###### ③ 目标要求

##### 2. 算法设计分析

###### ① 贝叶斯算法

###### ② TD-IDF

###### ② 代码实现

```
===== >> 数据处理 <<=====
===== >> 朴素贝叶斯分类器 <<=====
===== >> main函数 <<=====
```

#### 二、运行结果

截图一,词袋模型\_默认参数

截图二, TF\_IDF模型

截图三, 不设参数多次平均

#### 三、总结体会

程序设计心得:

### TASK3: 使用朴素贝叶斯对电影评论分类

#### 一、软件设计

##### 1. 设计目标

###### ① 数据集讲解:

###### ② 目标分析

##### 2. 算法设计分析

###### ① 决策树算法

###### ② 数据处理

###### ③ 文件操作

###### ④ main函数

#### 二、运行结果

结果截图1

结果截图二

#### 三、总结体会

程序设计心得:

# TASK1:使用朴素贝叶斯过滤垃圾邮件

## 一、软件设计

### 1. 设计目标

#### ① 目标分析

- 现有50封电子邮件，存放在数据集task1中，试基于朴素贝叶斯分类器原理，用Python编程实现对垃圾邮件和正常邮件的分类。采用交叉验证方式并且输出分类的错误率及分类错误的文档

### 2. 算法设计分析

所用项目环境 IDE: VS code    Python版本: python3.7.5

[跳转回task2](#) [跳转回task3](#)

#### ①朴素贝叶斯

##### • 贝叶斯定理

条件概率 (conditional probability) 是指在事件 B 发生的情况下，事件 A 发生的概率。通常记为  $P(A | B)$

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

因此

$$P(A \cap B) = P(A|B) P(B)$$

$$P(A \cap B) = P(B|A) P(A)$$

可得

$$P(A|B) P(B) = P(B|A) P(A)$$

由此可以推出贝叶斯公式

$$P(A|B) = \frac{P(B|A) P(A)}{P(B)}$$

这也是条件概率的计算公式

##### • 贝叶斯推断

(1)贝叶斯公式中， $P(A)$ 称为"先验概率" (Prior probability)，即在B事件发生之前，对A事件概率的一个判断。 $P(A|B)$ 称为"后验概率" (Posterior probability)，即在B事件发生之后，对A事件概率的重新评估。 $P(B|A)/P(B)$ 称为"可能性函数"，这是一个调整因子，使得预估概率更接近真实概率。

所以，条件概率可以理解成下面的式子：后验概率 = 先验概率 × 调整因子

(2)这就是贝叶斯推断的含义。我们先预估一个"先验概率"，然后加入实验结果，看这个实验到底是增强还是削弱了"先验概率"，由此得到更接近事实的"后验概率"。因为在分类中，只需要找出可能性最大的那个选项，而不需要知道具体那个类别的概率是多少，所以为了减少计算量，全概率公式在实际编程中可以不使用。

(3)而朴素贝叶斯推断，是在贝叶斯推断的基础上，对条件概率分布做了条件独立性的假设。因此可得朴素贝叶斯分类器的表达式。因为以自变量之间的独立（条件特征独立）性和连续变量的正态性假设为前提，就会导致算法精度在某种程度上受影响。

$$\hat{y} = \arg \max_{c \in Y} P(c) \prod_{i=1}^d P(x_i | c)$$

### • 朴素贝叶斯的参数推断

朴素贝叶斯分类器的训练过程就是基于训练集  $D$  来估计类先验概率  $P(c)$ ，并为每个属性估计条件概率  $P(x_i | c)$ 。这里就需要使用极大似然估计 (maximum likelihood estimation, 简称 MLE) 来估计相应的概率。

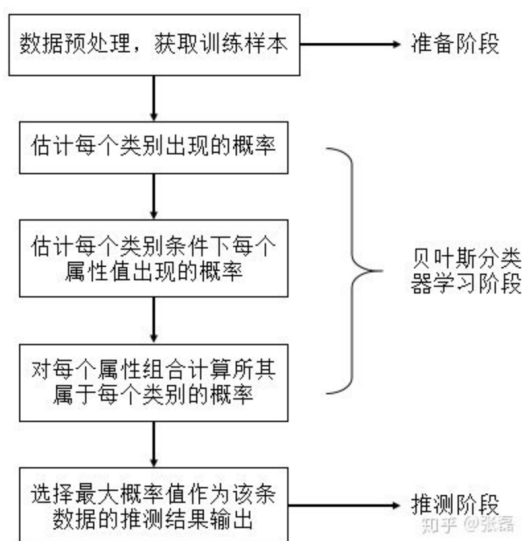
令  $D_c$  表示训练集  $D$  中的第  $c$  类样本组成的集合，若有充足的独立同分布样本，则可容易地估计出类别的先验概率：

$$P(c) = \frac{|D_c|}{D}$$

对于离散属性而言，令  $D_{c,x_i}$  表示  $D_c$  中在第  $i$  个属性上取值为  $x_i$  的样本组成的集合，则条件概率  $P(x_i | c)$  可估计为：

$$P(x_i | c) = \frac{|D_{c,x_i}|}{|D_c|}$$

### • 朴素贝叶斯算法流程：



贝叶斯方法流程

## ②代码实现

===== >> 数据预处理 <<=====

**算法介绍：**对提供的训练文件进行读取，并将邮件内容切分为单词训练向量，采用词袋模型构建向量

**数据整理：**拿到手的数据并不能直接使用，我们需要对数据进行预处理，变成向量的形式（这里采用**词袋模型**）。可以首先把所有字符转换成小写，去掉大小字符不统一的影响；然后构建一个包含在所有文档中出现的不重复的词列表；获得词汇表后，根据某个单词在一篇文档中出现的次数获得文档向量（这三步的代码可参考给出的demo）；最后利用构建的分类器进行训练。

- **文本切分函数：**将传入的文本字符串切分为单词列表，并去除非字母，下划线等特殊符号

```
# 准备数据：切分文本
def textParse(bigString):
    # listOfTokens = re.split(r'\w*', bigString) # 匹配非字母数字下划线
    listOfTokens = re.split(r'\W', bigString) # 匹配非字母数字下划线
    return [tok.lower() for tok in listOfTokens]
    if len(tok) > 2] # 若文本中有URL，对其进行切分时，会得到很多词，为避免该情况，限定字符创的长度
```

- 代码细节：

- (1) 采用正则表达式 `r'\W'` 匹配特殊字符，利用 `re.split()` 函数进行切分，形成列表
- (2) 对于长度小于2的单词，舍弃无效单词

- 词汇表函数：将传入的数据去重构成总的词汇表

```
# 将文档矩阵中的所有词构成词汇表
def creatVocabList(dataset):
    vocabSet = set([])
    for document in dataset:
        vocabSet = vocabSet | set(document) # 两个集合的并集
    return list(vocabSet)
```

- 代码细节：

- (1) `set()` 函数去重即可，`set` 对象不利于后续操作，进行类型转化为 `list` 后返回

- 词袋构建函数：根据传入的词汇表，对传入的数据构建出词袋模型的数据向量

```
# 将某一文档转换成词向量，该向量中所含数值数目与词汇表中词汇数目相同 词袋模型
def bagOfWords2Vec(vocabList, inputSet): # 参数分别为词汇表，输入文档
    returnVec = [0] * len(vocabList)
    for word in inputSet:
        if word in vocabList:
            returnVec[vocabList.index(word)] += 1
    return returnVec
```

- 代码细节：

- (1) `for` 循环依次遍历，将对应单词位置加1即可

## =====>> 贝叶斯算法实现 <<=====

**算法介绍：**利用处理好的数据，进行朴素贝叶斯分类器的构建，并进行邮件分类计算错误率

读取文件夹 `spam` 与 `ham` 下的文本文件，并将它们解析为词列表。接下来构建一个训练集和测试集，两个集合中的邮件都是随机选出的。经过一次迭代就能输出分类的错误率及分类错误的文档。

(注：由于测试集的选择是随机的，所以测试算法时每次的输出结果可能有些差别，如果想要更好的估计错误率，最好将上述过程重复多次，然后求平均值。)

- 朴素贝叶斯分类器训练函数：对处理好的数据向量矩阵计算相关特征的概率，用于生成分类器

```
# 朴素贝叶斯分类器训练函数,trainMatrix为文档词向量矩阵,trainCategory为每篇文档的类标签构成的向量
#  $p(c_i|w)=p(w|c_i)*p(c_i)/p(w)$  假设使其判定为垃圾邮件的每个标签 $w_i$ 互相独立
# 则有 $p(w|c_i)=p(w_0|c_i)p(w_1|c_i)...p(w_N|c_i)$ 
def trainNB0(trainMatrix, trainCategory):
    ....
    p1Vect = np.log(p1Num / p1Deom) # 在垃圾文档条件下词汇表中单词的出现概率
    p0Vect = np.log(p0Num / p0Deom) # 采用对数是为了解决下溢问题
    # pAbusive就是文档属于垃圾文档的概率 $p(c_i)$ , 先验概率
    return p0Vect, p1Vect, pAbusive
```

- 代码细节:

- (1) 避免某一个特征为0导致总结果为0,故采用拉普拉斯变换平滑,将分子pnum初始化为1,将分母pdeom初始化为标签个数2
- (2) 计算概率时,采用log将其转化为对数,为了避免概率过小,出现下溢的情况

- 朴素贝叶斯分类函数: 利用训练好的分类器判定输入的文档内容为垃圾邮件与否

```
# 朴素贝叶斯分类函数
# # 参数分别为: 要分类的向量以及使用trainNB0()计算得到的三个概率
def classifyNB(vec2Classify, p0Vec, p1Vec, pClass):
    p1 = sum(vec2Classify * p1Vec) + np.log(pClass)
    p0 = sum(vec2Classify * p0Vec) + np.log(1 - pClass)
    if p1 > p0:
        return 1 # 表示侮辱性文档
    else:
        return 0
```

- 代码细节:

- (1) 因为将概率转化为log对数形式了,所以 $p(w|c_i)=p(w_0|c_i)p(w_1|c_i)...p(w_N|c_i)$ 应该转为对数的求和
- (2) 计算文档为垃圾邮件与非垃圾邮件的概率,比较大小进行判定

- 朴素贝叶斯分类器测试函数: 训练朴素贝叶斯分类器,同时交叉验证,测试分类器的精度,保留分类错误的邮件

```
# 测试算法: 使用朴素贝叶斯交叉验证。同时保存分类模型的词汇表以及三个概率值,避免判断时重复求值
def spamTest():
    ..... # 具体代码省略部分
    for docIndex in testSet:
        wordVector = bagOfWords2Vec(vocabList, docList[docIndex])
        if classifyNB(wordVector, p0v, p1v, pAb) != classList[docIndex]:
            errorCount += 1
            if docIndex in test_error:
                pass
            else:
                test_error.append(docIndex)
    return (float(errorCount) / len(testSet)), test_error
```

- 代码细节:

- (1) 训练过程中生成的相关概率，及单词表可以保存为txt文件，用于后续直接读取
- (2) 交叉验证，将50封邮件设置10个测试集，40训练集，随机乱序。

- **分类器估计函数：**读取保存下来的分类器参数文件生成分类器，预估传入的文件的标签

```
def fileClassify(filepath):
    filewordList = textParse(open(filepath, mode='r').read())
    file = open('task1/result/vocabList.txt', mode='rb')
    vocabList = pickle.load(file)
    vocabList = vocabList
    filewordVec = bagOfWords2Vec(vocabList, filewordList) # 被判断文档的向量
    file = open('task1/result/threeRate.txt', mode='rb')
    rate = pickle.load(file)
    p0v = rate[0]
    p1v = rate[1]
    pAb = rate[2]
    return classifyNB(filewordVec, p0v, p1v, pAb)
```

- **代码细节：**

- (1) 调用classifyNB函数进行标签预测，使用python文件操作读取文件。

===== >> main函数 <<=====

**算法介绍：**调用上述编写的算法函数，实现邮件分类

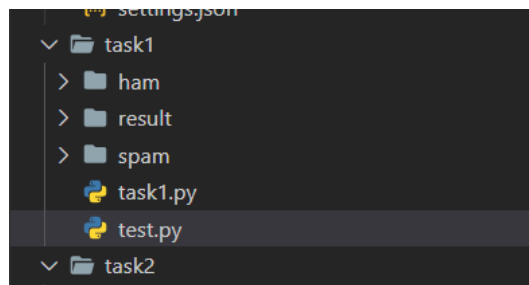
```
if __name__ == '__main__':
    rate = [0] * 100
    errorList, errorspam, errorham = [], [], []
    for i in range(0, 100):
        rate[i], test_error = spamTest()
        errorList = list(set(errorList) | set(test_error))
        # print('第{}次测试，朴素贝叶斯分类的错误率为: {}'.format(i + 1, rate[i])) #
    测试算法的错误率
    rate_average = np.mean(rate)
    print("共%d次测试，朴素贝叶斯分类平均错误率为: %f" % (i + 1, rate_average))
    for temp in errorList:
        if ((temp + 1) % 2 == 1):
            errorspam.append(int((temp + 3) / 2))
        else:
            errorham.append(int((temp + 1) / 2))
    print('预测错误的邮件为：垃圾邮件spam索引: {} 非垃圾邮件ham索引:
    {}'.format(errorspam, errorham))
```

- **代码细节：**

- (1) 循环调用100次，对于每一次预测错误的邮件进行保存，并用set去重
- (2) 使用np.mean()函数求列表平均值，计算平均精度。

## 二、运行结果

## 截图一\_文件目录



- 图片分析：上图显示项目文件目录，task1.py与test.py文件均为代码文件，result文件夹保存结果

## 截图二\_运行结果，少数循环

```
PS F:\own_file\课程文件\机器学习导论\贝叶斯模型\贝叶斯模型> python -u "f:\own_file\课程文件\机器学习导论\贝叶斯模型\贝叶斯模型\task1\test.py"
第1次测试，朴素贝叶斯分类的错误率为：0.0
第2次测试，朴素贝叶斯分类的错误率为：0.1
第3次测试，朴素贝叶斯分类的错误率为：0.0
第5次测试，朴素贝叶斯分类的错误率为：0.0
第6次测试，朴素贝叶斯分类的错误率为：0.0
第7次测试，朴素贝叶斯分类的错误率为：0.1
第8次测试，朴素贝叶斯分类的错误率为：0.1
第9次测试，朴素贝叶斯分类的错误率为：0.0
第10次测试，朴素贝叶斯分类的错误率为：0.1
10次测试，朴素贝叶斯分类平均错误率为：0.040000
垃圾邮件
PS F:\own_file\课程文件\机器学习导论\贝叶斯模型\贝叶斯模型> python -u "f:\own_file\课程文件\机器学习导论\贝叶斯模型\贝叶斯模型\task1\test.py"
共50次测试，朴素贝叶斯分类平均错误率为：0.076000
垃圾邮件
PS F:\own_file\课程文件\机器学习导论\贝叶斯模型\贝叶斯模型> []
```

- 图片分析：将主函数循环执行10次，显示出错误率不稳定，0~0.1之间浮动，取平均值的结果为：**10次测试错误率平均值为0.04。**

将主函数循环设为50次，错误率平均值为0.07，可以得出手写算法的精度还是比较高的

## 截图三\_运行结果，多次循环

```
PS F:\own_file\课程文件\机器学习导论\贝叶斯模型\贝叶斯模型> python -u "f:\own_file\课程文件\机器学习导论\贝叶斯模型\贝叶斯模型\task1\test.py"
共100次测试，朴素贝叶斯分类平均错误率为：0.062000
预测错误的邮件为：垃圾邮件spam索引：[17, 6] 非垃圾邮件ham索引：[17, 2, 23, 10, 14, 16]
PS F:\own_file\课程文件\机器学习导论\贝叶斯模型\贝叶斯模型> []
```

- 图片分析：将主函数循环100次取循环，保留每次测试错误的邮件，共同结果如上图所示。**错误率平均值为0.06左右**

```
PS F:\own_file\课程文件\机器学习导论\贝叶斯模型\贝叶斯模型> python -u "f:\own_file\课程文件\机器学习导论\贝叶斯模型\贝叶斯模型\task1\test.py"
共100次测试，朴素贝叶斯分类平均错误率为：0.064000
预测错误的邮件为：垃圾邮件spam索引：[17, 6, 25] 非垃圾邮件ham索引：[17, 2, 23, 14, 16]
PS F:\own_file\课程文件\机器学习导论\贝叶斯模型\贝叶斯模型> python -u "f:\own_file\课程文件\机器学习导论\贝叶斯模型\贝叶斯模型\task1\test.py"
共1000次测试，朴素贝叶斯分类平均错误率为：0.070500
预测错误的邮件为：垃圾邮件spam索引：[2, 6, 14, 17, 19, 20, 22, 25] 非垃圾邮件ham索引：[1, 2, 10, 14, 16, 17, 22, 23]
PS F:\own_file\课程文件\机器学习导论\贝叶斯模型\贝叶斯模型> python -u "f:\own_file\课程文件\机器学习导论\贝叶斯模型\贝叶斯模型\task1\test.py"
共1000次测试，朴素贝叶斯分类平均错误率为：0.062900
预测错误的邮件为：垃圾邮件spam索引：[4, 6, 14, 17, 20, 21, 22, 25] 非垃圾邮件ham索引：[2, 4, 6, 10, 14, 16, 17, 23]
PS F:\own_file\课程文件\机器学习导论\贝叶斯模型\贝叶斯模型> python -u "f:\own_file\课程文件\机器学习导论\贝叶斯模型\贝叶斯模型\task1\test.py"
共1000次测试，朴素贝叶斯分类平均错误率为：0.070300
预测错误的邮件为：垃圾邮件spam索引：[17, 6, 25] 非垃圾邮件ham索引：[17, 2, 4, 23, 10, 14, 16]
PS F:\own_file\课程文件\机器学习导论\贝叶斯模型\贝叶斯模型> python -u "f:\own_file\课程文件\机器学习导论\贝叶斯模型\贝叶斯模型\task1\test.py"
共500次测试，朴素贝叶斯分类平均错误率为：0.063400
预测错误的邮件为：垃圾邮件spam索引：[17, 6, 25] 非垃圾邮件ham索引：[17, 2, 4, 23, 10, 14, 16]
PS F:\own_file\课程文件\机器学习导论\贝叶斯模型\贝叶斯模型> python -u "f:\own_file\课程文件\机器学习导论\贝叶斯模型\贝叶斯模型\task1\test.py"
共500次测试，朴素贝叶斯分类平均错误率为：0.070000
预测错误的邮件为：垃圾邮件spam索引：[17, 20, 6, 22, 25] 非垃圾邮件ham索引：[17, 2, 23, 10, 14, 16]
PS F:\own_file\课程文件\机器学习导论\贝叶斯模型\贝叶斯模型> []
```

- 图片分析：上图为后续多次测试结果的一部分展示，最终结论为，**错误率平均值为0.06左右浮动**

## 三、总结体会



1. 本次算法过程中，遇到程序错误在正常不过，知错能改，善莫大焉，一个完整的工程背后一定是一次的debug，自身犯错最多的地方在于差错处理，可以多考虑极端情况，多进行程序的调试，对每一种特殊情况进行处理，避免程序运行的自行结束。通过这次软件课设，自身养成了不惧困难，不退缩，提高思维能力
2. 软件工程量较大，建议采用模块化设计，同时不同的内容可以分为不同的文件，优先设计函数头，确定参数及返回值后具体设计逻辑  
切记不要一个文件写完，那样导致调试修改复杂，不利于后续修改。
3. 朴素贝叶斯算法的构建，重点在于对于贝叶斯数学公式的理解，同时学会了词袋模型和词库模型处理数据，学会交叉验证方法，遇到难以解决的问题可以先单步调试，找到问题一步步修改

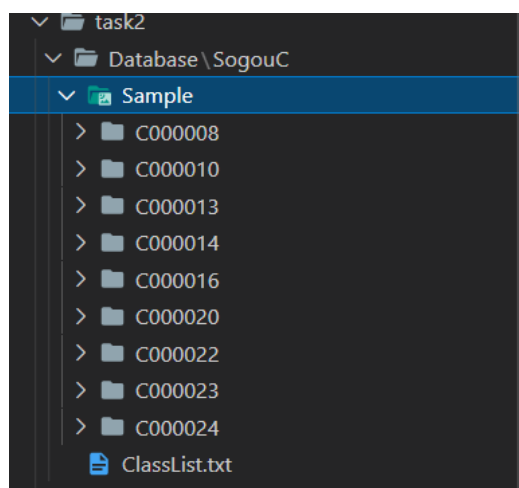
## TASK2: 使用朴素贝叶斯对搜狗新闻语料库进行分类

### 一、软件设计

#### 1. 设计目标

##### ① 数据集讲解:

数据集存放在task2/Database中，新闻一共分为9个类：财经、IT、健康、体育、旅游、教育、招聘、文化、军事，不同种类的新闻分别放在相应代号的文件夹中。NBC.py中，对新闻进行预处理的模块已经给出，请尝试补全分类器模块的代码（NBC.py中函数TextClassifier），自行划分训练集和测试集，对这些新闻进行训练和分类。



##### ② 设计提示思路

1. **数据预处理**：对语料库中的文本进行分词，并通过词频找到特征词，从而生成相应的训练集和测试集。这部分代码我们已经给出，请尝试看懂并理解它们。
2. **机器学习库sklearn**：可以手写朴素贝叶斯分类器，但是在本任务中，允许使用机器学习库sklearn快速实现分类器，以方便研究不同的参数设置对实验结果的影响，探讨影响朴素贝叶斯分类器分类效果的原因。
3. **分词工具**：本任务需要你安装分词库jieba。安装方法：pip install jieba。



### ③ 目标要求

- 可以在构建词典中尝试删去不同个数的高频词，观察实验结果的变化；
- 更改不同的特征词数量，观察对分类准确率的影响
- 或者更改训练集和测试集划分比例，观察不同训练集规模对实验结果的影响
- 也可以更换特征词的提取方式，如TF-IDF等等。

## 2. 算法设计分析

所用项目环境 IDE: VS code    Python版本: python3.7.5

### ① 贝叶斯算法

TASK1中详细描述了贝叶斯算法的实现，TASK2中调用sklearn库

下面就不再一一叙述，点击下方跳转到task1相关位置

[跳转到task1贝叶斯算法](#)

### ② TD-IDF

#### 1. 概念

- (1)TF-IDF (term frequency-inverse document frequency) 是一种用于资讯检索与资讯探勘的常用加权技术。
- (2)TF-IDF是一种统计方法，用以评估一字词对于一个文件集或一个语料库中的其中一份文件的重要程度。
- (3)字词的重要性随着它在文件中出现的次数成正比增加，但同时会随着它在语料库中出现的频率成反比下降。
- (4)TF-IDF加权的各种形式常被搜寻引擎应用，作为文件与用户查询之间相关程度的度量或评级。

#### 2. 主要思想

- (1)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。
- (2)TFIDF实际上是：TF \* IDF，TF词频(Term Frequency)，IDF反文档频率(Inverse Document Frequency)。TF表示词条在文档d中出现的频率（另一说：TF词频(Term Frequency)指的是某一个给定的词语在该文件中出现的次数）。
- (3)IDF的主要思想是：如果包含词条t的文档越少，也就是n越小，IDF越大（见后续公式），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。（另一说：IDF反文档频率(Inverse Document Frequency)是指果包含词条的文档越少，IDF越大，则说明词条具有很好的类别区分能力。）
- (4)但是实际上，有时候，如果一个词条在一个类的文档中频繁出现，则说明该词条能够很好代表这个类的文本的特征，这样的词条应该给它们赋予较高的权重，并选来作为该类文本的特征词以区别与其它类文档。这就是IDF的不足之处。

### ② 代码实现

较长的函数具体代码没有给出，给出函数定义及返回值

===== >> 数据处理 <<=====

**算法介绍：**将训练集的数据进行处理，使用词袋模型或者TF-IDF

- **文件读取算法：**

```
def MakeWordsSet(words_file):
    words_set = set()
    with open(words_file, 'r', encoding='utf-8') as fp:
        for line in fp.readlines():
            word = line.strip().encode('utf-8').decode("utf-8")
            if len(word) > 0 and word not in words_set: # 去重
                words_set.add(word)
    return words_set
```

- 代码细节:

- (1) 读取文件数据后, 用set去除
- (2) words\_file - 文件路径, words\_set - 读取的内容的set集合

- 文本处理及训练集测试集划分算法:

```
def TextProcessing(folder_path, test_size=0.2):
    folder_list = os.listdir(folder_path)
    data_list = []
    class_list = []

    .....

    return all_words_list, train_data_list, test_data_list, train_class_list,
    test_class_list
```

- 代码细节:

Parameters:

folder\_path - 文本存放的路径

test\_size - 测试集占比, 默认占所有数据集的百分之20

Returns:

all\_words\_list - 按词频降序排序的训练集列表

train\_data\_list - 训练集列表

test\_data\_list - 测试集列表

train\_class\_list - 训练集标签列表

test\_class\_list - 测试集标签列表

- (1) word\_cut = jieba.cut(raw, cut\_all=False) 精确模式, 返回的结构是一个可迭代的generator, 适合文本分析, 占用内存小
- (2) random.shuffle(data\_class\_list) 随机打乱数据来划分训练集和测试集
- (3) 数据的标签即为文件夹的名字, 循环读取文件目录保存为标签列表

- 文本特征读取算法:

```
def words_dict(all_words_list, deleteN, stopwords_set=set()):
    feature_words = []
    n = 1
    for t in range(deleteN, len(all_words_list), 1):
        if n > 1000: # feature_words词袋的维度1000
            break
        # 如果这个词不是数字，并且不是指定的结束语，并且单词长度大于1小于5，那么这个词就可以作为feature_word
        if not all_words_list[t].isdigit() and all_words_list[t] not in stopwords_set and 1 < len(all_words_list[t]) < 5:
            feature_words.append(all_words_list[t])
        n += 1
    return feature_words
```

- 代码细节:

Parameters:

all\_words\_list - 训练集所有文本列表  
 deleteN - 删除词频最高的deleteN个词  
 stopwords\_set - 指定的结束语

Returns:

feature\_words - 特征集

- (1) 读取停用词文件，若存在其中，便不是特征
- (2) 删除出现频率最高的特征词，同时特征词只选择单词长度大于1小于5

- 数据编码向量化算法:

```
def TextFeatures(train_data_list, test_data_list, feature_words):
    .....
    return train_feature_list, test_feature_list
```

- 代码细节:

Parameters:

train\_data\_list - 训练集  
 test\_data\_list - 测试集  
 feature\_words - 特征集

Returns:

train\_feature\_list - 训练集向量化列表  
 test\_feature\_list - 测试集向量化列表

- (1) 采用词袋模型
- (2) 删除出现频率最高的特征词，同时特征词只选择单词长度大于1小于5

- 数据编码TF\_IDF算法:

```
def TextFeatures_tf(train_data_list, test_data_list, stopwords):
    # from sklearn.feature_extraction.text import TfidfVectorizer
    # 使用tf-idf把文本转为向量
    tfidf_transformer = TfidfVectorizer(stop_words=stopwords, max_features=1000,
lowercase=False)
    train_feature_list = tfidf_transformer.fit_transform(train_data_list)
    test_feature_list = tfidf_transformer.transform(test_data_list)
    return train_feature_list, test_feature_list
```

- 代码细节:

Parameters:

train\_data\_list - 训练集

test\_data\_list - 测试集

stop\_words - 停止词

Returns:

train\_feature\_list - 训练集向量化列表

test\_feature\_list - 测试集向量化列表

- (1) 采用TF\_IDF, 调用sklearn库函数TfidfVectorizer
- (2) 去除停用词, 设置最大特征数量为1000

## =====>> 朴素贝叶斯分类器 <====

**算法介绍:** 利用得到的决策树生成相对应的分类器, 对测试集的数据进行测试

- 分类器生成函数:

```
def TextClassifier(train_feature_list, test_feature_list, train_class_list,
test_class_list):
    # from sklearn.naive_bayes import MultinomialNB
    classifier = MultinomialNB().fit(train_feature_list, train_class_list)
    test_accuracy = classifier.score(test_feature_list, test_class_list)
    return test_accuracy
```

- 代码细节:

Parameters:

train\_feature\_list - 训练集向量化的特征文本

test\_feature\_list - 测试集向量化的特征文本

train\_class\_list - 训练集分类标签

test\_class\_list - 测试集分类标签

Returns:

test\_accuracy - 分类器精度

- (1) 调用sklearn库函数MultinomialNB直接生成朴素贝叶斯分类器
- (2) 采用classifier.score计算模型精度

**算法介绍：**调用上述编写的算法函数，实现新闻文本分类

```
def main():
    # 文本预处理
    folder_path = './task2/Database/SogouC/Sample'
    all_words_list, train_data_list, test_data_list, train_class_list,
    test_class_list = TextProcessing(folder_path, test_size=0.2)
    # 生成stopwords_set
    stopwords_file = './task2/stopwords_cn.txt'
    stopwords_set = MakeWordsSet(stopwords_file)
    # 文本特征提取和分类
    deleteN = 450
    feature_words = words_dict(all_words_list, deleteN, stopwords_set)
    train_feature_list, test_feature_list = TextFeatures(train_data_list,
    test_data_list, feature_words)
    # train_feature_list, test_feature_list =
    TextFeatures_tf(list(train_data_list), list(test_data_list), list(stopwords_set))
    # 采用TF_IDF
    # 生成朴素贝叶斯分类器并得出精度
    accuracy = TextClassifier(train_feature_list, test_feature_list,
    train_class_list, test_class_list)
    # print('\n朴素贝叶斯分类器分类新闻文本的精度为: ', accuracy)
    return accuracy
```

- 代码细节：

(1) 调用之前设计的数个函数即可

## 二、运行结果

### 截图一,词袋模型\_默认参数

```
PS F:\own_file\课程文件\机器学习导论\贝叶斯模型\贝叶斯模型> python -u "f:\own_file\课程文件\机器学习导论\
Building prefix dict from the default dictionary ...
Loading model from cache C:\temp\jieba.cache
Loading model cost 0.844 seconds.
Prefix dict has been built successfully.
['C000016' 'C000014' 'C000016' 'C000020' 'C000024' 'C000010' 'C000020'
'C000024' 'C000016' 'C000013' 'C000016' 'C000022' 'C000014' 'C000022'
'C000016' 'C000023' 'C000020' 'C000013' 'C000024']
('C000016', 'C000014', 'C000016', 'C000020', 'C000022', 'C000010', 'C000020', 'C000024', 'C000016', 'C000013',
000016', 'C000023', 'C000020', 'C000010', 'C000024')
0.8947368421052632
```

```
朴素贝叶斯分类器分类新闻文本的精度为: 0.7368421052631579
PS F:\own_file\课程文件\机器学习导论\贝叶斯模型\贝叶斯模型> python -u "f:\own_file\课程文件\机器学习导论\贝叶斯模型\贝叶斯模型\task2\test.py"
Building prefix dict from the default dictionary ...
Loading model from cache C:\temp\jieba.cache
Loading model cost 0.745 seconds.
Prefix dict has been built successfully.
测试集测试结果: ['C000016' 'C000014' 'C000020' 'C000014' 'C000023' 'C000014' 'C000024'
'C000008' 'C000010' 'C000023' 'C000024' 'C000014' 'C000010' 'C000016'
'C000020' 'C000024' 'C000014' 'C000023' 'C000023']
测试集原始标签为: ('C000016', 'C000022', 'C000020', 'C000014', 'C000023', 'C000014', 'C000024', 'C000008', 'C000010', 'C000023', 'C000024', 'C000010', 'C000008', 'C000020', 'C000024', 'C000022', 'C000023', 'C000023')
朴素贝叶斯分类器分类新闻文本的精度为: 0.8421052631578947
```

- 图片分析：上图运行结果为两次的结果对比，结果显示，使用默认参数的情况下，朴素贝叶斯分类器的精度为0.8-0.9之间浮动，精度良好。
- 不断调整参数，得出默认参数效果最为理想，此时参数如下

```
classifier = MultinomialNB(alpha=1.0).fit(train_feature_list, train_class_list)
1 < len(all_words_list[t]) < 5, 特征长度大于1小于5
deleteN = 450, 删除频率最高的450个特征
```

## 截图二，TF\_IDF模型

```
PS F:\own_file\课程文件\机器学习导论\贝叶斯模型\贝叶斯模型> python -u "f:\own_file\课程文件\机器学习导论\贝叶斯模型\贝叶斯模型\task2\test2.py"
Building prefix dict from the default dictionary ...
Loading model from cache C:\temp\jieba.cache
Loading model cost 0.672 seconds.
Prefix dict has been built successfully.
测试集测试结果: ['c000008' 'c000008' 'c000016' 'c000022' 'c000008' 'c000020' 'c000020'
'c000016' 'c000023' 'c000016' 'c000010' 'c000024' 'c000022' 'c000016'
'c000008' 'c000016' 'c000023' 'c000016' 'c000008']

测试集原始标签为: ('c000023', 'c000014', 'c000016', 'c000013', 'c000013', 'c000020', 'c000020', 'c000016', 'c000023', 'c000016', 'c000010', 'c000024'
022', 'c000016', 'c000014', 'c000016', 'c000023', 'c000016', 'c000014')

朴素贝叶斯分类器分类新闻文本的精度为: 训练集精度: 1.000000 测试集精度: 0.684211
PS F:\own_file\课程文件\机器学习导论\贝叶斯模型\贝叶斯模型> python -u "f:\own_file\课程文件\机器学习导论\贝叶斯模型\贝叶斯模型\task2\test2.py"
Loading model from cache C:\temp\jieba.cache
Loading model cost 0.677 seconds.
Prefix dict has been built successfully.
测试集测试结果: ['c000008' 'c000023' 'c000010' 'c000023' 'c000023' 'c000020' 'c000014'
'c000024' 'c000014' 'c000016' 'c000023' 'c000014' 'c000016' 'c000022'
'c000023' 'c000024' 'c000008' 'c000020' 'c000008']

测试集原始标签为: ('c000008', 'c000023', 'c000010', 'c000013', 'c000013', 'c000020', 'c000022', 'c000024', 'c000022', 'c000016', 'c000010', 'c000014'
016', 'c000022', 'c000013', 'c000020', 'c000008', 'c000020', 'c000008')

朴素贝叶斯分类器分类新闻文本的精度为: 训练集精度: 1.000000 测试集精度: 0.631579
```

- 图片分析: 上图为使用TF\_IDF默认参数生成的分类器精度, 可以看出**精度不理想在0.6左右**。
- 不断修改参数后, 得出参数如下时, 精度达到较好的程度, **精度提高到0.8左右**

tfidf\_transformer = TfidfVectorizer(encoding='utf-8', stop\_words=stopwords,  
max\_features=1000, lowercase=False, max\_df=0.5, sublinear\_tf=True, smooth\_idf=True)  
encoding='utf-8',编码方式; stop\_words=stopwords,设置停用词;  
max\_features=1000最大特征数量; lowercase: 转化所有的字符为小写;  
max\_df=0.5忽略频率高于此的特征; sublinear\_tf: 对数缩放避免下溢; smooth\_idf: 添加平滑

```
PS F:\own_file\课程文件\机器学习导论\贝叶斯模型\贝叶斯模型> python -u "f:\own_file\课程文件\机器学习导论\贝叶斯模型\贝叶斯模型\task2\test2.py"
Building prefix dict from the default dictionary ...
Loading model from cache C:\temp\jieba.cache
Loading model cost 0.697 seconds.
Prefix dict has been built successfully.
测试集测试结果: ['c000014' 'c000010' 'c000020' 'c000008' 'c000016' 'c000023' 'c000016'
'c000024' 'c000016' 'c000022' 'c000008' 'c000014' 'c000023' 'c000020'
'c000024' 'c000023' 'c000020' 'c000010' 'c000016']

测试集原始标签为: ('c000014', 'c000013', 'c000020', 'c000008', 'c000016', 'c000023', 'c000016', 'c000024', 'c000016', 'c000022', 'c000008', 'c000014', 'c000
013', 'c000020', 'c000024', 'c000014', 'c000020', 'c000010', 'c000016')

朴素贝叶斯分类器分类新闻文本的精度为: 训练集精度: 1.000000 测试集精度: 0.842105
PS F:\own_file\课程文件\机器学习导论\贝叶斯模型\贝叶斯模型> |
```

## 截图三, 不设参数多次平均

```
朴素贝叶斯分类器分类新闻文本的精度为: 训练集精度: 1.000000 测试集精度: 0.631579
PS F:\own_file\课程文件\机器学习导论\贝叶斯模型\贝叶斯模型> python -u "f:\own_file\课程
Building prefix dict from the default dictionary ...
Loading model from cache C:\temp\jieba.cache
Loading model cost 0.676 seconds.
Prefix dict has been built successfully.
共100次测试, 朴素贝叶斯分类平均精度为: 0.640526
PS F:\own_file\课程文件\机器学习导论\贝叶斯模型\贝叶斯模型> python -u "f:\own_file\课程
Building prefix dict from the default dictionary ...
Loading model from cache C:\temp\jieba.cache
Loading model cost 0.687 seconds.
Prefix dict has been built successfully.
共10次测试, 朴素贝叶斯分类平均精度为: 0.642105
PS F:\own_file\课程文件\机器学习导论\贝叶斯模型\贝叶斯模型> |
```

- 图片分析: 上图为多次运行去平均值, 可以看出, **不设置参数时精度浮动较大, 平均值0.64, 很不理想**

## 三、总结体会

### 程序设计心得:

1. 本次算法过程中, 遇到程序错误在正常不过, 知错能改, 善莫大焉, 一个完整的工程背后一定是一次的debug, 自身犯错最多的地方在于差错处理, 可以多考虑极端情况, 多进行程序的调试, 对每一种特殊情况进行处理, 避免程序运行的自行结束。通过这次软件课设, 自身养成了不惧困难, 不退缩, 提高思维能力

# TASK3: 使用朴素贝叶斯对电影评论分类

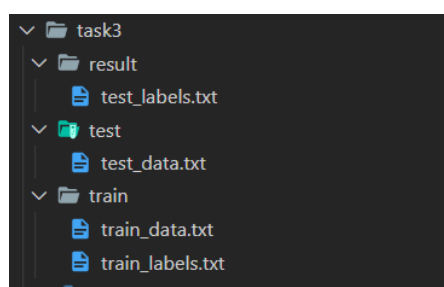
## 一、软件设计

### 1. 设计目标

#### ① 数据集讲解:

该数据集是IMDB电影数据集的一个子集，已经划分好了测试集和训练集，训练集包括25000条电影评论，测试集也有25000条，该数据集已经经过预处理，将每条评论的具体单词序列转化为词库里的整数序列，其中每个整数代表该单词在词库里的位置。例如，整数104代表该单词是词库的第104个单词。为实验简单，词库仅仅保留了10000个最常出现的单词，低频词汇被舍弃。每条评论都具有一个标签，0表示为负面评论，1表示为正面评论。

训练数据在train\_data.txt文件下，每一行为一条评论，训练集标签在train\_labels.txt文件下，每一行为一条评论的标签；测试数据在test\_data.txt文件下，测试数据标签未给出



#### ② 目标分析

- 对给出的训练集进行数据处理，one-hot编码，10000维
- 再使用sklearn库生成朴素贝叶斯分类器

自己写的决策树处理这么大的数据时间太长

- 对测试集的数据进行测试，得出测试集的预测标签结果

## 2. 算法设计分析

所用项目环境 IDE: VS code    Python版本: python3.7.5

#### ① 决策树算法

TASK1中详细描述了朴素贝叶斯算法的实现，TASK3的数据量太大，自己编写的算法处理时间高达几个小时，改用sklearn库

[跳转到task1贝叶斯算法](#)

- sklearn朴素贝叶斯算法:

```
def TextClassifier(train_feature_list, test_feature_list, train_class_list):
    # prior = prior_probability(train_class_list)
    classifier = MultinomialNB().fit(train_feature_list, train_class_list)
    train_accuracy = classifier.score(train_feature_list, train_class_list)
    print("分类器训练集拟合精度为: ", train_accuracy)
    test_class_list = classifier.predict(test_feature_list)
    return test_class_list
```

- 代码细节:



```
description: 使用sklearn中的MultinomialNB()生成朴素贝叶斯分类器,并预测测试集标签结果
            果保存为txt
            param {} train_feature_list, test_feature_list, train_class_list, file_path
            return {} test_class_list
```

- (1) 使用MultinomialNB()构造贝叶斯分类器, 使用clf.fit()导入训练数据及标签
- (2) 使用classifier.score计算训练集的拟合精度, classifier.predict预测测试集的标签结果

## ②数据处理

===== >> 数据处理 << =====

**算法介绍:** 将训练集的数据进行处理, 按照采集时间的不同生成不同矢量

$f_i = [BSSID_1:RSS_1, BSSID_2:RSS_2, \dots, RoomLabel]$

- 训练集数据处理算法:

```
def dataget(file_path1, file_path2):
    train_data = txtread(file_path1)    # 读取训练集数据
    label_data = txtread(file_path2)    # 读取训练集标签
    ...
    return traindata, label_data        # 处理后训练数据, 训练集标签
```

- 代码细节:

- (1) 将traindata初始化为25000\*10000的二维列表, 二维列表每一个元素, 将读取的训练集数据每一行出现的数字作为索引, 将对应位置设为1, 其余初始化为0, 使训练集数据每一行都形成10000个1, 0序列
- (2) txtread()函数实现读取txt文件, 并将其转化为二维的列表存储

## ③文件操作

===== >> 文件操作 << =====

**算法介绍:** txt文件的读取以及保存函数

- txt读取算法:

```
def txtread(filepath):
    with open(filepath, 'r') as fr:
        file_data = [inst.strip().replace('\n', '').split(' ') for inst in fr.readlines()]
        # print(file_data)
    print(filepath, '文件读取成功')
    return file_data
```

- 代码细节:

- (1) 使用python文件操作, 注意.strip()去除每一行的开始结尾; .split()将数据分片成为列表

- txt保存算法:

```
def txtsave(filename, data): # filename为写入txt文件的路径, data为要写入数据列表.
    file = open(filename, 'w')
    for i in range(len(data)):
        s = str(data[i]).replace('[', '').replace(']', '').replace(',', '\t') # 去除[]
        s = s.replace('"', '').replace(' ', '').replace('.', '\t') + '\n' # 去除单引号, 逗号, 每行末尾追加换行符
        file.write(s)
    file.close()
    print(filename, "保存文件成功")
```

- 代码细节:

(1) 使用python文件操作, 注意.replace()将特定字符替换, 同时行末写入换行符

#### ④ main函数

=====>> main函数 <<=====

```
def main():
    file_path_traindata = "task3\\train\\train_data.txt"
    file_path_testdata = "task3\\test\\test_data.txt"
    file_path_trainlabel = "task3\\train\\train_labels.txt"
    file_path_result = "task3\\result\\"
    traindata_list = dataget(file_path_traindata)
    testdatas_list = dataget(file_path_testdata)
    trainlabel_list = txtread(file_path_trainlabel)

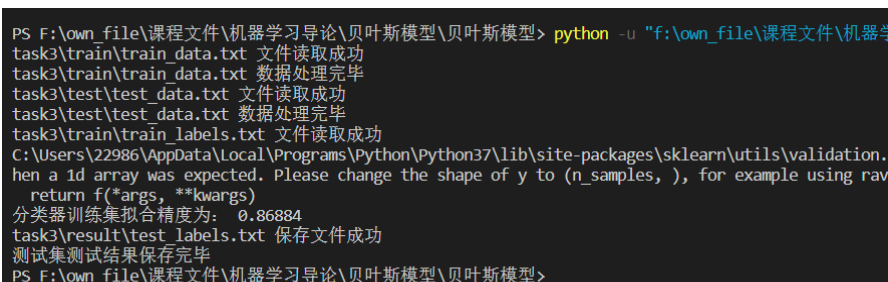
    test_class_list = TextClassifier(traindata_list, testdatas_list,
    trainlabel_list)

    txtsave(file_path_result + 'test_labels.txt', test_class_list)
    print('测试集测试结果保存完毕')
```

- 代码细节: 程序运行主函数, 调用编写的各个算法函数, 对电影评论进行预测分类

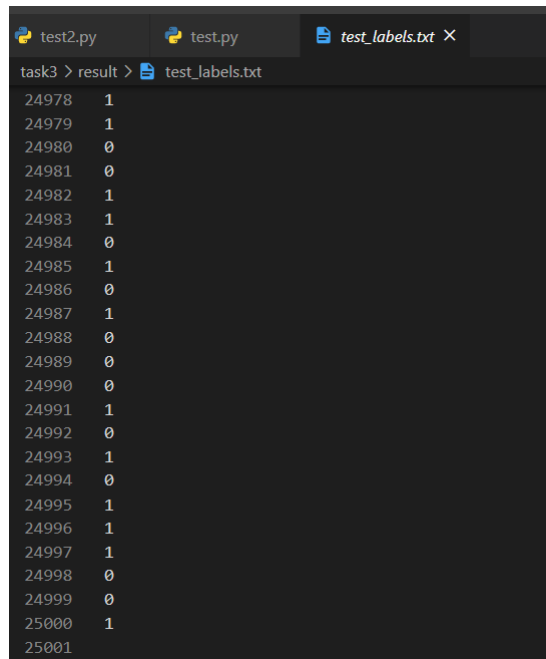
## 二、运行结果

### 结果截图1



```
PS F:\own_file\课程文件\机器学习导论\贝叶斯模型\贝叶斯模型> python -u "f:\own_file\课程文件\机器学习导论\贝叶斯模型\main.py"
task3\train\train_data.txt 文件读取成功
task3\train\train_data.txt 数据处理完毕
task3\test\test_data.txt 文件读取成功
task3\test\test_data.txt 数据处理完毕
task3\train\train_labels.txt 文件读取成功
C:\Users\22986\AppData\Local\Programs\Python\Python37\lib\site-packages\sklearn\utils\validation.py:103: FutureWarning:
When a 1d array was expected. Please change the shape of y to (n_samples, ), for example using.ravel().
return f(*args, **kwargs)
分类器训练集拟合精度为: 0.86884
task3\result\test_labels.txt 保存文件成功
测试集测试结果保存完毕
PS F:\own_file\课程文件\机器学习导论\贝叶斯模型\贝叶斯模型>
```

- 图片分析: 上图运行结果, 控制台打印信息显示程序一切运行正常, 预测结果标签保存为txt文件成功; 其中训练集拟合精度为0.869



```
task3 > result > test_labels.txt
24978 1
24979 1
24980 0
24981 0
24982 1
24983 1
24984 0
24985 1
24986 0
24987 1
24988 0
24989 0
24990 0
24991 1
24992 0
24993 1
24994 0
24995 1
24996 1
24997 1
24998 0
24999 0
25000 1
25001 1
```

- 图片分析：上图为预测结果的标签存储文件，与训练集的标签一致，共25000条标签数据

### 三、总结体会

#### 程序设计心得:

1. 本次算法过程中，遇到程序错误在正常不过，知错能改，善莫大焉，一个完整的工程背后一定是一次的debug，自身犯错最多的地方在于差错处理，可以多考虑极端情况，多进行程序的调试，对每一种特殊情况进行处理，避免程序运行的自行结束。通过这次软件课设，自身养成了不惧困难，不退缩，提高思维能力
2. 调用sklearn，重点在于对于该库的了解，可以多看看官方文档，以及找一些实战案例学习使用