

# 第 4 章 朴素贝叶斯实战之文本分类

## ——屏蔽侮辱性言论

### 4.1 概率的相关知识

#### 一 随机事件及其运算

##### (一) 事件间的关系

1. 包含关系:  $A \subset B$  —— A 被包含于 B 或 B 包含 A
2. 相等关系:  $A = B$  —— 事件 A 与事件 B 相等 (即  $A \subset B$  且  $B \subset A$ )
3. 互不相容:  $A \cap B = \emptyset$  —— 事件 A 与事件 B 不可能同时发生

##### (二) 事件间的运算

1. 事件 A 与 B 的并 ——  $A \cup B$
2. 事件 A 与 B 的交 ——  $A \cap B$  或简记为  $AB$
3. 事件 A 与 B 的差 ——  $A - B$  (事件 A 发生而 B 不发生)
4. 对立事件 ——  $\bar{A}$ , 即为事件 A 的对立事件, 含义为: 由在  $\Omega$  中而不在 A 中的样本点组成的新事件

注意:

1. 对立事件一定是互不相容的事件, 即  $A \cap \bar{A} = \emptyset$ . 但互不相容的事件不一定是对立事件
2.  $A - B$  可以记为  $A\bar{B}$

##### (三) 事件的运算性质

1. 交换律

$$A \cup B = B \cup A, \quad AB = BA$$

2. 结合律

$$(A \cup B) \cup C = A \cup (B \cup C) \\ (AB)C = A(BC)$$

3. 分配律

$$(A \cup B) \cap C = AC \cup BC \\ (A \cap B) \cup C = (A \cup C) \cap (B \cup C)$$

4. 对偶率 (德摩根公式)

$$\overline{A \cup B} = \bar{A} \cap \bar{B}$$

$$\overline{A \cap B} = \bar{A} \cup \bar{B}$$

#### 二 概率的性质

##### (一) 概率的可加性

**有限可加性:** 若有限个事件  $A_1, A_2, \dots, A_n$  互不相容, 则有

$$P\left(\bigcup_{i=1}^n A_i\right) = \sum_{i=1}^n P(A_i)$$

**推论:** 对任一事件  $A$ , 有:  $P(\bar{A}) = 1 - P(A)$

## (二) 概率单调性

1. **性质1:** 若  $A \supset B$ , 则  $P(A - B) = P(A) - P(B)$

2. **推论1(单调性):** 若  $A \supset B$ , 则  $P(A) \geq P(B)$

3. **性质2:** 对任意事件  $A, B$ , 有:

$$P(A - B) = P(A) - P(AB)$$

## (三) 概率的加法公式

1. **概率的加法公式:** 对任意事件  $A, B$ , 有

$$P(A \cup B) = P(A) + P(B) - P(AB)$$

2. **推论 (半可加性):** 对任意事件  $A, B$ , 有

$$P(A \cup B) \leq P(A) + P(B)$$

# 三 条件概率

---

## (一) 条件概率的定义

1. **条件概率:** 设  $A$  与  $B$  是样本空间  $\Omega$  中的两事件, 若  $P(B) > 0$ , 则称

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

为"在  $B$  发生下  $A$  的条件概率", 简称条件概率

## (二) 乘法公式

**乘法公式:**

- (1) 若  $P(B) > 0$ , 则  $P(AB) = P(B)P(A|B)$
- (2) 若  $P(A_1 A_2 \cdots A_{n-1}) > 0$ ,  
$$P(A_1 \cdots A_n) = P(A_1) P(A_2|A_1) P(A_3|A_1 A_2) \cdots P(A_n|A_1 \cdots A_{n-1})$$

## (二) 全概率公式

### 1. 全概率公式:

设  $B_1, B_2, \dots, B_n$  为样本空间  $\Omega$  的一个分割, 即  $B_1, B_2, \dots, B_n$  互不相容, 且  $\bigcup_{i=1}^n B_i = \Omega$ , 如果  $P(B_i) > 0, i = 1, 2, \dots, n$ , 则对任一事件  $A$  有:

$$P(A) = \sum_{i=1}^n P(B_i) P(A|B_i)$$

注:  $A = A\Omega = A(\bigcup_{i=1}^n B_i) = \bigcup_{i=1}^n (AB_i)$ , 那么,  $P(A) = P(\bigcup_{i=1}^n (AB_i)) = \sum_{i=1}^n P(AB_i)$ , 将  $P(AB_i) = P(B_i) P(A|B_i), i = 1, 2, \dots, n$  带入即可得到.

全概率的最简单形式:

$$P(A) = P(B)P(A|B) + P(\bar{B})P(A|\bar{B})$$

## (三) 贝叶斯公式

### 贝叶斯公式:

设  $B_1, B_2, \dots, B_n$  为样本空间  $\Omega$  的一个分割, 即  $B_1, B_2, \dots, B_n$  互不相容, 且  $\bigcup_{i=1}^n B_i = \Omega$ , 如果  $P(A) > 0, P(B_i) > 0, i = 1, 2, \dots, n$ , 则有:

$$P(B_i|A) = \frac{P(B_i) P(A|B_i)}{\sum_{j=1}^n P(B_j) P(A|B_j)}, \quad i = 1, 2, \dots, n$$

注: 根据条件概率的定义:

$$P(B_i|A) = \frac{P(AB_i)}{P(A)}$$

分子用乘法公式, 分母用全概率公式, 即

$$\begin{aligned} P(AB_i) &= P(B_i) P(A|B_i) \\ P(A) &= \sum_{j=1}^n P(B_j) P(A|B_j) \end{aligned}$$

带入即可得到.

上面的是一般形式, 常见的还有这样的形式:

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

便于理解:

$$P(\text{原因}|\text{结果}) = \frac{P(\text{原因})P(\text{结果}|\text{原因})}{P(\text{结果})}$$

## 4.2 朴素贝叶斯的理论基础

贝叶斯准则告诉我们如何交换条件概率中的条件与结果, 即如果已知  $P(x|c)$ , 要求  $P(c|x)$ , 那么可以使用下面的计算方法:

$$p(c|x) = \frac{p(x|c)p(c)}{p(x)}$$

- 如果  $p_1(x, y) > p_2(x, y)$ , 那么属于类别 1;
- 如果  $p_2(x, y) > p_1(x, y)$ , 那么属于类别 2.

这并不是贝叶斯决策理论的所有内容. 使用  $p_1()$  和  $p_2()$  只是为了尽可能简化描述, 而真正需要计算和比较的是  $p(c_1|x, y)$  和  $p(c_2|x, y)$ . 这些符号所代表的具体意义是: 给定某个由  $x$ 、 $y$  表示的数据点, 那么该数据点来自类别  $c_1$  的概率是多少? 数据点来自类别  $c_2$  的概率又是多少? 注意这些概率与概率  $p(x, y|c_1)$  并不一样, 不过可以使用贝叶斯准则来交换概率中条件与结果. 具体地, 应用贝叶斯准则得到:

$$p(c_i|x, y) = \frac{p(x, y|c_i) p(c_i)}{p(x, y)}$$

使用上面这些定义, 可以定义贝叶斯分类准则为:

- 如果  $P(c_1|x, y) > P(c_2|x, y)$ , 那么属于类别  $c_1$ ;
- 如果  $P(c_2|x, y) > P(c_1|x, y)$ , 那么属于类别  $c_2$ .

注: 上面的只是《机器学习》书上关于贝叶斯分类准则的简要理论介绍, 更为详细的理论知识, 参照《西瓜书》和《统计学习方法》相关内容

## 4.3 朴素贝叶斯的一般流程

机器学习的一个重要应用就是文档的自动分类. 在文档分类中, 整个文档 (如一封电子邮件) 是实例, 而电子邮件中的某些元素则构成特征. 虽然电子邮件是一种会不断增加的文本, 但我们同样也可以对新闻报道、用户留言、政府公文等其他任意类型的文本进行分类. 我们可以观察文档中出现的词, 并把每个词的出现或者不出现作为一个特征, 这样得到的特征数目就会跟词汇表中的词目一样多. 朴素贝叶斯是上节介绍的贝叶斯分类器的一个扩展, 是用于文档分类的常用算法.

### 4.3.1 朴素贝叶斯的一般过程

- 收集数据: 可以使用任何方法.
- 准备数据: 需要数值型或者布尔型数据.
- 分析数据: 有大量特征时, 绘制特征作用不大, 此时使用直方图效果更好.
- 训练算法: 计算不同的独立特征的条件概率.
- 测试算法: 计算错误率.
- 使用算法: 一个常见的朴素贝叶斯应用是文档分类. 可以在任意的分类场景中使用朴素贝叶斯分类器, 不一定非要是文本.

### 4.3.2 朴素贝叶斯的伪代码

```
计算每个类别中的文档数目
对每篇训练文档:
    对每个类别:
        如果词条出现在文档中→ 增加该词条的计数值
        增加所有词条的计数值
    对每个类别:
        对每个词条:
            将该词条的数目除以总词条数目得到条件概率
返回每个类别的条件概率
```

### 4.3.3 朴素贝叶斯的优缺点

优点：在数据较少的情况下仍然有效， 可以处理多类别问题。

缺点：对于输入数据的准备方式较为敏感。

适用数据类型：标称型数据。

## 4.4 项目案例1: 屏蔽社区留言板的侮辱性言论

### 4.4.1 项目概述

为了不影响社区的发展, 我们要屏蔽侮辱性的言论, 所以要构建一个快速过滤器, 如果某条留言使用了负面或者侮辱性的语言, 那么就将该留言标识为内容不当. 过滤这类内容是一个很常见的需求. 对此问题建立两个类别: 侮辱类和非侮辱类, 使用1和0分别表示.

### 4.4.2 项目流程

- 收集数据: 可以使用任何方法
- 准备数据: 从文本中构建词向量
- 分析数据: 检查词条确保解析的正确性
- 训练算法: 从词向量计算概率
- 测试算法: 根据现实情况修改分类器
- 使用算法: 对社区留言板言论进行分类

### 4.4.3 具体实现过程

#### 一 构造词汇表 (loadDataSet、setOfWords2Vec和createVocabList函数)

把文本看成单词向量或者词条向量, 也就是说将句子转换为向量. 考虑出现在所有文档中的所有单词, 再决定将哪些词纳入词汇表或者说所要的词汇集合, 然后必须要将每一篇文档转换为词汇表上的向量.

创建一个 Naive\_Bayes\_Hsu01.py 新文件

```
import numpy as np

"""
函数说明：创建样本

Parameters:
    无
Returns:
    postingList - 切分的词条
    classVec - 类别标签向量
"""
def loadDataSet():
    postingList = [['my', 'dog', 'has', 'flea', 'problems', 'help', 'please'],
                    ['maybe', 'not', 'take', 'him', 'to', 'dog', 'park',
                     'stupid'],
                    ['my', 'dalmation', 'is', 'so', 'cute', 'I', 'love', 'him'],
                    ['stop', 'posting', 'stupid', 'worthless', 'garbage']]
```

```

        ['mr', 'licks', 'ate', 'my', 'steak', 'how', 'to', 'stop',
'him'],
        ['quit', 'buying', 'worthless', 'dog', 'food', 'stupid']]
classVec = [0,1,0,1,0,1]
return postingList,classVec

```

"""

函数说明：将切分的样本词条整理成不重复的词条

Parameters:

dataSet - 样本数据集(即之前切分的样本词条)

Returns:

vocabSet - 不重复的词条

"""

```
def createVocabList(dataSet):
```

#初始化一个空的集合(集合是不重复的)

```
vocaSet = set([])
```

#遍历dataSet每个元素

```
for document in dataSet:
```

#取并集

```
vocaSet = vocaSet | set(document)
```

#返回不重复的列表

```
return list(vocaSet)
```

"""

函数说明：根据coclaList里的词汇表，将输入的inputSet向量化,向量的每个元素为1或者0

Parameters:

vocabList - createVocabList函数返回的词汇表

inputSet - 要输入的切分词条列表

Returns:

returnVec - 向量化后的文档

"""

```
def setOfWords2Vec(vocabList, inputSet):
```

#初始化returnVec列表，列表长度为vocabList长度，元素为0

```
returnVec = [0] * len(vocabList)
```

#遍历inputSet所有单词,如果出现了词汇表中的单词，则将输出的文档向量中的对应值设为1

```
for word in inputSet:
```

```
    if word in vocabList:
```

```
        returnVec[vocabList.index(word)] = 1
```

```
    else:
```

```
        print("the word: %s is not in my vocabulary!" % word)
```

```
return returnVec
```

```
if __name__ == '__main__':
```

```
    postingList,classVec = loadDataSet()
```

```
    vocabList = createVocabList(postingList)
```

```
    returnVec_1 = setOfWords2Vec(vocabList, postingList[0])
```

```
    returnVec_2 = setOfWords2Vec(vocabList, postingList[1])
```

```
    print(postingList)
```

```
    print(classVec)
```

```
    print(vocabList)
```

```
    print(returnVec_1)
```

```
    print(returnVec_2)
```

结果如下:

```
Python 3.6.4 |Anaconda, Inc.| (default, Jan 16 2018, 10:22:32) [MSC v.1900 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 6.2.1 -- An enhanced Interactive Python.

In [1]: runfile('C:/Users/HeatonHsu/.spyder-py3/temp.py', wdir='C:/Users/HeatonHsu/.spyder-py3')
[[['my', 'dog', 'has', 'flea', 'problems', 'help', 'please'], ['maybe', 'not', 'take', 'him', 'to', 'dog', 'park', 'stupid'],
['my', 'dalmation', 'is', 'so', 'cute', 'I', 'love', 'him'], ['stop', 'posting', 'stupid', 'worthless', 'garbage'], ['mr',
'licks', 'ate', 'my', 'steak', 'how', 'to', 'stop', 'him'], ['quit', 'buying', 'worthless', 'dog', 'food', 'stupid']]
[0, 1, 0, 1, 0, 1]
['food', 'take', 'help', 'dalmation', 'how', 'licks', 'flea', 'I', 'so', 'to', 'quit', 'stop', 'stupid', 'love', 'has', 'my',
'maybe', 'worthless', 'him', 'problems', 'buying', 'mr', 'not', 'please', 'dog', 'cute', 'steak', 'is', 'garbage', 'ate',
'posting', 'park']
[0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0]
[0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1]

In [2]:
```

## 二 训练算法 (trainNB0函数)

已经知道了一个词是否出现在一篇文档中, 也知道该文档所属的类别. 接下来我们重写贝叶斯准则, 将之前的  $x, y$  替换为  $w$  粗体的 表  $w$  示这是一个向量, 即它由多个值组成. 在这个例子中, 数值个数与词汇表中的词个数相同.

$$p(c_i|w) = \frac{p(w|c_i)p(c_i)}{p(w)}$$

其中  $p(w)$  是固定的, 因此只需要比较分子大小来做决策分类.

其中  $p(c_i)$  通过类别  $i$  中的文档数除以总的文档数来计算

而  $p(w|c_i)$  的计算, 基于朴素贝叶斯的假设, 可以写作  $p(w_0|c_i)p(w_1|c_i)\cdots p(w_n|c_i)$  来计算

那么具体代码如下:

```
"""
函数说明：朴素贝叶斯分类器训练函数

Parameters:
    trainMatrix - 训练文档矩阵，即函数setOfWords2Vec返回的returnVec构成的矩阵
    trainCategory - 训练类别(标签)向量，即函数loadDataSet返回的classVec

Returns:
    p0Vect - 非侮辱类的条件概率数组
    p1Vect - 侮辱类的条件概率数组
    pAbusive - 文档属于侮辱类的概率
"""

def trainNB0(trainMatrix, trainCategory):
    #计算文档的数目，即trainMatrix列表的长度(和trainCategory数目是对应的,相同的)
    #postingList每一行是一个文档，对应trainMatrix中的一个元素，也即是一个returnVec
    numTrainDocs = len(trainMatrix)
    #单词个数(trainMatrix[0]=trainMatrix[1]=....)
    numWords = len(trainMatrix[0])
    #侮辱性文件的出现概率，即trainCategory中所有的1的个数，
    #代表的就是多少个侮辱性文件，与文件的总数相除就得到了侮辱性文件的出现概率
    pAbusive = sum(trainCategory) / float(numTrainDocs)

    #初始化词条出现次数,初始为0
    p0Num = np.zeros(numWords)
    p1Num = np.zeros(numWords)
    #分母初始化为0
    p0Denom = 0
    p1Denom = 0
    for i in range(numTrainDocs):
        #是否是侮辱性文件
```

```

if trainCategory[i] == 1:
    #如果是侮辱性文件，对侮辱性文件的向量进行加和
    #注意array是可以直接和对应长度的列表相加的。
    p1Num += trainMatrix[i]    #[0,1,1,...] + [0,1,1,...]->[0,2,2,...]
    #对向量中的所有元素进行求和，也就是计算所有侮辱性文件中出现的单词总数
    p1Denom += sum(trainMatrix[i])
else:
    p0Num += trainMatrix[i]
    p0Denom += sum(trainMatrix[i])
#类别1，即侮辱性文档的[P(F1|C1),P(F2|C1),P(F3|C1),P(F4|C1),P(F5|C1)...]列表
#即在1类别下，每个单词出现的概率
p1Vect = p1Num / p1Denom #如# [1,2,3,5]/90->[1/90,...]
#类别0，即正常文档的[P(F1|C0),P(F2|C0),P(F3|C0),P(F4|C0),P(F5|C0)...]列表
#即在0类别下，每个单词出现的概率
p0Vect = p0Num / p0Denom
return p0Vect, p1Vect, pAbusive

if __name__ == '__main__':
    postingList, classVec = loadDataSet()
    vocabList = createVocabList(postingList)
    trainMat = []
    for postinDoc in postingList:
        trainMat.append(setOfWords2Vec(vocabList,postinDoc))
    p0Vect, p1Vect, pAbusive = trainNB0(trainMat,classVec)
    print(p0Vect)
    print(p1Vect)
    print(pAbusive)

```

运行结果如下:

```

In [7]: runfile('C:/Users/HeatonHsu/.spyder-py3/temp.py', wdir='C:/Users/HeatonHsu/.spyder-py3')
[0.      0.      0.04166667 0.04166667 0.04166667 0.04166667
 0.04166667 0.04166667 0.04166667 0.04166667 0.      0.04166667
 0.      0.04166667 0.04166667 0.125      0.      0.
 0.08333333 0.04166667 0.      0.04166667 0.      0.04166667
 0.04166667 0.04166667 0.04166667 0.04166667 0.      0.04166667
 0.      0.      ]
[0.05263158 0.05263158 0.      0.      0.      0.
 0.      0.      0.05263158 0.05263158 0.05263158 0.05263158
 0.15789474 0.      0.      0.      0.05263158 0.10526316
 0.05263158 0.      0.05263158 0.      0.05263158 0.
 0.10526316 0.      0.      0.      0.05263158 0.
 0.05263158 0.05263158]
0.5

```

In [8]:

### 三 训练算法的改进 (trainNB0函数)

tainNB0函数有两个问题:

1. 一是, 在计算  $p(\mathbf{w}_0|c_i)p(\mathbf{w}_1|c_i)\cdots p(\mathbf{w}_n|c_i)$  时, 如果其中一个概率值为0, 那么最后乘积也为0. 为降低这种影响, 可以将所有词的出现数初始化为 1, 并将分母初始化为 2
2. 另一个遇到的问题是下溢出, 这是由于太多很小的数相乘造成的. 在计算  $p(\mathbf{w}_0|c_i)p(\mathbf{w}_1|c_i)\cdots p(\mathbf{w}_n|c_i)$  时, 由于大部分因子都非常小, 所以程序会下溢出或者得到不正确的答案. 一种解决办法是对乘积取自然对数

基于以上分析, 对trainNB0函数进行改进, 结果如下:

```

def trainNB0(trainMatrix, trainCategory):
    #计算文档的数目, 即trainMatrix列表的长度(和trainCategory数目是对应的, 相同的)
    #postingList每一行是一个文档, 对应trainMatrix中的一个元素, 也即是一个returnVec

```



```

numTrainDocs = len(trainMatrix)
#单词个数(trainMatrix[0]=trainMatrix[1]=....)
numwords = len(trainMatrix[0])
#侮辱性文件的出现概率，即trainCategory中所有的1的个数，
#代表的就是多少个侮辱性文件，与文件的总数相除就得到了侮辱性文件的出现概率
pAbusive = sum(trainCategory) / float(numTrainDocs)

#初始化词条出现次数,初始为0,改进为1
p0Num = np.ones(numwords)
p1Num = np.ones(numwords)
#分母初始化为0啊,改进为2
p0Denom = 2
p1Denom = 2
for i in range(numTrainDocs):
    #是否是侮辱性文件
    if trainCategory[i] == 1:
        #如果是侮辱性文件，对侮辱性文件的向量进行加和
        #注意array是可以直接和对应长度的列表相加的。
        p1Num += trainMatrix[i] #[0,1,1,...] + [0,1,1,...]->[0,2,2,...]
        #对向量中的所有元素进行求和，也就是计算所有侮辱性文件中出现的单词总数
        p1Denom += sum(trainMatrix[i])
    else:
        p0Num += trainMatrix[i]
        p0Denom += sum(trainMatrix[i])
#类别1，即侮辱性文档的[P(F1|C1),P(F2|C1),P(F3|C1),P(F4|C1),P(F5|C1)...]列表
#即在1类别下，每个单词出现的概率,改进为对数
p1Vect = np.log(p1Num / p1Denom) #如# [1,2,3,5]/90->[1/90,...]
#类别0，即正常文档的[P(F1|C0),P(F2|C0),P(F3|C0),P(F4|C0),P(F5|C0)...]列表
#即在0类别下，每个单词出现的概率,改进为对数
p0Vect = np.log(p0Num / p0Denom)
return p0Vect, p1Vect, pAbusive

```

## 四 测试算法 (classifyNB和testingNB函数)

先分类, 再进行测试. 具体如下:

```

"""
函数说明：朴素贝叶斯分类函数

Parameters:
    vec2Classify - 待分类数组, 如[1,0,1,...], 注意是np.array类型的
    p0Vec - trainNB返回的p0Vect, 也就是类别为0, 即正常类文档的条件概率数组,
            即
            [log(P(X1|C0)), log(P(X2|C0)), log(P(X3|C0)), log(P(X4|C0)), log(P(X5|C0))...] 数组
    p1Vec - trainNB返回的p1Vect, 也即是类别为1, 即侮辱类文档的条件概率数组,
            即
            [log(P(X1|C1)), log(P(X2|C1)), log(P(X3|C1)), log(P(X4|C1)), log(P(X5|C1))...] 数组
    pClass1 - 文档属于侮辱类的概率, 即trainNB中的pAbusive

Returns:
    0 - 文档属于正常类, 非侮辱类
    1 - 文档属于侮辱类
"""
def classifyNB(vec2Classify, p0Vec, p1Vec, pClass1):

```

```

#因为p1Vect已经取了对数,所以要计算 $p(x_{\{i\}}|c1)$ 相乘,也就是计算求和,最后再加上
log(pClass1)即可
#同时, 因为分母 $p(w)$ 都是一致的, 所以只需要计算并比较分子大小即可
#同时, 根据训练得到的p1Vec,即条件概率数组, 乘以待分类向量,即可得到待分类的条件概率
p1 = sum(vec2Classify * p1Vec) + np.log(pClass1)
p0 = sum(vec2Classify * p0Vec) + np.log(1 - pClass1)
if p1 > p0:
    return 1
else:
    return 0

"""
函数说明: 测试朴素贝叶斯分类器

Parameters:
    无
Returns:
    无
"""
def testingNB():
    postingList, classVec = loadDataSet()
    vocabList = createVocabList(postingList)
    trainMat = []
    for postinDoc in postingList:
        trainMat.append(setOfWords2Vec(vocabList, postinDoc))
    p0Vect, p1Vect, pAbusive = trainNB0(trainMat, classVec)

    #测试第一个文档
    testEntry = ['love', 'my', 'dalmation']
    #这里注意一下,thisDoc可以是列表, array可以和对应长度的列表相乘的.参见trainNB0中的
    p1Num += trainMatrix[i]
    #所以, 也可以写成thisDoc = setOfWords2Vec(vocabList, testEntry)
    thisDoc = np.array(setOfWords2Vec(vocabList, testEntry))
    #如果结果是1,则是侮辱类, 否则为0, 就是非侮辱类的
    if classifyNB(thisDoc, p0Vect, p1Vect, pAbusive) == 1:
        print(testEntry, '属于侮辱类')
    elif classifyNB(thisDoc, p0Vect, p1Vect, pAbusive) == 0:
        print(testEntry, '属于非侮辱类')

    #测试另外一个文档
    testEntry = ['stupid', 'garbage']
    thisDoc = np.array(setOfWords2Vec(vocabList, testEntry))
    if classifyNB(thisDoc, p0Vect, p1Vect, pAbusive) == 1:
        print(testEntry, '属于侮辱类')
    elif classifyNB(thisDoc, p0Vect, p1Vect, pAbusive) == 0:
        print(testEntry, '属于非侮辱类')

```

## 五 完整代码

把前面的所有代码整合在一起, 就是完整的代码

```
import numpy as np
```

```
"""
```

函数说明：创建样本

Parameters:

无

Returns:

postingList - 切分的词条

classVec - 类别标签向量

"""

```
def loadDataSet():
```

```
    postingList = [['my', 'dog', 'has', 'flea', 'problems', 'help', 'please'],
                    ['maybe', 'not', 'take', 'him', 'to', 'dog', 'park',
                     'stupid'],
```

```
                    ['my', 'dalmation', 'is', 'so', 'cute', 'I', 'love', 'him'],
                    ['stop', 'posting', 'stupid', 'worthless', 'garbage'],
                    ['mr', 'licks', 'ate', 'my', 'steak', 'how', 'to', 'stop',
```

```
                    'him'],
```

```
                    ['quit', 'buying', 'worthless', 'dog', 'food', 'stupid']]
```

```
    classVec = [0,1,0,1,0,1]
```

```
    return postingList,classVec
```

"""

函数说明：将切分的样本词条整理成不重复的词条

Parameters:

dataSet - 样本数据集(即之前切分的样本词条)

Returns:

vocabSet - 不重复的词条

"""

```
def createVocabList(dataSet):
```

```
    #初始化一个空的集合(集合是不重复的)
```

```
    vocaSet = set([])
```

```
    #遍历dataSet每个元素
```

```
    for document in dataSet:
```

```
        #取并集
```

```
        vocaSet = vocaSet | set(document)
```

```
    #返回不重复的列表
```

```
    return list(vocaSet)
```

"""

函数说明：根据coclaList里的词汇表，将输入的inputSet向量化,向量的每个元素为1或者0

Parameters:

vocabList - createVocabList函数返回的词汇表

inputSet - 要输入的切分词条列表

Returns:

returnVec - 向量化后的文档

"""

```
def setOfWords2Vec(vocabList, inputSet):
```

```
    #初始化returnVec列表，列表长度为vocabList长度，元素为0
```

```
    returnVec = [0] * len(vocabList)
```

```
    #遍历inputSet所有单词,如果出现了词汇表中的单词，则将输出的文档向量中的对应值设为1
```

```
    for word in inputSet:
```

```
        if word in vocabList:
```

```
            returnVec[vocabList.index(word)] = 1
```

```
        else:
```

```
            print("the word: %s is not in my vocabulary!" % word)
```

```
    return returnVec
```

"""

函数说明：朴素贝叶斯分类器训练函数

Parameters:

**trainMatrix** - 训练文档矩阵，即函数setOfWords2Vec返回的returnVec构成的矩阵

**trainCategory** - 训练类别(标签)向量，即函数loadDataSet返回的classVec

Returns:

**p0Vect** - 非侮辱类的条件概率数组

**p1Vect** - 侮辱类的条件概率数组

**pAbusive** - 文档属于侮辱类的概率

"""

```
def trainNB0(trainMatrix, trainCategory):
```

```
    #计算文档的数目，即trainMatrix列表的长度(和trainCategory数目是对应的,相同的)
```

```
    #postingList每一行是一个文档，对应trainMatrix中的一个元素，也即是一个returnVec
```

```
    numTrainDocs = len(trainMatrix)
```

```
    #单词个数(trainMatrix[0]=trainMatrix[1]=....)
```

```
    numWords = len(trainMatrix[0])
```

```
    #侮辱性文件的出现概率，即trainCategory中所有的1的个数，
```

```
    #代表的就是多少个侮辱性文件，与文件的总数相除就得到了侮辱性文件的出现概率
```

```
    pAbusive = sum(trainCategory) / float(numTrainDocs)
```

```
    #初始化词条出现次数,初始为0,改进为1
```

```
    p0Num = np.ones(numWords)
```

```
    p1Num = np.ones(numWords)
```

```
    #分母初始化为0啊,改进为2
```

```
    p0Denom = 2
```

```
    p1Denom = 2
```

```
    for i in range(numTrainDocs):
```

```
        #是否是侮辱性文件
```

```
        if trainCategory[i] == 1:
```

```
            #如果是侮辱性文件，对侮辱性文件的向量进行加和
```

```
            #注意array是可以直接和对应长度的列表相加的。
```

```
            p1Num += trainMatrix[i]    #[0,1,1,...] + [0,1,1,...]->[0,2,2,...]
```

```
            #对向量中的所有元素进行求和，也就是计算所有侮辱性文件中出现的单词总数
```

```
            p1Denom += sum(trainMatrix[i])
```

```
        else:
```

```
            p0Num += trainMatrix[i]
```

```
            p0Denom += sum(trainMatrix[i])
```

```
    #类别1，即侮辱性文档的[P(F1|C1),P(F2|C1),P(F3|C1),P(F4|C1),P(F5|C1)....]列表
```

```
    #即在1类别下，每个单词出现的概率,改进为对数
```

```
    p1Vect = np.log(p1Num / p1Denom) #如# [1,2,3,5]/90->[1/90,...]
```

```
    #类别0，即正常文档的[P(F1|C0),P(F2|C0),P(F3|C0),P(F4|C0),P(F5|C0)....]列表
```

```
    #即在0类别下，每个单词出现的概率,改进为对数
```

```
    p0Vect = np.log(p0Num / p0Denom)
```

```
    return p0Vect, p1Vect, pAbusive
```

"""

函数说明：朴素贝叶斯分类函数

Parameters:

**vec2Classify** - 待分类数组,如[1,0,1,...],注意是np.array类型的

**p0Vec** - trainNB返回的p0Vect，也就是类别为0,即正常类文档的条件概率数组，

即

[log(P(X1|C0)),log(P(X2|C0)),log(P(X3|C0)),log(P(X4|C0)),log(P(X5|C0))....]数组

**p1Vec** - trainNB返回的p1Vect，也即是类别为1，即侮辱类文档的条件概率数组，

即

$[\log(P(X_1|C_1)), \log(P(X_2|C_1)), \log(P(X_3|C_1)), \log(P(X_4|C_1)), \log(P(X_5|C_1)) \dots]$  数组  
pClass1 - 文档属于侮辱类的概率, 即trainNB中的pAbusive

Returns:

0 - 文档属于正常类, 非侮辱类

1 - 文档属于侮辱类

"""

```
def classifyNB(vec2Classify, p0Vec, p1Vec, pClass1):
```

#因为p1Vect已经取了对数, 所以要计算 $p(x_{\{i\}}|c_1)$ 相乘, 也就是计算求和, 最后再加上

$\log(pClass1)$ 即可

#同时, 因为分母 $p(w)$ 都是一致的, 所以只需要计算并比较分子大小即可

#同时, 根据训练得到的p1Vec, 即条件概率数组, 乘以待分类向量, 即可得到待分类的条件概率

p1 = sum(vec2Classify \* p1Vec) + np.log(pClass1)

p0 = sum(vec2Classify \* p0Vec) + np.log(1 - pClass1)

if p1 > p0:

return 1

else:

return 0

"""

函数说明: 测试朴素贝叶斯分类器

Parameters:

无

Returns:

无

"""

```
def testingNB():
```

postingList, classVec = loadDataSet()

vocabList = createVocabList(postingList)

trainMat = []

for postinDoc in postingList:

trainMat.append(setOfWords2Vec(vocabList, postinDoc))

p0Vect, p1Vect, pAbusive = trainNB0(trainMat, classVec)

testEntry = ['love', 'my', 'dalmation']

#测试第一个文档

#这里注意一下, thisDoc可以是列表, array可以和对应长度的列表相乘的. 参见trainNB0中的

p1Num += trainMatrix[i]

#所以, 也可以写成thisDoc = setOfWords2Vec(vocabList, testEntry)

thisDoc = np.array(setOfWords2Vec(vocabList, testEntry))

#如果结果是1, 则是侮辱类, 否则为0, 就是非侮辱类的

if classifyNB(thisDoc, p0Vect, p1Vect, pAbusive) == 1:

print(testEntry, '属于侮辱类')

elif classifyNB(thisDoc, p0Vect, p1Vect, pAbusive) == 0:

print(testEntry, '属于非侮辱类')

#测试另外一个文档

testEntry = ['stupid', 'garbage']

thisDoc = np.array(setOfWords2Vec(vocabList, testEntry))

if classifyNB(thisDoc, p0Vect, p1Vect, pAbusive) == 1:

print(testEntry, '属于侮辱类')

elif classifyNB(thisDoc, p0Vect, p1Vect, pAbusive) == 0:

print(testEntry, '属于非侮辱类')

```
if __name__ == '__main__':
```

testingNB()

结果如下:

```
In [8]: runfile('C:/Users/HeatonHsu/.spyder-py3/temp.py', wdir='C:/Users/HeatonHsu/.spyder-py3')
['love', 'my', 'dalmation'] 属于非侮辱类
['stupid', 'garbage'] 属于侮辱类
```

In [9]:

## 4.5 项目案例2: 使用朴素贝叶斯过滤垃圾邮件

### 4.5.1 项目概述

朴素贝叶斯的一个最著名的应用: 电子邮件垃圾过滤

### 4.5.2 开发流程

- 收集数据: 提供文本文件.
- 准备数据: 将文本文件解析成词条向量.
- 分析数据: 检查词条确保解析的正确性.
- 训练算法: 使用我们之前建立的trainNB0()函数.
- 测试算法: 使用classifyNB(), 并且构建一个新的测试函数来计算文档集的错误率.
- 使用算法: 构建一个完整的程序对一组文档进行分类, 将错分的文档输出到屏幕上.

### 4.5.3 具体实现过程

#### 一 准备数据: 切分文本 (textParse函数)

前一节中的词向量是预先给定的, 下面介绍如何从文本文档中构建自己的词列表.

具体代码如下:

```
"""
函数说明: 接收一个大字符串并将其解析为字符串列表

Parameters:
    bigString - 一个大字符串
Reaturns:
    lowerString - 去掉少于2个字符的字符串, 并将所有字符串转换为小写, 返回字符串列表
"""
def textParse(bigString):
    # '\w*' 表示除单词, 数字意外的任意字符串
    # 两种模式, 一种是包含的正则表达式的字符串创建模式对象, 一种是直接使用
    # 下面的也可以这样写

    """
    regEx = re.compile('\w+')
```

```

listofTokens = regEx.split(a)
"""
listOfTokens = re.split('\w*',bigString) #listOfTokens本身也是列表
#列表生成式
lowerString = [tok.lower() for tok in listOfTokens if len(tok) > 2]
return lowerString

if __name__ == '__main__':
    bigString = 'Abdf,adfd,dfdwadf, if a you can'
    lowerString = textParse(bigString)
    print(lowerString)

```

运行结果如下:

```

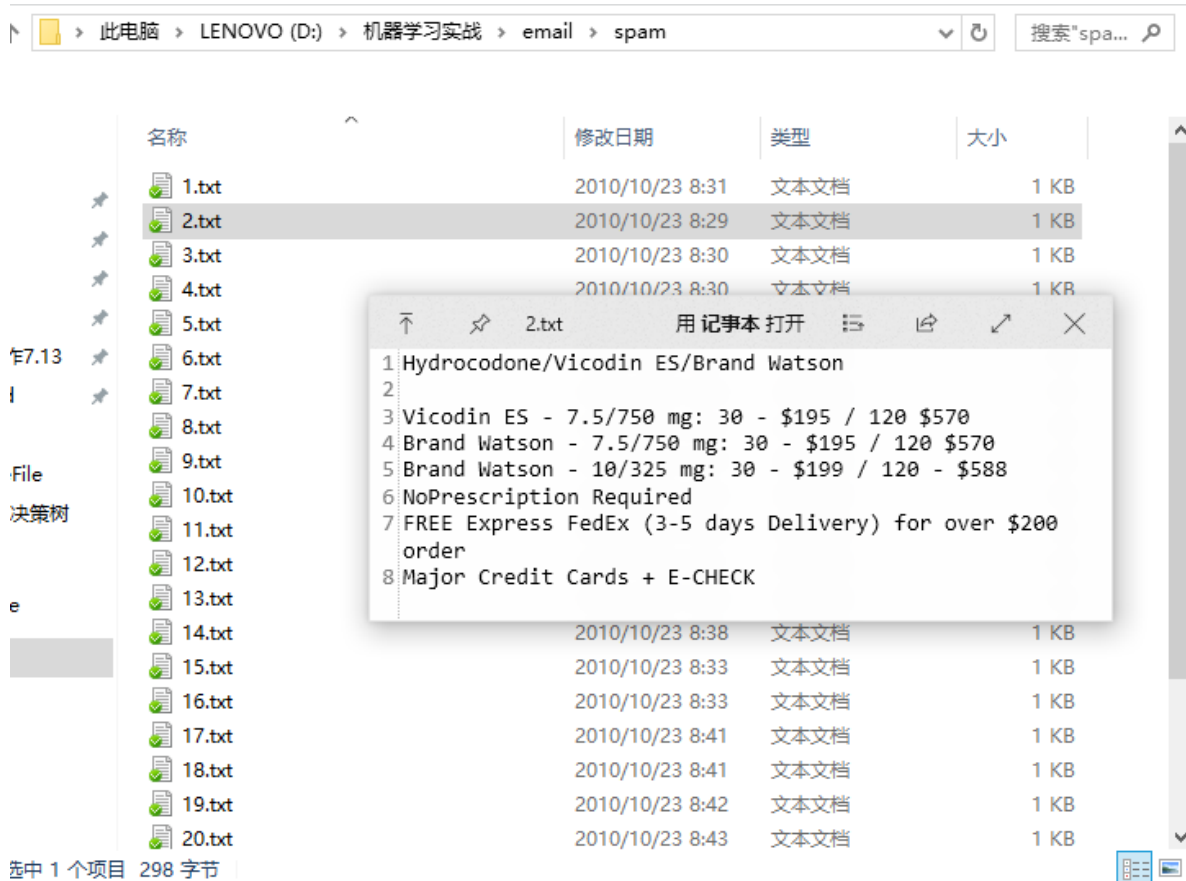
In [15]: runfile('C:/Users/HeatonHsu/.spyder-py3/temp.py', wdir='C:/Users/HeatonHsu/.spyder-py3')
['abdf', 'adfd', 'dfdwadf', 'you', 'can']

```

## 二 测试算法: 使用朴素贝叶斯进行交叉验证 (spamTest函数)

把需要的email数据集放在python目录中, 且email中包含spam和ham文件夹, 各自包含25个txt文件.

问价和txt内容如下:



File Explorer view of the 'spam' directory. The file list shows 20 .txt files. The context menu for '2.txt' displays the following text:

```

1 Hydrocodone/Vicodin ES/Brand Watson
2
3 Vicodin ES - 7.5/750 mg: 30 - $195 / 120 $570
4 Brand Watson - 7.5/750 mg: 30 - $195 / 120 $570
5 Brand Watson - 10/325 mg: 30 - $199 / 120 - $588
6 NoPrescription Required
7 FREE Express FedEx (3-5 days Delivery) for over $200
  order
8 Major Credit Cards + E-CHECK

```

具体处理代码如下:

```

"""
函数说明: 对贝叶斯垃圾邮件分类器进行自动化处理

Parameters:
    无

Returns:
    对测试集中的每封邮件进行分类, 若邮件分类错误, 则错误数加 1, 最后返回总的错误百分比
"""

```

```

def spamTest():
    docList = []
    classList = []
    fullText = []
    #遍历25个txt文件
    for i in range(1,26):
        #切分spam文件夹下的所有txt文件,并归类为1
        wordList = textParse(open('email/spam/%d.txt' %
i, 'r', errors='ignore').read())
        docList.append(wordList)
        fullText.extend(wordList)
        classList.append(1)
        #切分ham文件夹下的所有txt文件,并归类为0
        wordList = textParse(open('email/ham/%d.txt' %
i, 'r', errors='ignore').read())
        docList.append(wordList)
        fullText.extend(wordList)
        classList.append(0)

    #创建词汇表
    vocabList = createVocabList(docList)
    #创建存储训练集的索引值的列表和测试集的索引值的列表
    trainingSet = list(range(50))
    testSet = []
    #随机取10个邮件用来测试,另外40个作为训练
    #random.uniform(a,b),随机生成一个a到b之间的数
    for i in range(10):
        #随机选择索引
        randIndex = int(random.uniform(0, len(trainingSet)))
        #添加测试集的索引
        testSet.append(trainingSet[randIndex])
        #删除训练索引列表中删除选中的测试索引
        del(trainingSet[randIndex])
    #创建训练集矩阵和训练集类别标签系向量
    trainMat = []
    trainClasses = []
    #遍历训练集
    for docIndex in trainingSet:
        #将生成的词集模型添加到训练矩阵中
        trainMat.append(setOfWords2Vec(vocabList, docList[docIndex]))
        #将类别添加到训练集类别标签系向量中
        trainClasses.append(classList[docIndex])
    #训练朴素贝叶斯模型
    p0V, p1V, pSpam = trainNB0(np.array(trainMat), np.array(trainClasses))
    #错误分类计数
    errorCont = 0
    for docIndex in testSet:
        wordVector = setOfWords2Vec(vocabList, docList[docIndex])
        if classifyNB(np.array(wordVector), p0V, p1V, pSpam) !=
classList[docIndex]:
            errorCont += 1
        print('分类错误的测试集: ', docList[docIndex])
    print('错误率: %.2f%%' % (float(errorCont) / len(testSet) *100))

```



### 三 整合代码

把textParse函数和前面的creatVocabList,setOfWords2Vec,trainNB0和classifyNB函数整合在一起有:

```
import numpy as np
import re
import random

"""
函数说明：将切分的样本词条整理成不重复的词条

Parameters:
    dataSet - 样本数据集(即之前切分的样本词条)
Returns:
    vocabSet - 不重复的词条
"""

def createVocabList(dataSet):
    # 初始化一个空的集合(集合是不重复的)
    vocaSet = set([])
    # 遍历dataSet每个元素
    for document in dataSet:
        # 取并集
        vocaSet = vocaSet | set(document)
    # 返回不重复的列表
    return list(vocaSet)

"""
函数说明：根据coclaList里的词汇表，将输入的inputSet向量化,向量的每个元素为1或者0

Parameters:
    vocabList - createVocabList函数返回的词汇表
    inputSet - 要输入的切分词条列表
Returns:
    returnVec - 向量化后的文档
"""

def setOfWords2Vec(vocabList, inputSet):
    # 初始化returnVec列表，列表长度为vocabList长度，元素为0
    returnVec = [0] * len(vocabList)
    # 遍历inputSet所有单词,如果出现了词汇表中的单词，则将输出的文档向量中的对应值设为1
    for word in inputSet:
        if word in vocabList:
            returnVec[vocabList.index(word)] = 1
        else:
            print("the word: %s is not in my Vocabulary!" % word)
    return returnVec

"""
函数说明：朴素贝叶斯分类器训练函数

Parameters:
    trainMatrix - 训练文档矩阵，即函数setOfWords2Vec返回的returnVec构成的矩阵

```

```

trainCategory - 训练类别(标签)向量, 即函数loadDataSet返回的classVec
Returns:
p0Vect - 非侮辱类的条件概率数组
p1Vect - 侮辱类的条件概率数组
pAbusive - 文档属于侮辱类的概率
"""

def trainNB0(trainMatrix, trainCategory):
    # 计算文档的数目, 即trainMatrix列表的长度(和trainCategory数目是对应的, 相同的)
    # postingList每一行是一个文档, 对应trainMatrix中的一个元素, 也即是一个returnVec
    numTrainDocs = len(trainMatrix)
    # 单词个数(trainMatrix[0]=trainMatrix[1]=...)
    numWords = len(trainMatrix[0])
    # 侮辱性文件的出现概率, 即trainCategory中所有的1的个数,
    # 代表的就是多少个侮辱性文件, 与文件的总数相除就得到了侮辱性文件的出现概率
    pAbusive = sum(trainCategory) / float(numTrainDocs)

    # 初始化词条出现次数, 初始为0, 改进为1
    p0Num = np.ones(numWords)
    p1Num = np.ones(numWords)
    # 分母初始化为0啊, 改进为2
    p0Denom = 2
    p1Denom = 2
    for i in range(numTrainDocs):
        # 是否是侮辱性文件
        if trainCategory[i] == 1:
            # 如果是侮辱性文件, 对侮辱性文件的向量进行加和
            # 注意array是可以直接和对应长度的列表相加的.
            p1Num += trainMatrix[i] # [0,1,1,...] + [0,1,1,...]->[0,2,2,...]
            # 对向量中的所有元素进行求和, 也就是计算所有侮辱性文件中出现的单词总数
            p1Denom += sum(trainMatrix[i])
        else:
            p0Num += trainMatrix[i]
            p0Denom += sum(trainMatrix[i])
    # 类别1, 即侮辱性文档的[P(F1|C1), P(F2|C1), P(F3|C1), P(F4|C1), P(F5|C1)...]列表
    # 即在1类别下, 每个单词出现的概率, 改进为对数
    p1Vect = np.log(p1Num / p1Denom) # 如# [1,2,3,5]/90->[1/90,...]
    # 类别0, 即正常文档的[P(F1|C0), P(F2|C0), P(F3|C0), P(F4|C0), P(F5|C0)...]列表
    # 即在0类别下, 每个单词出现的概率, 改进为对数
    p0Vect = np.log(p0Num / p0Denom)
    return p0Vect, p1Vect, pAbusive

```

"""

函数说明: 朴素贝叶斯分类函数

Parameters:

vec2Classify - 待分类数组, 如[1,0,1,...], 注意是np.array类型的

p0Vec - trainNB返回的p0Vect, 也就是类别为0, 即正常类文档的条件概率数组,  
即

[log(P(X1|C0)), log(P(X2|C0)), log(P(X3|C0)), log(P(X4|C0)), log(P(X5|C0))...]数组

p1Vec - trainNB返回的p1Vect, 也就是类别为1, 即侮辱类文档的条件概率数组,  
即

[log(P(X1|C1)), log(P(X2|C1)), log(P(X3|C1)), log(P(X4|C1)), log(P(X5|C1))...]数组

pClass1 - 文档属于侮辱类的概率, 即trainNB中的pAbusive

Returns:

```

0 - 文档属于正常类,非侮辱类
1 - 文档属于侮辱类
"""

def classifyNB(vec2Classify, p0Vec, p1Vec, pClass1):
    # 因为p1Vect已经取了对数,所以要计算 $p(x_{\{i\}}|c1)$ 相乘,也就是计算求和,最后再加上
    log(pClass1)即可
    # 同时, 因为分母 $p(w)$ 都是一致的, 所以只需要计算并比较分子大小即可
    # 同时, 根据训练得到的p1Vec,即条件概率数组, 乘以待分类向量,即可得到待分类的条件概率
    p1 = sum(vec2Classify * p1Vec) + np.log(pClass1)
    p0 = sum(vec2Classify * p0Vec) + np.log(1 - pClass1)
    if p1 > p0:
        return 1
    else:
        return 0

```

"""

函数说明: 接收一个大字符串并将其解析为字符串列表

Parameters:

bigString - 一个大字符串

Returns:

lowerString - 去掉少于2个字符的字符串, 并将所有字符串转换为小写, 返回字符串列表

"""

```

def textParse(bigString):
    # '\w*'表示除单词,数字意外的任意字符串
    # 两种模式,一种是包含的正则表达式的字符串创建模式对象,一种是直接使用
    # 下面的也可以这样写

    """
    regex = re.compile('\w*')
    listOfTokens = regex.split(a)
    """

    listOfTokens = re.split('\w+', bigString) #listOfTokens本身也是列表
    #列表生成式
    lowerString = [tok.lower() for tok in listOfTokens if len(tok) > 2]
    return lowerString

```

"""

函数说明: 对贝叶斯垃圾邮件分类器进行自动化处理

Parameters:

无

Returns:

对测试集中的每封邮件进行分类, 若邮件分类错误, 则错误数加 1, 最后返回总的错误百分比

"""

```

def spamTest():
    docList = []
    classList = []
    fullText = []
    #遍历25个txt文件

```

```

for i in range(1,26):
    #切分spam文件夹下的所有txt文件,并归类为1
    wordList = textParse(open('email/spam/%d.txt' %
i, 'r', errors='ignore').read())
    docList.append(wordList)
    fullText.extend(wordList)
    classList.append(1)
    #切分ham文件夹下的所有txt文件,并归类为0
    wordList = textParse(open('email/ham/%d.txt' %
i, 'r', errors='ignore').read())
    docList.append(wordList)
    fullText.extend(wordList)
    classList.append(0)

#创建词汇表
vocabList = createVocabList(docList)
#创建存储训练集的索引值的列表和测试集的索引值的列表
trainingSet = list(range(50))
testSet = []
#随机取10个邮件用来测试,另外40个作为训练
#random.uniform(a,b),随机生成一个a到b之间的数
for i in range(10):
    #随机选择索引
    randIndex = int(random.uniform(0,len(trainingSet)))
    #添加测试集的索引
    testSet.append(trainingSet[randIndex])
    #删除训练索引列表中删除选中的测试索引
    del(trainingSet[randIndex])
#创建训练集矩阵和训练集类别标签系向量
trainMat = []
trainClasses = []
#遍历训练集
for docIndex in trainingSet:
    #将生成的词集模型添加到训练矩阵中
    trainMat.append(setOfWords2Vec(vocabList, docList[docIndex]))
    #将类别添加到训练集类别标签系向量中
    trainClasses.append(classList[docIndex])
#训练朴素贝叶斯模型
p0V, p1V, pSpam = trainNB0(np.array(trainMat), np.array(trainClasses))
#错误分类计数
errorCont = 0
for docIndex in testSet:
    wordVector = setOfWords2Vec(vocabList, docList[docIndex])
    if classifyNB(np.array(wordVector), p0V, p1V, pSpam) !=
classList[docIndex]:
        errorCont += 1
    print('分类错误的测试集: ', docList[docIndex])
print('错误率: %.2f%%' % (float(errorCont) / len(testSet) *100))

if __name__ == '__main__':
    spamTest()

```

运行结果如下:

```
Naive_Bayes_Hsu03 x
H:\Anaconda3\python.exe D:/机器学习实战/Naive_Bayes_Hsu03.py
分类错误的测试集: ['yeah', 'ready', 'may', 'not', 'here', 'because', 'jar', 'jar', 'has', 'plane', 'tickets', 'germany', 'for']
错误率: 10.00%

Process finished with exit code 0
```