菊安酱的机器学习第3期

菊安酱的直播间：https://live.bilibili.com/14988341

每周一晚8:00 菊安酱和你不见不散哦~(^o^)/~

更新日期：2018-11-19

作者：菊安酱

课件内容说明：

- 本文为作者原创内容，转载请注明作者和出处
- 如果想获得此课件及完整视频，可扫描下方二维码，回复"k"进群
- 若有任何疑问，请给作者留言。

完整版视频及课件：http://edu.cda.cn/course/966

12期完整版课纲

直播时间：每周一晚8:00

直播内容：

期数	算法
第1期	k-近邻算法
第2期	决策树
第3期	朴素贝叶斯
第4期	Logistic回归
第5期	支持向量机
第6期	AdaBoost 算法
第7期	线性回归
第8期	树回归
第9期	K-均值聚类算法
第10期	Apriori 算法
第11期	FP-growth 算法
第12期	奇异值分解SVD

朴素贝叶斯

CDA 数据分析师 www.cda.cn

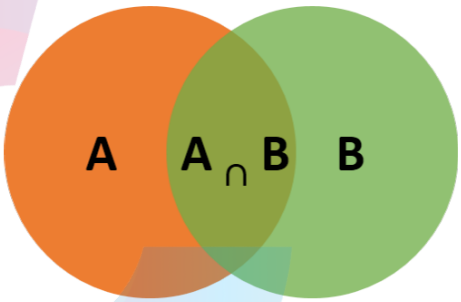
一、概述

贝叶斯分类算法是统计学的一种概率分类方法，朴素贝叶斯分类是贝叶斯分类中最简单的一种。其分类原理就是利用贝叶斯公式根据某特征的先验概率计算出其后验概率，然后选择具有最大后验概率的类作为该特征所属的类。之所以称之为"朴素"，是因为贝叶斯分类只做最原始、最简单的假设：所有的特征之间是统计独立的。

假设某样本X有$a_1, a_2, \ldots, a_n$个属性,那么有$P(X) = P(a_1, a_2, \ldots, a_n) = P(a_1) * P(a_2) * \ldots * P(a_n)$。满足这样的公式就说明特征统计独立。

1. 条件概率公式

条件概率(Condittional probability)，就是指在事件B发生的情况下，事件A发生的概率，用P(A|B)来表示。



根据文氏图可知：在事件B发生的情况下，事件A发生的概率就是P(A∩B)除以P(B)。

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

同理可得：

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

所以，

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

接着看全概率公式，如果事件$A_1, A_2, A_3, \ldots, A_n$构成一个完备事件且都有正概率，那么对于任意一个事件B则有：

$$P(B) = P(BA_1) + P(BA_2) + \ldots + P(BA_n)$$
$$= P(B|A_1)P(A_1) + P(B|A_2)P(A_2) + \ldots + P(B|A_n)P(A_n)$$

$$P(B) = \sum_{i=1}^{n} P(A_i)P(B|A_i)$$

2. 贝叶斯推断

根据条件概率和全概率公式，可以得到贝叶斯公式如下：

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

$$P(A_i|B) = P(A_i)\frac{P(B|A_i)}{\sum_{i=1}^{n} P(A_i)P(B|A_i)}$$

P(A)称为"先验概率"（Prior probability），即在B事件发生之前，我们对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，我们对A事件概率的重新评估。

P(B|A)/P(B)称为"可能性函数"（Likely hood），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为：**后验概率 = 先验概率 * 调整因子**

如果"可能性函数">1，意味着"先验概率"被增强，事件A的发生的可能性变大；

如果"可能性函数"=1，意味着B事件无助于判断事件A的可能性；

如果"可能性函数"<1，意味着"先验概率"被削弱，事件A的可能性变小。

3. 嫁？还是不嫁？这是一个问题……

为了加深对朴素贝叶斯的理解，我们……

颜值	性格	上进否	嫁与否
帅	好	上进	嫁
不帅	好	一般	不嫁
不帅	不好	不上进	不嫁
帅	好	一般	嫁
不帅	好	上进	嫁
帅	不好	一般	不嫁
帅	好	不上进	嫁
不帅	不好	上进	不嫁
帅	不好	上进	嫁
不帅	好	不上进	不嫁

假如某男（帅，性格不好，不上进）向女生求婚，该女生嫁还是不嫁?

根据贝叶斯公式:

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

转换成分类任务的表达式:

$$P(\text{类别}|\text{特征}) = P(\text{类别})\frac{P(\text{特征}|\text{类别})}{P(\text{特征})}$$

我们这个例子，按照朴素贝叶斯的求解，可以转换为计算$P(\text{嫁}|\text{帅 性格不好 不上进})$和$P(\text{不嫁}|\text{帅 性格不好 不上进})$，最终选择嫁与不嫁的答案。

根据贝叶斯公式可知

$$P(\text{嫁}|\text{帅 性格不好 不上进}) = P(\text{嫁})\frac{P(\text{帅}|\text{嫁})P(\text{性格不好}|\text{嫁})P(\text{不上进}|\text{嫁})}{P(\text{帅 性格不好 不上进})}$$

$$P(\text{不嫁}|\text{帅 性格不好 不上进}) = P(\text{不嫁})\frac{P(\text{帅}|\text{不嫁})P(\text{性格不好}|\text{不嫁})P(\text{不上进}|\text{不嫁})}{P(\text{帅 性格不好 不上进})}$$

分母的计算用到的是全概率公式:

$$P(B) = \sum_{i=1}^{n} P(A_i)P(B|A_i)$$

所以$P(\text{帅 性格不好 不上进}) =$

$$P(\text{嫁})P(\text{帅}|\text{嫁})P(\text{性格不好}|\text{嫁})P(\text{不上进}|\text{嫁})$$
$$+ P(\text{不嫁})P(\text{帅}|\text{不嫁})P(\text{性格不好}|\text{不嫁})P(\text{不上进}|\text{不嫁})$$

由上表可以得出:

P(嫁)= 5/10 = 1/2

P(不嫁)= 5/10 = 1/2

P(帅|嫁) * P(性格不好|嫁) * P(不上进|嫁)= 4/5 * 1/5 * 1/5

P(帅|不嫁) * P(性格不好|不嫁) * P(不上进|不嫁) = 1/5 * 3/5 * 2/5

对于类别“嫁”的贝叶斯分子为：

P(嫁) * P(帅|嫁) * P(性格不好|嫁) * P(不上进|嫁) = 1/2 * 4/5 * 1/5 * 1/5 = 2/125

对于类别“不嫁”的贝叶斯分子为：

P(不嫁) * P(帅|不嫁) * P(性格不好|不嫁) * P(不上进|不嫁) = 1/2 * 1/5 * 3/5 * 2/5 = 3/125

所以最终结果为：

P(嫁|帅\ 性格不好\ 不上进) = (2/125) / (2/125 + 3/125) = 40%

P(不嫁|帅\ 性格不好\ 不上进) = (3/125) / (2/125 + 3/125) = 60%

60% > 40%，该女生选择不嫁。

二、朴素贝叶斯种类

在scikit-learn中，一共有3个朴素贝叶斯的分类算法。分别是GaussianNB，MultinomialNB和BernoulliNB。

1. GaussianNB

GaussianNB就是先验为高斯分布（正态分布）的朴素贝叶斯，假设每个标签的数据都服从简单的正态分布。

$$P(X_j = x_j|Y = C_k) = \frac{1}{\sqrt{2\pi\sigma_k^2}}exp\left(-\frac{(x_j - \mu_k)^2}{2\sigma_k^2}\right)$$

其中C_k为Y的第k类类别。μ_k和σ_k^2为需要从训练集估计的值。

这里，用scikit-learn简单实现一下GaussianNB。

```
#导入包
import pandas as pd
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

#导入数据集
from sklearn import datasets
iris=datasets.load_iris()
```

```
#切分数据集
Xtrain, Xtest, ytrain, ytest = train_test_split(iris.data,
                                                iris.target,
                                                random_state=12)
#建模
clf = GaussianNB()
clf.fit(Xtrain, ytrain)

#在测试集上执行预测，proba导出的是每个样本属于某类的概率
clf.predict(Xtest)
clf.predict_proba(Xtest)

#测试准确率
accuracy_score(ytest, clf.predict(Xtest))
```

2. MultinomialNB

MultinomialNB就是先验为多项式分布的朴素贝叶斯。它假设特征是由一个简单多项式分布生成的。多项分布可以描述各种类型样本出现次数的概率，因此多项式朴素贝叶斯非常适合用于描述出现次数或者出现次数比例的特征。该模型常用于文本分类，特征表示的是次数，例如某个词语的出现次数。

多项式分布公式如下：

$$P(X_j = x_{jl}|Y = C_k) = \frac{x_{jl} + \lambda}{m_k + n\lambda}$$

其中，$P(X_j = x_{jl}|Y = C_k)$是第k个类别的第j维特征的第l个取值条件概率。m_k是训练集中输出为第k类的样本个数。λ 为一个大于0的常数，常常取为1，即拉普拉斯平滑。也可以取其他值。

3. BernoulliNB

BernoulliNB就是先验为伯努利分布的朴素贝叶斯。假设特征的先验概率为二元伯努利分布，即如下式：

$$P(X_j = x_{jl}|Y = C_k) = P(j|Y = C_k)x_{jl} + (1 - P(j|Y = C_k)(1 - x_{jl})$$

此时l只有两种取值。x_{jl}只能取值0或者1。

在伯努利模型中，每个特征的取值是布尔型的，即true和false，或者1和0。在文本分类中，就是一个特征有没有在一个文档中出现。

总结：

- 一般来说，如果样本特征的分布大部分是连续值，使用GaussianNB会比较好。
- 如果如果样本特征的分布大部分是多元离散值，使用MultinomialNB比较合适。
- 而如果样本特征是二元离散值或者很稀疏的多元离散值，应该使用BernoulliNB。

三、朴素贝叶斯之鸢尾花数据实验

应用GaussianNB对鸢尾花数据集进行分类。

1. 导入数据集

```
import numpy as np
import pandas as pd
import random


dataSet =pd.read_csv('iris.txt',header = None)
dataSet.head()
```

2. 切分训练集和测试集

```
import random

"""
函数功能：随机切分训练集和测试集
参数说明：
    dataSet:输入的数据集
    rate：训练集所占比例
返回：切分好的训练集和测试集
"""

def randSplit(dataSet, rate):

    l = list(dataSet.index)                 #提取出索引
    random.shuffle(l)                       #随机打乱索引
    dataSet.index = l                       #将打乱后的索引重新赋值给原数据集
    n = dataSet.shape[0]                    #总行数
    m = int(n * rate)                       #训练集的数量
    train = dataSet.loc[range(m), :]        #提取前m个记录作为训练集
    test = dataSet.loc[range(m, n), :]      #剩下的作为测试集
    dataSet.index = range(dataSet.shape[0]) #更新原数据集的索引
    test.index = range(test.shape[0])       #更新测试集的索引
    return train, test
```

```
train,test= randSplit(dataSet, 0.8)
```

3. 构建高斯朴素贝叶斯分类器

```
def gnb_classify(train,test):
    labels = train.iloc[:,-1].value_counts().index              #提取训练集的标签种类
    mean =[]                                                    #存放每个类别的均值
    std =[]                                                     #存放每个类别的方差
    result = []                                                 #存放测试集的预测结果
    for i in labels:
```

```
        item = train.loc[train.iloc[:,-1]==i,:]                #分别提取出每一种类别
        m = item.iloc[:,:-1].mean()                            #当前类别的平均值
        s = np.sum((item.iloc[:,:-1]-m)**2)/(item.shape[0])    #当前类别的方差
        mean.append(m)                                         #将当前类别的平均值追加至列表
        std.append(s)                                          #将当前类别的方差追加至列表
    means = pd.DataFrame(mean,index=labels)                    #变成DF格式，索引为类标签
    stds = pd.DataFrame(std,index=labels)                      #变成DF格式，索引为类标签
    for j in range(test.shape[0]):
        iset = test.iloc[j,:-1].tolist()                       #当前测试实例
        iprob = np.exp(-1*(iset-means)**2/(stds*2))/(np.sqrt(2*np.pi*stds)) #正态分布公式
        prob = 1                                               #初始化当前实例总概率
        for k in range(test.shape[1]-1):                       #遍历每个特征
            prob *= iprob[k]                                   #特征概率之积即为当前实例概率
            cla = prob.index[np.argmax(prob.values)]           #返回最大概率的类别
        result.append(cla)
    test['predict']=result
    acc = (test.iloc[:,-1]==test.iloc[:,-2]).mean()            #计算预测准确率
    print(f'模型预测准确率为{acc}')
    return test
```

4. 测试模型预测效果

将切分好的训练集和测试集带入模型，查看模型预测结果

```
gnb_classify(train,test)
```

运行10次，查看结果

```
for i in range(20):
    train,test= randSplit(dataSet, 0.8)
    gnb_classify(train,test)
```

四、使用朴素贝叶斯进行文档分类

朴素贝叶斯一个很重要的应用就是文本分类，所以我们以在线社区留言为例。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，那么就将该留言标志为内容不当。过滤这类内容是一个很常见的需求。对此问题建立两个类型：侮辱类和非侮辱类，使用1和0分别表示。

我们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。考虑出现所有文档中的单词，再决定将哪些单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。简单起见，我们先假设已经将本文切分完毕，存放到列表中，并对词汇向量进行分类标注。

1. 构建词向量

留言文本已经被切分好，并且人为标注好类别，用于训练模型。类别有两类，侮辱性（1）和非侮辱性（0）。

此案例所有的函数：

- loadDataSet：创建实验数据集
- createVocabList：生成词汇表
- setOfWords2Vec：生成词向量
- get_trainMat：所有词条向量列表
- trainNB：朴素贝叶斯分类器训练函数
- classifyNB：朴素贝叶斯分类器分类函数
- testingNB：朴素贝叶斯测试函数

```
"""
函数功能：创建实验数据集
参数说明：无参数
返回：
    postingList：切分好的样本词条
    classVec：类标签向量
"""
def loadDataSet():
    dataSet=[['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']] #切分好的词条
    classVec = [0,1,0,1,0,1]    #类别标签向量，1代表侮辱性词汇，0代表非侮辱性词汇
    return dataSet,classVec
```

```
dataSet,classVec=loadDataSet()
```

生成词汇表：

```
"""
函数功能：将切分的样本词条整理成词汇表（不重复）
参数说明：
    dataSet：切分好的样本词条
返回：
    vocabList：不重复的词汇表
"""
def createVocabList(dataSet):
    vocabSet = set()                        #创建一个空的集合
    for doc in dataSet:                     #遍历dataSet中的每一条言论
        vocabSet = vocabSet | set(doc)      #取并集
        vocabList = list(vocabSet)
    return vocabList
```

```
vocabList = createVocabList(dataSet)
```

生成词向量：

```
"""
```

```
函数功能：根据vocabList词汇表，将inputSet向量化，向量的每个元素为1或0
参数说明：
    vocabList：词汇表
    inputSet：切分好的词条列表中的一条
返回：
    returnVec：文档向量,词集模型
"""
def setOfWords2Vec(vocabList, inputSet):
    returnVec = [0] * len(vocabList)                    #创建一个其中所含元素都为0的向量
    for word in inputSet:                               #遍历每个词条
        if word in vocabList:                           #如果词条存在于词汇表中，则变为1
            returnVec[vocabList.index(word)] = 1
        else:
            print(f" {word} is not in my Vocabulary!" )
    return returnVec                                    #返回文档向量
```

所有词条向量列表：

```
"""
函数功能：生成训练集向量列表
参数说明：
    dataSet：切分好的样本词条
返回：
    trainMat：所有的词条向量组成的列表
"""
def get_trainMat(dataSet):
    trainMat = []                                       #初始化向量列表
    vocabList = createVocabList(dataSet)                #生成词汇表
    for inputSet in dataSet:                            #遍历样本词条中的每一条样本
        returnVec=setOfWords2Vec(vocabList, inputSet)   #将当前词条向量化
        trainMat.append(returnVec)                      #追加到向量列表中
    return trainMat
```

测试函数运行结果：

```
trainMat = get_trainMat(dataSet)
```

2. 朴素贝叶斯分类器训练函数

词向量构建好之后，我们就可以来构建朴素贝叶斯分类器的训练函数了。

```
"""
函数功能：朴素贝叶斯分类器训练函数
参数说明：
    trainMat：训练文档矩阵
    classVec：训练类别标签向量
返回：
    p0V：非侮辱类的条件概率数组
    p1V：侮辱类的条件概率数组
```

```
    pAb: 文档属于侮辱类的概率
"""
def trainNB(trainMat,classVec):
    n = len(trainMat)                       #计算训练的文档数目
    m = len(trainMat[0])                    #计算每篇文档的词条数
    pAb = sum(classVec)/n                   #文档属于侮辱类的概率
    p0Num = np.zeros(m)                     #词条出现数初始化为0
    p1Num = np.zeros(m)                     #词条出现数初始化为0
    p0Denom = 0                             #分母初始化为0
    p1Denom = 0                             #分母初始化为0
    for i in range(n):                      #遍历每一个文档
        if classVec[i] == 1:                #统计属于侮辱类的条件概率所需的数据
            p1Num += trainMat[i]
            p1Denom += sum(trainMat[i])
        else:                               #统计属于非侮辱类的条件概率所需的数据
            p0Num += trainMat[i]
            p0Denom += sum(trainMat[i])
    p1V = p1Num/p1Denom
    p0V = p0Num/p0Denom
    return p0V,p1V,pAb      #返回属于非侮辱类,侮辱类和文档属于侮辱类的概率
```

测试函数，查看结果

```
p0V,p1V,pAb = trainNB(trainMat, classVec)
```

```
print(vocabList)
```

```
['steak', 'flea', 'I', 'ate', 'dalmation', 'food', 'quit', 'stop', 'take', 'park', 'so', 'has', 'stupid', 'my', 'dog', 'worthless', 'buying', 'not', 'love', 'help', 'posting', 'maybe', 'problems', 'him', 'cute', 'garbage', 'licks', 'mr', 'to', 'please', 'is', 'how']
```

```
p0V
```

```
array([0.04166667, 0.04166667, 0.04166667, 0.04166667, 0.04166667,
       0.        , 0.        , 0.04166667, 0.        , 0.        ,
       0.04166667, 0.04166667, 0.        , 0.125     , 0.04166667,
       0.        , 0.        , 0.        , 0.04166667, 0.04166667,
       0.        , 0.        , 0.04166667, 0.08333333, 0.04166667,
       0.        , 0.04166667, 0.04166667, 0.04166667, 0.04166667,
       0.04166667, 0.04166667])
```

```
p1V
```

```
array([0.        , 0.        , 0.        , 0.        , 0.        ,
       0.05263158, 0.05263158, 0.05263158, 0.05263158, 0.05263158,
       0.        , 0.        , 0.15789474, 0.        , 0.10526316,
       0.10526316, 0.05263158, 0.05263158, 0.        , 0.        ,
       0.05263158, 0.05263158, 0.        , 0.05263158, 0.        ,
       0.05263158, 0.        , 0.        , 0.05263158, 0.        ,
       0.        , 0.        ])
```

倒数第7个

p0=0

p1=0.05

3. 测试朴素贝叶斯分类器

```
from functools import reduce
"""
函数功能：朴素贝叶斯分类器分类函数
参数说明：
    vec2Classify：待分类的词条数组
    p0V：非侮辱类的条件概率数组
```

```
    p1V：侮辱类的条件概率数组
    pAb：文档属于侮辱类的概率
返回：
    0：属于非侮辱类
    1：属于侮辱类
"""
def classifyNB(vec2Classify, p0V, p1V, pAb):
    p1 = reduce(lambda x,y:x*y, vec2Classify * p1V) * pAb          #对应元素相乘
    p0 = reduce(lambda x,y:x*y, vec2Classify * p0V) * (1 - pAb)
    print('p0:',p0)
    print('p1:',p1)
    if p1 > p0:
        return 1
    else:
        return 0
```

```
"""
函数功能：朴素贝叶斯测试函数
参数说明：
    testVec：测试样本
返回：测试样本的类别
"""
def testingNB(testVec):
    dataSet,classVec = loadDataSet()                    #创建实验样本
    vocabList = createVocabList(dataSet)                #创建词汇表
    trainMat= get_trainMat(dataSet)                     #将实验样本向量化
    p0V,p1V,pAb = trainNB(trainMat,classVec)            #训练朴素贝叶斯分类器
    thisone = setOfWords2Vec(vocabList, testVec)        #测试样本向量化
    if classifyNB(thisone,p0V,p1V,pAb):
        print(testVec,'属于侮辱类')                      #执行分类并打印分类结果
    else:
        print(testVec,'属于非侮辱类')                    #执行分类并打印分类结果
```

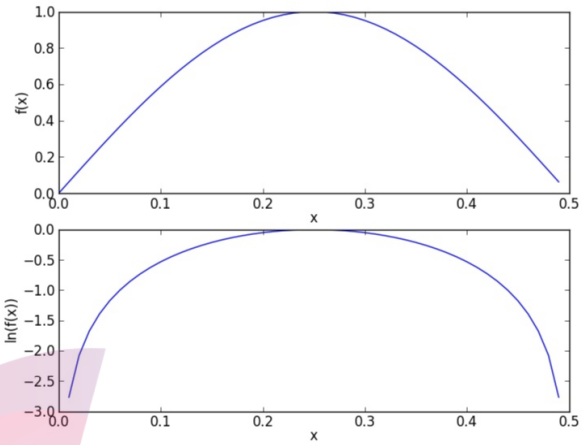
```
#测试样本1
testVec1 = ['love', 'my', 'dalmation']
testingNB(testVec1)

#测试样本2
testVec2 = ['stupid', 'garbage']
testingNB(testVec2)
```

你会发现，这样写的算法无法进行分类，p0和p1的计算结果都是0，显然结果错误。这是为什么呢?

4. 朴素贝叶斯改进之拉普拉斯平滑

利用贝叶斯分类器对文档进行分类时，要计算多个概率的乘积以获得文档属于某个类别的概率，即计算p(w0|1)p(w1|1)p(w2|1)。如果其中有一个概率值为0，那么最后的成绩也为0。显然，这样是不合理的，为了降低这种影响，可以将所有词的出现数初始化为1，并将分母初始化为2。这种做法就叫做拉普拉斯平滑(Laplace Smoothing)又被称为加1平滑，是比较常用的平滑方法，它就是为了解决0概率问题。

另外一个遇到的问题就是下溢出，这是由于太多很小的数相乘造成的。我们在计算乘积时，由于大部分因子都很小，所以程序会下溢或者得不到正确答案。为了解决这个问题，对乘积结果取自然对数。通过求对数可以避免下溢出或者浮点数舍入导致的错误。同时，采用自然对数进行处理不会有任何损失。下图给出函数f(x)和ln(f(x))的曲线。



检查这两条曲线就会发现它们在相同区域内同时增加或者减少，并且在相同点上取到极值。它们的取值虽然不同，但不影响最终结果。因此可以修改代码如下：

```
def trainNB(trainMat,classVec):
    n = len(trainMat)                       #计算训练的文档数目
    m = len(trainMat[0])                    #计算每篇文档的词条数
    pAb = sum(classVec)/n                   #文档属于侮辱类的概率
    p0Num = np.ones(m)                      #词条出现数初始化为1
    p1Num = np.ones(m)                      #词条出现数初始化为1
    p0Denom = 2                             #分母初始化为2
    p1Denom = 2                             #分母初始化为2
    for i in range(n):                      #遍历每一个文档
        if classVec[i] == 1:                #统计属于侮辱类的条件概率所需的数据
            p1Num += trainMat[i]
            p1Denom += sum(trainMat[i])
        else:                               #统计属于非侮辱类的条件概率所需的数据
            p0Num += trainMat[i]
            p0Denom += sum(trainMat[i])
    p1V = np.log(p1Num/p1Denom)
    p0V = np.log(p0Num/p0Denom)
    return p0V,p1V,pAb                      #返回属于非侮辱类,侮辱类和文档属于侮辱类的概率
```

查看代码运行结果：

```
p0V,p1V,pAb = trainNB(trainMat,classVec)
```

```
def classifyNB(vec2Classify, p0V, p1V, pAb):
    p1 = sum(vec2Classify * p1V) + np.log(pAb)        #对应元素相乘
    p0 = sum(vec2Classify * p0V) + np.log(1- pAb)     #对应元素相乘
    if p1 > p0:
        return 1
    else:
        return 0
```

测试代码运行结果：

```
#测试样本1
testVec1 = ['love', 'my', 'dalmation']
testingNB(testVec1)

#测试样本2
testVec2 = ['stupid', 'garbage']
testingNB(testVec2)
```

这样看，结果就没什么问题了。

其他

- 菊安酱的直播间：https://live.bilibili.com/14988341
- 下周一（2018/11/26）将讲解 Logistic 算法，欢迎各位进入菊安酱的直播间观看直播
- 如有问题，可以给我留言哦~