朴素贝叶斯进行文档分类

朴素贝叶斯分类器的特点：高维数据各个分量之间相互独立

朴素贝叶斯的使用场景：文档的自动分类。

在文档分类中，整个文档是一个实例，文档中的某些元素构成特征。我们可以观察文档中出现的词，并把每个词作为一个特征，而每个词的出现或者不出现作为该特征的值，这样得到的特征数目就会和词汇表中的词一样多。

贝叶斯决策的实质是在比较$P(D|h)P(h)$的大小，其中h指不同的类别，比如在辨别是否为垃圾邮件的时候就分为垃圾邮件和非垃圾邮件两类。

```
提取所有文档中的词条并进行去重（构建词典）
获取所有文档的类别
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（此类别下词条总数）
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 P(词条|类别)
返回该文档属于每个类别的条件概率 P(类别|文档的所有词条)
```

实践概述

构建一个快速过滤器在屏蔽社区留言板的侮辱性言论，即，如果某条留言使用了负面言论，就将其识别为内容不当。

建立两个类别：侮辱类言论和非侮辱类言论，分别表示为1和0

流程

```
收集数据（这里为了简单，我们自己构建数据集）
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
使用算法
```

收集数据

```
def loadDataSet():
    postingList=[['my', 'dog', 'has', 'flea', 'problems', 'help', 'please'],
  #切分的词条
                 ['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]#类别标签向量，1代表负面言论，0代表不是
    return postingList,classVec
```

准备数据

```
# 函数说明:将切分的实验样本词条整理成不重复的词条列表，也就是词汇表
def createVocabList(dataSet):
    vocabSet = set([])                          #创建一个空的不重复列表
    for document in dataSet:
        vocabSet = vocabSet | set(document) #取并集
    return list(vocabSet)

# 函数说明:根据vocabList词汇表，将inputSet向量化，向量的每个元素为1或0
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，则置1（此时只记录有没有出现，不记录出现次数）
            returnVec[vocabList.index(word)] = 1
        else: print("the word: %s is not in my Vocabulary!" % word)
    return returnVec                                                    #返回文档向量
```

现在知道了一个词是否出现在某个文档中，也知道每一个文档的类别，接下来就可以进行算法的训练。

训练算法

首先还是看贝叶斯准则

$$P(h|D)=\frac{P(D,h)}{P(D)}=\frac{P(D,h)P(h)}{P(D)}$$

将它重写为下面的形式

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

其中w为向量，由于这里我们的词向量使用one-hot表示法，故向量的长度与词汇表的长度相等。

在算法的训练中，我们要达到的主要目的是，计算出词汇表中的每个词在每个类别的文档中出现的概率，除此之外还需要计算每个类别的先验概率。

```
# 函数说明:朴素贝叶斯分类器训练函数
def trainNB0(trainMatrix,trainCategory):
```

```
    numTrainDocs = len(trainMatrix)                                 #计算训练的文档数目
    numWords = len(trainMatrix[0])                                  #计算每篇文档的词条数
    pAbusive = sum(trainCategory)/float(numTrainDocs)               #文档属于负面类的概率，即先
验概率
    p0Num = np.zeros(numWords); p1Num = np.zeros(numWords)#创建numpy.zeros数组,词条
出现数初始化为0
    p0Denom = 0.0; p1Denom = 0.0     #分母初始化为0
    for i in range(numTrainDocs):
        if trainCategory[i] == 1:    #统计属于负面言论类的条件概率所需的数据，即
P(w0|1),P(w1|1),P(w2|1)···
            p1Num += trainMatrix[i] #向量加和
            p1Denom += sum(trainMatrix[i])
        else:                        #统计属于正常言论类的条件概率所需的数据，即
P(w0|0),P(w1|0),P(w2|0)···
            p0Num += trainMatrix[i] #向量加和
            p0Denom += sum(trainMatrix[i])
    #进行完循环操作后，我们得到的是两个类别中每个词出现的总次数的数组，除以该类别下的总词数可以
得到出现的概率
    p1Vect = p1Num/p1Denom
    p0Vect = p0Num/p0Denom
    return p0Vect,p1Vect,pAbusive#返回属于负面言论类的条件概率数组，属于非负面言论类的条件
概率数组，先验概率
```

两个问题：一是需要数据平滑，防止在计算概率时出现乘0的现象；二是要防止小数相乘造成下溢出，失去函数特性；

改进：可以将所有词出现的次数初始化为1，分母初始化为2；

对概率值取自然对数，这样就可以避免下溢出。

```
# 函数说明:朴素贝叶斯分类器训练函数
def trainNB0(trainMatrix,trainCategory):
    numTrainDocs = len(trainMatrix)                                 #计算训练的文档数目
    numWords = len(trainMatrix[0])                                  #计算每篇文档的词条数
    pAbusive = sum(trainCategory)/float(numTrainDocs)    #文档属于负面言论类的概率，即
先验概率
    p0Num = np.ones(numWords); p1Num = np.ones(numWords)#创建numpy.ones数组,词条出
现数初始化为1
    p0Denom = 2.0; p1Denom = 2.0                                    #分母初始化为2
    for i in range(numTrainDocs):
        if trainCategory[i] == 1:#统计属于负面言论类的条件概率所需的数据，即
P(w0|1),P(w1|1),P(w2|1)···
            p1Num += trainMatrix[i]
            p1Denom += sum(trainMatrix[i])
        else:                    #统计属于非负面言论类的条件概率所需的数据，即
P(w0|0),P(w1|0),P(w2|0)···
            p0Num += trainMatrix[i]
            p0Denom += sum(trainMatrix[i])
    print("p1Num:")
    print(p1Num)
    print("p0Denom")
    print(p0Denom)

    p1Vect = np.log(p1Num/p1Denom)                                  #取对数，防止下溢出
    p0Vect = np.log(p0Num/p0Denom)
    #返回属于负面言论类的条件概率数组，属于非负面言论类的条件概率数组，先验概率
    return p0Vect,p1Vect,pAbusive
```

使用算法

```
def classifyNB(vec2Classify, p0Vec, p1Vec, pClass1):
    p1 = sum(vec2Classify * p1Vec) + np.log(pClass1)    #概率的乘，由于取了对数，变成了对数相加
    p0 = sum(vec2Classify * p0Vec) + np.log(1.0 - pClass1)
    if p1 > p0:
        return 1
    else:
        return 0

def testingNB():
    listOPosts,listClasses = loadDataSet()
    myVocabList = createVocabList(listOPosts)
    trainMat=[]
    for postinDoc in listOPosts:
        trainMat.append(setOfWords2Vec(myVocabList, postinDoc))
    p0V,p1V,pAb = trainNB0(np.array(trainMat),np.array(listClasses))
    testEntry = ['love', 'my', 'dalmation']
    thisDoc = np.array(setOfWords2Vec(myVocabList, testEntry))
    print (testEntry,'classified as: ',classifyNB(thisDoc,p0V,p1V,pAb))
    testEntry = ['Mr', 'stupid']
    thisDoc = np.array(setOfWords2Vec(myVocabList, testEntry))
    print (testEntry,'classified as: ',classifyNB(thisDoc,p0V,p1V,pAb))
```