实验 Naive Bayes

姓名：　李昀哲　　学号：　20123101

1 题目和数据：

题目：Naive Bayes

- 基于朴素贝叶斯算法的言论过滤器
- 基于朴素贝叶斯算法实现垃圾邮件过滤

数据描述:

- **言论过滤器**

  一组实验样本作为词条库，用以训练分类器，如图所示：

```
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']]
```

- **垃圾邮件过滤**

  bayes_email 目录下有两个文件夹：ham 和 spam，其中 spam 文件下的 txt 文件为垃圾邮件。

ham ▸ 1.txt

```
1  Hi Peter,
2
3  With Jose out of town, do you want to
4  meet once in a while to keep things
5  going and do some interesting stuff?
6
7  Let me know
8  Eugene
```

2 算法：

朴素贝叶斯法 (Naive Bayes model) 是基于贝叶斯定理与特征条件独立假设的分类方法，和决策树模型相比，朴素贝叶斯分类器(Naive Bayes Classifier 或 NBC)发源于古典数学理论，有着坚实的数学基础，以及稳定的分类效率。同时，NBC 模型所需估计的参数很少，对缺失数据不太敏感，算法也比较简单。理论上，NBC 模型与其他分类方法相比具有最小的误差率。但是实际上并非总是如此，这是因为 NBC 模型假设属性之间相互独立，这个假设在实际应用中往往是不成立的，这给 NBC 模型的正确分类带来了一定影响。

其优点是：朴素贝叶斯算法假设了数据集属性之间是相互独立的，因此算法的逻辑性十分简单，并且算法较为稳定，当数据呈现不同的特点时，朴素贝叶斯的分类性能不会有太大的差异。换句话说就是朴素贝叶斯算法的健壮性比较好，对于不同类型的数据集不会呈现出

太大的差异性。当数据集属性之间的关系相对比较独立时，朴素贝叶斯分类算法会有较好的效果。

缺点是：属性独立性的条件同时也是朴素贝叶斯分类器的不足之处。数据集属性的独立性在很多情况下是很难满足的，因为数据集的属性之间往往都存在着相互关联，如果在分类过程中出现这种问题，会导致分类的效果大大降低。

3 代码及结果

数据集读入

```
def loadDataSet():
    """
    函数说明:创建实验样本
    Parameters:
        无
    Returns:
        postingList - 实验样本切分的词条
        classVec - 类别标签向量
    """
    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
```

生成词条向量

```
def setOfWords2Vec(vocabList, inputSet):
    """
    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
    Parameters:
        vocabList - createVocabList返回的列表
        inputSet - 切分的词条列表
    Returns:
        returnVec - 文档向量,词集模型
    """
    returnVec = [0] * len(vocabList)                                    #创建
    for word in inputSet:                                               #遍历
        if word in vocabList:                                           #如果
            returnVec[vocabList.index(word)] = 1
        else:
            print("the word: %s is not in my Vocabulary!" % word)
    return returnVec
```

创建词汇库

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

    Parameters:
        dataSet - 整理的样本数据集
    Returns:
        vocabSet - 返回不重复的词条列表，也就是词汇表
    """
    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)
```

言论过滤

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。如果其中有一个概率值为 0，那么最后的乘积也为 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。这种做法就叫做拉普拉斯平滑(Laplace Smoothing)，是比较常用的平滑方法。
下溢问题。由于太多很小的数相乘会导致下溢问题。两个小数相乘，越乘越小，这样就造成了下溢出。在程序中，许多很小的数相乘，最后四舍五入计算结果可能就变成 0。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。

```
def trainNB(trainMatrix,trainCategory):
    """
    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setOfWords2Vec返回的returnVec构成的矩阵
        trainCategory - 训练类别标签向量，即loadDataSet返回的classVec
    Returns:
        p0Vect - 非的条件概率数组
        p1Vect - 侮辱类的条件概率数组
        pAbusive - 文档属于侮辱类的概率
    """
    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)       #文档属于侮辱类的概率
    p0Num = np.ones(numWords); p1Num = np.ones(numWords)    #创建numpy.ones数组,词条出现数初始化为1，拉普拉斯平滑
    p0Denom = 2.0; p1Denom = 2.0                            #分母初始化为2,拉普拉斯平滑
    for i in range(numTrainDocs):
        if trainCategory[i] == 1:                           #统计属于侮辱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:                                               #统计属于非侮辱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])
    p1Vect = np.log(p1Num/p1Denom)                          #取对数，防止下溢出
    p0Vect = np.log(p0Num/p0Denom)
    return p0Vect,p1Vect,pAbusive                           #返回属于侮辱类的条件概率数组，属于非侮辱类的条件概率数组，文档属于侮辱类的概率
```

P(词条属于侮辱类|测试词条中每个词汇的分类)是要求的结果，P(词条中每个词汇的分类|词条属于侮辱类)可以计算出来，P(词条中每个词汇的分类)是提前计算好的，P(词条属于侮辱类)也是已知的 0.5。最终计算结果，比较两个结果大小，值更大就表示测试词条属于该类。公式如下所示：

$$P(\text{词条属于侮辱类}|\text{测试词条中每个词汇的分类}) = P(\text{词条属于侮辱类}) \cdot \frac{P(\text{词条中每个词汇的分类}|\text{词条属于侮辱类})}{P(\text{词条中每个词汇的分类})}$$

$$P(\text{词条属于非侮辱类}|\text{测试词条中每个词汇的分类}) = P(\text{词条属于侮辱类}) \cdot \frac{P(\text{词条中每个词汇的分类}|\text{词条属于非侮辱类})}{P(\text{词条中每个词汇的分类})}$$

朴素贝叶斯分类

```
from functools import reduce
def classifyNB(vec2Classify, p0Vec, p1Vec, pClass1):
    """
    函数说明:朴素贝叶斯分类器分类函数

    Parameters:
        vec2Classify - 待分类的词条数组
        p0Vec - 非侮辱类的条件概率数组
        p1Vec -侮辱类的条件概率数组
        pClass1 - 文档属于侮辱类的概率
    Returns:
        0 - 属于非侮辱类
        1 - 属于侮辱类
    """
    p1 = sum(vec2Classify * p1Vec) + np.log(pClass1)        #对应元素相乘。logA * B = logA + logB，所以这里加上log(pClass1)
    p0 = sum(vec2Classify * p0Vec) + np.log(1.0 - pClass1)
    print('p0:',p0)
    print('p1:',p1)
    if p1 > p0:
        return 1
    else:
        return 0
```

测试

```
def testingNB():
    """
    函数说明:测试朴素贝叶斯分类器
    测试样本1：['love', 'my', 'dalmation']
    测试样本2：['stupid', 'garbage']
    Parameters:
        无
    Returns:
        无
    """
    listOPosts,listClasses = loadDataSet()                                  #创建实验样本
    myVocabList = createVocabList(listOPosts)                               #创建词汇表
    trainMat=[]
    for postinDoc in listOPosts:
        trainMat.append(setOfWords2Vec(myVocabList, postinDoc))             #将实验样本向量化
    p0V,p1V,pAb = trainNB(np.array(trainMat),np.array(listClasses))     #训练朴素贝叶斯分类器

    testEntry = ['love', 'my', 'dalmation']                                 #测试样本1
    thisDoc = np.array(setOfWords2Vec(myVocabList, testEntry))              #测试样本向量化
    if classifyNB(thisDoc,p0V,p1V,pAb):
        print(testEntry,'属于侮辱类')                                         #执行分类并打印分类结果
    else:
        print(testEntry,'属于非侮辱类')                                       #执行分类并打印分类结果

    testEntry = ['stupid', 'garbage']                                       #测试样本2
    thisDoc = np.array(setOfWords2Vec(myVocabList, testEntry))              #测试样本向量化
    if classifyNB(thisDoc,p0V,p1V,pAb):
        print(testEntry,'属于侮辱类')                                         #执行分类并打印分类结果
    else:
        print(testEntry,'属于非侮辱类')                                       #执行分类并打印分类结果
```

分类结果

```
p0: -7.69484807238
p1: -9.82671449373
['love', 'my', 'dalmation'] 属于非侮辱类
p0: -7.2093402566
p1: -4.70275051433
['stupid', 'garbage'] 属于侮辱类
```

文本解析

```
"""
函数说明:接收一个大字符串并将其解析为字符串列表

Parameters:
    无
Returns:
    无
"""
def textParse(bigString):                                           #将字符串转换为字符列表
    listOfTokens = re.split(r'\W*', bigString)                      #将特殊符号作为切分标志进行字符串切分，即非字母、非数字
    return [tok.lower() for tok in listOfTokens if len(tok) > 2]     #除了单个字母，例如大写的I，其它单词变成小写
```

过滤测试

```
def spamTest():
    """
    函数说明：将数据集分为训练集和测试集，使用交叉验证的方式测试朴素贝叶斯分类器的准确性
    """
    docList = []; classList = []; fullText = []
    for i in range(1, 26):                                          #遍历25个txt文件
        wordList = textParse(open('bayes_email1/spam/%d.txt' % i, 'r', encoding='ISO-8859-1').read())     #读取每个垃圾邮件，并字符串转换成
        docList.append(wordList)
        fullText.append(wordList)
        classList.append(1)                                         #标记垃圾邮件，1表示垃圾文件
        wordList = textParse(open('bayes_email1/ham/%d.txt' % i, 'r',encoding='ISO-8859-1').read())      #读取每个非垃圾邮件，并字符串转换成
        docList.append(wordList)
        fullText.append(wordList)
        classList.append(0)                                         #标记非垃圾邮件，1表示垃圾文件
    vocabList = createVocabList(docList)                            #创建词汇表，不重复
    trainingSet = list(range(50)); testSet = []                     #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
    for i in range(10):                                             #从50个邮件中，随机挑选出40个作为训练集,10个做测试集
        randIndex = int(random.uniform(0, len(trainingSet)))        #随机选取索索引值
        testSet.append(trainingSet[randIndex])                      #添加测试集的索引值
        del(trainingSet[randIndex])                                 #在训练集列表中删除添加到测试集的索引值
    trainMat = []; trainClasses = []                                #创建训练集矩阵和训练集类别标签系向量
    for docIndex in trainingSet:                                    #遍历训练集
        trainMat.append(setOfWords2Vec(vocabList, docList[docIndex]))       #将生成的词集模型添加到训练矩阵中
        trainClasses.append(classList[docIndex])                    #将类别添加到训练集类别标签系向量中
    p0V, p1V, pSpam = trainNB0(np.array(trainMat), np.array(trainClasses))  #训练朴素贝叶斯模型
    errorCount = 0                                                  #错误分类计数
    for docIndex in testSet:                                        #遍历测试集
        wordVector = setOfWords2Vec(vocabList, docList[docIndex])   #测试集的词集模型
        if classifyNB(np.array(wordVector), p0V, p1V, pSpam) != classList[docIndex]:    #如果分类错误
            errorCount += 1                                         #错误计数加1
            print("分类错误的测试集：",docList[docIndex])
    print('错误率：%.2f%%' % (float(errorCount) / len(testSet) * 100))
```

测试结果

```
if __name__ == '__main__':
    spamTest()
```

```
分类错误的测试集：  ['home', 'based', 'business', 'opportunity', 'knocking', 'your', 'door', 'don', 'rude', 'and', 'let', 'this', 'chance',
'you', 'can', 'earn', 'great', 'income', 'and', 'find', 'your', 'financial', 'life', 'transformed', 'learn', 'more', 'here', 'your', 'succ
ess', 'work', 'from', 'home', 'finder', 'experts']
错误率：10.00%
```