**华中科技大学计算机科学与技术学院**

**机器学习报告**



专 业： 计算机科学与技术

班 级： 计算机ACM1801班

学 号： U201814741

姓 名： 宋奕欣

成 绩：

指导教师： 何琨

**完成日期： 2020年 4月 24 日**

# 实验一

## 一、实验题目：KNN（K近邻）

## 二、实验要求

使用Python程序设计语言，给定最近邻数k和描述每个元组的数据n，实现k近邻分类算法，并根据实验结果进行分析与讨论。进一步理解KNN算法的优势与缺点。

## 三、算法设计

1. 读取MNIST数据集数据到程序中来，注意在将byte读成int时的读入方式（本数据集是大端读入），并预先归一化，即将像素值大于127的，设定为1，否则为0。
2. 将训练集分成训练集和evaluate集两部分。
3. 将数据集用numpy进行合理的拓展，可以在计算时不用循环计算。
4. 依次对evaluate集和测试集中的每一个图片的一组像素值与训练集中的每个图片的像素值求距离。（这里采用欧式距离）
5. 将距离最近的k个进行比较分析，选择出现频率最大的类标号作为测试数据的标签，并与真实值进行比较。
6. 分析数据结果。

## 四、实验环境与平台

Windows 10 + python IDLE

语言：python

## 五、程序实现

import os

import numpy as np

from matplotlib import pyplot as plt

import operator

IMAGE\_ROW = 28

IMAGE\_COL = 28

IMAGE\_SIZE = 28\*28

def readFileFirst(filename):

dimension = []

with open(filename,'rb') as pf:

data = pf.read(4)

magic\_num = int.from\_bytes(data,byteorder='big')

dimension\_cnt = magic\_num & 0xff

for i in range(dimension\_cnt):

data = pf.read(4)

dms = int.from\_bytes(data,byteorder='big')

dimension.append(dms)

return dimension

def read\_image\_vector(filename,head\_len,offset,amount):

image\_mat=np.zeros((amount,IMAGE\_SIZE),dtype=np.uint8)

with open(filename,'rb') as pf:

pf.seek(head\_len+IMAGE\_SIZE\*offset)

for ind in range(amount):

image = np.zeros((1,IMAGE\_SIZE),dtype=np.uint8)

for row in range(IMAGE\_SIZE):#处理28行数据，

data = pf.read(1)#单个读出数据

pix = int.from\_bytes(data,byteorder='big')#由byte转换为int类型，

#二值化

if pix >127:image[0][row] = 1

image\_mat[ind,:]=image

#print('read\_image\_vector：当前进度%0.2f%%'%(ind\*100.0/amount),end='\r')

print('图像解析完毕')

return image\_mat

def read\_label\_vector(filename,head\_len,offset,amount):

label\_list=[]

with open(filename,'rb') as pf:

#pf 指向label的第number个数据,magic\_num的长度为4，dimension\_cnt单个长度为4

pf.seek(head\_len+offset)

for ind in range(amount):

data = pf.read(1)

label = int.from\_bytes(data,byteorder='big')#由byte转换为int类型，

label\_list.append(label)

#print('read\_label\_vector：当前进度%0.2f%%'%(ind\*100.0/amount),end='\r')

print('标签解析完毕')

return label\_list

def read\_image\_label\_vector(image\_file,label\_file,offset,amount):

image\_dim = readFileFirst(image\_file)

label\_dim = readFileFirst(label\_file)

#获取样本image和label的头文件长度

image\_head\_len = 4 \* len(image\_dim) + 4

label\_head\_len = 4 \* label\_dim +4

#得到image和label的向量

image\_mat = read\_image\_vector(image\_file,image\_head\_len,offset,amount)

label\_list = read\_label\_vector(label\_file,label\_head\_len,offset,amount)

return image\_mat,label\_list

def knn\_classify(test\_data, train\_dataset, train\_label, k):

train\_dataset\_amount = train\_dataset.shape[0]#行数，也即训练样本的的个数，shape[1]为列数

#将输入test\_data变成了和train\_dataset行列数一样的矩阵

test\_rep\_mat = np.tile(test\_data, (train\_dataset\_amount,1))#tile(mat,(x,y)) Array类 mat 沿着行重复x次，列重复y次

diff\_mat = test\_rep\_mat - train\_dataset

sq\_diff\_mat = diff\_mat\*\*2

sq\_dist = sq\_diff\_mat.sum(axis=1)

dist\_index = sq\_dist.argsort()

for i in range(5):

label = train\_label[dist\_index[i]]

show\_picture(train\_dataset,label,dist\_index[i])

os.system("pause")

class\_count={}

for i in range(k):

label = train\_label[dist\_index[i]]

#如果属于某个类，在该类的基础上加1，相当于增加其权重，如果不是某个类则新建字典的一个key并且等于1

class\_count[label] = class\_count.get(label,0) + 1

#降序排列

class\_count\_list = sorted(class\_count.items(), key=operator.itemgetter(1), reverse=True)

#print('排序后的分类结果：',class\_count\_list)

return class\_count\_list[0][0]

def show\_picture(image\_mat,label,number):

image = np.zeros((IMAGE\_ROW,IMAGE\_COL),dtype = np.uint8)

for i in range(IMAGE\_ROW):

for j in range(IMAGE\_COL):

image[i][j] = image\_mat[number][i\*IMAGE\_ROW+j]

plt.imshow(image)

plt.title('picture number = %d,label = %d' %(number,label))

plt.show()

def run\_KNN(train\_image\_mat,train\_label\_list,test\_image\_mat,test\_label\_list,k):

err\_count = 0.0

for i in range(len(test\_image\_mat)):

#print('当前进度：%2.2f%%'%(100.0\*i/len(test\_image\_mat)))

#利用knn算法进行分类

show\_picture(test\_image\_mat,test\_label\_list[i],i)

class\_result = knn\_classify(test\_image\_mat[i], train\_image\_mat, train\_label\_list, k)#计算分类结果

if (class\_result != test\_label\_list[i]):

err\_count += 1.0

print( "\n总错误数: %d" % err\_count)

error\_rate = 100.0\*err\_count/len(test\_image\_mat)

print( "总错误率: %2.2f%%" % (error\_rate))

return error\_rate

if \_\_name\_\_ == '\_\_main\_\_':

train\_image\_file = 'F:\\syx\\机器学习\\KNN\\train-images.idx3-ubyte'

train\_label\_file = 'F:\\syx\\机器学习\\KNN\\train-labels.idx1-ubyte'

test\_image\_file = 'F:\\syx\\机器学习\\KNN\\t10k-images.idx3-ubyte'

test\_label\_file = 'F:\\syx\\机器学习\\KNN\\t10k-labels.idx1-ubyte'

train\_image\_mat,train\_label\_list=read\_image\_label\_vector(train\_image\_file,train\_label\_file,0,50000)

traineva\_image\_mat,traineva\_label\_list=read\_image\_label\_vector(train\_image\_file,train\_label\_file,50000,10000)

err\_count = 0.0

amount = 10000

K = [1,5,10,20,50,100,120]

#error\_rate = []

test\_image\_mat, test\_label\_list = read\_image\_label\_vector(test\_image\_file,test\_label\_file,0,amount)

#for i in range(len(K)):

rate = run\_KNN(train\_image\_mat,train\_label\_list,traineva\_image\_mat,traineva\_label\_list,5)

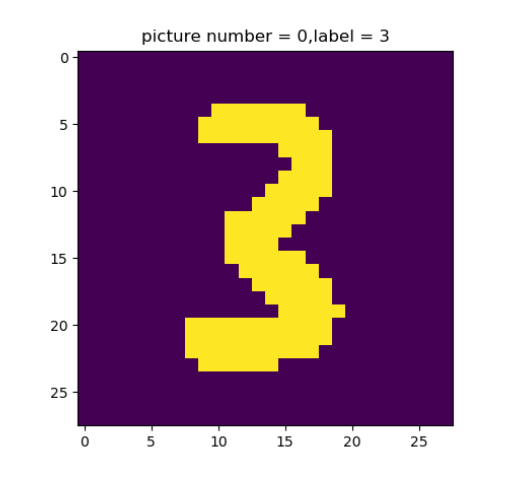
#error\_rate.append(rate)

#print(error\_rate)

## 六、实验结果

假设k=5

一张图片如图1-1，其五张距离最近的图，如图1-2。



1-1 测试图

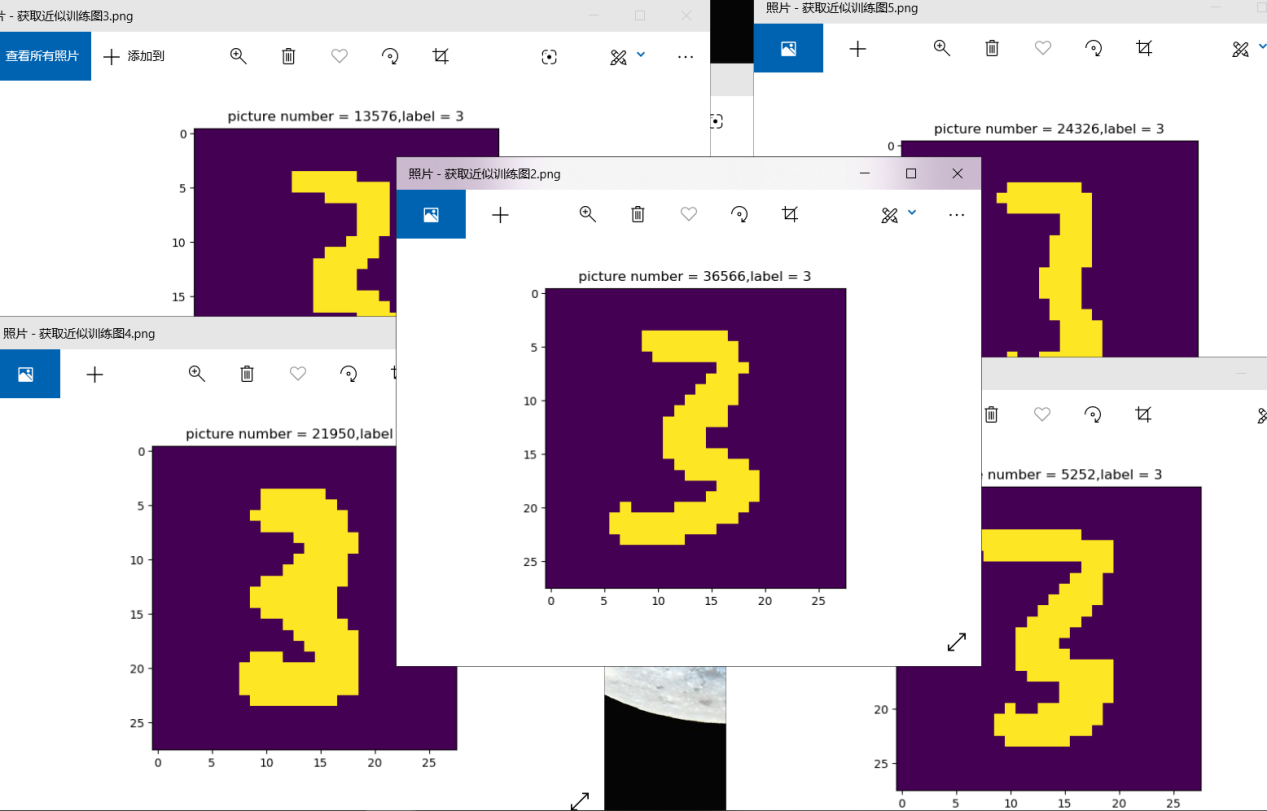


图1-2 k=5近邻训练集中的图

经过选取不同的k值，我们得到其错误率，如图1-3。

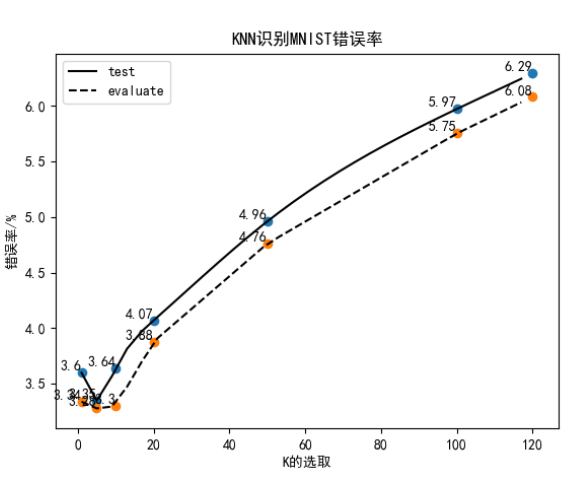


图1-3 错误率

## 七、结果分析

由实验结果可以看出，KNN算法虽然原理上比较简单，但在一些领域的使用上是非常有效的。在本实验的手写数字识别中，我们可以看到最高的正确率可以达到约97%，是非常不错的。

单就KNN算法的实现上，我们可以看到选取合适的K值是十分重要的，当K取得过小，则会导致参考值不够，从而导致错误率地提高。但K取得过大，则会可能因为过多的噪声而导致出现错误地识别。因此，在使用KNN时，一定要充分利用并行运算去加速实验地进行，这样我们可以有有效的时间里选取最合适的K值。

在使用KNN算法时，对数据的预处理也是很重要的。数据的维度如果过大，则有可能产生维度灾难。

# 实验二

## 一、实验题目：Naive Bayes（朴素贝叶斯）

## 二、实验要求

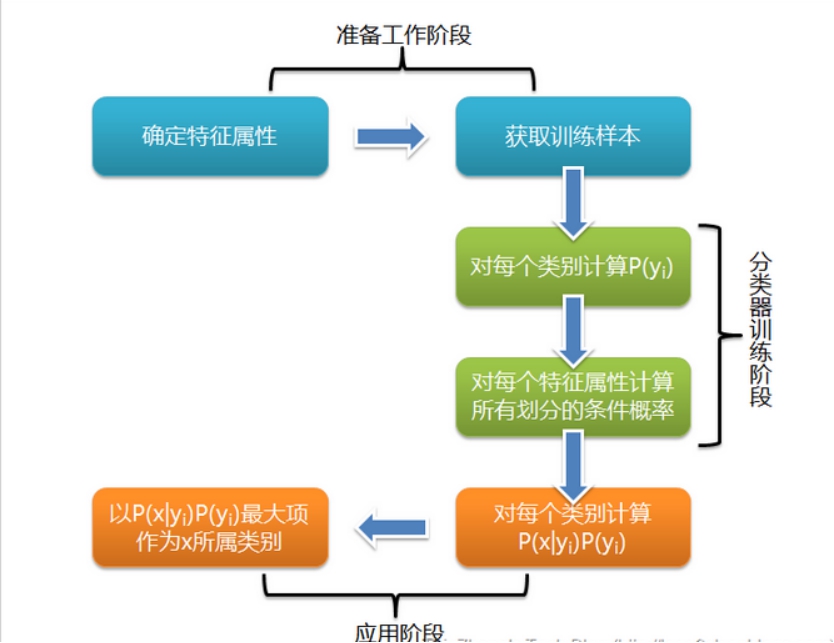
了解朴素贝叶斯算法的基本原理，能够使用朴素贝叶斯算法对数据进行分类，并能够分析训练好的分类器的性能的评估。

## 三、算法设计

整个朴素贝叶斯分类分为三个阶段：

第一阶段: 准备工作阶段，这个阶段的任务是为朴素贝叶斯分类做必要的准备，主要工作是根据具体情况确定特征属性，并对每个特征属性进行适当划分，然后由人工对一部分待分类项进行分类，形成训练样本集合。这一阶段的输入是所有待分类数据，输出是特征属性和训练样本。这一阶段是整个朴素贝叶斯分类中唯一需要人工完成的阶段，其质量对整个过程将有重要影响，分类器的质量很大程度上由特征属性、特征属性划分及训练样本质量决定。

第二阶段: 分类器训练阶段，这个阶段的任务就是生成分类器，主要工作是计算每个类别在训练样本中的出现频率及每个特征属性划分对每个类别的条件概率估计，并将结果记录。其输入是特征属性和训练样本，输出是分类器。这一阶段是机械性阶段，根据前面讨论的公式可以由程序自动计算完成。

第三阶段: 应用阶段。这个阶段的任务是使用分类器对待分类项进行分类，其输入是分类器和待分类项，输出是待分类项与类别的映射关系。这一阶段也是机械性阶段，由程序完成。

## 四、实验环境与平台

Windows 10 + python IDLE

语言：python

## 五、程序实现

中文：

import numpy as np

import re

import random

import jieba

import math

"""

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

"""

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:接收一个大字符串并将其解析为字符串列表

"""

def textParse(bigString): # 将字符串转换为字符列表

listOfTokens = re.split(r'\n', 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

return listOfTokens

"""

函数说明:测试朴素贝叶斯分类器，使用朴素贝叶斯进行交叉验证

"""

def spamTest():

docList = []

classList = []

fullText = []

stopList = textParse(open('stop.txt','r',encoding='UTF-8').read());

stopList.append('\n')

stopList.append('\t')

stopList.append('\u3000')

#print(stopList)

indexOfSet = open('newindex','r').read();

indexOfSet = indexOfSet.split('\n')

#seglist = jieba.cut(datas,cut\_all=False)

lengthOfFile = len(indexOfSet)

#j = 0

for i in indexOfSet:

filePath = i[4:]

filePath = "".join(filePath)

classType = i[0:4]

datas = open(filePath,'r').read();

datas = re.sub("[A-Za-z0-9\!\%\[\]\,\。\+\-\\_\.]","",datas)

seglist = jieba.cut(datas,cut\_all=False)

nowList = ""

for seg in seglist:

if seg not in stopList:

#print(seg)

nowList += seg

seglist = jieba.lcut(nowList,cut\_all=False)

docList.append(seglist)

if(classType=='ham '):

classList.append(0)

else:

classList.append(1)

#j += 1

#print(j)

print('预处理结束')

vocabList = createVocabList(docList) # 创建词汇表，不重复

print('词汇表创建完成')

#print(vocabList)

trainingSet = list(range(15320,lengthOfFile))

testSet = list(range(0,15320)) # 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

#print(testSet)

#print(trainingSet)

frequency1 = {}

frequency0 = {}

p1num = 0

p0num = 0

p1wordAllNum = 2

p0wordAllNum = 2

p1Vect = []

p1index = []

p0index = []

p0Vect = []

n = 0

for docIndex in trainingSet:

#print(docIndex)

if(classList[docIndex] == 1):#是spam

p1num += 1

for word in docList[docIndex]:

p1wordAllNum += 1

if word not in frequency1:

frequency1[word] = 1

else:

frequency1[word] += 1

else:

p0num += 1

for word in docList[docIndex]:

p0wordAllNum += 1

if word not in frequency0:

frequency0[word] = 1

else:

frequency0[word] += 1

print('the spam num is',p1num)

#print(frequency1)

#print(p1wordAllNum)

#print(p0wordAllNum)

pAbusive = p1num / (float)(p1num + p0num)

for key in frequency1:

#print(key)

gailv = math.log((frequency1[key]+1)/p1wordAllNum)

#print(key,':',gailv,':1:',frequency1[key]+1)

p1index.append(key)

p1Vect.append(gailv)

for key in frequency0:

gailv = math.log((frequency0[key]+1)/p0wordAllNum)

#print(key,':',gailv,':1:',frequency0[key]+1)

p0index.append(key)

p0Vect.append(gailv)

print(p1Vect)

falseNum = 0

print(frequency1)

print(frequency0)

for docIndex in testSet:

testDict = {}

ans1 = math.log(pAbusive)

ans0 = math.log(1-pAbusive)

for word in docList[docIndex]:

if word not in testDict:

testDict[word] = 1

if word in frequency1:

ans1 = ans1 + p1Vect[p1index.index(word)]

#print(p1Vect[p1index.index(word)],':',word)

else:

ans1 = ans1 + min(p1Vect)

if word in frequency0:

ans0 = ans0 + p0Vect[p0index.index(word)]

else:

ans0 = ans0 + min(p0Vect)

#print(ans1,':',ans0,':',classList[docIndex])

if ((ans1 > ans0) and classList[docIndex] != 1):

falseNum += 1

print(falseNum)

elif ((ans1 < ans0) and classList[docIndex] != 0):

falseNum += 1

print(falseNum)

print('错误率：%.2f%%' % (float(falseNum) / len(testSet) \* 100))

if \_\_name\_\_ == '\_\_main\_\_':

spamTest()

英文:

# -\*- coding: UTF-8 -\*-

import numpy as np

import re

import random

"""

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

"""

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

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

"""

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

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

else:

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

return returnVec #返回文档向量

"""

函数说明:根据vocabList词汇表，构建词袋模型

Parameters:

vocabList - createVocabList返回的列表

inputSet - 切分的词条列表

Returns:

returnVec - 文档向量,词袋模型

"""

def bagOfWords2VecMN(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，则计数加一

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

return returnVec # 返回词袋模型

"""

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 - 正常邮件类的条件概率数组

p1Vect - 垃圾邮件类的条件概率数组

pAbusive - 文档属于垃圾邮件类的概率

"""

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) # 文档属于垃圾邮件类的概率

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)

p0Vect = np.log(p0Num / p0Denom) #取对数，防止下溢出

return p0Vect, p1Vect, pAbusive # 返回属于正常邮件类的条件概率数组，属于侮辱垃圾邮件类的条件概率数组，文档属于垃圾邮件类的概率

"""

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 - 正常邮件类的条件概率数组

p1Vec - 垃圾邮件类的条件概率数组

pClass1 - 文档属于垃圾邮件的概率

Returns:

0 - 属于正常邮件类

1 - 属于垃圾邮件类

"""

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

#p1 = reduce(lambda x, y: x \* y, vec2Classify \* p1Vec) \* pClass1 # 对应元素相乘

#p0 = reduce(lambda x, y: x \* y, vec2Classify \* p0Vec) \* (1.0 - pClass1)

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

p0=sum(vec2Classify\*p0Vec)+np.log(1.0-pClass1)

print('p1:',p1)

print('p0:',p0)

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

"""

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

"""

函数说明:测试朴素贝叶斯分类器，使用朴素贝叶斯进行交叉验证

"""

def spamTest():

docList = []

classList = []

fullText = []

for i in range(1, 26): # 遍历25个txt文件

wordList = textParse(open('spam/%d.txt' % i, 'r').read()) # 读取每个垃圾邮件，并字符串转换成字符串列表

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('ham/%d.txt' % i, 'r').read()) # 读取每个非垃圾邮件，并字符串转换成字符串列表

docList.append(wordList)

fullText.append(wordList)

classList.append(0) # 标记正常邮件，0表示正常文件

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])

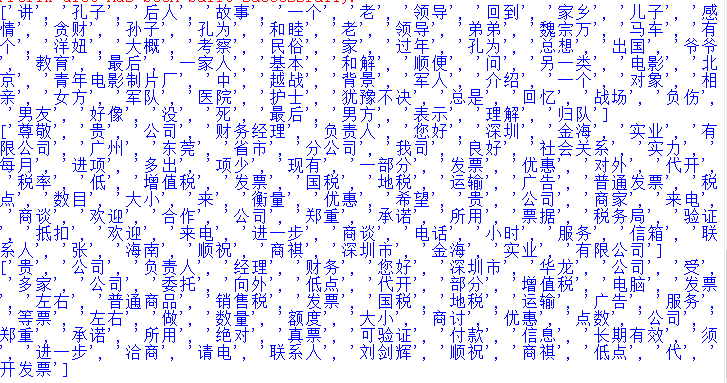
print('错误率：%.2f%%' % (float(errorCount) / len(testSet) \* 100))

if \_\_name\_\_ == '\_\_main\_\_':

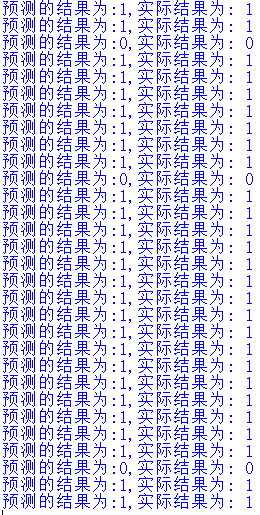
spamTest()

## 实验结果

分词结果：



利用多项式特征，下图为部分验证过程。



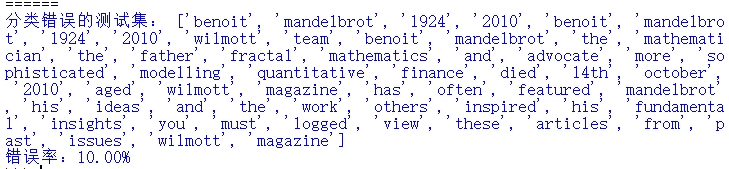
得到的错误率为：，即正确率达到了95.46%。

对分类错误的情况进行判断，可知将垃圾邮件判断成正常邮件是判断错误的主要原因，如图。



不去掉停用词对实验结果有较大影响，错误率为：

对应英文的由于采用随机的取测试样本，故错误率有时在百分之十，有时在百分之五。由于测试集只有十个，故百分之十也仅代表错误识别了极个别个。



## 七、结果分析

从结果可以看出朴素贝叶斯在垃圾邮件上的效果还是不错的，我做的最初版本是快到96%，但通过调整更合适的拉普拉斯平滑常数，以及其它的一些优化策略，比如说词袋中不存在测试集中的文字时的处理方式等等可以使得结果有更好的表现。

从结果可以看出，大部分的错误判断都是分类器把垃圾邮件错判为正常邮件。

停用词的处理对结果也是有很大的影响，去掉停用词后的效果明显高于没有去掉停用词的结果。主要由于停用词是一些很常见的，比如说连接词之类的，对于邮件本身内容不具有判别性，故将这些词语加入判断训练之中则会干扰实际的效果。

# 参考文献

[1]https://blog.csdn.net/sinat\_26811377/article/details/101691336.Python 中文分词并去除停用词

[2]https://blog.csdn.net/codejas/article/details/80356544.Python入门：jieba库的使用

[3]https://www.cnblogs.com/beikew/p/10246883.html.KNNpython实现