**Lab 8 Naive Bayes应用实践**

|  |  |  |  |
| --- | --- | --- | --- |
| 学号 | 姓名 | 年级 | 专业 |
| 18353070 | 谭嘉伟 | 2018 | 软件工程 |

1. 实验目的：
2. 熟悉朴素贝叶斯分类器的基本原理，了解如何用Python实现朴素贝叶斯分类器
3. 了解一定的数据平滑化的处理方法
4. 练习用Python实现朴素贝叶斯分类器分类数据的能力
5. 进一步锻炼使用Python语言编程的能力
6. 进一步提升用tensorflow实现算法的能力，尝试用tensorflow实现贝叶斯分类器
7. 比较了解贝叶斯分类器和KNN等分类算法的优缺点
8. 实验环境：

1、硬件环境：PC 1.60GHz四核处理器 8.00G内存

2、软件环境：Windows10 64位操作系统

Python3.5.6 tensorflow1.4.0 Jupyter Notebook

1. 实验步骤：
2. 准备数据：从文本中构建词向量
3. 切分文本成词

首先解决将一封邮件分词的问题，可以用Python的方法轻松实现英文分词。

A 用split()方法切分

mySent = 'This book is the best book on Python or M.L. I have ever laid eyes upon.'

mySent.split()

B 用正则表达式分词

import re

regEx = re.compile('\\W+')

listOfTokens = regEx.split(mySent)

listOfTokens

C 用列表推导式分词

#[tok for tok in listOfTokens if len(tok) > 0]

emailText = open('D://PUB128/email/ham/6.txt').read()

listOfTokens = regEx.split(emailText)

listOfTokens=[tok.lower() for tok in listOfTokens if len(tok) > 0]

print(listOfTokens)

1. 生成词汇表

运行下面的代码建立词汇表

listOPost,listClasses=loadDataSet()

myVocabList=createVocabList(listOPost)

myVocabList

1. 生成词向量

对bagOfWords2Vec函数进行适当修改如下

def bagOfWords2Vec(vocabList,inputSet):

# vocablist为词汇表，inputSet为输入的邮件

returnVec=[0]\*len(vocabList)

for word in inputSet:

if word in vocabList:

#查找单词的索引

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

else: print ("the word is not in my vocabulary")

return returnVec

运行以下代码测试：

setOfWords2Vec0=setOfWords2Vec(myVocabList,listOPost[0])

print(setOfWords2Vec0)

setOfWords2Vec3=setOfWords2Vec(myVocabList,listOPost[3])

print(setOfWords2Vec3)

1. 训练算法：计算概率

输入以下代码进行训练

trainMat = []

for postinDoc in listOPost:

trainMat.append(bagOfWords2Vec (myVocabList, postinDoc))

print(trainMat)

p0V, p1V, pAb = train(trainMat, listClasses)

print(pAb)

print(p0V)

print(p1V)

1. 测试过程：根据现实情况修改分类器

spamTest()实现了随机选取40封邮件作为训练样本，10封邮件做测试

用下面的代码计算10次测试的平均错误率

ave=0

for i in range(10):

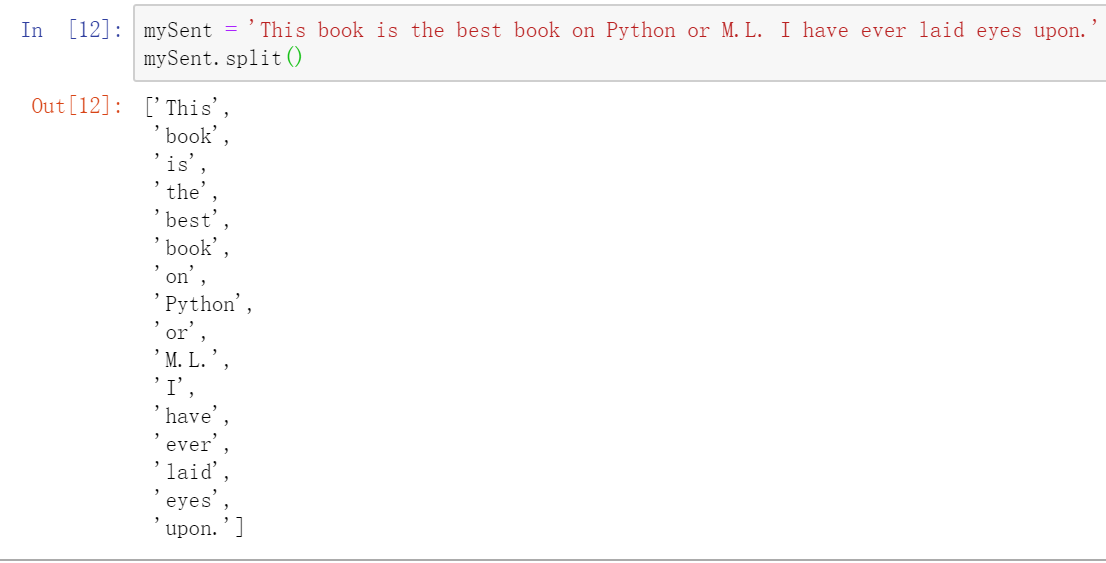
ave=ave+spamTest()

print('average error rate:',ave/10)

1. 实验结果与分析：
2. 准备数据：从文本中构建词向量
3. 切分文本成词

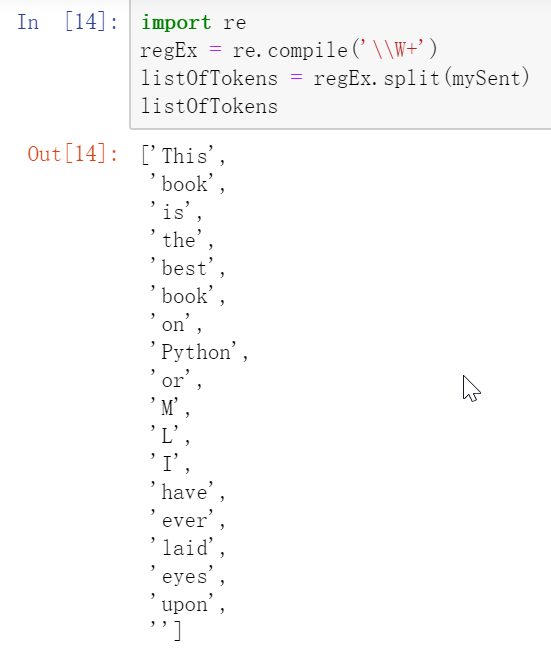
首先解决将一封邮件分词的问题，可以用Python的方法轻松实现英文分词。

A 用split()方法切分



B 用正则表达式分词

可以看到分割方法与A有些不同，M.L.的.也被当做为分隔符而删除，M和L则被当成了两个单词而分开



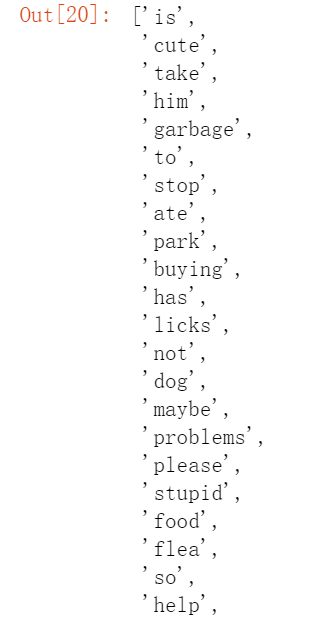
C 用列表推导式分词

可以看到这段语句成功读取了第六封正常邮件的内容并且成功将单词切分

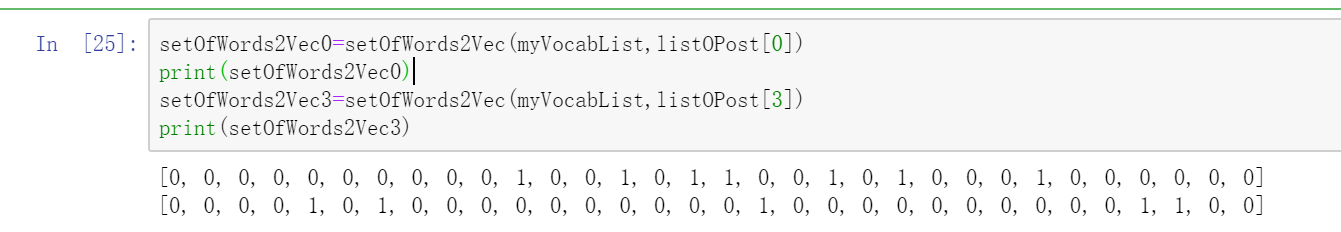


1. 创建词汇表

词汇得到了去重，一个单词只出现了一次



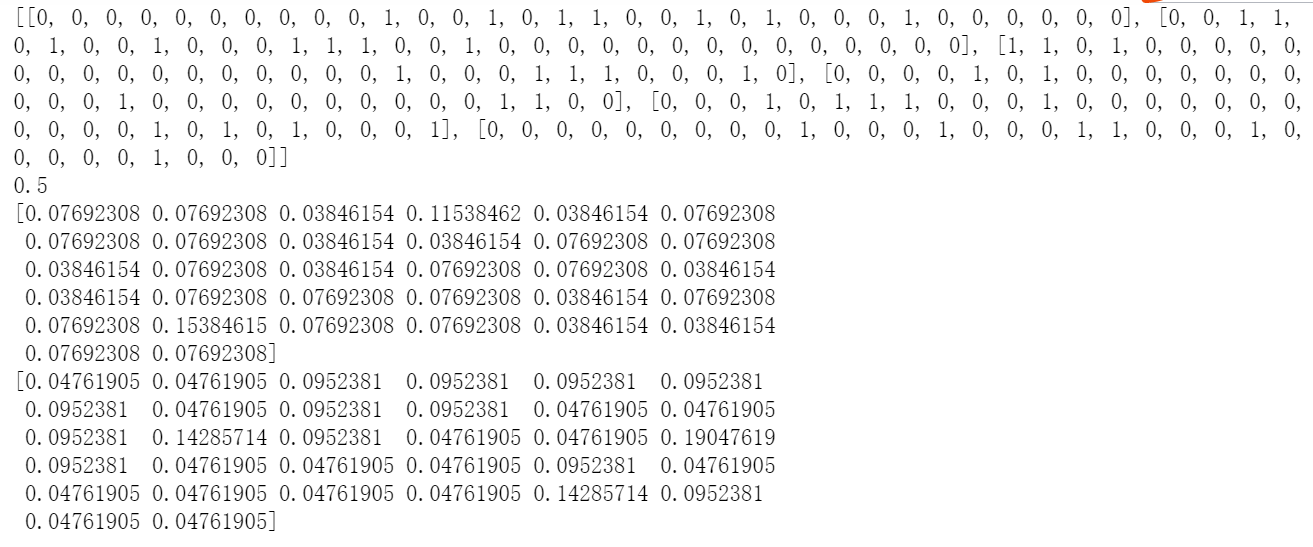
1. 得到的词向量如下



现在已经成功地实现将文本信息数字化为向量信息

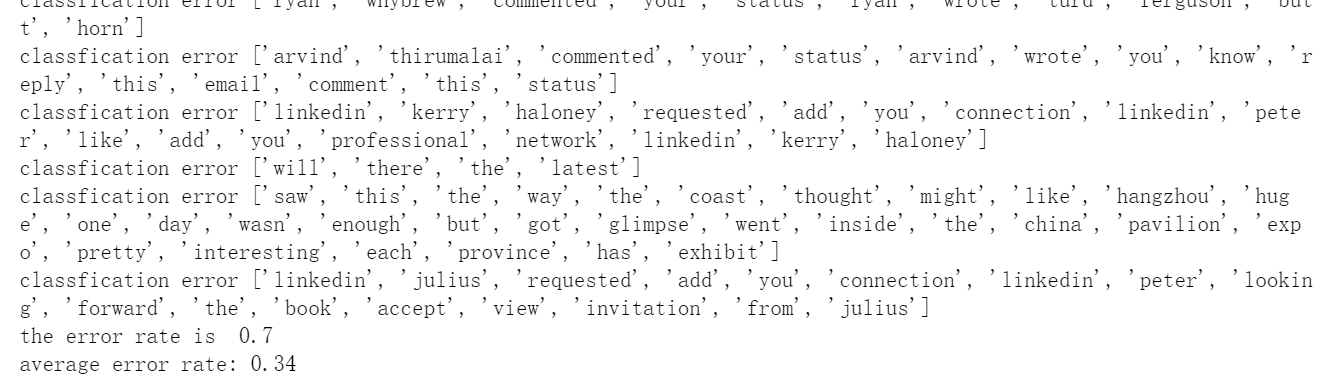
1. 训练算法：计算概率

P(词|垃圾邮件)和P(词|正常邮件)的概率被计算出来(取对数后的概率)



1. 测试过程：根据现实情况修改分类器

得到平均错误率结果如下



平均错误率为0.34，还是比较高的。且有时单次测试错误率会很大，达到0.7，这样的分类效果不太理想。与训练集较少，文字邮件的特性丰富有一定关系。改进需要一方面增大训练集，一方面尽可能多地考虑不同类型的垃圾邮件需要放入到训练集中。另一方面，把垃圾邮件分类为正常邮件比把正常邮件分类为垃圾邮件好一些，这个错误率对分类错误一视同仁。应该设计更科学的评价指标，更认可算法将垃圾邮件分类为正常邮件的错误，因为这样的垃圾邮件识别标准更能被邮件用户所接受，避免用户的重要邮件被误判为垃圾邮件。

1. 作业：
2. 利用sklearn中BernoulliNB 分类该邮件数据集

from sklearn.naive\_bayes import BernoulliNB

import numpy as np

def myskBernoulliNB():

fullTest = [];docList = [];classList= []

# it only 25 doc in every class

for i in range(1,26):

wordList = textParse(open('D://PUB128/email/spam/%d.txt' % i,encoding="ISO-8859-1").read())

docList.append(wordList)

fullTest.extend(wordList)

classList.append(1)

wordList = textParse(open('D://PUB128/email/ham/%d.txt' % i,encoding="ISO-8859-1").read())

docList.append(wordList)

fullTest.extend(wordList)

classList.append(0)

# create vocabulary

vocabList = createVocabList(docList)

trainSet = list(range(50));testSet=[]

# choose 10 sample to test ,it index of trainMat

for i in range(10):

randIndex = int(np.random.uniform(0,len(trainSet)))#num in 0-49

testSet.append(trainSet[randIndex])

del(trainSet[randIndex])

trainMat = []; trainClass = []

for docIndex in trainSet:

trainMat.append(bagOfWords2Vec(vocabList,docList[docIndex]))

trainClass.append(classList[docIndex])

clf=BernoulliNB()

#print(type(np.array(trainMat)))

clf.fit(np.array(trainMat),np.array(trainClass))

errCount = 0

for docIndex in testSet:

wordVec=bagOfWords2Vec(vocabList,docList[docIndex])

#print(wordVec)

if clf.predict(np.array([wordVec])) != classList[docIndex]:

errCount += 1

print (("classfication error"), docList[docIndex])

print (("the error rate is ") , float(errCount)/len(testSet))

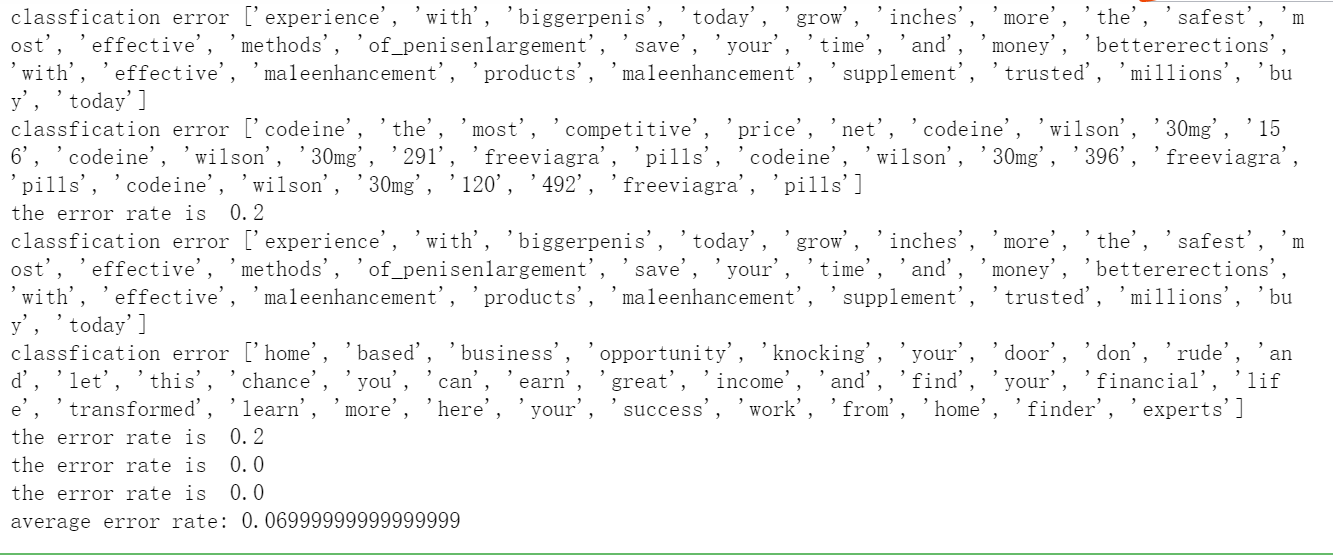
return float(errCount)/len(testSet)

ave=0

for i in range(10):

ave=ave+myskBernoulliNB()

print('average error rate:',ave/10)



平均错误率为0.07 表现情况良好 sklearn的伯努利分类器有良好的分类效果

1. 将词集向量用 TF-IDF 词向量替代，测试分析结果。

有的词如to等虚词在很多邮件里出现，其并不代表实际内容，其与是否为垃圾邮件没有直接关系，这些词的重要性就比money，card等词低一些。

TF-IDF词向量Term Frequency - Inverse Document Frequency的缩写，即“词频-逆文本频率”。它由两部分组成，TF和IDF。IDF解决在大量文本中都有的出现的重要性低的词的问题。

TF=某个词组在文章中出现次数/该文章的总词组数

IDF=log(语料库中文档总数/(包含该词组的文章总数+1))

可以运用sklearn的tf-idf进行转换处理

运行以下代码：

from sklearn.naive\_bayes import BernoulliNB

from sklearn.feature\_extraction.text import TfidfTransformer

from sklearn.feature\_extraction.text import CountVectorizer

import numpy as np

import scipy

def myskBernoulliNB2():

fullTest = [];docList = [];classList= []

# it only 25 doc in every class

for i in range(1,26):

wordList = open('D://PUB128/email/spam/%d.txt' % i,encoding="ISO-8859-1").read()

docList.append(wordList)

fullTest.extend(wordList)

classList.append(1)

wordList = open('D://PUB128/email/ham/%d.txt' % i,encoding="ISO-8859-1").read()

docList.append(wordList)

fullTest.extend(wordList)

classList.append(0)

# create vocabulary

vocabList = createVocabList(docList)

trainSet = list(range(50));testSet=[]

# choose 10 sample to test ,it index of trainMat

for i in range(10):

randIndex = int(np.random.uniform(0,len(trainSet)))#num in 0-49

testSet.append(trainSet[randIndex])

del(trainSet[randIndex])

transformer = TfidfTransformer()

vectorizer = CountVectorizer()

#print(transformer)

#print(docList)

tfidf = transformer.fit\_transform(vectorizer.fit\_transform(docList)).A

#print(type(tfidf.A))

#print(tfidf.A)

trainmail=[tfidf[i] for i in trainSet]

trainClass=[classList[i] for i in trainSet]

testmail=[tfidf[i] for i in testSet]

testClass=[classList[i] for i in testSet]

#print(np.array(trainmail[1]))

#print(trainmail)

#trainMat.voc=vectorizer.vocabulary\_

#print(trainMat)

clf=BernoulliNB()

#print(type(np.array(trainMat)))

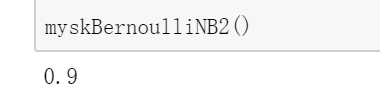
#print(type(tfidf))

#print(trainmail)

clf.fit(scipy.mat(trainmail),trainClass)

print(clf.score(scipy.mat(testmail),testClass))

myskBernoulliNB2()



可见用tf-idf词向量来进行分类，分类效果好，正确率高。

1. 编程实现 PPT 中的例 1，例 2

用多项分布实现PPT中的例1代码如下，训练集与测试集相同：

from sklearn.naive\_bayes import MultinomialNB

data81=["youth high no fair no",

"youth high no excellent no",

"middle\_aged high no fair yes",

"senior medium no fair yes",

"senior low yes fair yes",

"senior low yes excellent no",

"middle\_aged low yes excellent yes",

"youth medium no fair no",

"youth low yes fair yes",

"senior medium yes fair yes",

"youth medium yes excellent yes",

"middle\_aged medium no excellent yes",

"middle\_aged high yes fair yes",

"senior medium no excellent no",

]

n=len(data81)

mat81=[]

cls81=[]

for i in range(n):

spl=data81[i].split()

tmp=[spl[j] for j in range(4)]

mat81.append(tmp)

if (spl[4]=='yes'):

cls81.append(1)

else:

cls81.append(0)

for j in range(4):

dict={}

cnt=0

for i in range(n):

if not (mat81[i][j] in dict):

dict[mat81[i][j]]=cnt

cnt+=1

for i in range(n):

mat81[i][j]=dict[mat81[i][j]]

print(mat81)

print(cls81)

#mat81=[[1,1],[0,1]]

#cls81=[0,1]

clf=MultinomialNB()

clf.fit(np.array(mat81),np.array(cls81))

err=0

tot=0

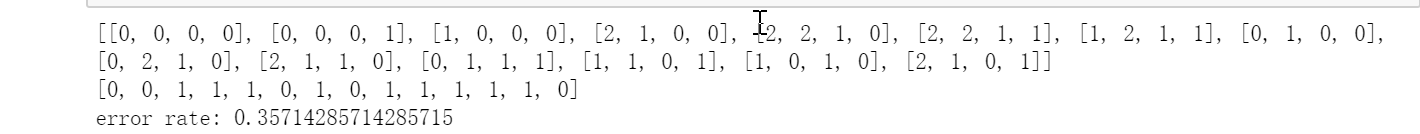
for i in range(n):

tot+=1

if clf.predict(np.array([mat81[i]]))!=cls81[i]:

err+=1

print('error rate:',err/tot)



错误率为0.35左右

用高斯分布实现PPT中例2的代码如下：

#example8-2

from sklearn.naive\_bayes import GaussianNB

data82=["男 6 180 12",

"男 5.92 190 11",

"男 5.58 170 12",

"男 5.92 165 10",

"女 5 100 6",

"女 5.5 150 8",

"女 5.42 130 7",

"女 5.75 150 9"]

n=len(data82)

mat82=[]

cls82=[]

for i in range(n):

spl=data82[i].split()

tmp=[spl[j] for j in range(1,4)]

mat82.append(tmp)

if (spl[0]=='男'):

cls82.append(1)

else:

cls82.append(0)

print(mat82)

print(cls82)

#mat81=[[1,1],[0,1]]

#cls81=[0,1]

clf=GaussianNB()

clf.fit(np.array(mat81),np.array(cls81))

err=0

tot=0

for i in range(n):

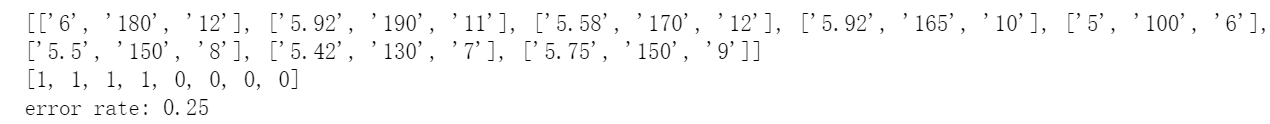
tot+=1

if clf.predict(np.array([mat81[i]]))!=cls81[i]:

err+=1

print('error rate:',err/tot)

训练集与测试集相同，错误率为0.25，分类效果较好



1. 利用朴素贝叶斯算法实现对 lab6 的两个数据集分类

使用高斯分布来实现lab6的约会数据集分类：

#lab6\_2

def file2matrix(filename):

fr = open(filename)

#得到文件行数

arrayOfLines = fr.readlines()

numberOfLines = len(arrayOfLines)

#创建返回的Numpy矩阵

returnMat = np.zeros((numberOfLines,3))

classLabelVector = []

#解析文件数据到列表

index = 0

for line in arrayOfLines:

line = line.strip() #注释1

listFromLine = line.split('\t') #注释2

returnMat[index,:] = listFromLine[0:3]

classLabelVector.append(int(listFromLine[-1]))

index +=1

return returnMat,classLabelVector

datingDataMat,datingLabels=file2matrix("D:\\datingTestSet2.txt")

clf=GaussianNB()

clf.fit(np.array(datingDataMat),np.array(datingLabels))

err=0

tot=0

for i in range(len(datingDataMat)):

tot+=1

if clf.predict(np.array([datingDataMat[i]]))!=datingLabels[i]:

err+=1

print('error rate:',err/tot)

错误率为0.063，分类效果不错，可见高斯分布适应于这个数据集的特征



使用伯努利分布来实现lab6的手写数字集分类：

#lab6\_3

def getTestDataSet():

hwLabels = []

# 获取目录traningDigits内容(即数据集文件名)，并储存在一个list中

trainingFileList = listdir('D:\\testDigits')

m = len(trainingFileList) #当前目录文件数

# 初始化m维向量的训练集，每个向量1024维

trainingMat = np.zeros((m,1024))

for i in range(m):

fileNameStr = trainingFileList[i]

# 从文件名中解析分类数字，作为分类标签

hwLabels.append(classnumCut(fileNameStr))

# 将图片矩阵转换为向量并储存在新的矩阵中

trainingMat[i,:] = img2vector('D:\\testDigits/%s' % fileNameStr)

return hwLabels,trainingMat

hwLabels,trainingMat = trainingDataSet()

testLabels,testMat=getTestDataSet()

clf=BernoulliNB()

clf.fit(np.array(trainingMat),np.array(hwLabels))

tot=0

err=0

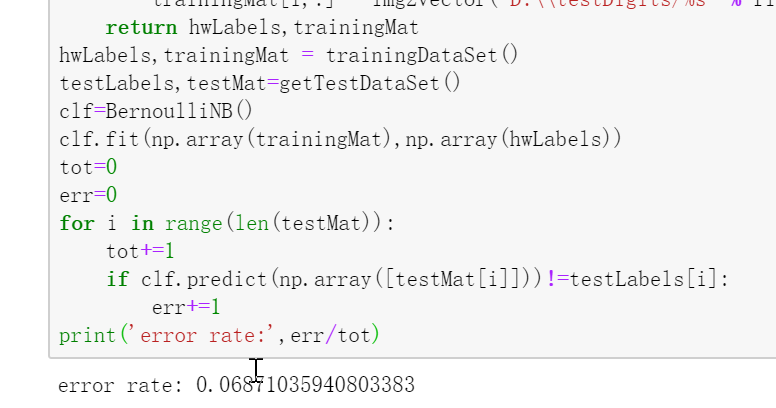
for i in range(len(testMat)):

tot+=1

if clf.predict(np.array([testMat[i]]))!=testLabels[i]:

err+=1

print('error rate:',err/tot)



错误率为6.8%左右，分类效果与当时的KNN算法相比没有那么好，但是贝叶斯分类器的执行速度较快，时间复杂度与当时的knn算法相比相对较低。

1. 基于 Tensorflow 实现朴素贝叶斯分类器（扩展）

使用Tensorflow实现朴素贝叶斯分类器解决手写识别数字的代码如下(进行了拉普拉斯数据平滑化处理):

tftrain=tf.placeholder("float",[None,1024])#1\*1024的01向量

tftest=tf.placeholder("float",[1024])

#distance=tf.reduce\_sum(tf.abs(tf.add(tftrain,tf.negative(tftest))),axis=1)

#distance=tf.sqrt(tf.reduce\_sum(tf.square(tf.add(tftrain,tf.negative(tftest))),axis=1))#欧几里得距离

pmat=tf.placeholder("float",[10])

pc=tf.placeholder("float",[10])

n=len(trainingMat)

dic=[1 for i in range(10)]

for i in range(n):

dic[hwLabels[i]]+=1

for i in range(10):

dic[i]=np.log(dic[i]/(n+10))

pp=np.ones((10,1024),dtype=np.float)

for i in range(10):

tot=0

for j in range(n):

if hwLabels[j]==i:

tot+=1

for k in range(1024):

if trainingMat[j][k]:

pp[i][k]+=1

for k in range(1024):

pp[i][k]/=(tot+2)

print(pp)

pro=tf.add(pmat,pc)

pred=tf.argmax(pro,0)

accuracy=0

init = tf.global\_variables\_initializer()

with tf.Session() as sess:

sess.run(init)

m=testMat.shape[0]

for i in range(m):

tmp=[0 for j in range(10)]

for j in range(10):

for k in range(1024):

if testMat[i][k]:

tmp[j]+=np.log(pp[j][k])

else:

tmp[j]+=np.log(1-pp[j][k])

index=sess.run([pred,pro],feed\_dict={pc:dic,pmat:tmp})

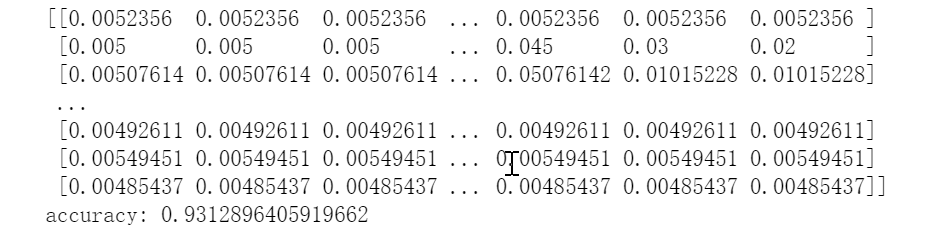
pred\_label=index[0]#index[0]为pred预测值

true\_label=testLabels[i]#真实标签

if pred\_label==true\_label:

accuracy+=1

print("accuracy:",accuracy/m)



正确率达到93%，尽管未达到当时KNN算法的准确率，但也是比较高的水平，与sklearn的伯努利贝叶斯分类器的准确率差不多。

1. 实验总结：
2. csr\_matrix转换为ndarray: 假设S为csr\_matrix类型，则S.A可以轻易转换为ndarray
3. ndarray转换为matrix:scipy.mat()方法可以将一个ndarray转换为一个matrix
4. transformer = TfidfTransformer();vectorizer = CountVectorizer();tfidf = transformer.fit\_transform(vectorizer.fit\_transform(docList)).A可以得到tf-idf处理后的词向量矩阵(ndarray类型) 然后将其转换为mat类型后可以调用sklearn的贝叶斯分类器的fit方法。
5. 可以先生成一个0~len-1的索引列表，然后使用随机数取走这个列表的一部分元素，然后得到一个新的包含原来索引子集的列表。然后可以用trainmail=[tfidf[i] for i in trainSet]的列表推导式来用这个索引列表将原来数据集分为测试集和训练集。
6. 朴素贝叶斯分类器可能分类效果会比KNN算法差，但是其时间复杂度比KNN要低。KNN算法单次预测时间复杂度高达O(nmk)或者O(nmlogk)(需要数据结构维护最近的k个样本),n为样本集样本的数目，m为数据的维数。而贝叶斯分类器的样本训练时间为O(nm)，单次预测时间复杂度仅为O(m)，时间复杂度较低，测试集大时，贝叶斯分类器明显快于KNN算法。这样在实际应用时可以将两种算法结合起来，在贝叶斯分类器分到一个类别概率明显高于其他类别时，预测成功概率较大，可以直接运用这个结果。而当贝叶斯分类器无法得到一个概率特别大的可能类别时，再用KNN算法辅助判断预测出最合适的类别。
7. 多项分布是二项分布的特例，理论上用多项分布拟合可以胜任二项分布的数据。
8. 当数据的特征取值为字符串离散值时，不可以直接用sklearn的朴素贝叶斯分类器来分类。需要实现一个字典，将原来的离散的字符串特征转换为整数，然后才能用sklearn的朴素贝叶斯分类器。例如：

for j in range(4):

dict={}

cnt=0

for i in range(n):

if not (mat81[i][j] in dict):

dict[mat81[i][j]]=cnt

cnt+=1

for i in range(n):

mat81[i][j]=dict[mat81[i][j]]

1. 离散型特征数据一般可以直接用频率估计概率，计算出每个特征对应每个分类的概率，或者用离散型均匀分布、二项分布、多项分布、几何分布、负二项分布、泊松分布等离散型随机变量来拟合估计概率。而连续性特征数据可以先求出数据集的统计值，然后用连续型均匀分布、指数分布、正态分布、伽马分布、贝塔分布、t分布、F分布来作为估计概率的概率密度函数。
2. 朴素贝叶斯算法假设了特征间相互独立，贝叶斯信念网络可以解决属性子集间有依赖关系的分类问题。
3. 英文垃圾邮件分词较为简单，可以直接通过分隔符来分词，而中文垃圾邮件分词复杂些，可以通过Python的jieba库来帮助分词
4. sklearn的贝叶斯分类器的参数类型需要是二维数组类型。即使在预测时，也需要传入一个二维数组。测试错误率的时候，传入一个向量trainX[i]会报错，应该要将向量转化为二维数组后传给函数。也就是将np.array([trainX[i]])传给函数
5. 朴素贝叶斯分类器只能应用于分类问题，其算法有一定的局限性
6. 可以用拉普拉斯平滑解决零概率问题，即让所有特征的出现次数加1，保证所有未出现过的特征概率不为0，这样可以保障算法稳定性，避免因为一个概率为0导致累乘时所有概率为0
7. 实现贝叶斯分类器时，分母P(X)可以不用在计算式中处理。因为每个类别计算的概率都有一个/P(X)的公共部分。每一类的相对概率大小不会因为同除P(X)发生改变，所以不同除P(X)依然可以找出最优的分类。而同除P(X)不仅是无用的操作，而且还会引入精度误差。