**实验一**  **朴素贝叶斯算法（naive bayes）**

1. 实验目的

熟悉素朴贝叶斯算法原理以及实现朴素贝叶斯算法。

二、 实验内容

用python代码实现朴素贝叶斯算法，用数据验证算法的正确性。

三、 实验环境

Windows环境，

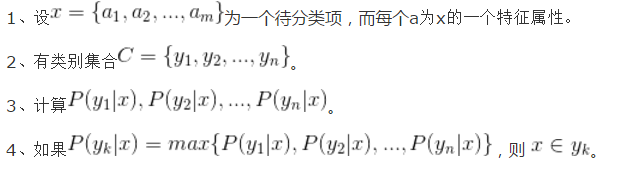
安装python，

安装pythyon IDE

四、 实验相关理论

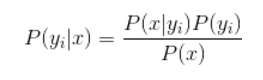
贝叶斯分类是一类分类算法的总称，这类算法均以贝叶斯定理为基础，故统称为贝叶斯分类。而朴素贝叶斯分类是贝叶斯分类中最简单，也是常见的一种分类方法。

朴素贝叶斯分类的正式定义如下：

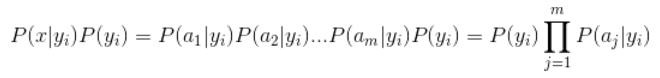


那么现在的关键就是如何计算第3步中的各个条件概率。我们可以这么做：

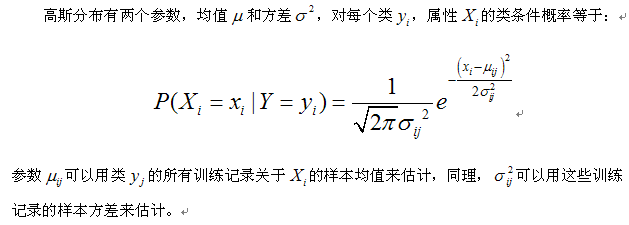
1. 找到一个已知分类的待分类项集合，这个集合叫做训练样本集。
2. 统计得到在各类别下各个特征属性的条件概率估计。
3. 如果各个特征属性是条件独立的，则根据贝叶斯定理有如下推导：



因为分母对于所有类别为常数，因为我们只要将分子最大化即可。又因为各特征属性是条件独立的，所以有：



当属性是‌连续型时，有两种方法来估计属性的类条件概率。第一种方法是把每一个连续的属性离散化，然后用相应的离散区间替换连续属性值，但这种方法不好控制离散区间划分的粒度，如果粒度太细，就会因为每一个区间中训练记录太少而不能对P(X/Y)做出可靠的估计，如果粒度太粗，那么有些区间就会含有来自不同类的记录，因此失去了正确的决策边界。第二种方法是，可以假设连续变量服从某种概率分布，然后使用训练数据估计分布的参数，高斯分布通常被用来表示连续属性的类条件概率分布。



五、 实验步骤

（1）收集数据：可以使用任何方法。

（2）准备数据：需要数值型或者布尔型数据。

（3）分析数据：有大量特征时，绘制特征作用不大，此时使用直方图效果更好。

（4）训练算法：计算不同的独立特征的条件概率。

（5）测试算法：计算错误率。

（6）使用算法：应用朴素贝叶斯处理连续数据。

六、 实验报告要求

按时完成实验要求，提交完整代码及实验报告，进行实验总结。

import numpy as np

import matplotlib.pyplot as plt

def loadDataSet(path,training\_sample):

dataMat = [];labelMat = []#定义列表

filename=path+training\_sample

fr = open(filename)

for line in fr.readlines():

line = line.strip('\n')

lineArr = line.strip().split(' ') #文件中数据的分隔符

dataMat.append([float(lineArr[0]), float(lineArr[1]),float(lineArr[2])]) #前三列数据

labelMat.append(int(lineArr[2])) # 标准答案

return dataMat,labelMat

def getSubCol(dataSet,col1,col2):

rownum = len(dataSet)

list = []

for featVec in dataSet: # 统计每一类的数量

list.append([featVec[col1],featVec[col2]])

return list

def getSubRow(dataSet,value):

rownum = len(dataSet)

list = []

for featVec in dataSet:

if featVec[-1] == value:

list.append(featVec)

return list

def sample\_average(data\_sample):

num = len(data\_sample)

sum = 0

for i in range(num):

sum += data\_sample[i][0]

return sum / num

def sample\_variance(data\_sample, mean\_value):

num = len(data\_sample)

sum = 0

for i in range(num):

sum += np.square(data\_sample[i][0]-mean\_value)

return sum/(num-1)

def Gaussian\_distribution(data\_sample,mean\_value,variance):

molecule = 0 # 分子

denominator = 0 # 分母

equation = 0

molecule = np.exp(-(np.square(data\_sample - mean\_value)) / (2 \* variance)) #分子部分

denominator = np.sqrt(2\*np.pi\*variance) #分母部分

equation = (molecule/denominator)

return equation

def percentage(dataSet,value):

num = len(dataSet)

count = 0

for featVec in dataSet:

if featVec[-1] == value:

count += 1

return (count/num)

def plotBestFit(dataArr,labelMat1,labelMat2):

n = len(dataArr) #取行数

xcord1 = []; ycord1 = []

xcord2 = []; ycord2 = []

xcord3 = []; ycord3 = []

xcord4 = []; ycord4 = []

for i in range(n): #将训练前的数据分类存储

if int(labelMat1[i])== 1: #分类为1

xcord1.append(dataArr[i][0]); ycord1.append(dataArr[i][1])

else:

xcord2.append(dataArr[i][0]); ycord2.append(dataArr[i][1])

for i in range(n): #将训练后的数据分类存储

if int(labelMat2[i]) == 1: # 分类为1

xcord3.append(dataArr[i][0]);ycord3.append(dataArr[i][1])

else:

xcord4.append(dataArr[i][0]);ycord4.append(dataArr[i][1])

fig = plt.figure("Naive Bayes1") #新建一个画图窗口

ax = fig.add\_subplot(111) #添加一个子窗口

ax.set\_title('Original')

ax.scatter(xcord1, ycord1, s=30, c='red', marker='s') #画点并标记颜色

ax.scatter(xcord2, ycord2, s=30, c='green') #画点并标记颜色

plt.xlabel('X1'); plt.ylabel('X2')

plt.figure("Naive Bayes2")

plt.title('Forecast')

plt.scatter(xcord3, ycord3, s=30, c='red', marker='s')

plt.scatter(xcord4, ycord4, s=30, c='green')

plt.xlabel('X1');plt.ylabel('X2')

plt.show()

def getResult(trainingSet,testingSet):

p0 = percentage(trainingSet,0) #初始0的频率

p1 = percentage(trainingSet,1) #初始1的频率

h = []

mean\_value0 = [1,1]

variance0 = [1,1]

mean\_value1 = [1,1]

variance1 = [1,1]

for i in range(2): #求均值和方差

featList = getSubCol(trainingSet, i, 2) # 取部分特征

featList0 = getSubRow(featList, 0) # 取结果值为0的行

featList1 = getSubRow(featList, 1) # 取结果值为1的行

mean\_value0[i] = sample\_average(featList0) # 值为0的均值

variance0[i] = sample\_variance(featList0, mean\_value0[i]) # 值为0的方差

mean\_value1[i] = sample\_average(featList1) # 值为1的均值

variance1[i] = sample\_variance(featList1, mean\_value1[i]) # 值为1的方差

for featVec in testingSet: #计算数据样本的高斯值

result0 = 1 #初始化

result1 = 1 #初始化

for j in range(2):

Gaussian0 = Gaussian\_distribution(featVec[j],mean\_value0[j],variance0[j]) #计算结果为0的高斯值

Gaussian1 = Gaussian\_distribution(featVec[j], mean\_value1[j], variance1[j]) #计算结果为1的高斯值

result0 \*= Gaussian0 #迭乘运算

result1 \*= Gaussian1 #迭乘运算

result0 \*= result0\*p0 #为0的可能值

result1 \*= result1\*p1 #为1的可能值

if(result0 > result1): #分类

h.append(0)

else:

h.append(1)

return h

from NaiveBayes import \*

def main():

path = "D:\\AI\\data\\" #文件目录

training\_sample = 'trainingSet.txt' #训练数据文件

testing\_sample = 'testingSet.txt' #测试数据文件

trainingSet,label = loadDataSet(path,training\_sample) #获取训练数据

testingSet,label = loadDataSet(path,testing\_sample) #获取测试数据

h = getResult(trainingSet,testingSet) #计算结果向量

plotBestFit(testingSet,label,h) #图形化展示

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

main()