**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，

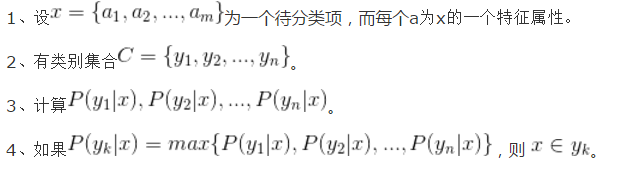
安装python，

安装pythyon IDE

四、 实验相关理论

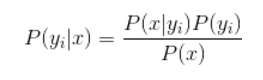
贝叶斯分类是一类分类算法的总称，这类算法均以贝叶斯定理为基础，故统称为贝叶斯分类。而朴素贝叶斯分类是贝叶斯分类中最简单，也是常见的一种分类方法。

朴素贝叶斯分类的正式定义如下：

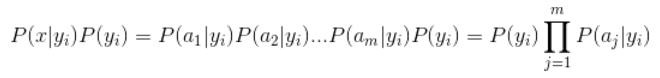


那么现在的关键就是如何计算第3步中的各个条件概率。我们可以这么做：

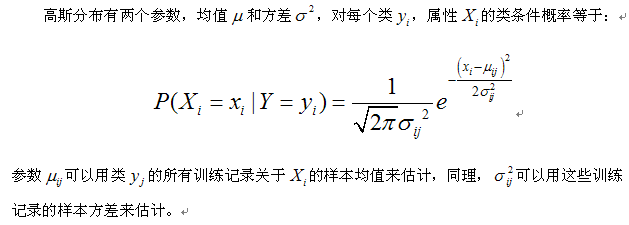
1. 找到一个已知分类的待分类项集合，这个集合叫做训练样本集。
2. 统计得到在各类别下各个特征属性的条件概率估计。
3. 如果各个特征属性是条件独立的，则根据贝叶斯定理有如下推导：



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。

代码：

**def** loadDataSet():  
 postingList=[[**'my'**,**'dog'**,**'has'**,**'flea'**,**'problem'**,**'help'**,**'please'**],  
 [**'maybe'**,**'not'**,**'take'**,**'him'**,**'to'**,**'dog'**,**'park'**,**'stupid'**],  
 [**'my'**,**'dalmation'**,**'is'**,**'so'**,**'cute'**,**'I'**,**'love'**,**'him'**],  
 [**'stop'**,**'posting'**,**'ate'**,**'my'**,**'steak'**,**'how'**,**'to'**,**'stop'**,**'him'**],  
 [**'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  
 *#定义一个简单的文本数据集，由6个简单的文本以及对应的标签构成。1表示侮辱性文档，0表示正常文档。***def** createVocabList(dataSet):  
 vocabSet=set([])  
 **for** document **in** dataSet:  
 vocabSet=vocabSet|set(document)  
 **return** list(vocabSet)  
 **def** setOfWords2Vec(vocabList,inputSet):  
 returnVec=[0]\*len(vocabList) *#每个文档的大小与词典保持一致，此时returnVec是空表* **for** word **in** inputSet:  
 **if** word **in** vocabList:  
 returnVec[vocabList.index(word)]=1 *#当前文档中有某个词条，则根据词典获取其位置并赋值1* **else**:print **"the word :%s is not in my vocabulary"** %word  
 **return** returnVec  
 **def** bagOfWords2Vec(vocabList,inputSet):  
 returnVec=[0]\*len(vocabList)  
 **for** word **in** inputSet:  
 **if** word **in** vocabList:  
 returnVec[vocabList.index(word)]+=1 *# 与词集模型的唯一区别就表现在这里* **else**:print **"the word :%s is not in my vocabulary"** %word  
 **return** returnVec  
 *#### 文档向量化，这里是词袋模型，不知关心某个词条出现与否，还考虑该词条在本文档中的出现频率* **def** trainNB(trainMatrix,trainCategory):  
 numTrainDocs=len(trainMatrix)  
 numWords=len(trainMatrix[0])  
 pAbusive=sum(trainCategory)/float(numTrainDocs) *#统计侮辱性文档的总个数，然后除以总文档个数  
 #p0Num=zeros(numWords);p1Num=zeros(numWords) # 把属于同一类的文本向量加起来  
 #p0Denom=0.0;p1Denom=0.0* p0Num=ones(numWords);p1Num=ones(numWords)  
 p0Denom=2.0;p1Denom=2.0  
 **for** i **in** range(numTrainDocs):  
 **if** trainCategory[i]==1:  
 p1Num+=trainMatrix[i]*#把属于同一类的文本向量相加，实质是统计某个词条在该类文本中出现频率* p1Denom+=sum(trainMatrix[i]) *#把侮辱性文档向量的所有元素加起来* **else**:  
 p0Num+=trainMatrix[i]  
 p0Denom+=sum(trainMatrix[i])  
 *#p1Vec=p1Num/float(p1Denom)  
 #p0Vec=p0Num/float(p0Denom)* p1Vec=log(p1Num/p1Denom) *#统计词典中所有词条在侮辱性文档中出现的概率* p0Vec=log(p0Num/p0Denom) *#统计词典中所有词条在正常文档中出现的概率* **return** pAbusive,p1Vec,p0Vec  
 *#### 训练生成朴素贝叶斯模型，实质上相当于是计算P（x，y|Ci）P（Ci）的权重。  
 ### 注意：被注释掉的代码代表不太好的初始化方式，在那种情况下某些词条的概率值可能会非常非常小，甚至约  
 ###等于0，那么在不同词条的概率在相乘时结果就近似于0* **def** classifyNB(vec2classify,p0Vec,p1Vec,pClass1): 　*# 参数1是测试文档向量，参数2和参数3是词条在各个  
 #类别中出现的概率，参数4是P（C1）* p1=sum(vec2classify\*p1Vec)+log(pClass1)　　　*# 这里没有直接计算P（x，y|C1）P（C1），而是取其对数  
 #这样做也是防止概率之积太小，以至于为0* p0=sum(vec2classify\*p0Vec)+log(1.0-pClass1)　*#取对数后虽然P（C1|x，y）和P(C0|x，y)的值变了，但是  
 #不影响它们的大小关系。* **if** p1>p0:  
 **return** 1  
 **else**:  
 **return** 0