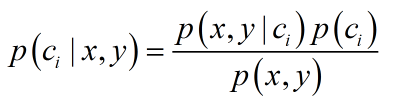
****朴素贝叶斯公式：****  
*P*(*yk*∣*x*)=*P*(*yk*)∗∏*P*(*xi*∣*yk*)P(yk∣x)=P(yk)∗∏P(xi∣yk) P(y\_k|x) =P(y\_k) \* ∏P(x\_i|y\_k) *P*(*yk*​∣*x*)=*P*(*yk*​)∗∏*P*(*xi*​∣*yk*​)

****理论基础——条件概率，词集模型、词袋模型****

* ****条件概率****：朴素贝叶斯最核心的部分是贝叶斯法则，而贝叶斯法则的基石是条件概率。贝叶斯法则如下：



这里的C表示类别，输入待判断数据，式子给出要求解的某一类的概率。我们的最终目的是比较各类别的概率值大小

Python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