第 6 章　生成模型与贝叶斯分类器

能够生成出一个事物，才能说真正理解了它

到目前为止，无论是有监督还是无监督，本文都关注的是给定数据$\boldsymbol{x}$，预测目标变量y（比如有监督的类别、无监督的簇），用概率的术语来讲，模型学习的是条件概率分布$p(y|\boldsymbol{x})$。由于这类模型的目标是把$\boldsymbol{x}$的不同y区分开来（即概率高的y作为预测值），因此被称为区分模型或判别模型。相对而言，如果一个模型学习的不是$p(y|\boldsymbol{x})$，而是联合概率分布$p(\boldsymbol{x},y)$，就被称为生成模型。这类模型的优势在于，一旦学习到$p(\boldsymbol{x},y)$，既可以得到$p(y|\boldsymbol{x})$，从而完成对$\boldsymbol{x}$的区分与判别；也可以根据$p(\boldsymbol{x}|y)$，指定y而采样生成出样本数据$\boldsymbol{x}$，这也是为什么称为生成模型的原因。总的来说，判别模型和生成模型各有优缺点和适应场合。

源于贝叶斯公式的贝叶斯分类器实际上形成了一个模型“谱系”，包括具有理论意义的贝叶斯最优分类器、假定特征条件独立的朴素贝叶斯分类器、假定部分特征条件独立的半朴素贝叶斯分类器、以及不假定特征条件独立的贝叶斯网（也称为有向概率图模型）。本章将介绍贝叶斯最优分类器和朴素贝叶斯分类器。

6.1　贝叶斯最优分类器

首先回忆一下概率论的重要公式——贝叶斯公式。如式（4-1）所示：

$$p(y|\boldsymbol{x})=\frac{p(\boldsymbol{x}|y)p(y)}{p(\boldsymbol{x})},p(\boldsymbol{x})=\sum_y p(\boldsymbol{x}|y)p(y) \tag{6.1}$$

其中，$p(y|\boldsymbol{x})$称为后验概率，$p(\boldsymbol{x}|y)$称为观察概率或似然概率，$p(y)$称为先验概率，$p(\boldsymbol{x})$称为证据因子。由概率的积规则可知，联合概率$p(\boldsymbol{x},y)=p(\boldsymbol{x}|y)p(y)$。如果$p(\boldsymbol{x}|y)$和$p(y)$都可以从数据样本中估计得到，那么就可得到$p(\boldsymbol{x},y)$，这就是生成模型的基本思路。

以多分类任务为例。假设有C个类别，标签$y=\{a_1,\dots,a_C\}$，L_{ij}是将一个真实标签为a_i的样本误分类为a_j所产生的损失。则基于后验概率$p(y|\boldsymbol{x})$和L_{ij}可以得到将样本$\boldsymbol{x}$分类为a_i的期望损失：

$$R(a_i|\boldsymbol{x})=\sum_{j=1}^{C}L_{ij}p\left(a_j\middle|\boldsymbol{x}\right) \tag{6.2}$$

对训练集D上所有N个样本的期望损失求和，得到总体损失R：

$$R=\sum_{\boldsymbol{x}}R(a_i|\boldsymbol{x}) \tag{6.3}$$

显然，要使得R最小，只需要每个$R(a_i|\boldsymbol{x})$最小，这就得到了贝叶斯判定准则：在每个样本$\boldsymbol{x}$上选择能使期望损失$R(a_i|\boldsymbol{x})$最小的那个标签a_i，即

$$b^*(\boldsymbol{x})=\underset{a_i\in y}{\operatorname{argmin}}R(a_i|\boldsymbol{x}) \tag{6.4}$$

此时，称$b^*(\boldsymbol{x})$为贝叶斯最优分类器，对应的总体损失R^*称为贝叶斯风险或贝叶斯误差。$1-R^*$就是模型能够达到的最高精度。

如果损失L_{ij}采用 1.2.3 节定义的 0-1 损失（式（1.1）），则由式（6.2）可得到：

$$R(a_i|\boldsymbol{x})=1-p(a_i|\boldsymbol{x}) \tag{6.5}$$

此时，式（6.4）成为：

$$b^*(\boldsymbol{x})=\underset{a_i\in y}{\operatorname{argmax}}p(a_i|\boldsymbol{x}) \tag{6.6}$$

即在每个样本$\boldsymbol{x}$上选择能使后验概率$p(a_i|\boldsymbol{x})$最大的那个标签a_i。

那么，如何得到后验概率$p(a_i|\boldsymbol{x})$呢？这就回到了本章开头提到的两种做法。一种是直接建模$p(a_i|\boldsymbol{x})$，被称为区分模型或判别模型。另一种是对联合概率$p(\boldsymbol{x},a_i)$进行建模，然后由式（6.1）得到$p(a_i|\boldsymbol{x})$，被称为生成模型。此处，生成模型可写为：

$$p(a_i|\boldsymbol{x}) = \frac{p(\boldsymbol{x},a_i)}{p(\boldsymbol{x})} = \frac{p(\boldsymbol{x}|a_i)p(a_i)}{p(\boldsymbol{x})} \tag{6.7}$$

$p(a_i)$是类别a_i的先验概率——类先验概率，$p(\boldsymbol{x}|a_i)$是样本$\boldsymbol{x}$相对于类别a_i的条件概率——类条件概率。对于给定的样本$\boldsymbol{x}$，证据因子$p(\boldsymbol{x})$与类别a_i无关，因此基于训练集D估计后验概率$p(a_i|\boldsymbol{x})$的问题就转化为了如何估计类先验概率$p(a_i)$和类条件概率$p(\boldsymbol{x}|a_i)$。

估计类先验概率$p(a_i)$通常容易办到，只要训练集D中有足够多的独立同分布样本，就可以用每个类别在D中出现的频率来进行估计。困难在于估计类条件概率$p(\boldsymbol{x}|a_i)$，因为样本$\boldsymbol{x}$一般为F维特征向量（构成F维特征空间），其分布$p(\boldsymbol{x})$实际上是F个特征的联合分布$p(x_1,\dots,x_F)$。正如 4.1 节谈到的，为了把F维特征空间填满，需要的数据样本将随着维数F的增加而呈指数增长，这就意味着，在含N个有限样本的训练集D中，很多特征x_i的取值根本不会出现，从而没有办法通过频率来估计$p(\boldsymbol{x}|a_i)$。注意：“不出现”仅仅意味着在D中没有观察到或采样到，并不意味着“出现概率为 0”。

可以采用 2.3.3 节介绍的极大似然估计来对类条件概率$p(\boldsymbol{x}|a_i)$进行估计。类似 2.3.3 节的做法，先假定$p(\boldsymbol{x}|a_i;\boldsymbol{\theta})$是一个含参模型，并且具有某种确定的概率分布形式，再基于训练样本和对数似然函数，对参数$\boldsymbol{\theta}$进行估计。显然，极大似然估计严重依赖于假定的概率分布形式和真实数据分布（未知）的符合程度。为此，关于具体应用任务的经验知识往往能够有所帮助。比如，以抛硬币为例，经验告诉我们，硬币要么是正面要么是反面，所以假定伯努利分布（$p(\boldsymbol{x}|\mu)$，其中μ是唯一的参数）是与真实数据分布相符的。再比如，统计一个班的成绩分布，经验告诉我们，如果学生人数足够多，采用高斯分布比较合适。更复杂的情况，一般还需对经验知识进行简化，从而实现建模。接下来要介绍的朴素贝叶斯模型就是一个典型例子。

6.2 朴素贝叶斯分类器

为了实现对类条件概率$p(\boldsymbol{x}|a_i)$进行估计，朴素贝叶斯分类器假设所有特征相互独立——特征条件独立性假设。这就意味着，每个特征独立地对分类结果产生影响。特征条件独立性假设是一个很强的假设，在实践中一般并不成立。比如一个西瓜，其“敲声”与其“成熟度”、“密度”、“含糖率”这些往往关系紧密。再比如，短语“第一季度”经常出现在商业分析文章中，其出现的概率大于“第一”的概率和“季度”的概率相乘的结果。虽然如此，但有趣的是，朴素贝叶斯分类器在很多情形下都能获得相当好的性能（后面给出了两个应用实例）。一种解释是，虽然违背独立性会导致后验概率更接近 1 或 0，但其各个类别概率值的大小顺序一般受到的影响并不大。

基于特征条件独立性假设，式（6.7）可重写为：

$$p(a_i|\boldsymbol{x}) = \frac{p(\boldsymbol{x},a_i)}{p(\boldsymbol{x})} = \frac{p(a_i)}{p(\boldsymbol{x})}\prod_{j=1}^{F} p\left(x_j|a_i\right) \tag{6.8}$$

其中，F为特征数目（即样本$\boldsymbol{x}$的维数），x_j为$\boldsymbol{x}$在第j个特征上的取值。

由于式（6.8）中，证据因子$p(\boldsymbol{x})$与类别a_i无关，因此由贝叶斯判定准则（式（6.6））可得：

$$b_n(\boldsymbol{x}) = \underset{a_i\in y}{\operatorname{argmax}}\, p(a_i)\prod_{j=1}^{F} p\left(x_j|a_i\right) \tag{6.9}$$

这就是朴素贝叶斯分类器的表达式。

显然，朴素贝叶斯分类器的训练过程就是基于训练集D，对类先验概率$p(a_i)$和类条件概率$p(x_j|a_i)$进行估计。设D_c表示D中第c类样本组成的集合，则类先验概率可估计为：

$$p(a_c)=\frac{|D_c|}{|D|} \tag{6.10}$$

对于类条件概率，需要分别考虑特征x_j为离散或连续两种情况。如果x_j为离散值，设$D_{c,j}$表示D_c中第j个特征取值为x_j的样本组成的集合，则类条件概率可估计为：

$$p(x_j|a_c)=\frac{|D_{c,j}|}{|D_c|} \tag{6.11}$$

式（6.11）还存在一个问题：如果D_c中x_j未出现该怎么办？一方面，正如 6.1 节所说，“未出现”仅仅意味着没有观察到或采样到，并不意味着“出现概率为 0”。另一方面，在这里，还可以看到，如果某个$p(x_j|a_c)$为 0，则将导致式（6.9）中的连乘式$\prod_{j=1}^{F}p(x_j|a_i)$为 0，其他的不为 0 的$p(x_j|a_c)$也被“抹掉了”。这启发我们，需要对式（6.10）和（6.11）进行合适的修正，以做到不会出现为 0 的$p(x_j|a_c)$。

实际上，式（6.10）和（6.11）就是极大似然估计的结果——概率等于频率，没出现的概率就为 0。因此，可以采用贝叶斯的方式，通过引入先验来解决这个问题，这就是所谓的贝叶斯估计：

$$p(a_c)=\frac{|D_c|+\beta}{|D|+C\beta} \tag{6.12}$$

$$p(x_j|a_c)=\frac{|D_{c,j}|+\beta}{|D_c|+F_j\beta} \tag{6.13}$$

其中，$\beta\geq 0$，C是类别数，F_j是第j个特征取值个数。显然，如果$\beta=0$，就回到式（6.10）和（6.11）。常取$\beta=1$，称之为“拉普拉斯平滑”。以$\beta=1$为例，如果$|D_{c,j}|=0$，则$p(x_j|a_c)=1/(|D_c|+F_j)$，可见拉普拉斯平滑实质上假定了类别和特征取值的均匀分布先验。另外，随着训练集D的增大（从而$|D|$、$|D_c|$和$|D_{c,j}|$增大），先验的影响将减小，估计值将更接近实际概率值。

如果x_j为连续值，可基于概率密度函数来考虑。比如设$p(x_j|a_c)$服从均值为$\mu_{c,j}$方差为$\sigma_{c,j}^2$的高斯分布，则有：

$$p(x_j|a_c)=\frac{1}{\sqrt{2\pi}\sigma_{c,j}}\exp\left(-\frac{(x_i-\mu_{c,j})^2}{2\sigma_{c,j}^2}\right) \tag{6.14}$$

其中，$\mu_{c,j}$和$\sigma_{c,j}^2$分别是第c类样本在第j个特征上取值的均值和方差。

6.3 半朴素贝叶斯分类器和贝叶斯网

为了完整性，这一节简单介绍一下半朴素贝叶斯分类器和贝叶斯网。

正如前面已经提到的，朴素贝叶斯分类器假定所有特征条件独立，而半朴素贝叶斯分类器仅假定部分特征条件独立，贝叶斯网则对于特征的条件独立性不做任何假设。这就构成了一个完整的贝叶斯分类器模型“谱”：朴素贝叶斯分类器和贝叶斯网分别位于“谱”的两端，而介于两者之间的就是一系列半朴素贝叶斯分类器。

如果将式（6.8）改写为：

$$p(a_i|\boldsymbol{x})=\frac{p(\boldsymbol{x},a_i)}{p(\boldsymbol{x})}=\frac{p(a_i)}{p(\boldsymbol{x})}\prod_{j=1}^{F}p(x_j|a_i,p_j) \tag{6.15}$$

其中，p_j表示x_j所依赖的特征，称为x_j的父特征。式（6.15）就是一种常见的半朴素贝叶斯分类器：每个特征在类别之外仅依赖于一个其他特征。选择p_j的方式不同，就得到不同的半朴素贝叶斯分类器。（见 E:\CDUwork\教材建设立项\AI 专业-ML 课程教材\chap9-实验\第 9 章

9.16 朴素贝叶斯与文本分类 - 习题解答.docx）

贝叶斯网是一个有向概率图模型（对应一个有向无环图），可以用联合概率的方式写为：

$$p(\boldsymbol{x}) = \prod_{j=1}^{F} p(x_j|p_j) \quad (6.16)$$

其中，p_j表示x_j所依赖特征的集合，称为x_j的父特征集合。

举个贝叶斯网的例子，如图 8.1 所示，这个贝叶斯网对应的联合概率分布为：$p(x_1,\dots,x_7) = p(x_1)p(x_2)p(x_3)p(x_4|x_1,x_2,x_3)p(x_5|x_1,x_3)p(x_6|x_4)p(x_7|x_4,x_5)$。

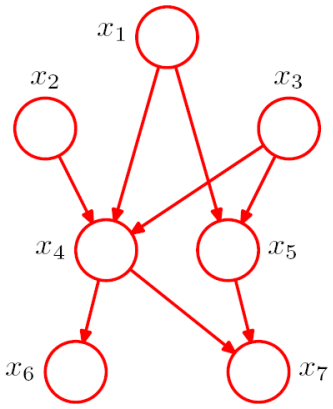

图 8.1 一个贝叶斯网

6.4 朴素贝叶斯分类器核心代码实现

本节用 Python 实现朴素贝叶斯分类器，采用“词集”和“词袋”两种方式进行实现，并将其应用到情绪分类和垃圾邮件过滤两个任务上。

6.4.1 词集与情绪分类

1、首先装入数据：

```
def loadDataSet():
    postingList=[['my', 'dog', 'has', 'flea', 'problems', 'help', 'please'],
                 ['maybe', 'not', 'take', 'him', 'to', 'dog', 'park', 'stupid'],
                 ['my', 'dalmation', 'is', 'so', 'cute', 'I', 'love', 'him'],
                 ['stop', 'posting', 'stupid', 'worthless', 'garbage'],
                 ['mr', 'licks', 'ate', 'my', 'steak', 'how', 'to', 'stop', 'him'],
                 ['quit', 'buying', 'worthless', 'dog', 'food', 'stupid']]
    classVec = [0,1,0,1,0,1]       #1 is abusive, 0 not
    return postingList, classVec
```

嵌套列表 postingList 包含 6 句话，每句话已经分成了一个个单词。标签向量 classVec 用 0 表示正面情绪，用 1 表示负面情绪。

2、接下来创建“词集”：

```
def createVocabList(dataSet):
    vocabSet = set([])  #创建空词集
    for document in dataSet:
        vocabSet = vocabSet | set(document) #集合的并
    return list(vocabSet)
```

集合 vocabSet（称为“词集”）通过“并”运算将 dataSet 里不同的单词记录下来，以列表类型返回。

3、基于词集创建文本向量：

```
def setOfWords2Vec(vocabList, inputSet):
    returnVec = [0]*len(vocabList)
    for word in inputSet:
        if word in vocabList:
            returnVec[vocabList.index(word)] = 1
        else: print("the word: %s is not in my Vocabulary!" % word)
    return returnVec
```

文本向量 returnVec（初值为 0）的长度为词集 vocabList 的长度，如果输入句子 inputSet 里出现词集里的单词，则将文本向量对应元素置为 1；如果 inputSet 里出现词集里没有的单词，则打印提示信息。这样，returnVec 就反映了词集里的单词在 inputSet 里出现的情况，0 表示未出现，1 表示出现。

4、训练朴素贝叶斯分类器：

```
def trainNB0(trainArray, trainCategory):
    numTrainDocs = len(trainArray)
    numWords = len(trainArray[0])
    pAbusive = sum(trainCategory)/float(numTrainDocs)
    p0Num = np.ones(numWords); p1Num = np.ones(numWords)
    p0Denom = 2.0; p1Denom = 2.0
    for i in range(numTrainDocs):
        if trainCategory[i] == 1:
            p1Num += trainArray[i]
            p1Denom += sum(trainArray[i])
        else:
            p0Num += trainArray[i]
            p0Denom += sum(trainArray[i])
    p1Vect = np.log(p1Num/p1Denom)
    p0Vect = np.log(p0Num/p0Denom)
    return p0Vect, p1Vect, pAbusive
```

trainArray 和 trainCategory 都是 Numpy 数组，前者的每个元素就是一个文本向量，后者则是其对应的标签。pAbusive 是负面情绪样本数占总样本数的比例，而 1-pAbusive 当然就是负面情绪的比例，对应式（6.10）给出的类先验概率。向量 p0Num 和 p1Num（都定义为 Numpy 数组）则分别用于记录正面情绪和负面情绪样本中，各单词各自出现的总次数；p0Denom 和 p1Denom 则分别用于记录正面情绪和负面情绪样本中，所有单词出现的总次数。由此，向量 p0Vect 和 p1Vect 对应式（6.11）给出的类条件概率。

为什么向量 p0Num 和 p1Num 的各元素的初值都设为 1 呢？类似的，为什么 p0Denom 和 p1Denom 的初值都设为 2 呢？实际上，这是式（6.13）当$\beta = 1$时给出的拉普拉斯平滑。

还有一点，为什么计算 p0Vect 和 p1Vect 时要取对数呢？聪明的读者可能马上想到了，这是因为式（6.9）中是概率值的连乘，取对数后将其转换为加法运算，可以有效地防止下溢。

5、分类和测试函数：

```
def classifyNB(vec2Classify, p0Vec, p1Vec, pClass1):
    p1 = np.sum(vec2Classify * p1Vec) + np.log(pClass1) #‘*’是向量的逐元素相乘
    p0 = np.sum(vec2Classify * p0Vec) + np.log(1.0 - pClass1)
```

```
        print('p1:',p1,'p0:',p0)
        if p1 > p0:
            return 1
        else:
            return 0
```

这个函数输入一个文本向量 vec2Classify，利用 trainNB0()的训练结果进行情绪分类。这个函数就是套式（6.9）。

```
    def testingNB():
        listOPosts, listClasses = loadDataSet()
        myVocabList = createVocabList(listOPosts)
        trainList=[]
        for postinDoc in listOPosts:
            trainList.append(setOfWords2Vec(myVocabList, postinDoc))
        p0V, p1V, pAb = trainNB0(np.array(trainList), np.array(listClasses))
        testEntry = ['love', 'my', 'dalmation']
        thisDoc = np.array(setOfWords2Vec(myVocabList, testEntry))
        print(testEntry, 'classified as: ', classifyNB(thisDoc, p0V, p1V, pAb))
        testEntry = ['stupid', 'garbage']
        thisDoc = np.array(setOfWords2Vec(myVocabList, testEntry))
        print(testEntry, 'classified as: ', classifyNB(thisDoc, p0V, p1V, pAb))
        testEntry = ['bad', 'garbage']
        thisDoc = np.array(setOfWords2Vec(myVocabList, testEntry))
        print(testEntry, 'classified as: ', classifyNB(thisDoc,p0V,p1V,pAb))
        testEntry = ['bad', 'person']
        thisDoc = np.array(setOfWords2Vec(myVocabList, testEntry))
        print(testEntry, 'classified as: ', classifyNB(thisDoc,p0V,p1V,pAb))
```

这个函数调用 loadDataSet()装入训练数据，调用 createVocabList()创建词集，调用 setOfWords2Vec()生成文本向量。完成这些准备工作后，调用 trainNB0()完成训练，然后依次测试了 4 个样本，运行结果如下：

```
p1: -9.826714493730215 p0: -7.694848072384611
['love', 'my', 'dalmation'] classified as:  0
p1: -4.702750514326955 p0: -7.20934025660291
['stupid', 'garbage'] classified as:  1
the word: bad is not in my Vocabulary!
p1: -3.044522437723423 p0: -3.9512437185814275
['bad', 'garbage'] classified as:  1
the word: bad is not in my Vocabulary!
the word: person is not in my Vocabulary!
p1: -0.6931471805599453 p0: -0.6931471805599453
['bad', 'person'] classified as:  0
```

可见，前两个样本分类正确，这两个样本都没有出现词集里没有的单词。第三个样本的单词'bad'未在词集里出现，所以打印了提示信息。由于词集里未出现的单词在计算中就不予考虑，所以第三个样本仅依据'garbage'这个单词进行判断，进而得出负面情绪的结果。第四个样本的所有单词都未在词集里出现，所以正面和负面等概率。

6.4.2 词袋与垃圾邮件过滤

1、邮件数据集介绍：

用于实验的邮件数据集共 50 个样本，正常邮件和垃圾邮件各 25 个。比如第一个正常邮件内容如下：

Hi Peter,

With Jose out of town, do you want to
meet once in a while to keep things
going and do some interesting stuff?

Let me know
Eugene

对应的，第一个垃圾邮件内容如下：

--- Codeine 15mg -- 30 for $203.70 -- VISA Only!!! --

-- Codeine (Methylmorphine) is a narcotic (opioid) pain reliever
-- We have 15mg & 30mg pills -- 30/15mg for $203.70 - 60/15mg for $385.80 - 90/15mg for $562.50 -- VISA Only!!! ---

2、基于词集的垃圾邮件过滤：

```
def spamTest():
    docList=[]; classList = []; fullText =[]
    for i in range(1, 26):
        wordList = textParse(open('email/spam/%d.txt' % i).read())
        docList.append(wordList)
        classList.append(1)
        wordList = textParse(open('email/ham/%d.txt' % i).read())
        docList.append(wordList)
        classList.append(0)
    vocabList = createVocabList(docList) #创建词集
    trainingSet = list(range(50)); testSet=[] #创建训练集和测试集
    for i in range(10):
        randIndex = int(np.random.uniform(0, len(trainingSet)))
        testSet.append(trainingSet[randIndex])
        del(trainingSet[randIndex])
    trainList=[]; trainClasses = []
    for docIndex in trainingSet:
        trainList.append(setOfWords2Vec(vocabList, docList[docIndex]))
        trainClasses.append(classList[docIndex])
    p0V, p1V, pSpam = trainNB0(np.array(trainList), np.array(trainClasses))
    errorCount = 0
    for docIndex in testSet:
        wordVector = setOfWords2Vec(vocabList, docList[docIndex])
        if classifyNB(np.array(wordVector), p0V, p1V, pSpam) != classList[docIndex]:
            errorCount += 1
```

```
            print("classification error", docList[docIndex])
    print('the error rate is: ', float(errorCount)/len(testSet))
```

这个函数首先调用 textParse()完成邮件数据的解析，得到正常邮件和垃圾邮件各 25 条。接着，生成词集 vocabList。为了进行交叉验证，从 50 个样本中随机选取 10 个用于测试。之后就是训练和测试过程，并打印出测试集上的错误率。

运行 spamTest()共 30 次，得到平均错误率为 3%。

3、基于词袋的垃圾邮件分类：

“词集”只考虑某个单词是否出现，“词袋”则进一步记录某个单词在文本中的出现次数。

```
def bagOfWords2VecMN(vocabList, inputSet):
    returnVec = [0]*len(vocabList)
    for word in inputSet:
        if word in vocabList:
            returnVec[vocabList.index(word)] += 1
    return returnVec
```

将前面的 spamTest()函数测试阶段的 setOfWords2Vec()替换为 bagOfWords2VecMN()，同样运行 30 次，得到平均错误率为 2.67%。确实性能有进一步提升。

习题 4

6.1 对于式（6.12）和（6.13），仅考虑了$|D_{c,i}| = 0$的情况，你认为$|D_c| = 0$的情况也需要考虑吗？为什么？请写出$\beta = 1$时$p(a_c)$和$p(x_i|a_c)$的表达式，分析其意义。

6.2 为什么向量 p0Num 和 p1Num 的各元素的初值都设为 1 呢？类似的，为什么 p0Denom 和 p1Denom 的初值都设为 2 呢？请结合式（6.13）当$\beta = 1$时给出的拉普拉斯平滑进行具体分析。

6.3 请分析 6.4.1 节的代码实现中是否考虑了式（6.12）给出的贝叶斯估计估计呢？为什么这样考虑，合理性在哪里？

$$p(a_c) = \frac{|D_c|+\beta}{|D|+C\beta} \tag{6.12}$$

$$p(x_j|a_c) = \frac{|D_{c,j}|+\beta}{|D_c|+F_j\beta} \tag{6.13}$$

6.4 请解释 6.4.1 节代码运行结果，为何 p1 和 p0 不是正常的概率值，而是负值呢？你认为可以怎样改进？

6.5 请解释 6.4.1 节代码运行结果，为何第四个样本的 p1 和 p0 都是-0.6931471805599453，这个值是如何算出来的？

6.6 spamTest()函数中，为了进行交叉验证，从 50 个样本中随机选取 10 个用于测试。请分析代码是否能保证随机选取的样本是类别平衡的，并实际验证。

6.7 请实现 textParse()函数，完成邮件数据的解析，返回按单词（字母小写）分割的列表，要求单词长度大于 2。

6.8 短语“第一季度”经常出现在商业分析文章中，其出现的概率大于“第一”的概率和“季度”的概率相乘的结果。请解释原因。

6.9 朴素贝叶斯分类器在很多情形下都能获得相当好的性能，6.2 节给出了一种解释，请分析这种解释的合理性。你认为还可能有哪些解释呢？跟具体的应用领域是否相关呢？

6.10 classifyNB()函数中，vec2Classify * p1Vec 的作用是什么？

6.11 基于下图中给出的西瓜数据集，应用朴素贝叶斯分类器判断“色泽为乌黑、根蒂为硬挺、敲声为清脆、纹理为模糊、脐部为平坦、密度为 0.732、含糖率为 0.315”的西瓜是否为好瓜。注意：下图中，密度和含糖率为连续值，假定其符合高斯分布（见式（6.14））。

编号	色泽	根蒂	敲声	纹理	脐部	触感	密度	含糖率	好瓜
1	青绿	蜷缩	浊响	清晰	凹陷	硬滑	0.697	0.460	是
2	乌黑	蜷缩	沉闷	清晰	凹陷	硬滑	0.774	0.376	是
3	乌黑	蜷缩	浊响	清晰	凹陷	硬滑	0.634	0.264	是
4	青绿	蜷缩	沉闷	清晰	凹陷	硬滑	0.608	0.318	是
5	浅白	蜷缩	浊响	清晰	凹陷	硬滑	0.556	0.215	是
6	青绿	稍蜷	浊响	清晰	稍凹	软粘	0.403	0.237	是
7	乌黑	稍蜷	浊响	稍糊	稍凹	软粘	0.481	0.149	是
8	乌黑	稍蜷	浊响	清晰	稍凹	硬滑	0.437	0.211	是
9	乌黑	稍蜷	沉闷	稍糊	稍凹	硬滑	0.666	0.091	否
10	青绿	硬挺	清脆	清晰	平坦	软粘	0.243	0.267	否
11	浅白	硬挺	清脆	模糊	平坦	硬滑	0.245	0.057	否
12	浅白	蜷缩	浊响	模糊	平坦	软粘	0.343	0.099	否
13	青绿	稍蜷	浊响	稍糊	凹陷	硬滑	0.639	0.161	否
14	浅白	稍蜷	沉闷	稍糊	凹陷	硬滑	0.657	0.198	否
15	乌黑	稍蜷	浊响	清晰	稍凹	软粘	0.360	0.370	否
16	浅白	蜷缩	浊响	模糊	平坦	硬滑	0.593	0.042	否
17	青绿	蜷缩	沉闷	稍糊	稍凹	硬滑	0.719	0.103	否

6.12 在 6.4.2 节中，将 spamTest() 函数测试阶段的 setOfWords2Vec() 替换为 bagOfWords2VecMN()，同样运行 30 次，得到平均错误率为 2.67%。确实性能有进一步提升。请进一步比较训练阶段和测试阶段采用词集或词袋的其他情况。并给出你对比较结果的分析和解释。