贝叶斯分类器及Python实现

0. 前言

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。本文由本人学习贝叶斯分类器过程中的笔记，再加上使用Python进行文本分类实战组成。

1. 贝叶斯决策论（Bayesian decision theory）

贝叶斯决策论是概率框架下实施决策的基本方法。对于分类任务来说，在所有相关概率都已知的情况下，贝叶斯决策论考虑如何基于这些概率和误判损失来选择最优的类别标记。

假设由N种可能的类别标记，$\mathcal{Y}=c_1,c_2,\cdots,c_N$,$\lambda_{ij}$是将一个真实标记为$c_j$的样本误差分类为$c_i$所产生的损失，基于后验概率$P(c_i|\boldsymbol{x})$可获得将样本$\boldsymbol{x}$分类为$c_i$所产生的期望损失(expected loss)，即在样本$\boldsymbol{x}$上的“条件风险”(conditional risk)

$R(c_i|\boldsymbol{x})=\sum_{j=1}^{N}\lambda_{ij}P(c_j|\boldsymbol{x})$

主要任务是寻找一个判定准则$h:\mathcal{X}\mapsto\mathcal{Y}$以最小化总体风险

$R(h)=\mathbb{E}_{\boldsymbol{x}}[R(h(\boldsymbol{x})|\boldsymbol{x})]$

显然，对每个样本$\boldsymbol{x}$,若h能最小化条件风险$R(h(\boldsymbol{x})|\boldsymbol{x})$,则总体风险$R(h)$也将被最小化

由此产生了贝叶斯准则(Bayes decision rule):

为最小化总体风险，只需在每个样本上选择那个能使条件风险$R(c|\boldsymbol{x})$最小的类别标记

$$h^*(\boldsymbol{x}) = \underset{c \in \mathcal{Y}}{argmin} R(c|(x))$$

此时，$h^*(\boldsymbol{x})$称为贝叶斯最优分类器（Bayse optimal classifier），与之对应的总体风险$R(h^*)$称为贝叶斯风险（Bayes risk）. $1 - R(h^*)$反映了分类器所能达到的最好性能。

若目标是最小化分类错误率，则误判损失λ_{ij}可写为

$$\lambda_{ij} = \begin{cases} 0, & \text{if} \quad i == j \\ 1, & otherwise \end{cases}$$

此时风险条件为：

$$R(c|\boldsymbol{x}) = 1 - P(c|\boldsymbol{x})$$

最小化分类错误率的贝叶斯最优分类器为

$$h^*(\boldsymbol{x}) = \underset{c \in \mathcal{Y}}{argmax} P(c|\boldsymbol{x})$$

基于贝叶斯定理，$P(c|\boldsymbol{x}) = \frac{P(c)P(\boldsymbol{x}|c)}{P(\boldsymbol{x})}$

关于这个公式的证明，很容易，依照条件概率的定义即可得到，感兴趣的读者可以百度一下。这个定理虽然很简单，但是却建立起先验概率和后验概率相互转化的桥梁。

$P(c)$为类先验概率

$P(\boldsymbol{x}|c)$称为类条件概率或者似然

> - 先验概率： 根据以往经验和分析得到的概率。
> - 后验概率：后验概率是基于新的信息，修正原来的先验概率后所获得的更接近实际情况的概率估计。

对于类先验概率就是样本空间中各类样本所占的比例，根据大数定理（当样本足够多时，频率趋于稳定等于其概率），这样当训练样本充足时，$P(c)$可以使用各类出现的频率来代替。因此只剩下类条件概率$P(\boldsymbol{x}|c)$,它表达的意思是在类别c中出现x的概率，它涉及到属性的联合概率问题，若只有一个离散属性还好，当属性多时采用频率估计起来就十分困难，因此这里一般采用极大似然法进行估计。

2. 极大似然估计

极大似然估计（Maximum Likelihood Estimation，简称MLE），是一种根据数据采样来估计概率分布的经典方法。常用的策略是先假定总体具有某种确定的概率分布，再基于训练样本对概率分布的参数进行估计。运用到类条件概率$p(\boldsymbol{x}|c)$中，假设$p(\boldsymbol{x}|c)$服从一个参数为θ的分布，问题就变为根据已知的训练样本来估计$\boldsymbol{\theta}$。极大似然法的核心思想就是：估计出的参数使得已知样本出现的概率最大，即使得训练数据的似然最大。

令D_c表示训练集D中第c类样本组成的集合，假设这些样本是独立同分布的，则参数$\boldsymbol{\theta}_c$对数据集D_c的似然是

$$P(D_c|\boldsymbol{\theta}_c) = \prod_{\boldsymbol{x}\in D_c} P(\boldsymbol{x}|\boldsymbol{\theta}_c)$$

对$\boldsymbol{\theta}_c$进行极大似然估计，就是去寻找能最大似然$P(D_c|\boldsymbol{\theta}_c)$的参数值$\hat{\boldsymbol{\theta}}_c$. 直观上看，极大似然估计是视图在$\boldsymbol{\theta}_c$所有可能取值中，找到一个能使数据出现的“可能性”最大的值

连乘操作使得求解变得复杂（容易造成下溢），一般我们使用对数似然(log-likelihood)

$$LL(\boldsymbol{\theta}_c) = logP(D_c|\boldsymbol{\theta}_c) = \sum_{\boldsymbol{x}\in D_c} logP(\boldsymbol{x}|\boldsymbol{\theta}_c)$$

此时参数$\boldsymbol{\theta}_c$的极大似然估计$\hat{\boldsymbol{\theta}}_c$为

$$\hat{\boldsymbol{\theta}}_c = \underset{\boldsymbol{\theta}_c}{argmax} LL(\boldsymbol{\theta}_c)$$

所以，贝叶斯分类器的训练过程就是参数估计。总结最大似然法估计参数的过程，一般分为以下四个步骤：

> - 1.写出似然函数；
> - 2.对似然函数取对数，并整理；
> - 3.求导数，令偏导数为0，得到似然方程组；
> - 4.解似然方程组，得到所有参数即为所求。

3. 朴素贝叶斯分类器

朴素贝叶斯分类是一种十分简单的分类算法，叫它朴素贝叶斯分类是因为这种方法的思想真的很朴素，朴素贝叶斯的思想基础是这样的：对于给出的待分类项，求解在此项出现的条件下各个类别出现的概率，哪个最大，就认为此待分类项属于哪个类别。通俗来说，就好比这么个道理，你在街上看到一个黑人，我问你你猜这哥们哪里来的，你十有八九猜非洲。为什么呢？因为黑人中非洲人的比率最高，当然人家也可能是美洲人或亚洲人，但在没有其它可用信息下，我们会选择条件概率最大的类别，这就是朴素贝叶斯的思想基础。

基于属性条件独立性假设，可以得到

$$P(c|\boldsymbol{x}) = \frac{P(c)P(\boldsymbol{x}|c)}{P(\boldsymbol{x})} = \frac{P(c)}{P(\boldsymbol{x})}\prod_{i=1}^{d} P(x_i|c)$$

d为属性数目，x_i为$\boldsymbol{x}$在第i个属性上的取值

$$h_{nb}(\boldsymbol{x}) = \underset{c\in\mathcal{Y}}{argmax} P(c)\prod_{i=1}^{d} P(x_i|c)$$

这就是朴素贝叶斯的表达式

这样，为每个样本估计类条件概率变成为每个样本的每个属性估计类条件概率。

对于**离散属性**，属性的条件概率可以估计为：$P(x_i|c) = \frac{|D_{c,x_i}|}{|D_c|}$

对于**连续属性**，若假设属性服从正态分布，则条件概率可以估计为：

$$p(x_i|c) = \frac{1}{\sqrt{2\pi}\sigma_{c,i}} exp(-\frac{(x_i-\mu c,i)^2}{2\sigma_{c,i}^2})$$

实例

这里用西瓜数据集3.0为例

--《周志华-西瓜书》

```
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,0.697,0.46,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,0.36,0.37,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
```

用上面数据训练一个朴素贝叶斯分类器，对测试例“测1”进行分类：

```
编号,色泽,根蒂,敲声,纹理,脐部,触感,密度,含糖率,好瓜
测1,青绿,蜷缩,浊响,清晰,凹陷,硬滑,0.697,0.46,是
```

注意，当样本数目足够多时才能进行有意义的概率估计. 本书仅是以西瓜数据集 3.0 对估计过程做一个简单的演示.

然后，为每个属性估计条件概率 $P(x_i \mid c)$:

$$P_{\text{青绿}|\text{是}} = P(\text{色泽} = \text{青绿} \mid \text{好瓜} = \text{是}) = \frac{3}{8} = 0.375,$$

$$P_{\text{青绿}|\text{否}} = P(\text{色泽} = \text{青绿} \mid \text{好瓜} = \text{否}) = \frac{3}{9} \approx 0.333,$$

$$P_{\text{蜷缩}|\text{是}} = P(\text{根蒂} = \text{蜷缩} \mid \text{好瓜} = \text{是}) = \frac{5}{8} = 0.625,$$

$$P_{\text{蜷缩}|\text{否}} = P(\text{根蒂} = \text{蜷缩} \mid \text{好瓜} = \text{否}) = \frac{3}{9} \approx 0.333,$$

$$P_{\text{浊响}|\text{是}} = P(\text{敲声} = \text{浊响} \mid \text{好瓜} = \text{是}) = \frac{6}{8} = 0.750,$$

$$P_{\text{浊响}|\text{否}} = P(\text{敲声} = \text{浊响} \mid \text{好瓜} = \text{否}) = \frac{4}{9} \approx 0.444,$$

$$P_{\text{清晰}|\text{是}} = P(\text{纹理} = \text{清晰} \mid \text{好瓜} = \text{是}) = \frac{7}{8} = 0.875,$$

$$P_{\text{清晰}|\text{否}} = P(\text{纹理} = \text{清晰} \mid \text{好瓜} = \text{否}) = \frac{2}{9} \approx 0.222,$$

$$P_{\text{凹陷}|\text{是}} = P(\text{脐部} = \text{凹陷} \mid \text{好瓜} = \text{是}) = \frac{6}{8} = 0.750,$$

这里应该是 5/8=0.625

$$P_{\text{凹陷}|\text{否}} = P(\text{脐部} = \text{凹陷} \mid \text{好瓜} = \text{否}) = \frac{2}{9} \approx 0.222,$$

$$P_{\text{硬滑}|\text{是}} = P(\text{触感} = \text{硬滑} \mid \text{好瓜} = \text{是}) = \frac{6}{8} = 0.750,$$

$$P_{\text{硬滑}|\text{否}} = P(\text{触感} = \text{硬滑} \mid \text{好瓜} = \text{否}) = \frac{6}{9} \approx 0.667,$$

$$p_{\text{密度: 0.697}|\text{是}} = p(\text{密度} = 0.697 \mid \text{好瓜} = \text{是})$$

$$= \frac{1}{\sqrt{2\pi} \cdot 0.129} \exp\left(-\frac{(0.697 - 0.574)^2}{2 \cdot 0.129^2}\right) \approx 1.959,$$

$$p_{\text{密度: 0.697}|\text{否}} = p(\text{密度} = 0.697 \mid \text{好瓜} = \text{否})$$

$$= \frac{1}{\sqrt{2\pi} \cdot 0.195} \exp\left(-\frac{(0.697 - 0.496)^2}{2 \cdot 0.195^2}\right) \approx 1.203,$$

$$p_{\text{含糖: 0.460}|\text{是}} = p(\text{含糖率} = 0.460 \mid \text{好瓜} = \text{是})$$

$$= \frac{1}{\sqrt{2\pi} \cdot 0.101} \exp\left(-\frac{(0.460 - 0.279)^2}{2 \cdot 0.101^2}\right) \approx 0.788,$$

$$p_{\text{含糖: 0.460}|\text{否}} = p(\text{含糖率} = 0.460 \mid \text{好瓜} = \text{否})$$

$$= \frac{1}{\sqrt{2\pi} \cdot 0.108} \exp\left(-\frac{(0.460 - 0.154)^2}{2 \cdot 0.108^2}\right) \approx 0.066.$$

于是, 有

中常通过取对数的
将“连乘”转化为
以避免数值下溢.

$$P(\text{好瓜}=\text{是}) \times P_{\text{青绿}|\text{是}} \times P_{\text{蜷缩}|\text{是}} \times P_{\text{浊响}|\text{是}} \times P_{\text{清晰}|\text{是}} \times P_{\text{凹陷}|\text{是}} \times P_{\text{硬滑}|\text{是}} \times p_{\text{密度: 0.697}|\text{是}} \times p_{\text{含糖: 0.460}|\text{是}} \approx 0.063 ,$$

$$P(\text{好瓜}=\text{否}) \times P_{\text{青绿}|\text{否}} \times P_{\text{蜷缩}|\text{否}} \times P_{\text{浊响}|\text{否}} \times P_{\text{清晰}|\text{否}} \times P_{\text{凹陷}|\text{否}} \times P_{\text{硬滑}|\text{否}} \times p_{\text{密度: 0.697}|\text{否}} \times p_{\text{含糖: 0.460}|\text{否}} \approx 6.80 \times 10^{-5} .$$

由于 $0.063 > 6.80 \times 10^{-5}$, 因此, 朴素贝叶斯分类器将测试样本“测 1”判别为“好瓜”.

需注意, 若某个属性值在训练集中没有与某个类同时出现过, 则直接基于式(7.17)进行概率估计, 再根据式(7.15) 进行判别将出现问题. 例如, 在使用西瓜数据集 3.0 训练朴素贝叶斯分类器时, 对一个“敲声=清脆”的测试例, 有

$$P_{\text{清脆}|\text{是}} = P(\text{敲声}=\text{清脆} \mid \text{好瓜}=\text{是}) = \frac{0}{8} = 0 ,$$

由于式(7.15)的连乘式计算出的概率值为零, 因此, 无论该样本的其他属性是什么, 哪怕在其他属性上明显像好瓜, 分类的结果都将是“好瓜=否”, 这显然不太合理.

为了避免其他属性携带的信息被训练集中未出现的属性值“抹去”, 在估计概率值时通常要进行“平滑”(smoothing), 常用“拉普拉斯修正”(Laplacian correction). 具体来说, 令 N 表示训练集 D 中可能的类别数, N_i 表示第 i 个属性可能的取值数, 则式(7.16)和(7.17)分别修正为

避免出现0的情况.

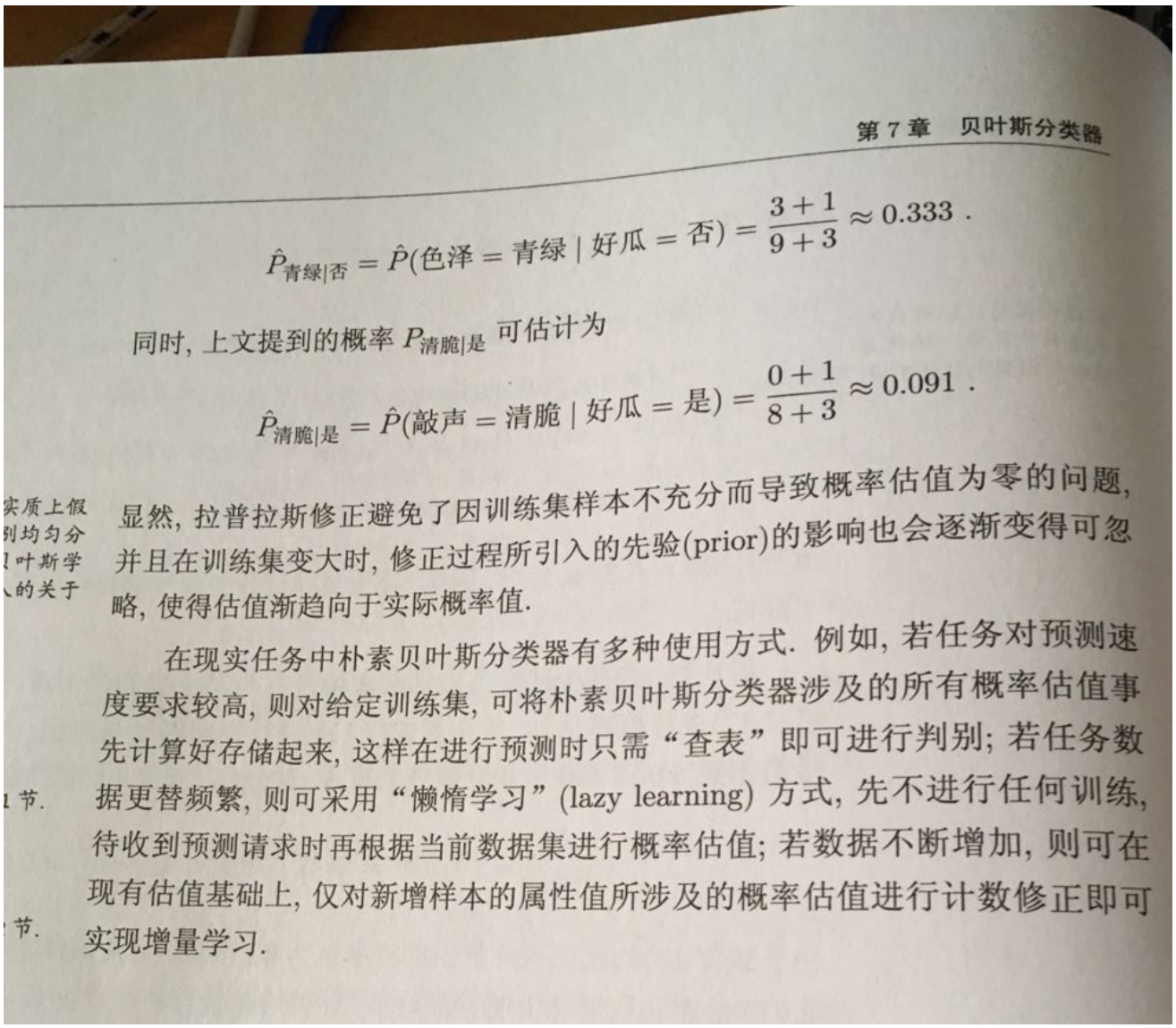
$$\hat{P}(c) = \frac{|D_c|+1}{|D|+N} , \tag{7.19}$$

$$\hat{P}(x_i \mid c) = \frac{|D_{c,x_i}|+1}{|D_c|+N_i} . \tag{7.20}$$

例如, 在本节的例子中, 类先验概率可估计为

$$\hat{P}(\text{好瓜}=\text{是}) = \frac{8+1}{17+2} \approx 0.474 , \quad \hat{P}(\text{好瓜}=\text{否}) = \frac{9+1}{17+2} \approx 0.526 .$$

类似地, $P_{\text{青绿}|\text{是}}$和$P_{\text{青绿}|\text{否}}$ 可估计为



第 7 章　贝叶斯分类器

$$\hat{P}_{青绿|否} = \hat{P}(色泽 = 青绿 \mid 好瓜 = 否) = \frac{3+1}{9+3} \approx 0.333 .$$

同时, 上文提到的概率 $P_{清脆|是}$ 可估计为

$$\hat{P}_{清脆|是} = \hat{P}(敲声 = 清脆 \mid 好瓜 = 是) = \frac{0+1}{8+3} \approx 0.091 .$$

显然, 拉普拉斯修正避免了因训练集样本不充分而导致概率估值为零的问题, 并且在训练集变大时, 修正过程所引入的先验(prior)的影响也会逐渐变得可忽略, 使得估值渐趋向于实际概率值.

在现实任务中朴素贝叶斯分类器有多种使用方式. 例如, 若任务对预测速度要求较高, 则对给定训练集, 可将朴素贝叶斯分类器涉及的所有概率估值事先计算好存储起来, 这样在进行预测时只需“查表”即可进行判别; 若任务数据更替频繁, 则可采用“懒惰学习”(lazy learning) 方式, 先不进行任何训练, 待收到预测请求时再根据当前数据集进行概率估值; 若数据不断增加, 则可在现有估值基础上, 仅对新增样本的属性值所涉及的概率估值进行计数修正即可实现增量学习.

实战 - 使用Python进行文本分类

要从文本中获取特征，需要先拆分文本。具体如何做？这里的特征是来自文本的词条（token），一个词条是字符的任意组合。可以把词条想象为单词，也可以使用非单词词条，如URL、IP地址或者任意其他字符串。

以在线社区的留言板为例。为了不影响社区的发展，我们要屏蔽侮辱性的言论，所以要构建一个快速过滤器，如果某条留言使用了负面或者侮辱性的语言，那么就该留言标识为内容不当。过滤这类内容是一个很常见的需求。对此问题建立两个类别，侮辱性和非侮辱性，使用1和0分别表示。

数据：

```
"my","dog","has","flea","problems","help","please","null"
"maybe","not","take","him","to","dog","park","stupid","null"
"my","dalmation","is","so","cute","I","love","him","null"
"stop","posting","stupid","worthless","garbage","null"
 "mr","licks","ate","my","steak","how","to","stop","him","null"
"quit","buying","worthless","dog","food","stupid","null"
```

Python实现：

```python
# _*_ coding:utf-8 _*_
import numpy as np

def loadDataSet():
    """
    导入数据， 1代表脏话
    @ return postingList: 数据集
    @ return classVec: 分类向量
    """
    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]
    return postingList, classVec

def createVocabList(dataSet):
    """
    创建词库
    @ param dataSet: 数据集
    @ return vocabSet: 词库
    """
    vocabSet = set([])
    for document in dataSet:
        # 求并集
        vocabSet = vocabSet | set(document)
```

```
29      return list(vocabSet)
30
31  def setOfWords2Vec(vocabList, inputSet):
32      """
33      文本词向量.词库中每个词当作一个特征，文本中就该词，该词特征就是1，没有就是0
34      @ param vocabList: 词表
35      @ param inputSet: 输入的数据集
36      @ return returnVec: 返回的向量
37      """
38      returnVec = [0] * len(vocabList)
39      for word in inputSet:
40          if word in vocabList:
41              returnVec[vocabList.index(word)] = 1
42          else:
43              print("单词: %s 不在词库中!" % word)
44      return returnVec
45
46
47  def trainNB0(trainMatrix, trainCategory):
48      """
49      训练
50      @ param trainMatrix: 训练集
51      @ param trainCategory: 分类
52      """
53      numTrainDocs = len(trainMatrix)
54      numWords = len(trainMatrix[0])
55      pAbusive = sum(trainCategory) / float(numTrainDocs)
56      #防止某个类别计算出的概率为0，导致最后相乘都为0，所以初始词都赋值1，分母赋值为2.
57      p0Num = np.ones(numWords)
58      p1Num = np.ones(numWords)
59      p0Denom = 2

60      p1Denom = 2
```

```
61      for i in range(numTrainDocs):
62          if trainCategory[i] == 1:
63              p1Num += trainMatrix[i]
64              p1Denom += sum(trainMatrix[i])
65          else:
66              p0Num += trainMatrix[i]
67              p0Denom += sum(trainMatrix[i])
68      # 这里使用log函数，方便计算，因为最后是比较大小，所有对结果
    没有影响。
69      p1Vect = np.log(p1Num / p1Denom)
70      p0Vect = np.log(p0Num / p0Denom)
71      return p0Vect, p1Vect, pAbusive
72
73  def classifyNB(vec2Classify,p0Vec,p1Vec,pClass1):
74      """
75      判断大小
76      """
77      p1 = sum(vec2Classify*p1Vec)+np.log(pClass1)
78      p0 = sum(vec2Classify*p0Vec)+np.log(1-pClass1)
79      if p1>p0:
80          return 1
81      else:
82          return 0
83
84  def testingNB():
85      listOPosts,listClasses = loadDataSet()
86      myVocabList = createVocabList(listOPosts)
87      trainMat=[]
88      for postinDoc in listOPosts:
89          trainMat.append(setOfWords2Vec(myVocabList,
    postinDoc))
90      p0V,p1V,pAb =
    trainNB0(np.array(trainMat),np.array(listClasses))

91      testEntry = ['love', 'my', 'dalmation']
```

```
92      thisDoc = np.array(setOfWords2Vec(myVocabList,
    testEntry))
93      print(testEntry,'classified as:
    ',classifyNB(thisDoc,p0V,p1V,pAb))
94      testEntry = ['stupid', 'garbage']
95      thisDoc = np.array(setOfWords2Vec(myVocabList,
    testEntry))
96      print(testEntry,'classified as:
    ',classifyNB(thisDoc,p0V,p1V,pAb))
97
98  if __name__=='__main__':
99      testingNB()
```