**​​​​​​​【机器学习】朴素贝叶斯（Naive Bayes）**

订阅专栏

* [【机器学习】k近邻算法（KNN）](https://blog.csdn.net/haha0332/article/details/112322847)
* [【机器学习】决策树（Decision Tree）](https://blog.csdn.net/haha0332/article/details/112515098)
* [【机器学习】朴素贝叶斯（Naive Bayes）](https://blog.csdn.net/haha0332/article/details/112575122)

**1. 概述**

**贝叶斯分类算法**是统计学的一种概率分类方法，[朴素贝叶斯](https://so.csdn.net/so/search?q=%E6%9C%B4%E7%B4%A0%E8%B4%9D%E5%8F%B6%E6%96%AF&spm=1001.2101.3001.7020)分类（Naive Bayes）是贝叶斯分类中最简单的一种。

分类原理：利用[贝叶斯](https://so.csdn.net/so/search?q=%E8%B4%9D%E5%8F%B6%E6%96%AF&spm=1001.2101.3001.7020)公式根据某特征的先验概率计算出其后验概率，然后选择具有最大后验概率的类作为该特征所属的类。

之所以称之为”朴素”，是因为贝叶斯分类只做最原始、最简单的假设：所有的特征之间是统计独立的。

**相关概念：**

**（1）条件概率**

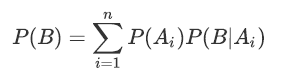
条件概率(Condittional probability)，就是指在事件B发生的情况下，事件A发生的概率，用P(A|B)来表示。

表达如下：

https://i-blog.csdnimg.cn/blog_migrate/d3dea4ce7cea4bd5bad56c398d4aaf87.png

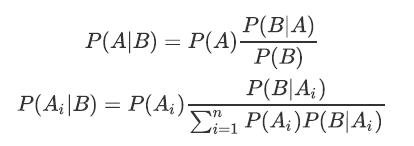
**（2）全概率**

如果事件 *A1,A2,...,An*构成一个完备事件且都有正概率，那么对于任意一个事件B则有：



**（3）贝叶斯推断**

根据条件概率和全概率公式，可以得到贝叶斯公式如下：



**P(A)**称为"**先验概率**"（Prior probability），即在B事件发生之前，我们对A事件概率的一个判断。  
**P(A|B)**称为"**后验概率**"（Posterior probability），即在B事件发生之后，我们对A事件概率的重新评估。  
**P(B|A)/P(B)**称为"**可能性函数**"（Likely hood），这是一个**调整因子**，使得预估概率更接近真实概率。

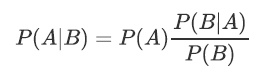
因此，我们就是通过**先验概率**乘以**调整因子**来计算其**后验概率**。即：

条件概率可以理解为：**后验概率** = **先验概率** \* **调整因子**

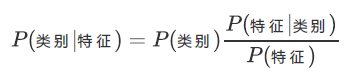
其中：

* 调整因子>1，表示事件B发生时，事件A发生的可能性变大，先验概率被增强；
* 调整因子=1，表示事件B无助于判断事件A的可能性；
* 调整因子<1，表示事件B发生时，事件A发生的可能性变小，先验概率呗削弱。

根据贝叶斯公式：

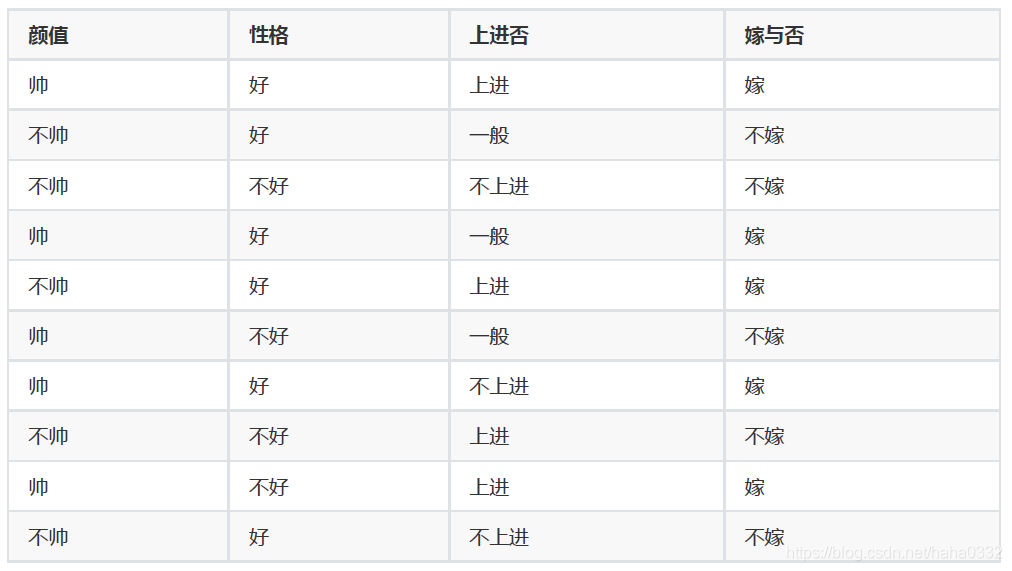


转换成分类任务的表达式：



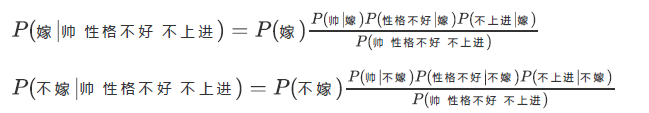
那我们结合这个公式，就可以**在相同特征下**，**对于不同类别的概率大小的判断**

若有以下数据：

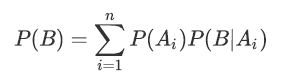


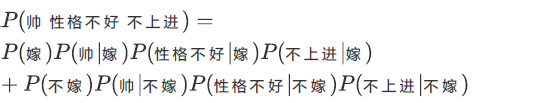
假如某男（帅，性格不好，不上进）向女生求婚，该女生嫁还是不嫁？

那实际上我们需要计算 *P(嫁 | 帅 性格不好 不上进)*和 *P(不嫁 | 帅 性格不好 不上进)*，即：

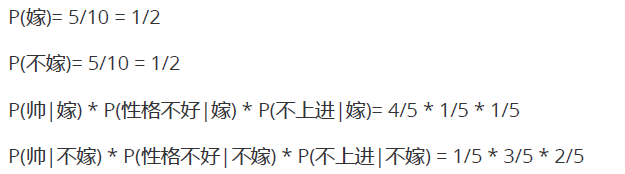


通过全概率公式计算分母，公式如下：





**对表中数据统计可得：**



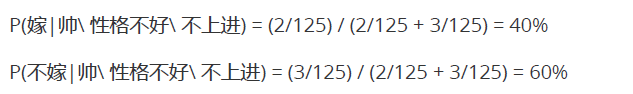
**则对于类别“嫁”的贝叶斯分子为：**

https://i-blog.csdnimg.cn/blog_migrate/2bc9fd567f68e580a1beace945caed48.png

**对于类别“不嫁”的贝叶斯分子为：**

https://i-blog.csdnimg.cn/blog_migrate/ca6d9e0780e055678d73ebd605306941.png

**最终计算结果：**



因此选择不嫁

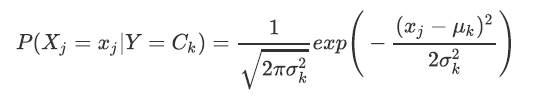
**2. 朴素贝叶斯算法种类**

在scikit-learn中，一共有3个朴素贝叶斯的[分类算法](https://so.csdn.net/so/search?q=%E5%88%86%E7%B1%BB%E7%AE%97%E6%B3%95&spm=1001.2101.3001.7020)。

分别是**GaussianNB**，**MultinomialNB**和**BernoulliNB**。

**（1）GaussianNB**

**GaussianNB**就是先验为高斯分布（正态分布）的朴素贝叶斯，假设每个标签的数据都服从简单的正态分布。



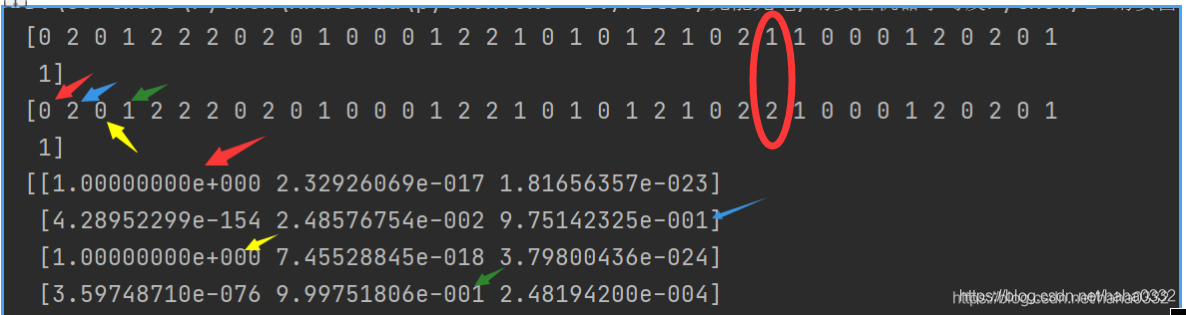
https://i-blog.csdnimg.cn/blog_migrate/a611780daaa87e205efd844988cb86b1.png

调用scikit-learn的包实现GaussianNB:

数据集为4中特征，3种类别的数据。

1. *# 导入包*
2. import pandas as pd
3. from sklearn.naive\_bayes import GaussianNB
4. from sklearn.model\_selection import train\_test\_split
5. from sklearn.metrics import accuracy\_score
6. from sklearn import datasets
8. *# 导入数据集*
9. iris = datasets.load\_iris()
11. *# 切分数据集,将数据的内容余目标切分成训练集和测试集*
12. Xtrain, Xtest, ytrain, ytest = train\_test\_split(iris.data,
13. iris.target,
14. random\_state=12)
15. *# 建模*
16. clf = GaussianNB()
17. clf.fit(Xtrain, ytrain)
18. *# 在测试集上执行预测，proba导出的是每个样本属于某类的概率*
19. pre\_result=clf.predict(Xtest) *# 预测结果*
20. pro\_result=clf.predict\_proba(Xtest) *# 预测的每一类的百分比*
21. *# 测试准确率*
22. accuracy\_value=accuracy\_score(ytest, pre\_result)
23. *# 打印输出结果*
24. print(ytest) *# 测试集的正确值*
25. print(pre\_result) *# 测试集的预测值*
26. print(pro\_result) *# 测试集的计算结果 每一类的可能性百分比*
27. print(accuracy\_value) *# 最后预测准确率*

https://csdnimg.cn/release/blogv2/dist/pc/img/newCodeMoreWhite.png



准确率为0.9736842105263158

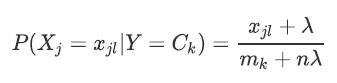
**（2）MultinomialNB**

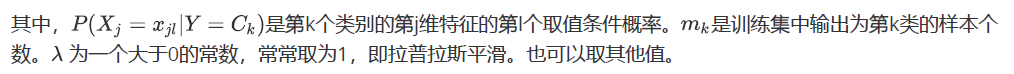
**MultinomialNB**就是先验为多项式分布的朴素贝叶斯。它假设特征是由一个简单多项式分布生成的。

多项分布可以描述各种类型样本出现次数的概率，因此多项式朴素贝叶斯非常适合用于描述出现次数或者出现次数比例的特征。

该模型常用于文本分类，特征表示的是次数，例如某个词语的出现次数。

多项式分布公式如下：





**（3）BernoulliNB**

**BernoulliNB**就是先验为伯努利分布的朴素贝叶斯。

假设特征的先验概率为二元伯努利分布，即如下式：

https://i-blog.csdnimg.cn/blog_migrate/d0a89f3a3c752473535e3be9184e0484.png

此时 l只有两种取值, x_{jl}只能取值0或者1。  
在伯努利模型中，每个特征的取值是布尔型的，即true和false，或者1和0。

在文本分类中，就是一个特征有没有在一个文档中出现。

**总结：**

* 如果样本特征的分布大部分是**连续值**，使用GaussianNB会比较好。
* 如果样本特征的分布大部分是**多元离散值**，使用MultinomialNB比较合适(常用于文本分类)。
* 如果样本特征是**二元离散值**或者很稀疏的多元离散值，应该使用BernoulliNB。

**3. 使用朴素贝叶斯进行文档分类**

朴素贝叶斯一个很重要的应用就是文本分类

**目的：**构建一个快速过滤器屏蔽掉侮辱性的言论

对此问题建立两个类型：侮辱类和非侮辱类，使用1和0分别表示

**思路**：我们把文本看成单词向量或者词条向量，也就是说将句子转换为向量。考虑出现所有文档中的单词，再决定将哪些单词纳入词汇表或者说所要的词汇集合，然后必须要将每一篇文档转换为词汇表上的向量。简单起见，我们先假设已经将本文切分完毕，存放到列表中，并对词汇向量进行分类标注。

此案例所有的[函数](https://marketing.csdn.net/p/3127db09a98e0723b83b2914d9256174?pId=2782&utm_source=glcblog&spm=1001.2101.3001.7020)：

* loadDataSet：创建实验数据集
* createVocabList：生成词汇表
* setOfWords2Vec：生成词向量
* get\_trainMat：所有词条向量列表
* trainNB：朴素贝叶斯分类器训练函数
* classifyNB：朴素贝叶斯分类器分类函数
* testingNB：朴素贝叶斯测试函数

**（1）创建实验数据集 loadDataSet**

1. """
2. 函数功能：创建实验数据集
3. 参数说明：无参数
4. 返回：
5. postingList：切分好的样本词条
6. classVec：类标签向量
7. """

10. def loadDataSet():
11. dataSet = [['my', 'dog', 'has', 'flea', 'problems', 'help', 'please'],
12. ['maybe', 'not', 'take', 'him', 'to', 'dog', 'park', 'stupid'],
13. ['my', 'dalmation', 'is', 'so', 'cute', 'I', 'love', 'him'],
14. ['stop', 'posting', 'stupid', 'worthless', 'garbage'],
15. ['mr', 'licks', 'ate', 'my', 'steak', 'how', 'to', 'stop', 'him'],
16. ['quit', 'buying', 'worthless', 'dog', 'food', 'stupid']] *# 切分好的词条*
17. classVec = [0, 1, 0, 1, 0, 1] *# 类别标签向量，1代表侮辱性词汇，0代表非侮辱性词汇*
18. return dataSet, classVec

https://csdnimg.cn/release/blogv2/dist/pc/img/newCodeMoreWhite.png

**（2）生成词汇表 createVocabList**

1. """
2. 函数功能：生成词汇表：将切分的样本词条整理成词汇表（不重复）
3. 参数说明：
4. dataSet：切分好的样本词条
5. 返回：
6. vocabList：不重复的词汇表
7. """

10. def createVocabList(dataSet):
11. vocabSet = set() *# 创建一个空的集合*
12. for doc in dataSet: *# 遍历dataSet中的每一条言论*
13. vocabSet = vocabSet | set(doc) *# 取并集*
14. vocabList = list(vocabSet)
15. return vocabList

**（3）生成词向量 setOfWords2Vec**

1. """
2. 函数功能：根据vocabList词汇表，将inputSet向量化，向量的每个元素为1或0
3. 参数说明：
4. vocabList：词汇表
5. inputSet：切分好的词条列表中的一条
6. 返回：
7. returnVec：文档向量,词集模型
8. """

11. def setOfWords2Vec(vocabList, inputSet):
12. returnVec = [0] \* len(vocabList) *# 创建一个其中所含元素都为0的向量*
13. for word in inputSet: *# 遍历每个词条*
14. if word in vocabList: *# 如果词条存在于词汇表中，则变为1*
15. returnVec[vocabList.index(word)] = 1
16. else:
17. print(f" {word} is not in my Vocabulary!")
18. return returnVec *# 返回文档向量*

https://csdnimg.cn/release/blogv2/dist/pc/img/newCodeMoreWhite.png

**（4）所有词条向量列表 get\_trainMat**

1. """
2. 函数功能：生成训练集向量列表
3. 参数说明：
4. dataSet：切分好的样本词条
5. 返回：
6. trainMat：所有的词条向量组成的列表
7. """

10. def get\_trainMat(dataSet):
11. trainMat = [] *# 初始化向量列表*
12. vocabList = createVocabList(dataSet) *# 生成词汇表*
13. for inputSet in dataSet: *# 遍历样本词条中的每一条样本*
14. returnVec = setOfWords2Vec(vocabList, inputSet) *# 将当前词条向量化*
15. trainMat.append(returnVec) *# 追加到向量列表中*
16. return trainMat

https://csdnimg.cn/release/blogv2/dist/pc/img/newCodeMoreWhite.png

**（5）朴素贝叶斯分类器训练函数 trainNB**

1. """
2. 函数功能：朴素贝叶斯分类器训练函数
3. 参数说明：
4. trainMat：训练文档矩阵
5. classVec：训练类别标签向量
6. 返回：
7. p0V：非侮辱类的条件概率数组
8. p1V：侮辱类的条件概率数组
9. pAb：文档属于侮辱类的概率
10. """

13. def trainNB(trainMat, classVec):
14. n = len(trainMat) # 计算训练的文档数目
15. m = len(trainMat[0]) # 计算每篇文档的词条数
16. pAb = sum(classVec) / n # 文档属于侮辱类的概率
17. p0Num = np.zeros(m) # 词条出现数初始化为0
18. p1Num = np.zeros(m) # 词条出现数初始化为0
19. p0Denom = 0 # 分母初始化为0
20. p1Denom = 0 # 分母初始化为0
21. for i in range(n): # 遍历每一个文档
22. if classVec[i] == 1: # 统计属于侮辱类的条件概率所需的数据
23. p1Num += trainMat[i]
24. p1Denom += sum(trainMat[i])
25. else: # 统计属于非侮辱类的条件概率所需的数据
26. p0Num += trainMat[i]
27. p0Denom += sum(trainMat[i])
28. p1V = p1Num / p1Denom
29. p0V = p0Num / p0Denom
30. return p0V, p1V, pAb # 返回属于非侮辱类,侮辱类和文档属于侮辱类的概率

https://csdnimg.cn/release/blogv2/dist/pc/img/newCodeMoreWhite.png

**（6）朴素贝叶斯分类器分类函数 classifyNB**

1. """
2. 函数功能：朴素贝叶斯分类器分类函数
3. 参数说明：
4. vec2Classify：待分类的词条数组
5. p0V：非侮辱类的条件概率数组
6. p1V：侮辱类的条件概率数组
7. pAb：文档属于侮辱类的概率
8. 返回：
9. 0：属于非侮辱类
10. 1：属于侮辱类
11. """

14. def classifyNB(vec2Classify, p0V, p1V, pAb):
15. p1 = reduce(lambda x, y: x \* y, vec2Classify \* p1V) \* pAb # 对应元素相乘
16. p0 = reduce(lambda x, y: x \* y, vec2Classify \* p0V) \* (1 - pAb)
17. print('p0:', p0)
18. print('p1:', p1)
19. if p1 > p0:
20. return 1
21. else:
22. return 0

https://csdnimg.cn/release/blogv2/dist/pc/img/newCodeMoreWhite.png

**（7）朴素贝叶斯测试函数 testingNB**

1. """
2. 函数功能：朴素贝叶斯测试函数
3. 参数说明：
4. testVec：测试样本
5. 返回：测试样本的类别
6. """

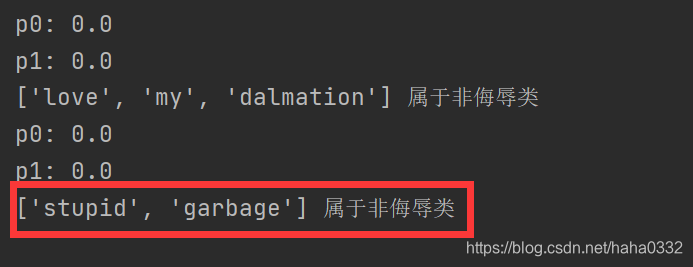
9. def testingNB(testVec):
10. dataSet, classVec = loadDataSet() *# 创建实验样本*
11. vocabList = createVocabList(dataSet) *# 创建词汇表*
12. trainMat = get\_trainMat(dataSet) *# 将实验样本向量化*
13. p0V, p1V, pAb = trainNB(trainMat, classVec) *# 训练朴素贝叶斯分类器*
14. thisone = setOfWords2Vec(vocabList, testVec) *# 测试样本向量化*
15. if classifyNB(thisone, p0V, p1V, pAb):
16. print(testVec, '属于侮辱类') *# 执行分类并打印分类结果*
17. else:
18. print(testVec, '属于非侮辱类') *# 执行分类并打印分类结果*

https://csdnimg.cn/release/blogv2/dist/pc/img/newCodeMoreWhite.png

最后测试：

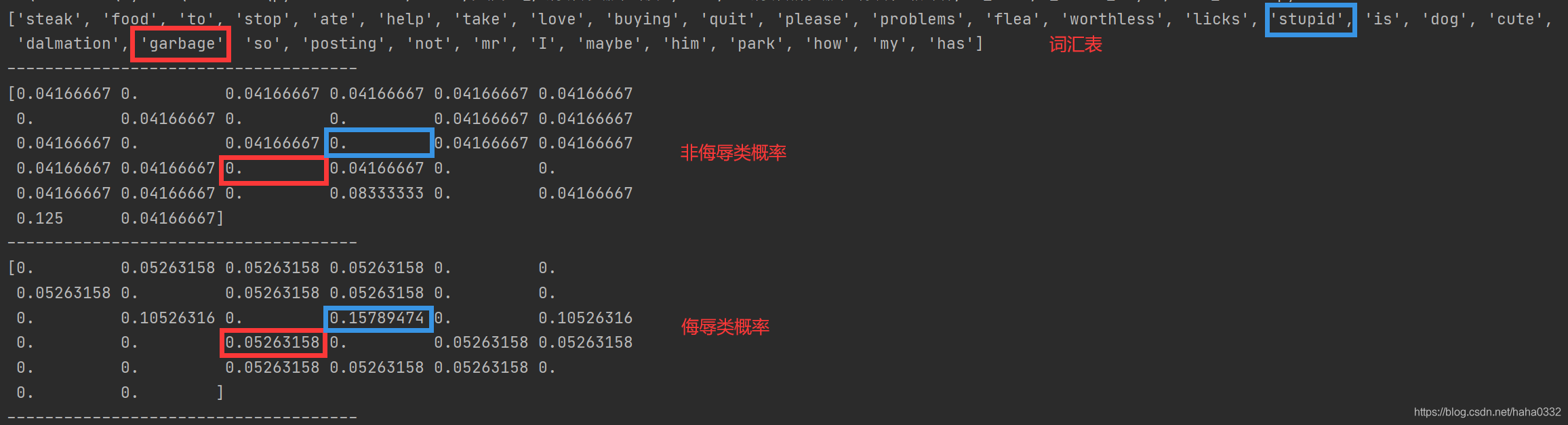
1. # 测试样本1
2. testVec1 = ['love', 'my', 'dalmation']
3. testingNB(testVec1)
4. # 测试样本2
5. testVec2 = ['stupid', 'garbage']
6. testingNB(testVec2)

结果：

​

这是出现问题，明显‘stupid’与‘garbage’属于侮辱词汇，而最后判断为非侮辱的，探究一下原因。

由于最后的概率是非侮辱和侮辱概率相乘，因此打印样本2的这两个的概率看看：

​

可以看到，即使这两个词汇侮辱类的概率值是存在的，而非侮辱值由于初始化设置为零因此相乘之后也为 零，所以判断为非侮辱类

**利用贝叶斯分类器对文档进行分类时，要计算多个概率的乘积以获得文档属于某个类别的概率，即计算p(w0|1)p(w1|1)p(w2|1)。如果其中有一个概率值为0，那么最后的成绩也为0**

处理该问题的方法：

可以将所有词的出现数初始化为1，并将分母初始化为2。这种做法就叫做**拉普拉斯平滑**(Laplace Smoothing)又被称为**加1平滑**，是比较常用的平滑方法，它就是为了解决0概率问题。

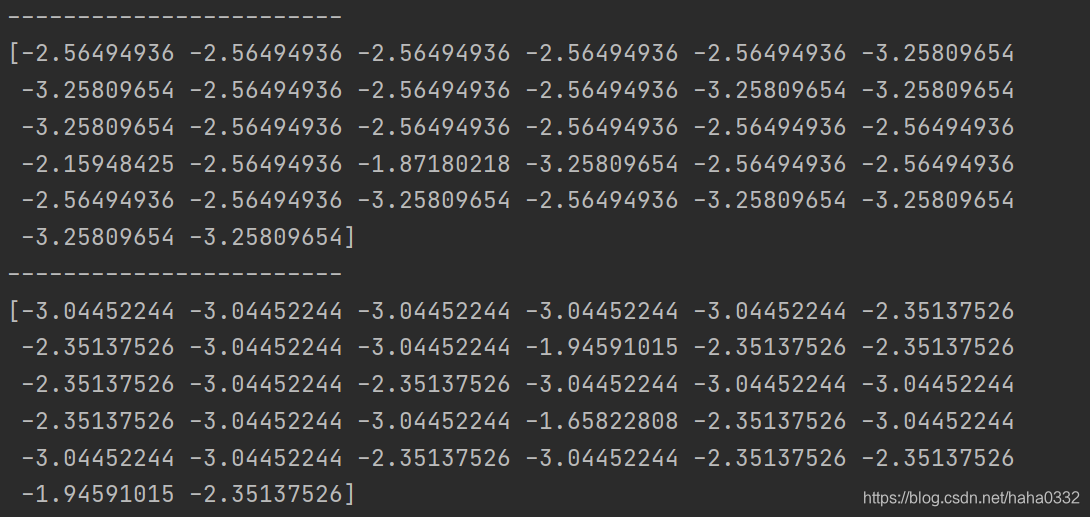
因此修改

* trainNB：朴素贝叶斯分类器训练函数
* testingNB：朴素贝叶斯测试函数

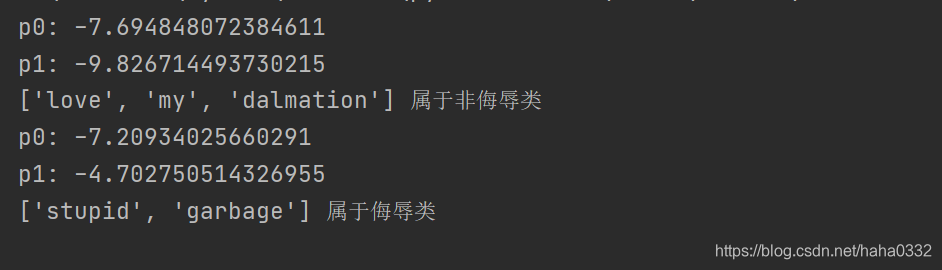
1. # 使用拉普拉斯平滑处理的
2. def trainNB(trainMat,classVec):
3. n = len(trainMat) #计算训练的文档数目
4. m = len(trainMat[0]) #计算每篇文档的词条数
5. pAb = sum(classVec)/n #文档属于侮辱类的概率
6. p0Num = np.ones(m) #词条出现数初始化为1
7. p1Num = np.ones(m) #词条出现数初始化为1
8. p0Denom = 2 #分母初始化为2
9. p1Denom = 2 #分母初始化为2
10. for i in range(n): #遍历每一个文档
11. if classVec[i] == 1: #统计属于侮辱类的条件概率所需的数据
12. p1Num += trainMat[i]
13. p1Denom += sum(trainMat[i])
14. else: #统计属于非侮辱类的条件概率所需的数据
15. p0Num += trainMat[i]
16. p0Denom += sum(trainMat[i])
17. p1V = np.log(p1Num/p1Denom)
18. p0V = np.log(p0Num/p0Denom)
19. return p0V,p1V,pAb #返回属于非侮辱类,侮辱类和文档属于侮辱类的概率
21. # 使用拉普拉斯平滑处理的
22. def classifyNB(vec2Classify, p0V, p1V, pAb):
23. p1 = sum(vec2Classify \* p1V) + np.log(pAb) # 对应元素相乘
24. p0 = sum(vec2Classify \* p0V) + np.log(1 - pAb) # 对应元素相乘
25. if p1 > p0:
26. return 1
27. else:
28. return 0

https://csdnimg.cn/release/blogv2/dist/pc/img/newCodeMoreWhite.png

计算概率不再是零：



运行结果正确：

​

**学习来源：**[**菊安酱的机器学习实战**](https://www.bilibili.com/video/BV18J411h7tQ)