

《机器学习实战》学习笔记（四）：基于概率论的分类方法 - 朴素贝叶斯

原创 我是管小亮 2019-08-28 19:09:27 4201 收藏 25

版权

分类专栏: Machine Learning 文章标签: 机器学习 机器学习实战 学习笔记 基于概率论的分类方法 - 朴素贝叶斯 朴素贝叶斯

欢迎关注WX公众号：【程序员管小亮】

【机器学习】《机器学习实战》读书笔记及代码 总目录

- <https://blog.csdn.net/TeFuirnever/article/details/99701256>

GitHub代码地址:

- <https://github.com/TeFuirnever/Machine-Learning-in-Action>

目录

欢迎关注WX公众号：【程序员管小亮】

本章内容

- 1、朴素贝叶斯
 - 2、基于贝叶斯决策理论的分类方法
 - 3、数学知识准备
 - 1) 条件概率
 - 2) 全概率公式
 - 3) 贝叶斯推断
 - 4、使用条件概率来分类
 - 5、朴素贝叶斯推断
 - 6、使用朴素贝叶斯进行文档分类
 - 7、使用Python 进行文本分类
 - 1) 准备数据：从文本中构建词向量
 - 2) 训练算法：从词向量计算概率
 - 3) 测试算法：根据现实情况修改分类器
 - 8、示例：使用朴素贝叶斯过滤垃圾邮件
 - 1) 准备数据：切分文本
 - 2) 测试算法：使用朴素贝叶斯进行交叉验证
 - 9、Sklearn构建朴素贝叶斯分类器用于新浪新闻分类
 - 1) 中文语句切分
 - 2) 文本特征选择
 - 10、sklearn.naive_bayes.MultinomialNB
 - 11、总结
- 参考文章

本章内容

- 使用概率分布进行分类
- 学习朴素贝叶斯分类器
- 解析RSS源数据
- 使用朴素贝叶斯来分析不同地区的态度

关于朴素贝叶斯的西瓜书笔记和课后习题的博客如下:

- 《机器学习》周志华西瓜书学习笔记（七）：贝叶斯分类器
- 《机器学习》周志华西瓜书习题参考答案：第7章 - 贝叶斯分类器

1、朴素贝叶斯

前两章要求分类器做出艰难决策，给出“该数据实例属于哪一类”这类问题的明确答案。不过，分类器有时会产生错误结果，这时可以要求 **分类器给出一个最优的类别猜测结果，同时给出这个猜测的概率估计值。**

概率论是许多机器学习算法的基础，所以深刻理解这一主题就显得十分重要。第3章（《机器学习实战》学习笔记（三）：决策树）在计算特征值取某个值的概率时涉及了一些概率知识，在那里先统计特征在数据集中取某个特定值的次数，然后除以数据集的实例总数，就得到了特征取该值的概率。本章会给出一些使用概率论进行分类的方法。首先从一个最简单的概率分类器开始，然后给出一些假设来学习朴素贝叶斯分类器。**称之为“朴素”，是因为整个形式化过程只做最原始、最简单的假设。**

朴素贝叶斯法是基于贝叶斯定理与特征条件独立假设的分类方法，是有监督的学习算法。现实生活中朴素贝叶斯算法应用非常广泛，如文本分类，垃圾邮件的分类，信用评估，钓鱼网站检测等等。

2、基于贝叶斯决策理论的分类方法

朴素贝叶斯

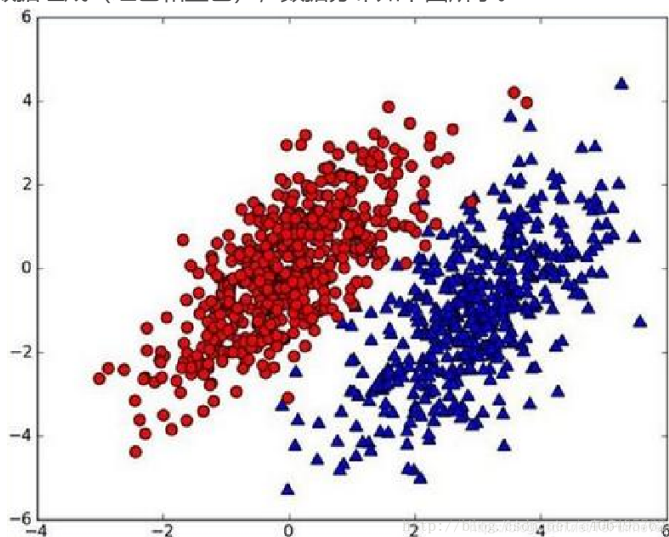
优点：在数据较少的情况下仍然有效，可以处理多类别问题。

缺点：对于输入数据的准备方式较为敏感。

适用数据类型：标称型数据。

朴素贝叶斯是贝叶斯决策理论的一部分，所以讲述朴素贝叶斯之前有必要快速了解一下贝叶斯决策理论。

假设现在有一个数据集，它由两类数据组成（红色和蓝色），数据分布如下图所示。



我们现在用 $p1(x, y)$ 表示数据点 (x, y) 属于类别（图中红色圆点表示的类别）的概率，用 $p2(x, y)$ 表示数据点 (x, y) 属于类别2（图中蓝色三角形表示的类别）的概率，那么对于一个新数据点 (x, y) ，可以用下面的规则来判断它的类别：

- 如果 $p1(x, y) > p2(x, y)$ ，那么类别为1
- 如果 $p2(x, y) > p1(x, y)$ ，那么类别为2

也就是说，会 **选择高概率所对应的类别**。这就是贝叶斯决策理论的核心思想，即选择具有最高概率的决策。回到上图中，如果该图中的整个数据使用6个浮点数（整个数据由两类不同分布的数据构成，有可能只需要6个统计参数来描述）来表示，并且计算类别概率的Python代码只有两行，那么你会更倾向于使用下面哪种方法来对该数据点进行分类？

1. 使用第1章的kNN，进行1000次距离计算；
2. 使用第2章的决策树，分别沿x轴、y轴划分数据；
3. 计算数据点属于每个类别的概率，并进行比较。

使用决策树不会非常成功；而和简单的概率计算相比，kNN的计算量太大。因此，对于上述问题，最佳选择是使用刚才提到的概率比较方法。

贝叶斯???

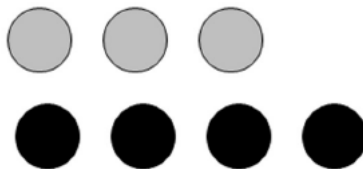
这里使用的概率解释属于贝叶斯概率理论的范畴，该理论非常流行且效果良好。贝叶斯概率以18世纪的一位神学家托马斯·贝叶斯（Thomas Bayes）的名字命名。贝叶斯概率 **引入先验知识和逻辑推理** 来处理不确定命题。另一种概率解释称为频数概率（frequency probability），它只从数据本身获得结论，并不考虑逻辑推理及先验知识。

3、数学知识准备

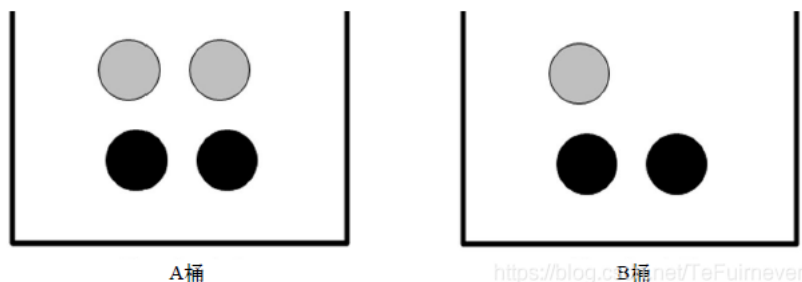
1) 条件概率

接下来，必须要详述 p_1 及 p_2 概率计算方法。为了能够计算 p_1 与 p_2 ，有必要讨论一下条件概率。

举个例子来说明，假设现在有一个装了7块石头的罐子，其中3块是灰色的，4块是黑色的。如果从罐子中随机取出一块石头，那么是灰色石头的可能性是多少？由于取石头有7种可能，其中3种为灰色，所以取出灰色石头的概率为 $3/7$ 。那么取到黑色石头的概率又是多少呢？很显然，是 $4/7$ 。我们使用 $P(\text{gray})$ 来表示取到灰色石头的概率，其概率值可以通过灰色石头数目除以总的石头数目来得到。



如果这7块石头如下图所示放在两个桶中，那么上述概率应该如何计算？



要计算 $P(\text{gray})$ 或者 $P(\text{black})$ ，事先得知道石头所在桶的信息会不会改变结果？你有可能已经想到计算从B桶中取到灰色石头的概率的办法，这就是所谓的 **条件概率 (conditional probability)**。假定计算的是从B桶取到灰色石头的概率，这个概率可以记作 $P(\text{gray}|\text{bucketB})$ ，我们称之为“**在已知石头出自B桶的条件下，取出灰色石头的概率**”。不难得到， $P(\text{gray}|\text{bucketA})$ 值为 $2/4$ ， $P(\text{gray}|\text{bucketB})$ 的值为 $1/3$ 。

条件概率的计算公式如下所示：

$$P(\text{gray}|\text{bucketB}) = P(\text{gray and bucketB}) / P(\text{bucketB})$$

来看看上述公式是否合理：

- 首先，用B桶中灰色石头的个数除以两个桶中总的石头数，得到 $P(\text{gray and bucketB}) = 1/7$ 。
- 其次，由于B桶中有3块石头，而总石头数为7，于是 $P(\text{bucketB})$ 就等于 $3/7$ 。
- 最后，有 $P(\text{gray}|\text{bucketB}) = P(\text{gray and bucketB}) / P(\text{bucketB}) = (1/7) / (3/7) = 1/3$ 。

这个公式虽然对于这个简单例子来说有点复杂，但当存在更多特征时是非常有效的。用代数方法计算条件概率时，该公式也很有用。

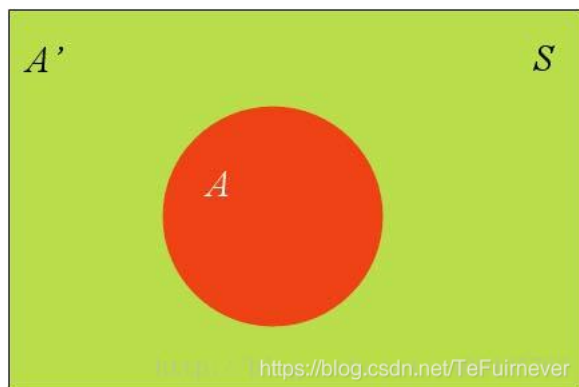
另一种有效计算条件概率的方法称为 **贝叶斯准则**。贝叶斯准则告诉我们如何交换条件概率中的条件与结果，即如果已知 $P(x|c)$ ，要求 $P(c|x)$ ，那么可以使用下面的计算方法：

$$p(c|x) = \frac{p(x|c)p(c)}{p(x)}$$

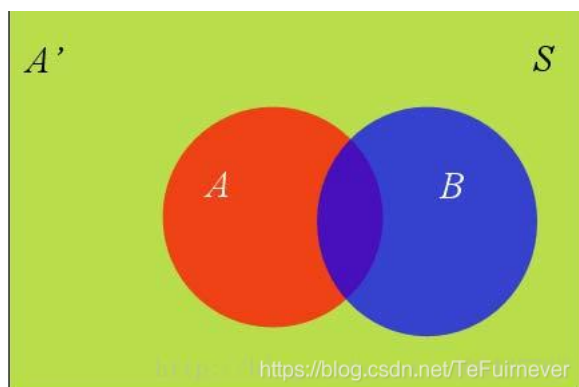
2) 全概率公式

除了条件概率以外，在计算 p_1 和 p_2 的时候，还要用到全概率公式，因此，这里继续讨论一下全概率公式。

举个例子，假设样本空间S是两个事件A与A'的和，其中红色部分是事件A，绿色部分是事件A'，如下图：



而事件B的位置如下图，和事件A有交集但是并不是包含关系，这个时候 $P(B)=??$ ？



很明显， $P(B)$ 是两部分，其中一部分是和事件A重合的，另一部分是和事件A'重合的，这个时候分别去求两个概率相加就是 $P(B)$ 了。

- 和事件A重合的部分，根据条件概率公式，等于A的概率乘上A条件下B的概率，即 $P_1=P(B|A)P(A)$ ；
- 和事件A'重合的部分，根据条件概率公式，等于A'的概率乘上A'条件下B的概率，即 $P_2=P(B|A')P(A')$ ；

综上可以得出全概率公式：

$$P(B) = P(B|A)P(A) + P(B|A')P(A')$$

其实全概率就是 **表示达到某个目的的多种方式各自概率的和**。

3) 贝叶斯推断

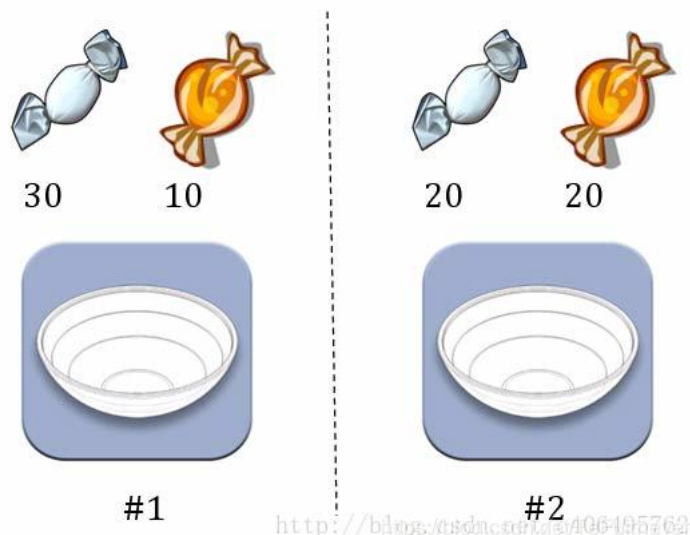
对条件概率公式进行变形，可以得到如下形式：

$$P(A|B) = P(A) \frac{P(B|A)}{P(B)}$$

- $P(A)$ 称为"**先验概率**" (Prior probability)，即在B事件发生之前，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，对A事件概率的重新评估。
- $P(B|A)/P(B)$ 称为"**可能性函数**" (Likelihood)，这是一个调整因子，使得预估概率更接近真实概率。

所以，条件概率可以理解成下面的式子：**后验概率 = 先验概率 × 调整因子**

下面做一个实验来验证一下，先预估一个"先验概率"，然后加入实验结果，这样到底是增强还是削弱了"先验概率"，由此得到更接近事实的"后验概率"。在这里，如果"可能性函数" $P(B|A)/P(B) > 1$ ，意味着"先验概率"被增强，事件A的發生的可能性变大；如果"可能性函数" $= 1$ ，意味着B事件无助于判断事件A的可能性；如果"可能性函数" < 1 ，意味着"先验概率"被削弱，事件A的可能性变小。



两个一模一样的碗，一号碗有30颗水果糖和10颗巧克力糖，二号碗有水果糖和巧克力糖各20颗。现在随机选择一个碗，从中摸出一颗糖，发现是水果糖。请问这颗水果糖来自一号碗的概率有多大？（眼熟吗，这不就是个大学概率题哈哈 😊）

现在假定， H_1 表示一号碗， H_2 表示二号碗。由于这两个碗是一样的，在取出水果糖之前，这两个碗被选中的概率相同，也就是等概率的，所以 $P(H_1)=P(H_2)=0.5$ 。这个概率就叫做“**先验概率**”，即没有做实验之前，来自一号碗的概率是0.5。

再假定， E 表示水果糖，所以问题就变成了在已知拿出来的是 E 的情况下，这个 E 来自一号碗的概率有多大？即求 $P(H_1|E)$ 。这个概率叫做“**后验概率**”，即在 E 事件发生之后，对 $P(H_1)$ 的修正结果。

根据条件概率变形公式，可以得到

$$P(H_1|E) = P(H_1) \frac{P(E|H_1)}{P(E)}$$

现在这个题就是一个纯粹的数学问题了，已知 $P(H_1)$ 等于0.5， $P(E|H_1)$ 为从一号碗中取出的糖是水果糖的概率，等于 $30 / (30+10)=0.75$ ，那么求出 $P(E)$ 就可以得到答案。根据全概率公式

$$P(E) = P(E|H_1)P(H_1) + P(E|H_2)P(H_2)$$

$$P(E) = 0.75 \times 0.5 + 0.5 \times 0.5 = 0.625$$

所以，得出 $P(H_1|E)$ 。

$$P(H_1|E) = 0.5 \times \frac{0.75}{0.625} = 0.6$$

本来是0.5，但是通过调整之后，来自一号碗的概率是0.6。也就是说，取出水果糖之后， H_1 事件的可能性得到了增强。

调整因子 = $0.75 / 0.625 = 1.2$ ，是 > 1 的，意味着“先验概率”被增强，事件A的发生的可能性变大。

4、使用条件概率来分类

贝叶斯决策理论要求计算两个概率 $p_1(x, y)$ 和 $p_2(x, y)$ ：

- 如果 $p_1(x, y) > p_2(x, y)$ ，那么属于类别1；
- 如果 $p_2(x, y) > p_1(x, y)$ ，那么属于类别2。

但这两个准则并不是贝叶斯决策理论的所有内容。使用 $p_1()$ 和 $p_2()$ 只是为了尽可能简化描述，而真正需要计算和比较的是 $p(c_1 | x, y)$ 和 $p(c_2 | x, y)$ 。这些符号所代表的具体意义是：给定某个由 x, y 表示的数据点，那么该数据点来自类别 c_1 的概率是多少？数据点来自类别 c_2 的概率又是多少？注意这些概率与刚才给出的概率 $p(x, y | c_1)$ 并不一样，不过可以使用贝叶斯准则来交换概率中条件与结果。具体地，应用贝叶斯准则得到：

$$p(c_i | x, y) = \frac{p(x, y | c_i) p(c_i)}{p(x, y)}$$

使用这些定义，可以定义贝叶斯分类准则为：

- 如果 $P(c1|x, y) > P(c2|x, y)$, 那么属于类别 $c1$ 。
- 如果 $P(c1|x, y) < P(c2|x, y)$, 那么属于类别 $c2$ 。

使用贝叶斯准则，可以通过已知的三个概率值来计算未知的概率值。

5、朴素贝叶斯推断

我们来看一看朴素贝叶斯推断，对比之前讲过的贝叶斯推断，你可以发现贝叶斯和朴素贝叶斯的概念是不同的，区别就在于“朴素”二字，**朴素贝叶斯对条件概率分布做了条件独立性的假设**。比如下面的公式，假设有 n 个特征：

$$P(a|X) = p(X|a)p(a) = p(x_1, x_2, x_3, \dots, x_n|a)p(a)$$

由于每个特征都是独立的，我们可以进一步拆分公式：

$$\begin{aligned} p(a|X) &= p(X|a)p(a) \\ &= \{p(x_1|a) * p(x_2|a) * p(x_3|a) * \dots * p(x_n|a)\}p(a) \end{aligned}$$

这样就可以分布计算了，举一个例子，某个医院早上来了六个门诊的病人，他们的情况如下表所示：

现在又来了第七个病人，是一个打喷嚏的建筑工人。请问他患上感冒的概率有多大？

根据贝叶斯定理：

$$P(A|B) = \frac{P(B|A) P(A)}{P(B)}$$

应用实例可得：

$$\begin{aligned} &P(\text{感冒}|\text{打喷嚏} \times \text{建筑工人}) \\ &= \frac{P(\text{打喷嚏} \times \text{建筑工人}|\text{感冒}) \times P(\text{感冒})}{P(\text{打喷嚏} \times \text{建筑工人})} \end{aligned}$$

根据朴素贝叶斯条件独立性的假设可知，“打喷嚏”和“建筑工人”这两个特征是独立的，因此：

$$\begin{aligned} &P(\text{感冒}|\text{打喷嚏} \times \text{建筑工人}) \\ &= \frac{P(\text{打喷嚏}|\text{感冒}) \times P(\text{建筑工人}|\text{感冒}) \times P(\text{感冒})}{P(\text{打喷嚏}) \times P(\text{建筑工人})} \end{aligned}$$

这里带入数字可以计算：

$$P(\text{感冒}|\text{打喷嚏} \times \text{建筑工人}) = \frac{0.66 \times 0.33 \times 0.5}{0.5 \times 0.33} = 0.66$$

因此，这个打喷嚏的建筑工人，有66%的概率是得了感冒。同理，可以得到


```

1 | P(过敏 | 打喷嚏 x 建筑工人)
2 | = (P(打喷嚏 | 过敏) x P(建筑工人 | 过敏) x P(过敏)) / (P(打喷嚏) x P(建筑工人))
3 | = (1 x 0 x 1/6) / (0.5 x 0.33)
4 | = 0

1 | P(脑震荡 | 打喷嚏 x 建筑工人)
2 | = (P(打喷嚏 | 脑震荡) x P(建筑工人 | 脑震荡) x P(脑震荡)) / (P(打喷嚏) x P(建筑工人))
3 | = (0 x 0.5 x 1/3) / (0.5 x 0.33)
4 | = 0

```

比较这几个概率，就可以知道他最可能得的是感冒。

这就是贝叶斯分类器的基本方法：**在统计资料的基础上，依据某些特征，计算各个类别的概率，从而实现分类。**

6、使用朴素贝叶斯进行文档分类

机器学习的一个重要应用就是**文档的自动分类**。在文档分类中，整个文档（如一封电子邮件）是实例，而电子邮件中的某些元素则构成特征。虽然电子邮件是一种会不断增加的文本，但同样也可以对新闻报道、用户留言、政府公文等其他任意类型的文本进行分类。我们可以观察文档中出现的词，并把每个词的出现或者不出现作为一个特征，这样得到的特征数目就会跟词汇表中的词目一样多。**朴素贝叶斯是上节介绍的贝叶斯分类器的一个扩展，是用于文档分类的常用算法。**

使用每个词作为特征并观察它们是否出现，这样得到的特征数目会有多少呢？针对的是哪一种人类语言呢？当然不止一种语言。据统计，仅在英语中，单词的总数就有500000之多。为了能进行英文阅读，估计需要掌握数千单词。

朴素贝叶斯的一般过程

- (1) 收集数据：可以使用任何方法。本章使用RSS源。
- (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：一个常见的朴素贝叶斯应用是文档分类。可以在任意的分类场景中使用朴素贝叶斯分类器，不一定非要是文本。

假设词汇表中有1000个单词。要得到好的概率分布，就需要足够的数据样本，假定样本数为N。前面讲到的约会网站示例中有1000个实例，手写识别示例中每个数字有200个样本，而决策树示例中有24个样本。其中，24个样本有点少，200个样本好一些，而1000个样本就非常好了。约会网站例子中有三个特征。由统计学知，如果每个特征需要N个样本，那么对于10个特征将需要 N^{10} 个样本，对于包含1000个特征的词汇表将需要 N^{1000} 个样本。可以看到，所需要的样本数会随着特征数目增大而迅速增长。

如果特征之间相互独立，那么样本数就可以从 N^{1000} 减少到1000×N。所谓**独立（independence）**指的是统计意义上的独立，即**一个特征或者单词出现的可能性与它和其他单词相邻没有关系。**

举个例子讲，假设单词bacon出现在unhealthy后面与出现在delicious后面的概率相同。当然，我们知道这种假设并不正确，bacon常常出现在delicious附近，而很少出现在unhealthy附近，这个假设正是朴素贝叶斯分类器中朴素（naive）一词的含义。朴素贝叶斯分类器中的另一个假设是，**每个特征同等重要**。其实这个假设也有问题。如果要判断留言板的留言是否得当，那么可能不需要看完所有的1000个单词，而只需要看10~20个特征就足以做出判断了。尽管上述假设存在一些小的瑕疵，但朴素贝叶斯的实际效果却很好。

7、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。具体如何做呢？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。然后将每一个文本片段表示为一个词条向量，其中值为1表示词条出现在文档中，0表示词条未出现。

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，那么就将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：侮辱类和非侮辱类，使用1和0分别表示。

1) 准备数据：从文本中构建词向量

我们将把文本看成**单词向量**或者**词条向量**，也就是说将句子转换为向量。考虑出现在所有文档中的所有单词，再决定将哪些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，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。

编写代码如下：

```

1  ...
2  Parameters:
3      无
4  Returns:
5      postingList - 实验样本切分的词条
6      classVec - 类别标签向量
7  ...
8  # 函数说明: 创建实验样本
9  def loadDataSet():
10     postingList=[['my', 'dog', 'has', 'flea', 'problems', 'help', 'please'],      #切分的词条
11                  ['maybe', 'not', 'take', 'him', 'to', 'dog', 'park', 'stupid'],
12                  ['my', 'dalmation', 'is', 'so', 'cute', 'I', 'love', 'him'],
13                  ['stop', 'posting', 'stupid', 'worthless', 'garbage'],
14                  ['mr', 'licks', 'ate', 'my', 'steak', 'how', 'to', 'stop', 'him'],
15                  ['quit', 'buying', 'worthless', 'dog', 'food', 'stupid']]
16     classVec = [0,1,0,1,0,1]#类别标签向量, 1代表侮辱性词汇, 0代表不是
17     return postingList,classVec
18
19  ...
20  Parameters:
21     vocabList - createVocabList返回的列表
22     inputSet - 切分的词条列表
23  Returns:
24     returnVec - 文档向量,词集模型
25  ...
26  # 函数说明: 根据vocabList词汇表, 将inputSet向量化, 向量的每个元素为1或0
27  def setOfWords2Vec(vocabList, inputSet):
28     returnVec = [0] * len(vocabList)      #创建一个其中所含元素都为0的向量
29     for word in inputSet:                  #遍历每个词条
30         if word in vocabList:               #如果词条存在于词汇表中, 则置1
31             returnVec[vocabList.index(word)] = 1
32         else: print("the word: %s is not in my Vocabulary!" % word)
33     return returnVec                      #返回文档向量
34
35  ...
36  Parameters:
37     dataSet - 整理的样本数据集
38  Returns:
39     vocabSet - 返回不重复的词条列表, 也就是词汇表
40  ...
41  # 函数说明: 将切分的实验样本词条整理成不重复的词条列表, 也就是词汇表
42  def createVocabList(dataSet):
43     vocabSet = set([])                      #创建一个空的不重复列表
44     for document in dataSet:
45         vocabSet = vocabSet | set(document) #取并集
46     return list(vocabSet)
47
48  if __name__ == '__main__':
49     postingList, classVec = loadDataSet()
50     print('postingList:\n',postingList)
51     myVocabList = createVocabList(postingList)
52     print('myVocabList:\n',myVocabList)
53     trainMat = []
54     for postinDoc in postingList:
55         trainMat.append(setOfWords2Vec(myVocabList, postinDoc))
56     print('trainMat:\n', trainMat)
57
58
59  >>>
60  postingList:
61  [['my', 'dog', 'has', 'flea', 'problems', 'help', 'please'], ['maybe', 'not', 'take', 'him', 'to', 'dog', 'park', 'stupid'], ['
62  myVocabList:
63  ['stop', 'please', 'problems', 'worthless', 'mr', 'is', 'I', 'help', 'love', 'maybe', 'has', 'take', 'quit', 'ate', 'licks', 't
64  trainMat:
65  [[0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

```

从运行结果可以看出：

- postingList 是原始的 **词条列表**；
- myVocabList 是 **词汇表**，是所有单词出现的集合，没有重复的元素；
- trainMat 是所有的词条向量组成的列表，它里面存放的是根据 myVocabList 向量化的 **词条向量**。

词汇表是用来干什么的？没错，它是用来将词条向量化的，一个单词在词汇表中出现过一次，那么就在相应位置记作1，如果没有出现就在相应位置记作0。

2) 训练算法：从词向量计算概率

前面介绍了如何将一组单词转换为一组数字，接下来看看如何使用这些数字计算概率。现在已经知道一个词是否出现在一篇文档中，也知道该文档所属的类别。还记得 **贝叶斯准则**？我们重写贝叶斯准则，将之前的x、y 替换为 \mathbf{w} 。粗体 \mathbf{w} 表示这是一个向量，即它由多个数值组成。在这个例子中，数值个数与词汇表中的词个数相同。

$$p(c_i | \mathbf{w}) = \frac{p(\mathbf{w} | c_i)p(c_i)}{p(\mathbf{w})}$$

使用上述公式，对每个类计算该值，然后比较这两个概率值的大小。如何计算呢？首先可以通过类别 i （侮辱性留言或非侮辱性留言）中文档数除以总的文档数来计算概率 $p(c_i)$ 。接下来计算 $p(\mathbf{w} | c_i)$ ，这里就要用到 **朴素贝叶斯假设**。如果将 \mathbf{w} 展开为一个个独立特征，那么就可以将上述概率写作 $p(w_0, w_1, w_2 \dots w_N | c_i)$ 。这里假设所有词都互相独立，该假设也称作条件独立性假设，它意味着可以使用 $p(w_0 | c_i)p(w_1 | c_i)p(w_2 | c_i) \dots p(w_N | c_i)$ 来计算上述概率，这就极大地简化了计算的过程。

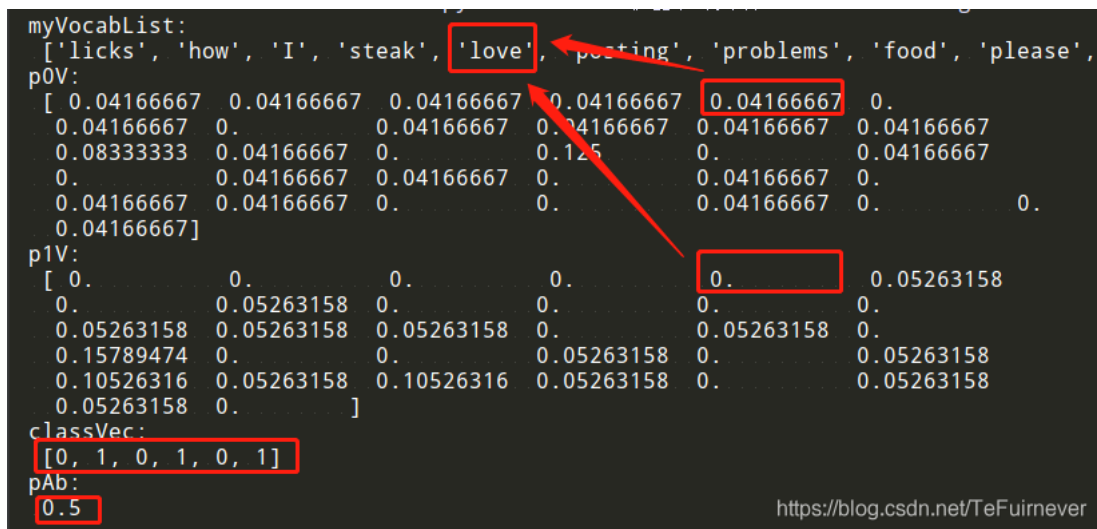
接下来，通过词条向量训练朴素贝叶斯分类器。代码如下：

```
1 import numpy as np
2
3 ...
4 Parameters:
5     无
6 Returns:
7     postingList - 实验样本切分的词条
8     classVec - 类别标签向量
9 ...
10 # 函数说明: 创建实验样本
11 def loadDataSet():
12     postingList=[['my', 'dog', 'has', 'flea', 'problems', 'help', 'please'],      #切分的词条
13                  ['maybe', 'not', 'take', 'him', 'to', 'dog', 'park', 'stupid'],
14                  ['my', 'dalmation', 'is', 'so', 'cute', 'I', 'love', 'him'],
15                  ['stop', 'posting', 'stupid', 'worthless', 'garbage'],
16                  ['mr', 'licks', 'ate', 'my', 'steak', 'how', 'to', 'stop', 'him'],
17                  ['quit', 'buying', 'worthless', 'dog', 'food', 'stupid']]
18     classVec = [0,1,0,1,0,1]#类别标签向量, 1代表侮辱性词汇, 0代表不是
19     return postingList,classVec
20
21 ...
22 Parameters:
23     vocabList - createVocabList返回的列表
24     inputSet - 切分的词条列表
25 Returns:
26     returnVec - 文档向量,词集模型
27 ...
28 # 函数说明: 根据vocabList词汇表, 将inputSet向量化, 向量的每个元素为1或0
29 def setOfWords2Vec(vocabList, inputSet):
30     returnVec = [0] * len(vocabList)      #创建一个其中所含元素都为0的向量
31     for word in inputSet:                  #遍历每个词条
32         if word in vocabList:               #如果词条存在于词汇表中, 则置1
33             returnVec[vocabList.index(word)] = 1
34         else: print("the word: %s is not in my Vocabulary!" % word)
35     return returnVec                      #返回文档向量
36
37 ...
38 Parameters:
39     dataSet - 整理的样本数据集
40 Returns:
41     vocabSet - 返回不重复的词条列表, 也就是词汇表
42 ...
43 # 函数说明: 将切分的实验样本词条整理成不重复的词条列表, 也就是词汇表
44 def createVocabList(dataSet):
45     vocabSet = set([])                      #创建一个空的不重复列表
46     for document in dataSet:
47         vocabSet = vocabSet | set(document) #取并集
48     return list(vocabSet)
49
50 ...
51 Parameters:
52     trainMatrix - 训练文档矩阵, 即setOfWords2Vec返回的returnVec构成的矩阵
53     trainCategory - 训练类别标签向量, 即loadDataSet返回的classVec
54 Returns:
55     model - 训练好的朴素贝叶斯模型
```

```

51     p0Vect - 侮辱类的条件概率数组
52     p1Vect - 非侮辱类的条件概率数组
53     pAbusive - 文档属于侮辱类的概率
54     ...
55 # 函数说明: 朴素贝叶斯分类器训练函数
56 def trainNB0(trainMatrix, trainCategory):
57     numTrainDocs = len(trainMatrix)          # 计算训练的文档数目
58     numWords = len(trainMatrix[0])           # 计算每篇文档的词条数
59     pAbusive = sum(trainCategory)/float(numTrainDocs) # 文档属于侮辱类的概率
60     p0Num = np.zeros(numWords); p1Num = np.zeros(numWords) # 创建numpy.zeros数组, 词条出现数初始化为0
61     p0Denom = 0.0; p1Denom = 0.0           # 分母初始化为0
62     for i in range(numTrainDocs):
63         if trainCategory[i] == 1:           # 统计属于侮辱类的条件概率所需的数据, 即P(w0|1), P(w1|1), P(w2|1)...
64             p1Num += trainMatrix[i]
65             p1Denom += sum(trainMatrix[i])
66         else:                                # 统计属于非侮辱类的条件概率所需的数据, 即P(w0|0), P(w1|0), P(w2|0)...
67             p0Num += trainMatrix[i]
68             p0Denom += sum(trainMatrix[i])
69     p1Vect = p1Num/p1Denom
70     p0Vect = p0Num/p0Denom
71     return p0Vect, p1Vect, pAbusive # 返回属于侮辱类的条件概率数组, 属于非侮辱类的条件概率数组, 文档属于侮辱类的概率
72
73 if __name__ == '__main__':
74     postingList, classVec = loadDataSet()
75     myVocabList = createVocabList(postingList)
76     print('myVocabList:\n', myVocabList)
77     trainMat = []
78     for postinDoc in postingList:
79         trainMat.append(setOfWords2Vec(myVocabList, postinDoc))
80     p0V, p1V, pAb = trainNB0(trainMat, classVec)
81     print('p0V:\n', p0V)
82     print('p1V:\n', p1V)
83     print('classVec:\n', classVec)
84     print('pAb:\n', pAb)
85

```



```

myVocabList:
['licks', 'how', 'I', 'steak', 'love', 'posting', 'problems', 'food', 'please',
p0V:
[ 0.04166667  0.04166667  0.04166667  0.04166667  0.04166667  0.
 0.04166667  0.          0.04166667  0.04166667  0.04166667
 0.08333333  0.04166667  0.          0.125          0.          0.04166667
 0.          0.04166667  0.04166667  0.          0.04166667  0.
 0.04166667  0.04166667  0.          0.          0.04166667  0.          0.
 0.04166667]
p1V:
[ 0.          0.          0.          0.          0.          0.05263158
 0.          0.05263158  0.          0.          0.          0.
 0.05263158  0.05263158  0.05263158  0.          0.05263158  0.
 0.15789474  0.          0.          0.05263158  0.          0.05263158
 0.10526316  0.05263158  0.10526316  0.05263158  0.          0.05263158
 0.05263158  0.          ]
classVec:
[0, 1, 0, 1, 0, 1]
pAb:
0.5

```

运行结果如下, p0V存放的是属于类别0的单词的概率, 也就是非侮辱类词汇的概率。比如p0V的正数第5个概率, 就是love这个单词属于非侮辱类的概率为0.04166667, 换算成百分比, 也就是4.17%。同理, p1V的正数第5个概率, 就是love这个单词属于侮辱类的概率为0。简单的单词love, 大家都知道是属于非侮辱类的, 这么看, 分类还是比较准确的。pAb是所有侮辱类的样本占有所有样本的概率, 从classVec中可以看出, 一共有3个侮辱类, 3个非侮辱类。所以侮辱类的概率是0.5。

因此, p0V和p1V存放的就是myVocabList中单词的条件概率, 而pAb就是先验概率。

3) 测试算法: 根据现实情况修改分类器

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:

$$P(w_0 | 1)P(w_1 | 1)P(w_2 | 1)$$

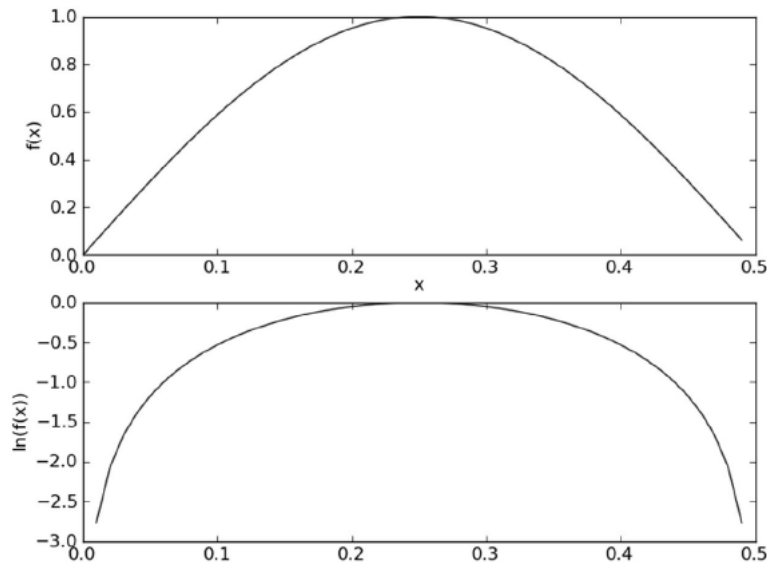
如果其中一个概率值为0，那么最后的乘积也为0。为降低这种影响，可以将所有词的出现数初始化为1，并将分母初始化为2。这种做法就叫做 **拉普拉斯平滑(Laplace Smoothing)** 又被称为 **加1平滑**，是比较常用的平滑方法，它就是为了解决0概率问题。

除了这个问题之外，另一个遇到的问题是下溢出，这是由于太多很小的数相乘造成的。当计算乘积：

$$p(w_0 | c_i)p(w_1 | c_i)p(w_2 | c_i)...p(w_N | c_i)$$

由于大部分因子都非常小，所以程序会下溢出或者得到不正确的答案。（读者可以用Python尝试相乘许多很小的数，最后四舍五入后会得到0。）一种解决办法是对乘积取 **自然对数**。在代数中有 $\ln(a * b) = \ln(a) + \ln(b)$ ，于是通过求对数可以避免下溢出或者浮点数舍入导致的错误。同时，采用 **自然对数** 进行处理不会有任何损失。

下图给出函数 $f(x)$ 与 $\ln(f(x))$ 的曲线：



4 函数 $f(x)$ 与 $\ln(f(x))$ 会一块增大。这表明想求函数的最大值时，可以使用该函数的自然对数来替换原函数进行求解

检查这两条曲线，就会发现它们在相同区域内同时增加或者减少，并且在相同点上取到极值。它们的取值虽然不同，但不影响最终结果。

替换trainNB0这个函数的代码如下：

```
1  ...
2  Parameters:
3      trainMatrix - 训练文档矩阵，即setOfWords2Vec返回的returnVec构成的矩阵
4      trainCategory - 训练类别标签向量，即loadDataSet返回的classVec
5  Returns:
6      p0Vect - 侮辱类的条件概率数组
7      p1Vect - 非侮辱类的条件概率数组
8      pAbusive - 文档属于侮辱类的概率
9  ...
10 # 函数说明: 朴素贝叶斯分类器训练函数
11 def trainNB0(trainMatrix, trainCategory):
12     numTrainDocs = len(trainMatrix)          # 计算训练的文档数目
13     numWords = len(trainMatrix[0])           # 计算每篇文档的词条数
14     pAbusive = sum(trainCategory)/float(numTrainDocs) # 文档属于侮辱类的概率
15     p0Num = np.ones(numWords); p1Num = np.ones(numWords) # 创建numpy.ones数组，词条出现数初始化为1，拉普拉斯平滑
16     p0Denom = 2.0; p1Denom = 2.0             # 分母初始化为2，拉普拉斯平滑
17     for i in range(numTrainDocs):
18         if trainCategory[i] == 1: # 统计属于侮辱类的条件概率所需的数据，即P(w0|1), P(w1|1), P(w2|1) ...
19             p1Num += trainMatrix[i]
20             p1Denom += sum(trainMatrix[i])
21         else: # 统计属于非侮辱类的条件概率所需的数据，即P(w0|0), P(w1|0), P(w2|0) ...
22             p0Num += trainMatrix[i]
23             p0Denom += sum(trainMatrix[i])
24     p1Vect = np.log(p1Num/p1Denom)            # 取对数，防止下溢出
25     p0Vect = np.log(p0Num/p0Denom)
26     # 返回属于侮辱类的条件概率数组，属于非侮辱类的条件概率数组，文档属于侮辱类的概率
27     return p0Vect, p1Vect, pAbusive
```

```

myVocabList:
['posting', 'maybe', 'cute', 'ate', 'licks', 'buying', 'has', 'park', 'I'
p0V:
[-3.25809654 -3.25809654 -2.56494936 -2.56494936 -2.56494936 -3.25809654
-2.56494936 -3.25809654 -2.56494936 -3.25809654 -2.56494936 -2.56494936
-2.56494936 -3.25809654 -1.87180218 -3.25809654 -2.56494936 -3.25809654
-2.56494936 -3.25809654 -3.25809654 -2.56494936 -2.56494936 -2.56494936
-2.56494936 -2.56494936 -2.15948425 -2.56494936 -2.56494936 -2.56494936
-3.25809654 -2.56494936]
p1V:
[-2.35137526 -2.35137526 -3.04452244 -3.04452244 -3.04452244 -2.35137526
-3.04452244 -2.35137526 -3.04452244 -2.35137526 -3.04452244 -3.04452244
-3.04452244 -1.65822808 -3.04452244 -2.35137526 -3.04452244 -2.35137526
-3.04452244 -2.35137526 -1.94591015 -3.04452244 -2.35137526 -3.04452244
-2.35137526 -1.94591015 -2.35137526 -3.04452244 -3.04452244 -3.04452244
-2.35137526 -3.04452244]
classVec:
[0, 1, 0, 1, 0, 1]
pAb:
0.5

```

<https://blog.csdn.net/TeFuirnever>

没有0概率了，完美的解决了。

增加一个测试函数，对我们的分类器进行测试。

```

1  ...
2  Parameters:
3      vec2Classify - 待分类的词条数组
4      p0Vec - 侮辱类的条件概率数组
5      p1Vec - 非侮辱类的条件概率数组
6      pClass1 - 文档属于侮辱类的概率
7  Returns:
8      0 - 属于非侮辱类
9      1 - 属于侮辱类
10 ...
11 # 函数说明: 朴素贝叶斯分类器分类函数
12 def classifyNB(vec2Classify, p0Vec, p1Vec, pClass1):
13     p1 = sum(vec2Classify * p1Vec) + log(pClass1) #element-wise mult
14     p0 = sum(vec2Classify * p0Vec) + log(1.0 - pClass1)
15     if p1 > p0:
16         return 1
17     else:
18         return 0
19
20 # 函数说明: 朴素贝叶斯分类器测试函数
21 def testingNB():
22     list0Posts, listClasses = loadDataSet()
23     myVocabList = createVocabList(list0Posts)
24     trainMat = []
25     for postinDoc in list0Posts:
26         trainMat.append(setOfWords2Vec(myVocabList, postinDoc))
27     p0V, p1V, pAb = trainNB0(np.array(trainMat), np.array(listClasses))
28     testEntry = ['love', 'my', 'dalmation']
29     thisDoc = np.array(setOfWords2Vec(myVocabList, testEntry))
30     print(testEntry, 'classified as: ', classifyNB(thisDoc, p0V, p1V, pAb))
31     testEntry = ['stupid', 'garbage']
32     thisDoc = np.array(setOfWords2Vec(myVocabList, testEntry))
33     print(testEntry, 'classified as: ', classifyNB(thisDoc, p0V, p1V, pAb))
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

人眼验证一下，['love', 'my', 'dalmation']这三个确实没有侮辱性，也就是0；但是['stupid', 'garbage']这两个确实是侮辱性词汇，也就是1。

8、示例：使用朴素贝叶斯过滤垃圾邮件

在前面那个简单的例子中，我们引入了字符串列表。使用朴素贝叶斯解决一些现实生活中的问题时，需要先从文本内容得到字符串列表，然后生成词向量。下面这个例子中，我们将了解朴素贝叶斯的一个最著名的应用：**电子邮件垃圾过滤**。首先看一下如何使用通用框架来解决该问题。

- 示例：使用朴素贝叶斯对电子邮件进行分类
- (1) 收集数据：提供文本文件。
 - (2) 准备数据：将文本文件解析成词条向量。
 - (3) 分析数据：检查词条确保解析的正确性。
 - (4) 训练算法：使用我们之前建立的trainNB0()函数。
 - (5) 测试算法：使用classifyNB(), 并且构建一个新的测试函数来计算文档集的错误率。
 - (6) 使用算法：构建一个完整的程序对一组文档进行分类，将错分的文档输出到屏幕上。

1) 准备数据：切分文本

ham	2011/12/20 11:36	文件夹
spam	2011/12/20 11:36	文件夹

脑 > 文档 (E:) > 机器学习实战 > machinelearninginaction > Ch04 > email > ham				
名称	修改日期	类型	大小	
1.txt	2010/10/23 17:11	文本文档	1 KB	
2.txt	2010/10/23 8:48	文本文档	1 KB	
3.txt	2010/10/23 8:49	文本文档	1 KB	
4.txt	2010/10/23 8:50	文本文档	1 KB	
5.txt	2010/10/23 17:11	文本文档	1 KB	
6.txt	2010/10/23 17:12	文本文档	2 KB	
7.txt	2010/10/23 17:12	文本文档	1 KB	
8.txt	2010/10/23 8:58	文本文档	1 KB	
9.txt	2010/10/23 9:01	文本文档	1 KB	
10.txt	2010/10/23 17:13	文本文档	1 KB	
11.txt	2010/10/23 17:13	文本文档	1 KB	
12.txt	2010/10/23 9:16	文本文档	1 KB	
13.txt	2010/10/23 17:13	文本文档	1 KB	
14.txt	2010/10/23 17:13	文本文档	1 KB	
15.txt	2010/10/23 9:21	文本文档	1 KB	
16.txt	2010/10/23 9:21	文本文档	1 KB	
17.txt	2010/10/23 9:22	文本文档	1 KB	
18.txt	2010/10/23 9:23	文本文档	1 KB	
19.txt	2010/10/23 17:14	文本文档	1 KB	
20.txt	2010/10/23 9:26	文本文档	1 KB	
21.txt	2010/10/23 9:27	文本文档	1 KB	
22.txt	2010/10/23 9:28	文本文档	1 KB	
23.txt	2010/10/23 17:15	文本文档	1 KB	

1.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

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 <https://blog.csdn.net/TeFuirn>

名称	修改日期	类型	大小
1.txt	2010/10/23 8:28	文本文档	1 KB
2.txt	2010/10/23 8:29	文本文档	1 KB
3.txt	2010/10/23 8:30	文本文档	1 KB
4.txt	2010/10/23 8:30	文本文档	1 KB
5.txt	2010/10/23 8:31	文本文档	1 KB
6.txt	2010/10/23 8:32	文本文档	1 KB
7.txt	2010/10/23 8:32	文本文档	1 KB
8.txt	2010/10/23 8:33	文本文档	1 KB
9.txt	2010/10/23 8:34	文本文档	1 KB
10.txt	2010/10/23 8:36	文本文档	1 KB
11.txt	2010/10/23 8:37	文本文档	1 KB
12.txt	2010/10/23 8:37	文本文档	1 KB
13.txt	2010/10/23 8:38	文本文档	1 KB
14.txt	2010/10/23 8:38	文本文档	1 KB
15.txt	2010/10/23 8:39	文本文档	1 KB
16.txt	2010/10/23 8:40	文本文档	1 KB
17.txt	2010/10/23 8:41	文本文档	1 KB
18.txt	2010/10/23 8:41	文本文档	1 KB
19.txt	2010/10/23 8:42	文本文档	1 KB
20.txt	2010/10/23 8:43	文本文档	1 KB
21.txt	2010/10/23 8:43	文本文档	1 KB
22.txt	2010/10/23 8:43	文本文档	1 KB
23.txt	2010/10/23 8:44	文本文档	1 KB

```

1.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
--- 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!!! ---

```

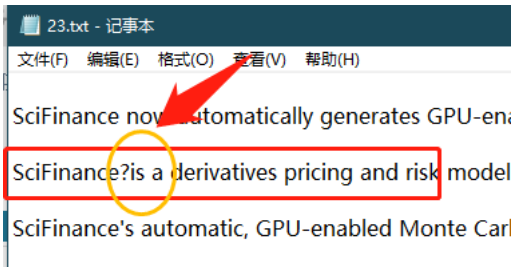
对于英文文本，我们可以以非字母、非数字作为符号进行切分，使用split函数即可。编写代码如下：

```

1  # -*- coding: UTF-8 -*-
2  import re
3
4  # 函数说明: 接收一个大字符串并将其解析为字符串列表
5  def textParse(bigString):                                #将字符串转换为字符串列表
6      #将特殊符号作为切分标志进行字符串切分, 即非字母、非数字
7      listOfTokens = re.split(r'\W*', bigString)
8      return [tok.lower() for tok in listOfTokens if len(tok) > 2]#除了单个字母, 例如大写的H, 其它单词变成小写
9
10 ...
11 Parameters:
12     dataSet - 整理的样本数据集
13 Returns:
14     vocabSet - 返回不重复的词条列表, 也就是词汇表
15
16 # 函数说明: 将切分的实验样本词条整理成不重复的词条列表, 也就是词汇表
17 def createVocabList(dataSet):
18     vocabSet = set([])                                     #创建一个空的不重复列表
19     for document in dataSet:
20         vocabSet = vocabSet | set(document) #取并集
21     return list(vocabSet)
22
23 if __name__ == '__main__':
24     docList = []; classList = []
25     for i in range(1, 26):                                #遍历25个txt文件
26         wordList = textParse(open('email/spam/%d.txt' % i, 'r').read())#读取每个垃圾邮件, 并字符串转换成字符串列表
27         docList.append(wordList)
28         classList.append(1)                                #标记垃圾邮件, 1表示垃圾文件
29         wordList = textParse(open('email/ham/%d.txt' % i, 'r').read())#读取每个非垃圾邮件, 并字符串转换成字符串列表
30         docList.append(wordList)
31         classList.append(0)                                #标记非垃圾邮件, 1表示垃圾文件
32
33     ...
34     ...
35     ...
36     ...
37     ...
38     ...
39     ...
40     ...
41     ...
42     ...
43     ...
44     ...
45     ...
46     ...
47     ...
48     ...
49     ...
50     ...
51     ...
52     ...
53     ...
54     ...
55     ...
56     ...
57     ...
58     ...
59     ...
60     ...
61     ...
62     ...
63     ...
64     ...
65     ...
66     ...
67     ...
68     ...
69     ...
70     ...
71     ...
72     ...
73     ...
74     ...
75     ...
76     ...
77     ...
78     ...
79     ...
80     ...
81     ...
82     ...
83     ...
84     ...
85     ...
86     ...
87     ...
88     ...
89     ...
90     ...
91     ...
92     ...
93     ...
94     ...
95     ...
96     ...
97     ...
98     ...
99     ...
100    ...

```


直接运行程序会出现报错的情况, `UnicodeDecodeError: 'gbk' codec can't decode byte 0xae in position 199: illegal multibyte`, 这时候发现报错位置是: `wordList = textParse(open('./email/ham/%d.txt' % i, 'r').read())`, 是文件读取有问题, 这个时候打开相应的文件, 发现是这个问号的问题, 先删除再输入即可解决错误。



这样就得到了词汇表, 结果如下:

```
>>>
['email', 'assistance', 'treat', 'forward', 'codeine', 'edit', 'wallets', 'looking', '396', 'this', 'had', 'storedetailview_98',
```

2) 测试算法: 使用朴素贝叶斯进行交叉验证

根据词汇表就可以将每个文本向量化。首先将数据集分为训练集和测试集, 使用 **交叉验证** 的方式测试朴素贝叶斯分类器的准确性。编写代码如下:

```
1 import numpy as np
2 import random
3 import re
4
5 ...
6 Parameters:
7     dataSet - 整理的样本数据集
8 Returns:
9     vocabSet - 返回不重复的词条列表, 也就是词汇表
10 # 函数说明: 将切分的实验样本词条整理成不重复的词条列表, 也就是词汇表
11 def createVocabList(dataSet):
12     vocabSet = set([]) # 创建一个空的不重复列表
13     for document in dataSet:
14         vocabSet = vocabSet | set(document) # 取并集
15     return list(vocabSet)
16
17 ...
18 Parameters:
19     vocabList - createVocabList返回的列表
20     inputSet - 切分的词条列表
21 Returns:
22     returnVec - 文档向量, 词集模型
23 ...
24 # 函数说明: 根据vocabList词汇表, 将inputSet向量化, 向量的每个元素为1或0
25 def setOfWords2Vec(vocabList, inputSet):
26     returnVec = [0] * len(vocabList) # 创建一个其中所含元素都为0的向量
27     for word in inputSet: # 遍历每个词条
28         if word in vocabList: # 如果词条存在于词汇表中, 则置1
29             returnVec[vocabList.index(word)] = 1
30         else: print("the word: %s is not in my Vocabulary!" % word)
31     return returnVec # 返回文档向量
32
33 ...
34 Parameters:
35     vocabList - createVocabList返回的列表
36     inputSet - 切分的词条列表
37 Returns:
38     returnVec - 文档向量, 词袋模型
39 ...
40 # 函数说明: 根据vocabList词汇表, 构建词袋模型
41 def bagOfWords2VecMN(vocabList, inputSet):
42     returnVec = [0] * len(vocabList) # 创建一个其中所含元素都为0的向量
43     for word in inputSet: # 遍历每个词条
```

```

41         if word in vocabList:                                #如果词条存在于词汇表中, 则计数加一
42             returnVec[vocabList.index(word)] += 1
43     return returnVec                                         #返回词袋模型
44
45     '''
46 Parameters:
47     trainMatrix - 训练文档矩阵, 即setOfWords2Vec返回的returnVec构成的矩阵
48     trainCategory - 训练类别标签向量, 即loadDataSet返回的classVec
49 Returns:
50     p0Vect - 侮辱类的条件概率数组
51     p1Vect - 非侮辱类的条件概率数组
52     pAbusive - 文档属于侮辱类的概率
53     ...
54 # 函数说明: 朴素贝叶斯分类器训练函数
55 def trainNB0(trainMatrix, trainCategory):
56     numTrainDocs = len(trainMatrix)                         #计算训练的文档数目
57     numWords = len(trainMatrix[0])                          #计算每篇文档的词条数
58     pAbusive = sum(trainCategory)/float(numTrainDocs)       #文档属于侮辱类的概率
59     p0Num = np.ones(numWords); p1Num = np.ones(numWords)   #创建numpy.ones数组, 词条出现数初始化为1, 拉普拉斯平滑
60     p0Denom = 2.0; p1Denom = 2.0                           #分母初始化为2, 拉普拉斯平滑
61     for i in range(numTrainDocs):
62         if trainCategory[i] == 1:                            #统计属于侮辱类的条件概率所需的数据,
63             #即P(w0|1), P(w1|1), P(w2|
64             p1Num += trainMatrix[i]
65             p1Denom += sum(trainMatrix[i])
66         else:                                                  #统计属于非侮辱类的条件概率所需的数据,
67             #即P(w0|0), P(w1|0), P(w2|
68             p0Num += trainMatrix[i]
69             p0Denom += sum(trainMatrix[i])
70     p1Vect = np.log(p1Num/p1Denom)                           #取对数, 防止下溢出
71     p0Vect = np.log(p0Num/p0Denom)
72     return p0Vect, p1Vect, pAbusive                          #返回属于侮辱类的条件概率数组, 属于非侮辱类的条件概率数组, 文档属于侮辱类的概率
73
74     '''
75 Parameters:
76     vec2Classify - 待分类的词条数组
77     p0Vec - 侮辱类的条件概率数组
78     p1Vec - 非侮辱类的条件概率数组
79     pClass1 - 文档属于侮辱类的概率
80 Returns:
81     0 - 属于非侮辱类
82     1 - 属于侮辱类
83     ...
84 # 函数说明: 朴素贝叶斯分类器分类函数
85 def classifyNB(vec2Classify, p0Vec, p1Vec, pClass1):
86     p1 = sum(vec2Classify * p1Vec) + np.log(pClass1)         #对应元素相乘。LogA * B = LogA + LogB,
87                                                                #所以这里加上Log(
88     p0 = sum(vec2Classify * p0Vec) + np.log(1.0 - pClass1)
89     if p1 > p0:
90         return 1
91     else:
92         return 0
93
94     '''
95 Parameters:
96     trainMatrix - 训练文档矩阵, 即setOfWords2Vec返回的returnVec构成的矩阵
97     trainCategory - 训练类别标签向量, 即loadDataSet返回的classVec
98 Returns:
99     p0Vect - 侮辱类的条件概率数组
100    p1Vect - 非侮辱类的条件概率数组
101    pAbusive - 文档属于侮辱类的概率
102    ...
103 # 函数说明: 朴素贝叶斯分类器训练函数
104 def trainNB0(trainMatrix, trainCategory):
105     numTrainDocs = len(trainMatrix)                         #计算训练的文档数目
106     numWords = len(trainMatrix[0])                          #计算每篇文档的词条数
107     pAbusive = sum(trainCategory)/float(numTrainDocs)       #文档属于侮辱类的概率
108     p0Num = np.ones(numWords); p1Num = np.ones(numWords)   #创建numpy.ones数组, 词条出现数初始化为1, 拉普拉斯平滑
109     p0Denom = 2.0; p1Denom = 2.0                           #分母初始化为2, 拉普拉斯平滑
110     for i in range(numTrainDocs):
111         if trainCategory[i] == 1:                            #统计属于侮辱类的条件概率所需的数据,
112             #即P(w0|1), P(w1|1), P(w2|
113             p1Num += trainMatrix[i]
114             p1Denom += sum(trainMatrix[i])
115         else:                                                  #统计属于非侮辱类的条件概率所需的数据,
116             #即P(w0|0), P(w1|0), P(w2|

```

```

112         p0Num += trainMatrix[i]
113         p0Denom += sum(trainMatrix[i])
114         p1Vect = np.log(p1Num/p1Denom)          #取对数，防止下溢出
115         p0Vect = np.log(p0Num/p0Denom)
116         return p0Vect, p1Vect, pAbusive #返回属于侮辱类的条件概率数组，属于非侮辱类的条件概率数组，文档属于侮辱类的概率
117
118 # 函数说明: 接收一个大字符串并将其解析为字符串列表
119 def textParse(bigString):                      #将字符串转换为字符串列表
120     listOfTokens = re.split(r'\W+', bigString) #将特殊符号作为切分标志进行字符串切分，即非字母、非数字
121     return [tok.lower() for tok in listOfTokens if len(tok) > 2] #除了单个字母，例如大写的I，其它单词变成小写
122
123 # 函数说明: 测试朴素贝叶斯分类器
124 def spamTest():
125     docList = []; classList = []; fullText = []
126     for i in range(1, 26):                    #遍历25个txt文件
127         wordList = textParse(open('email/spam/%d.txt' % i, 'r').read()) #读取每个垃圾邮件，并字符串转换成字符串列表
128         docList.append(wordList)
129         fullText.append(wordList)
130         classList.append(1)                   #标记垃圾邮件，1表示垃圾文件
131         wordList = textParse(open('email/ham/%d.txt' % i, 'r').read()) #读取每个非垃圾邮件，并字符串转换成字符串列表
132         docList.append(wordList)
133         fullText.append(wordList)
134         classList.append(0)                   #标记非垃圾邮件，1表示垃圾文件
135     vocabList = createVocabList(docList)        #创建词汇表，不重复
136     trainingSet = list(range(50)); testSet = [] #创建存储训练集的索引值的列表和测试集的索引值的列表
137     for i in range(10): #从50个邮件中，随机挑选出40个作为训练集，10个做测试集
138         randIndex = int(random.uniform(0, len(trainingSet))) #随机选取索引值
139         testSet.append(trainingSet[randIndex])                #添加测试集的索引值
140         del(trainingSet[randIndex])                            #在训练集列表中删除添加到测试集的索引值
141     trainMat = []; trainClasses = []                          #创建训练集矩阵和训练集类别标签系向量
142     for docIndex in trainingSet:                               #遍历训练集
143         trainMat.append(setOfWords2Vec(vocabList, docList[docIndex])) #将生成的词集模型添加到训练矩阵中
144         trainClasses.append(classList[docIndex])               #将类别添加到训练集类别标签系向量中
145     p0V, p1V, pSpam = trainNB0(np.array(trainMat), np.array(trainClasses)) #训练朴素贝叶斯模型
146     errorCount = 0                                             #错误分类计数
147     for docIndex in testSet:                                   #遍历测试集
148         wordVector = setOfWords2Vec(vocabList, docList[docIndex]) #测试集的词集模型
149         if classifyNB(np.array(wordVector), p0V, p1V, pSpam) != classList[docIndex]: #如果分类错误
150             errorCount += 1                                     #错误计数加1
151         print("分类错误的测试集: ", docList[docIndex])
152     print('错误率: %.2f%%' % (float(errorCount) / len(testSet) * 100))
153
154
155 if __name__ == '__main__':
156     spamTest()
157

```

```

1 >>>
2 分类错误的测试集: ['scifinance', 'now', 'automatically', 'generates', 'gpu', 'enabled', 'pricing', 'risk', 'model', 'source', 'c
3 分类错误的测试集: ['home', 'based', 'business', 'opportunity', 'knocking', 'your', 'door', 'don抰', 'rude', 'and', 'let', 'this',
4 分类错误的测试集: ['benoit', 'mandelbrot', '1924', '2010', 'benoit', 'mandelbrot', '1924', '2010', 'wilmott', 'team', 'benoit',
5 错误率: 30.00%

```

函数spamTest()会输出在10封随机选择的电子邮件上的分类错误概率。所以存在误判的情况，将垃圾邮件误判为正常邮件要比将正常邮件归为垃圾邮件好。为了避免错误，有多种方式可以用来修正分类器，这些内容会在后续文章中进行讨论。

第九章转载自 <https://blog.csdn.net/c406495762/article/details/77500679>

9、Sklearn构建朴素贝叶斯分类器用于新浪新闻分类

1) 中文语句切分

前面我们说，英文的语句可以通过非字母和非数字进行切分，但是汉语句子呢，该如何进行切分呢？

答案是可以直接使用第三方分词组件，即jieba。

新闻分类数据集下载地址: <https://github.com/Jack-Cherish/Machine-Learning/tree/master/Naive%20Bayes/SogouC>

数据集已经做好分类, 分文件夹保存, 分类结果如下:

```
C000008 财经
C000010 IT
C000013 健康
C000014 体育
C000016 旅游
C000020 教育
C000022 招聘
C000023 文化
C000024 军事
```

切分中文语句, 编写如下代码:

```
1  # -*- coding: UTF-8 -*-
2  import os
3  import jieba
4
5  def TextProcessing(folder_path):
6      folder_list = os.listdir(folder_path)          #查看folder_path下的文件
7      data_list = []                                #训练集
8      class_list = []
9
10     #遍历每个子文件夹
11     for folder in folder_list:
12         new_folder_path = os.path.join(folder_path, folder) #根据子文件夹, 生成新的路径
13         files = os.listdir(new_folder_path)             #存放子文件夹下的txt文件的列表
14
15         j = 1
16         #遍历每个txt文件
17         for file in files:
18             if j > 100:                                #每类txt样本数最多100个
19                 break
20             with open(os.path.join(new_folder_path, file), 'r', encoding = 'utf-8') as f: #打开txt文件
21                 raw = f.read()
22
23                 word_cut = jieba.cut(raw, cut_all = False) #精简模式, 返回一个可迭代的generator
24                 word_list = list(word_cut)                #generator转换为list
25
26                 data_list.append(word_list)
27                 class_list.append(folder)
28                 j += 1
29         print(data_list)
30         print(class_list)
31
32
33 if __name__ == '__main__':
34     #文本预处理
35     folder_path = './SogouC/Sample'                    #训练集存放地址
36     TextProcessing(folder_path)
```


这些标点符号是不能作为新闻分类的特征的。为了降低这些高频的符号对分类结果的影响，需要放弃这些符号，除了这些，还有“在”，“了”这样对新闻分类无关痛痒的词，并且还有一些数字，数字显然也不能作为分类新闻的特征。

所以要消除它们对分类结果的影响，可以定制一个规则：首先去掉高频词，至于去掉多少个高频词，可以通过观察去掉高频词个数和最终检测准确率的关系来确定。除此之外，去除数字，不把数字作为分类特征。同时，去除一些特定的词语，比如：“的”，“一”，“在”，“不”，“当然”，“怎么”这类的对新闻分类无影响的介词、代词、连词。怎么去除这些词呢？可以使用整理好的stopwords_cn.txt文本。

下载地址：https://github.com/Jack-Cherish/Machine-Learning/blob/master/Naive%20Bayes/stopwords_cn.txt

文件内容如下：

```
37 起
38 最
39 再
40 今
41 去
42 好
43 只
44 又
45 或
46 很
47 亦
48 某
49 把
50 那
51 你
52 乃
53 它
54 怎么
55 任何
56 连同
57 开外
58 再有
59 哪些
60 甚至于
61 又及
62 当然
63 就是
64 遵照
65 以来
66 赖以
67 否则
68 此间
69 后者
70 按照
71 才是
72 自身
73 再则
74 就算
75 即便
76 有些
77 例如
78 它们
79 虽然
80 为此
81 以免
82 别处
```

所以可以根据文档去除单词。我们先去除前100个高频词汇，然后编写代码如下：

```
1 # -*- coding: UTF-8 -*-
2 import os
3 import random
4 import jieba
5
6 """
7 函数说明:中文文本处理
8
9 Parameters:
10     folder_path - 文本存放的路径
11     test_size - 测试集占比，默认占所有数据集的百分之20
12 Returns:
13     all_words_list - 按词频降序排序的训练集列表
14     train_data_list - 训练集列表
15     test_data_list - 测试集列表
```

```

14     train_class_list - 训练集标签列表
15     test_class_list - 测试集标签列表
16 """
17 def TextProcessing(folder_path, test_size = 0.2):
18     folder_list = os.listdir(folder_path)                #查看folder_path下的文件
19     data_list = []                                       #数据集数据
20     class_list = []                                     #数据集类别
21
22     #遍历每个子文件夹
23     for folder in folder_list:
24         new_folder_path = os.path.join(folder_path, folder)    #根据子文件夹,生成新的路径
25         files = os.listdir(new_folder_path)                  #存放子文件夹下的txt文件的列表
26
27         j = 1
28         #遍历每个txt文件
29         for file in files:
30             if j > 100:                                       #每类txt样本数最多100个
31                 break
32             with open(os.path.join(new_folder_path, file), 'r', encoding = 'utf-8') as f:    #打开txt文件
33                 raw = f.read()
34
35             word_cut = jieba.cut(raw, cut_all = False)        #精简模式, 返回一个可迭代的generator
36             word_list = list(word_cut)                        #generator转换为list
37
38             data_list.append(word_list)                       #添加数据集数据
39             class_list.append(folder)                         #添加数据集类别
40             j += 1
41
42     data_class_list = list(zip(data_list, class_list))         #zip压缩合并, 将数据与标签对应压缩
43     random.shuffle(data_class_list)                           #将data_class_list乱序
44     index = int(len(data_class_list) * test_size) + 1        #训练集和测试集切分的索引值
45     train_list = data_class_list[:index]                     #训练集
46     test_list = data_class_list[index:]                      #测试集
47     train_data_list, train_class_list = zip(*train_list)     #训练集解压缩
48     test_data_list, test_class_list = zip(*test_list)        #测试集解压缩
49
50     all_words_dict = {}                                       #统计训练集词频
51     for word_list in train_data_list:
52         for word in word_list:
53             if word in all_words_dict.keys():
54                 all_words_dict[word] += 1
55             else:
56                 all_words_dict[word] = 1
57
58     #根据键的值倒序排序
59     all_words_tuple_list = sorted(all_words_dict.items(), key = lambda f:f[1], reverse = True)
60     all_words_list, all_words_nums = zip(*all_words_tuple_list) #解压缩
61     all_words_list = list(all_words_list)                    #转换成列表
62     return all_words_list, train_data_list, test_data_list, train_class_list, test_class_list
63
64 """
65 函数说明:读取文件里的内容, 并去重
66
67 Parameters:
68     words_file - 文件路径
69 Returns:
70     words_set - 读取的内容的set集合
71 """
72 def MakeWordsSet(words_file):
73     words_set = set()                                       #创建set集合
74     with open(words_file, 'r', encoding = 'utf-8') as f:    #打开文件
75         for line in f.readlines():                          #一行一行读取
76             word = line.strip()                             #去回车
77             if len(word) > 0:                                #有文本, 则添加到words_set中
78                 words_set.add(word)
79     return words_set                                         #返回处理结果
80
81 """
82 函数说明:文本特征选取
83
84 Parameters:
85     all_words_list - 训练集所有文本列表
86     deleteN - 删除词频最高的deleteN个词
87     stopwords_set - 指定的结束语
88 Returns:

```

```

85     feature_words - 特征集
86     """
87     def words_dict(all_words_list, deleteN, stopwords_set = set()):
88         feature_words = [] #特征列表
89         n = 1
90         for t in range(deleteN, len(all_words_list), 1):
91             if n > 1000: #feature_words的维度为1000
92                 break
93             #如果这个词不是数字, 并且不是指定的结束语, 并且单词长度大于1小于5, 那么这个词就可以作为特征词
94             if not all_words_list[t].isdigit() and all_words_list[t] not in stopwords_set and 1 < len(all_words_list[t]) < 5:
95                 feature_words.append(all_words_list[t])
96             n += 1
97         return feature_words
98
99     if __name__ == '__main__':
100         #文本预处理
101         folder_path = './SogouC/Sample' #训练集存放地址
102         all_words_list, train_data_list, test_data_list, train_class_list, test_class_list = TextProcessing(folder_path, test_size=0.1)
103
104         #生成stopwords_set
105         stopwords_file = './stopwords_cn.txt'
106         stopwords_set = MakeWordsSet(stopwords_file)
107
108         feature_words = words_dict(all_words_list, 100, stopwords_set)
109         print(feature_words)
110

```

运行结果如下：

```

187 feature_words = words_dict(all_words_list, 100, stopwords_set)
188 print(feature_words)

Building prefix dict from D:\Python3.4.4-32bit\lib\site-packages\jieba\dictionary.txt ...
Loading model from cache C:\Users\ADMINI~1\AppData\Local\Temp\jieba.cache
Prefix dict cost 1.0470600128173828 seconds.
Prefix dict has been built successfully.
['作战', '复习', '仿制', '支付', '很多', '工作', '选择', '可能', '一定', '远程', '比赛', '分析', '问题', '主要', '学校', '时候', '射程', '成为', '目前', '基础', '亿美元',
'考试', '词汇', '通过', '编辑', '部署', '部分', '完全', '文章', '技术', '能力', '增长', '专业', '毕业生', '学习', '黄金周', '品牌', '银行', '使用', '填报', '需要',
'vs', '接待', '降地', '影响', '用户', '训练', '拥有', '坦克', '重要', '达到', '资料', '万人', '次', '销售', '要求', '收入', '几乎', '表现', '计划', '新浪', '开始', '阅读',
'军事', '发展', '期间', '相对', '上海', '五', '设计', '提供', '历史', '网络', '提高', '管理', '比较', '老师', '英语', '写作', '重点', '最后', '今年', '我国', '专利',
'MBA', '实验室', '参加', '公里', '考研', '希望', '不用', '压制', '情况', '方面', '大批', '一直', '注意', '来源', '岛屿', '耿大勇', '阿里', '游戏', '电话', '距离',
'必须', '员工', '相关', '手机', '显示', '机会', '指数', '活动', '服务', '睡眠', '不会', '力气', '海底', '沿海', '专家', '一次', '国内', '人数', '角度', '包括', '产品', '成功',
'协议', '找替', '作用', '最大', '进入', '机台', '不同', '摧毁', '准备', '单独', '数字', '海基', '指挥', '平台', '这种', '众多', '平壤', '纳斯', '金贵', '开赛', '装备',
'现在', '自寻死路', '之内', '行业', '型号', '招聘', '世界', '看到', '目标', '牛奶', '排名', '药', '香港', '预期', '人才', '战场', '著名', '新型', '掌握', '一家',
'获得', '沈阳市', '之间', '未来', '东引号', '应该', '一下', '出现', '止痛膏', '录取', '了解', '全国', '建议', '决定', '系统', '第一', '顾客', '容易', '结果', '连续', '非常',
'理解', '整个', '电脑', '知识点', '两个', '利用', '发现', '得到', '语法', '完成', '句子', '概念', '组织', '教育', '左右', '辽宁', '队', '每个', '理由', '能够', '普救区', '交易', '网',
'考虑', '分钟', '内容', '镇铺药', '愿意', '之后', '本场', '建设', '知识', '对手', '方员', '消息', '组织', '教育', '左右', '辽宁', '队', '每个', '理由', '能够', '普救区', '交易', '网',
'业务', '超过', '调查', '不要', '詹姆斯', '上市', '努力', '数据', '其实', '之前', '学员', '住往', '关键', '思路', '原因', '此前', '王治郅', '客户', '是否', '代表', '公布', '事情',
'利苑', '补充', '相当', '过程', '詹姆斯', '产生', '方式', '领域', '大量', '之', '指出', '小小时', '特点', '万元', '伯德', '练习', '支持', '年', '环境', '每天', '蓝军', '基本', '老', '回家', '解題', '全球', '过年', '东莞', '职业',
'人力', '数量', '各型', '喜欢', '增加', '知名', '更', '加', '再次', '这场', '家', '的', '实现', '到', '支持', '年', '环境', '每天', '蓝军', '基本', '老', '回家', '解題', '全球', '过年', '东莞', '职业',
'一批', '全面', '国际', '姚明', '美国', '光', '更加', '复试', '再', '次', '介绍', '战斗', '小小时', '特点', '万元', '伯德', '练习', '支持', '年', '环境', '每天', '蓝军', '基本', '老', '回家', '解題', '全球', '过年', '东莞', '职业',
'公式', '大眼', '每股', '口技', '告诉', '下套', '进攻', '集团', '同学', '电子', '标题', '最近', '食物', '日本', '实施', '孩子', '功能', '教材', '项目', '加', '清', '空',
'不断', '同事', '信息', '世界', '运动', '本数', '批发', '效果', '客场', '不足', '翻译', '上午', '最好', '吸引', '传统', 'cup', 'hubs', '自动', 'gunner', 'ref', 'forever']

```

可以看到，已经滤除了那些没有用的词组，这个feature_words就是最终选出的用于新闻分类的特征。随后就可以根据feature_words，将文本向量化，然后用于训练朴素贝叶斯分类器。

通过观察取不同的去掉前deleteN个高频词的个数与最终检测准确率的关系, 确定deleteN的取值:

```

1 # -*- coding: UTF-8 -*-
2 from sklearn.naive_bayes import MultinomialNB
3 import matplotlib.pyplot as plt
4 import os
5 import random
6 import jieba
7
8 """
9 函数说明:中文文本处理
10
11 Parameters:
12     folder_path - 文本存放的路径
13     test_size - 测试集占比,默认占所有数据集的百分之20
14 Returns:
15     all_words_list - 按词频降序排序的训练集列表
16     train_data_list - 训练集列表
17     test_data_list - 测试集列表
18     train_class_list - 训练集标签列表
19     test_class_list - 测试集标签列表
20 """
21
22 def TextProcessing(folder_path, test_size = 0.2):
23     folder_list = os.listdir(folder_path)
24     data_list = []
25
26     #查看folder_path下的文件
27     #数据集数据

```

```

21 class_list = [] #数据集类别
22
23 #遍历每个子文件夹
24 for folder in folder_list:
25     new_folder_path = os.path.join(folder_path, folder) #根据子文件夹, 生成新的路径
26     files = os.listdir(new_folder_path) #存放子文件夹下的txt文件的列表
27
28     j = 1
29     #遍历每个txt文件
30     for file in files:
31         if j > 100: #每类txt样本数最多100个
32             break
33         with open(os.path.join(new_folder_path, file), 'r', encoding = 'utf-8') as f: #打开txt文件
34             raw = f.read()
35
36             word_cut = jieba.cut(raw, cut_all = False) #精简模式, 返回一个可迭代的generator
37             word_list = list(word_cut) #generator转换为List
38
39             data_list.append(word_list) #添加数据集数据
40             class_list.append(folder) #添加数据集类别
41             j += 1
42
43     data_class_list = list(zip(data_list, class_list)) #zip压缩合并, 将数据与标签对应压缩
44     random.shuffle(data_class_list) #将data_class_list乱序
45     index = int(len(data_class_list) * test_size) + 1 #训练集和测试集切分的索引值
46     train_list = data_class_list[index:] #训练集
47     test_list = data_class_list[:index] #测试集
48     train_data_list, train_class_list = zip(*train_list) #训练集解压缩
49     test_data_list, test_class_list = zip(*test_list) #测试集解压缩
50
51     all_words_dict = {} #统计训练集词频
52     for word_list in train_data_list:
53         for word in word_list:
54             if word in all_words_dict.keys():
55                 all_words_dict[word] += 1
56             else:
57                 all_words_dict[word] = 1
58
59     #根据键的值倒序排序
60     all_words_tuple_list = sorted(all_words_dict.items(), key = lambda f:f[1], reverse = True)
61     all_words_list, all_words_nums = zip(*all_words_tuple_list) #解压缩
62     all_words_list = list(all_words_list) #转换成列表
63     return all_words_list, train_data_list, test_data_list, train_class_list, test_class_list
64
65 """
66 函数说明:读取文件里的内容, 并去重
67
68 Parameters:
69     words_file - 文件路径
70 Returns:
71     words_set - 读取的内容的set集合
72 """
73 def MakeWordsSet(words_file):
74     words_set = set() #创建set集合
75     with open(words_file, 'r', encoding = 'utf-8') as f: #打开文件
76         for line in f.readlines(): #一行一行读取
77             word = line.strip() #去回车
78             if len(word) > 0: #有文本, 则添加到words_set中
79                 words_set.add(word)
80     return words_set #返回处理结果
81
82 """
83 函数说明:根据feature_words将文本向量化
84
85 Parameters:
86     train_data_list - 训练集
87     test_data_list - 测试集
88     feature_words - 特征集
89 Returns:
90     train_feature_list - 训练集向量化列表
91     test_feature_list - 测试集向量化列表
92 """
93 def TextFeatures(train_data_list, test_data_list, feature_words):
94     def text_features(text, feature_words): #出现在特征集中, 则置1
95         text_words = set(text)

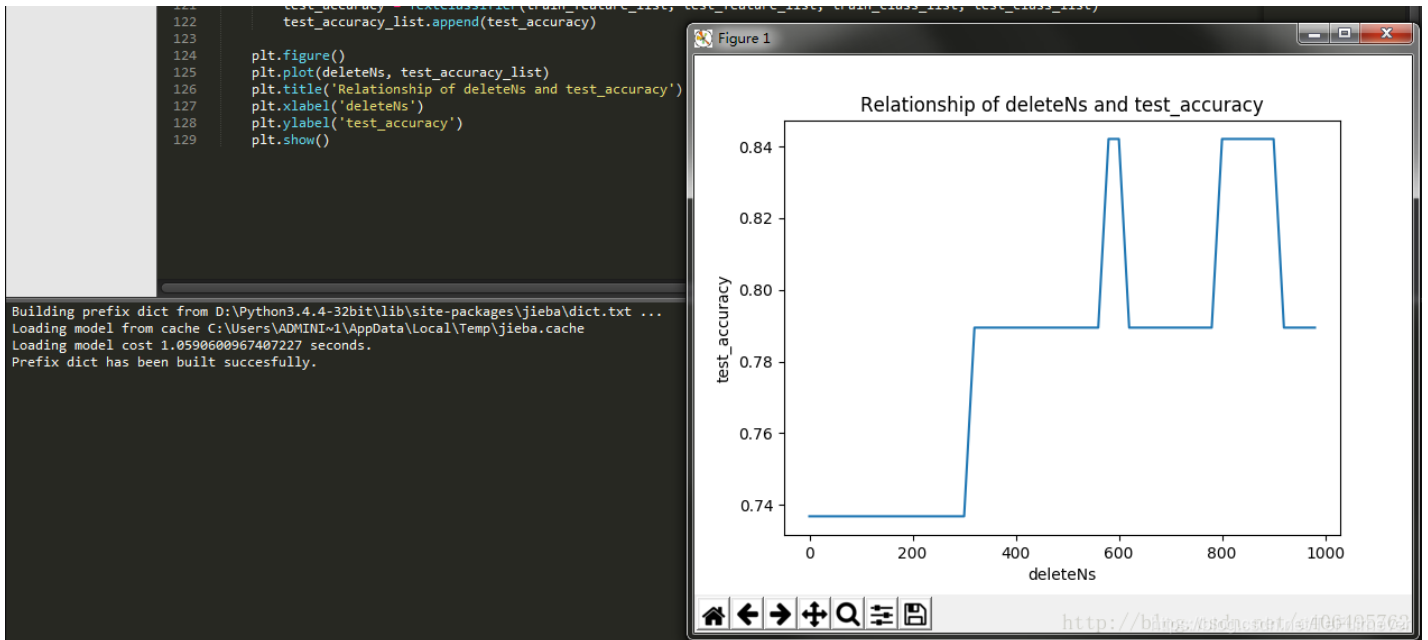
```

```

92     features = [1 if word in text_words else 0 for word in feature_words]
93     return features
94 train_feature_list = [text_features(text, feature_words) for text in train_data_list]
95 test_feature_list = [text_features(text, feature_words) for text in test_data_list]
96 return train_feature_list, test_feature_list          #返回结果
97
98
99 """
100 函数说明:文本特征选取
101 Parameters:
102     all_words_list - 训练集所有文本列表
103     deleteN - 删除词频最高的deleteN个词
104     stopwords_set - 指定的结束语
105 Returns:
106     feature_words - 特征集
107 """
108 def words_dict(all_words_list, deleteN, stopwords_set = set()):
109     feature_words = []          #特征列表
110     n = 1
111     for t in range(deleteN, len(all_words_list), 1):
112         if n > 1000:            #feature_words的维度为1000
113             break
114         #如果这个词不是数字, 并且不是指定的结束语, 并且单词长度大于1小于5, 那么这个词就可以作为特征词
115         if not all_words_list[t].isdigit() and all_words_list[t] not in stopwords_set and 1 < len(all_words_list[t]) < 5:
116             feature_words.append(all_words_list[t])
117         n += 1
118     return feature_words
119
120 """
121 函数说明:新闻分类器
122 Parameters:
123     train_feature_list - 训练集向量化的特征文本
124     test_feature_list - 测试集向量化的特征文本
125     train_class_list - 训练集分类标签
126     test_class_list - 测试集分类标签
127 Returns:
128     test_accuracy - 分类器精度
129 """
130 def TextClassifier(train_feature_list, test_feature_list, train_class_list, test_class_list):
131     classifier = MultinomialNB().fit(train_feature_list, train_class_list)
132     test_accuracy = classifier.score(test_feature_list, test_class_list)
133     return test_accuracy
134
135 if __name__ == '__main__':
136     #文本预处理
137     folder_path = './SogouC/Sample'          #训练集存放地址
138     all_words_list, train_data_list, test_data_list, train_class_list, test_class_list = TextProcessing(folder_path, test_size=0.2)
139
140     # 生成stopwords_set
141     stopwords_file = './stopwords_cn.txt'
142     stopwords_set = MakeWordsSet(stopwords_file)
143
144     test_accuracy_list = []
145     deleteNs = range(0, 1000, 20)            #0 20 40 60 ... 980
146     for deleteN in deleteNs:
147         feature_words = words_dict(all_words_list, deleteN, stopwords_set)
148         train_feature_list, test_feature_list = TextFeatures(train_data_list, test_data_list, feature_words)
149         test_accuracy = TextClassifier(train_feature_list, test_feature_list, train_class_list, test_class_list)
150         test_accuracy_list.append(test_accuracy)
151
152     plt.figure()
153     plt.plot(deleteNs, test_accuracy_list)
154     plt.title('Relationship of deleteNs and test_accuracy')
155     plt.xlabel('deleteNs')
156     plt.ylabel('test_accuracy')
157     plt.show()

```

运行结果如下:



绘制出deleteNs和test_accuracy的关系，这样就可以大致确定去掉前多少的高频词汇了。每次运行程序，绘制的图形可能不尽相同，可以通过多次测试，来决定这个deleteN的取值，然后确定这个参数，这样就可以顺利构建出用于新闻分类的朴素贝叶斯分类器了。

我测试感觉450还不错，最差的分类准确率也可以达到百分之50以上。将 `if __name__ == '__main__':` 下的代码修改如下：

```
1 if __name__ == '__main__':
2     # 文本预处理
3     folder_path = './SogouC/Sample' # 训练集存放地址
4     all_words_list, train_data_list, test_data_list, train_class_list, test_class_list = TextProcessing(folder_path, test_size=0.2)
5
6     # 生成stopwords_set
7     stopwords_file = './stopwords_cn.txt'
8     stopwords_set = MakeWordsSet(stopwords_file)
9
10    test_accuracy_list = []
11    feature_words = words_dict(all_words_list, 450, stopwords_set)
12    train_feature_list, test_feature_list = TextFeatures(train_data_list, test_data_list, feature_words)
13    test_accuracy = TextClassifier(train_feature_list, test_feature_list, train_class_list, test_class_list)
14    test_accuracy_list.append(test_accuracy)
15    ave = lambda c: sum(c) / len(c)
```

运行结果：

```
186 feature_words = words_dict(all_words_list, 450, stopwords_set)
187 train_feature_list, test_feature_list = TextFeatures(train_data_list, test_data_list, feature_words)
188 test_accuracy = TextClassifier(train_feature_list, test_feature_list, train_class_list, test_class_list)
189 test_accuracy_list.append(test_accuracy)
190 ave = lambda c: sum(c) / len(c)
191
192 print(ave(test_accuracy_list))
193
```

Building prefix dict from D:\Python3.4.4-32bit\lib\site-packages\jieba\dict.txt ...
Loading model from cache C:\Users\ADMINI~1\AppData\Local\Temp\jieba.cache
Loading model cost 1.061061143875122 seconds.
Prefix dict has been built successfully.
0.736842105263
[Finished in 3.0s]

<https://blog.csdn.net/TeFurnever>

10、sklearn.naive_bayes.MultinomialNB

sklearn.naive_bayes.MultinomialNB是一个很好的模型，决策树算法就是通过它实现的，详细的看这个博客——[sklearn.naive_bayes.MultinomialNB\(\)函数解析（最清晰的解释）](#)

11、总结

对于分类而言，使用 **概率** 有时要比使用 **硬规则** 更为有效。贝叶斯概率及贝叶斯准则提供了一种利用已知值来估计未知概率的有效方法。可以通过 **特征之间的条件独立性假设**，降低对数据量的需求。独立性假设是指一个词的出现概率并不依赖于文档中的其他词。当然我们也知道这个假设过于简单。这就是之所以称为 **朴素贝叶斯** 的原因。尽管条件独立性假设并不正确，但是朴素贝叶斯仍然是一种有效的分类器。

利用现代编程语言来实现朴素贝叶斯时需要考虑很多实际因素。**下溢出** 就是其中一个问题，它可以通过对概率取对数来解决。还有其他一些方面的改进，比如说移除停用词，当然也可以花大量时间对切分器进行优化。

本章学习到的概率理论将在后续章节中用到，另外本章也给出了有关贝叶斯概率理论全面具体的介绍。接下来的一章将暂时不再讨论概率理论这一话题，介绍另一种称作Logistic回归的分类方法及一些优化算法。