贝叶斯分类

一文读懂贝叶斯分类算法

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。本文首先介绍分类问题，给出分类问题的定义。随后介绍贝叶斯分类算法的基础——贝叶斯定理。最后介绍贝叶斯分类中最简单的一种——朴素贝叶斯分类，并结合应用案例进一步阐释。

贝叶斯分类

1. 分类问题综述

对于分类问题，我们每一个人都并不陌生，因为在日常生活中我们都在或多或少地运用它。例如，当你看到一个陌生人，你的脑子下意识判断 TA 是男是女；你可能经常会走在路上对身旁的朋友说“这个人一看就很有钱、那边有个非主流”之类的话，其实这就是一种分类操作。

从数学角度来说，分类问题可做如下定义：

已知集合：和，确定映射规则，使得 0 任意有且仅有一个使得成立。其中 C 叫做类别集合，其中每一个元素是一个类别，而 I 叫做项集合，其中每一个元素是一个待分类项，f 叫做分类器。分类算法的任务就是构造分类器 f。

这里要强调的是，分类问题往往采用经验性方法构造映射规则，即一般情况下的分类问题缺少足够的信息来构造 100%正确的映射规则，而是通过对经验数据的学习从而实现一定概率意义上正确的分类，因此所训练出的分类器并不是一定能将每个待分类项准确映射到其分类，分类器的质量与分类器构造方法、待分类数据的特性以及训练样本数量等诸多因素有关。

例如，医生对病人进行诊断就是一个典型的分类过程，任何一个医生都无法直接看到病人的病情，只能观察病人表现出的症状和各种化验检测数据来推断病情，这时医生就好比一个分类器，而这个医生诊断的准确率，与他当初受到的教育方式（构造方法）、病人的症状是否突出（待分类数据的特性）以及医生的经验多少（训练样本数量）都有密切关系。

2. 贝叶斯分类的基础——贝叶斯定理

这个定理解决了现实生活里经常遇到的问题：已知某条件概率，如何得到两个事件交换后的概率，也就是在已知 P(A|B)的情况下如何求得 P(B|A)。这里先解释什么是条件概率：

P(A|B)表示事件 B 已经发生的前提下，事件 A 发生的概率，叫做事件 B 发生下事件 A 的条件概率。其基本求解公式为：

贝叶斯定理之所以有用，是因为我们在生活中经常遇到这种情况：我们可以很容易直接得出 P(A|B)，P(B|A)则很难直接得出，但我们更关心 P(B|A)，贝叶斯定理就为我们打通从 P(A|B)获得 P(B|A)的道路。

下面不加证明地直接给出贝叶斯定理：

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

3. 朴素贝叶斯分类

朴素贝叶斯分类的原理与流程：

朴素贝叶斯（分类器）是一种生成模型，它会基于训练样本对每个可能的类别建模。之所以叫朴素贝叶斯，是因为采用了属性条件独立性假设，就是假设每个属性独立地对分类结果产生影响。即有下面的公式：

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

后面连乘的地方要注意的是，如果有一项概率值为 0 会影响后面估计，所以我们对未出现的属性概率设置一个很小的值，并不为 0,这就是拉普拉斯修正（Laplacian correction）。

拉普拉斯修正实际上假设了属性值和类别的均匀分布，在学习过程中额外引入了先验识。

整个朴素贝叶斯分类分为三个阶段：

Stage1：准备工作阶段，这个阶段的任务是为朴素贝叶斯分类做必要的准备，主要工作是根据具体情况确定特征属性，并对每个特征属性进行适当划分，然后由人工对一部分待分类项进行分类，形成训练样本集合。这一阶段的输入是所有待分类数据，输出是特征属性和训练样本。这一阶段是整个朴素贝叶斯分类中唯一需要人工完成的阶段，其质量对整个过程将有重要影响，分类器的质量很大程度上由特征属性、特征属性划分及训练样本质量决定。

Stage2：分类器训练阶段，这个阶段的任务就是生成分类器，主要工作是计算每个类别在训练样本中的出现频率及每个特征属性划分对每个类别的条件概率估计，并将结果记录。其输入是特征属性和训练样本，输出是分类器。这一阶段是机械性阶段，根据前面讨论的公式可以由程序自动计算完成。

Stage3：应用阶段，这个阶段的任务是使用分类器对待分类项进行分类，其输入是分类器和待分类项，输出是待分类项与类别的映射关系。这一阶段也是机械性阶段，由程序完成。

4. 半朴素贝叶斯分类

在朴素的分类中，我们假定了各个属性之间的独立，这是为了计算方便，防止过多的属性之间的依赖导致的大量计算。这正是朴素的含义，虽然朴素贝叶斯的分类效果不错，但是属性之间毕竟是有关联的，某个属性依赖于另外的属性，于是就有了半朴素贝叶斯分类器。

为了计算量不至于太大，假定每个属性只依赖另外的一个。这样，更能准确描述真实情况。

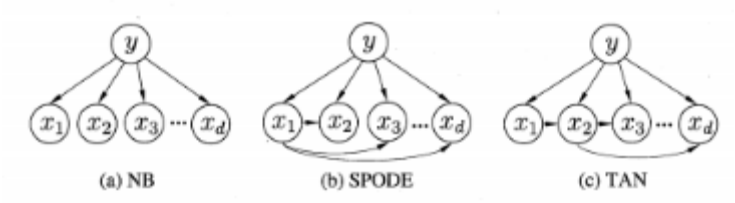
公式就变成：

$$P(c|x) \propto P(c)\prod_{i=1}^{d} P(x_i|c,pa_i)$$

属性为所依赖的属性，成为的父属性。

确定父属性有如下方法：

- **SOPDE 方法。**这种方法是假定所有的属性都依赖于共同的一个父属性。
- **TAN 方法。**每个属性依赖的另外的属性由最大带权生成树来确定。
- 先求每个属性之间的互信息来作为他们之间的权值。
- 构件完全图。权重是刚才求得的互信息。然后用最大带权生成树算法求得此图的最大带权的生成树。
- 找一个根变量，然后依次将图变为有向图。
- 添加类别 y 到每个属性的的有向边。



三种方法的属性依赖关系

贝叶斯算法应用举例

以下我们再举一些实际例子来说明贝叶斯方法被运用的普遍性，这里主要集中在机器学习方面。

2.1 中文分词

Google 研究员吴军在《数学之美》系列中就有一篇是介绍中文分词的，这里只介绍一下核心的思想。

分词问题的描述为：给定一个句子（字串），如：南京市长江大桥。如何对这个句子进行分词（词串）才是最靠谱的。例如：

- 南京市/长江大桥
- 南京/市长/江大桥

这两个分词，到底哪个更靠谱呢？

我们用贝叶斯公式来形式化地描述这个问题，令 X 为字串（句子），Y 为词串（一种特定的分词假设）。我们就是需要寻找使得 P(Y|X) 最大的 Y ，使用一次贝叶斯可得：

用自然语言来说就是这种分词方式（词串）的可能性乘这个词串生成我们的句子的可能性。我们进一步容易看到：可以近似地将 P(X|Y) 看作是恒等于 1 的，因为任意假想的一种分词方式之下生成我们的句子总是精准地生成的（只需把分词之间的分界符号扔掉即可）。于是，我们就变成了去最大化 P(Y) ，也就是寻找一种分词使得这个词串（句子）的概率最大化。而如何计算一个词串：W1, W2, W3, W4 ..的可能性呢？

我们知道，根据联合概率的公式展开：P(W1, W2, W3, W4 ..) = P(W1) * P(W2|W1) * P(W3|W2, W1) * P(W4|W1,W2,W3) * .. 于是我们可以通过一系列的条件概率（右式）的乘积来求整个联合概率。然而不幸的是随着条件数目的增加（P(Wn|Wn-1,Wn-2,..,W1) 的条件有 n-1 个），数据稀疏问题也会越来越严重，即便语料库再大也无法统计出一个靠谱的 P(Wn|Wn-1,Wn-2,..,W1) 来。

为了缓解这个问题，计算机科学家们一如既往地使用了“天真”假设：我们假设句子中一个词的出现概率只依赖于它前面的有限的 k 个词（k 一般不超过 3，如果只依赖于前面的一个词，就是 2 元语言模型（2-gram），同理有 3-gram 、 4-gram 等），这个就是所谓的“有限地平线”假设。虽然这个假设似乎有些理想化，但结果却表明它的结果往往是很强大的，后面要提到的朴素贝叶斯方法使用的假设跟这个精神上是完全一致的，我们会解释为什么像这样一个理想化假设能够得到强大的结果。

目前我们只要知道，有了这个假设，刚才那个乘积就可以改写成： P(W1) * P(W2|W1) * P(W3|W2) * P(W4|W3) .. （假设每个词只依赖于它前面的一个词）。而统计 P(W2|W1) 就不再受到数据稀疏问题的困扰了。对于我们上面提到的例子“南京市长江大桥”，如果按照自左到右的贪婪方法分词的话，结果就成了“南京市长/江大桥”。但如果按照贝叶斯分词的话（假设使用 3-gram），由于“南京市长”和“江大桥”在语料库中一起出现的频率为 0 ，这个整句的概率便会被判定为 0 。从而使得“南京市/长江大桥”这一分词方式胜出。

2.2 贝叶斯垃圾邮件过滤器

给定一封邮件，判定它是否属于垃圾邮件。按照先例，我们还是用 D 来表示这封邮件，注意 D 由 N 个单词组成。我们用 h+来表示垃圾邮件，h-表示正常邮件。问题可以形式化地描述为求：

$$P(h+|D) = P(h+) * P(D|h+) / P(D)$$

$$P(h-|D) = P(h-) * P(D|h-) / P(D)$$

其中 P(h+) 和 P(h-) 这两个先验概率都是很容易求出来的，只需要计算一个邮件库里面垃圾邮件和正常邮件的比例就行了。然而 P(D|h+) 却不容易求，因为 D 里面含有 N 个单词 d1, d2, d3, .. ，所以 P(D|h+) = P(d1,d2,..,dn|h+) 。我们又一次遇到了数据稀疏性，为何这么说呢？

P(d1,d2,..,dn|h+) 就是说在垃圾邮件当中出现跟我们目前这封邮件一模一样的一封邮件的概率是非常小的，因为每封邮件都是不同的，世界上有无穷多封邮件。这就是数据稀疏性，因为可以肯定地说，你收集的训练数据库不管里面含了多少封邮件，也不可能找出一封跟目前这封一模一样的。结果呢？我们又该如何来计算 P(d1,d2,..,dn|h+) 呢？

我们将 P(d1,d2,..,dn|h+)扩展为： P(d1|h+) * P(d2|d1, h+) * P(d3|d2,d1, h+) * .. 。这个式子想必大家并不陌生，这里我们会使用一个更激进的假设，我们假设 di 与 di-1 是完全条件无关的，于是式子就简化为 P(d1|h+) * P(d2|h+) * P(d3|h+) * .. 。这个就是所谓的条件独立假设，也正是朴素贝叶斯方法的朴素之处。而计算 P(d1|h+) * P(d2|h+) * P(d3|h+) * .. 问题至此就变得简单了，只要统计 di 这个单词在垃圾邮件中出现的频率即可。

通过以上学习我们发现，由于无法穷举所有可能性，贝叶斯推断基本上不能给出肯定的结果。尽管如此，在进行大量的测试后，如果获得的测试结果都无误，我们也会对自己的算法很有信心（即便算法的准确性尚未确认）。事实上，随着新的测试结果出现，算法无误的可信度也在逐渐改变。

生活中，我们经常有意或无意地运用着贝叶斯定理，从算法的角度讲，如果想真正搞懂其中的原理，还需要对数理统计知识进行更加深入地学习并且不断实践，最终相信大家一定能够有所收获。

机器学习经典算法之朴素贝叶斯分类

很多人都听说过贝叶斯原理，在哪听说过？基本上是在学概率统计的时候知道的。有些人可能会说，我记不住这些概率论的公式，没关系，我尽量用通俗易懂的语言进行讲解。

/*请尊重作者劳动成果，转载请标明原文链接：*/

/* https://www.cnblogs.com/jpcflyer/p/11069659.html */

贝叶斯原理是英国数学家托马斯·贝叶斯提出的。贝叶斯是个很神奇的人，他的经历类似梵高。生前没有得到重视，死后，他写的一篇关于归纳推理的论文被朋友翻了出来，并发表了。这一发表不要紧，结果这篇论文的思想直接影响了接下来两个多世纪的统计学，是科学史上著名的论文之一。

贝叶斯原理跟我们的生活联系非常紧密。举个例子，如果你看到一个人总是花钱，那么会推断这个人多半是个有钱人。当然这也不是绝对，也就是说，当你不能准确预知一个事物本质的时候，你可以依靠和事物本质相关的事件来进行判断，如果事情发生的频次多，则证明这个属性更有可能存在。

一、 贝叶斯原理

贝叶斯原理是怎么来的呢？贝叶斯为了解决一个叫“逆向概率”问题写了一篇文章，尝试解答在没有太多可靠证据的情况下，怎样做出更符合数学逻辑的推测。

什么是“逆向概率”呢？

所谓“逆向概率”是相对“正向概率”而言。正向概率的问题很容易理解，比如我们已经知道袋子里面有 N 个球，不是黑球就是白球，其中 M 个是黑球，那么把手伸进去摸一个球，就能知道摸出黑球的概率是多少。但这种情况往往是上帝视角，即了解了事情的全貌再做判断。

在现实生活中，我们很难知道事情的全貌。贝叶斯则从实际场景出发，提了一个问题：如果我们事先不知道袋子里面黑球和白球的比例，而是通过我们摸出来的球的颜色，能判断出袋子里面黑白球的比例么？

正是这样的一个问题，影响了接下来近 200 年的统计学理论。这是因为，贝叶斯原理与其他统计学推断方法截然不同，它是建立在主观判断的基础上：在我们不了解所有客观事实的情况下，同样可以先估计一个值，然后根据实际结果不断进行修正。

我们用一个题目来体会下：假设有一种病叫做“贝叶死”，它的发病率是万分之一，即 10000 人中会有 1 个人得病。现有一种测试可以检验一个人是否得病的准确率是 99.9%，它的误报率是 0.1%，那么现在的问题是，如果一个人被查出来患有“叶贝死”，实际上患有的可能性有多大？

你可能会想说，既然查出患有“贝叶死”的准确率是 99.9%，那是不是实际上患“贝叶死”的概率也是 99.9% 呢？实际上不是的。你自己想想，在 10000 个人中，还存在 0.1% 的误查的情况，也就是 10 个人没有患病但是被诊断成阳性。当然 10000 个人中，也确实存在一个患有贝叶死的人，他有 99.9% 的概率被检查出来。所以你可以粗算下，患病的这个人实际上是这 11 个人里面的一员，即实际患病比例是 1/11≈9%。

上面这个例子中，实际上涉及到了贝叶斯原理中的几个概念：

先验概率

通过经验来判断事情发生的概率，比如说“贝叶死”的发病率是万分之一，就是先验概率。再比如南方的梅雨季是 6-7 月，就是通过往年的气候总结出来的经验，这个时候下雨的概率就比其他时间高出很多。

后验概率

后验概率就是发生结果之后，推测原因的概率。比如说某人查出来了患有“贝叶死”，那么患病的原因可能是 A、B 或 C。患有“贝叶死”是因为原因 A 的概率就是后验概率。它是属于条件概率的一种。

条件概率

事件 A 在另外一个事件 B 已经发生条件下的发生概率，表示为 P(A|B)，读作“在 B 发生的条件下 A 发生的概率”。比如原因 A 的条件下，患有“贝叶死”的概率，就是条件概率。

似然函数（likelihood function）

你可以把概率模型的训练过程理解为求参数估计的过程。举个例子，如果一个硬币在 10 次抛落中正面均朝上。那么你肯定在想，这个硬币是均匀的可能性是多少？这里硬币均匀就是个参数，似然函数就是用来衡量这个模型的参数。似然在这里就是可能性的意思，它是关于统计参数的函数。

介绍完贝叶斯原理中的这几个概念，我们再来看下贝叶斯原理，实际上贝叶斯原理就是求解后验概率，我们假设：A 表示事件 “测出为阳性”，用 B1 表示“患有贝叶死”，B2 表示“没有患贝叶死”。根据上面那道题，我们可以得到下面的信息。

患有贝叶死的情况下，测出为阳性的概率为 P(A|B1)=99.9%，没有患贝叶死，但测出为阳性的概率为 P(A|B2)=0.1%。另外患有贝叶死的概率为 P(B1)=0.01%，没有患贝叶死的概率 P(B2)=99.99%。
那么我们检测出来为阳性，而且是贝叶死的概率 P(B1，A) =P(B1)*P(A|B1)=0.01%*99.9%=0.00999%。
这里 P(B1,A) 代表的是联合概率，同样我们可以求得 P(B2,A)=P(B2)*P(A|B2)=99.99%*0.1%=0.09999%。
然后我们想求得是检查为阳性的情况下，患有贝叶死的概率，也即是 P(B1|A)。

所以检查出阳性，且患有贝叶死的概率为：

$$P(B_1 \mid A) = \frac{0.01\%}{0.01\% + 0.1\%} \approx 9\%$$

检查出是阳性，但没有患有贝叶死的概率为：

$$P(B_2 \mid A) = \frac{0.1\%}{0.01\% + 0.1\%} \approx 90.9\%$$

这里我们能看出来 0.01%+0.1% 均出现在了 P(B1|A) 和 P(B2|A) 的计算中作为分母。我们把它称之为论据因子，也相当于一个权值因子。
其中 P(B1）、P(B2) 就是先验概率，我们现在知道了观测值，就是被检测出来是阳性，来求患贝叶死的概率，也就是求后验概率。求后验概率就是贝叶斯原理要求的，基于刚才求得的 P(B1|A)，P(B2|A)，我们可以总结出贝叶斯公式为：

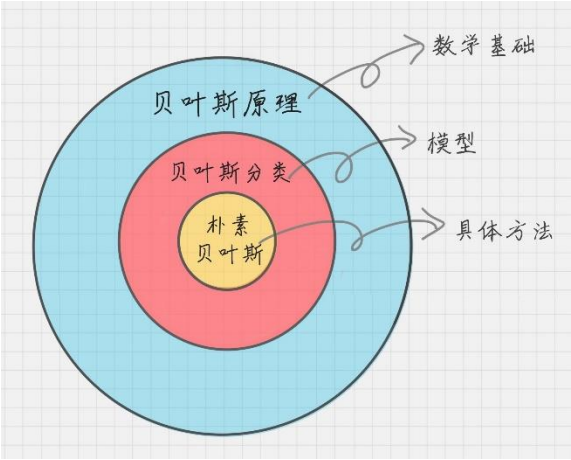
$$P(B_i \mid A) = \frac{P(B_i)P(A \mid B_i)}{P(B_1)P(A \mid B_1) + P(B_2)P(A \mid B_2)}$$

由此，我们可以得出通用的贝叶斯公式：

$$P(B_i \mid A) = \frac{P(B_i)P(A \mid B_i)}{\sum_{i=1}^{n} P(B_i)P(A \mid B_i)}$$

朴素贝叶斯

讲完贝叶斯原理之后，我们再来看下今天重点要讲的算法，朴素贝叶斯。 它是一种简单但极为强大的预测建模算法 。之所以称为朴素贝叶斯，是因为它假设每个输入变量是独立的。这是一个强硬的假设，实际情况并不一定，但是这项技术对于绝大部分的复杂问题仍然非常有效。
朴素贝叶斯模型由两种类型的概率组成：
每个 类别的概率 P(Cj)；
每个属性的 条件概率 P(Ai|Cj)。
我来举个例子说明下什么是类别概率和条件概率。假设我有 7 个棋子，其中 3 个是白色的，4 个是黑色的。那么棋子是白色的概率就是 3/7，黑色的概率就是 4/7，这个就是类别概率。
假设我把这 7 个棋子放到了两个盒子里，其中盒子 A 里面有 2 个白棋，2 个黑棋；盒子 B 里面有 1 个白棋，2 个黑棋。那么在盒子 A 中抓到白棋的概率就是 1/2，抓到黑棋的概率也是 1/2，这个就是条件概率，也就是在某个条件（比如在盒子 A 中）下的概率。
在朴素贝叶斯中，我们要统计的是属性的条件概率，也就是假设取出来的是白色的棋子，那么它属于盒子 A 的概率是 2/3。
为了训练朴素贝叶斯模型，我们需要先给出训练数据，以及这些数据对应的分类。那么上面这两个概率，也就是类别概率和条件概率。他们都可以从给出的训练数据中计算出来。一旦计算出来，概率模型就可以使用贝叶斯原理对新数据进行预测。
另外我想告诉你的是，贝叶斯原理、贝叶斯分类和朴素贝叶斯这三者之间是有区别的。
贝叶斯原理是最大的概念，它解决了概率论中“逆向概率”的问题，在这个理论基础上，人们设计出了贝叶斯分类器，朴素贝叶斯分类是贝叶斯分类器中的一种，也是最简单，最常用的分类器。朴素贝叶斯之所以朴素是因为它假设属性是相互独立的，因此对实际情况有所约束，如果属性之间存在关联，分类准确率会降低。不过好在对于大部分情况下，朴素贝叶斯的分类效果都不错。



二、 朴素贝叶斯分类工作原理

朴素贝叶斯分类是常用的贝叶斯分类方法。我们日常生活中看到一个陌生人，要做的第一件事情就是判断 TA 的性别，判断性别的过程就是一个分类的过程。根据以往的经验，我们通常会从身高、体重、鞋码、头发长短、服饰、声音等角度进行判断。这里的“经验”就是一个训练好的关于性别判断的模型，其训练数据是日常中遇到的各式各样的人，以及这些人实际的性别数据。
离散数据案例

我们遇到的数据可以分为两种，一种是离散数据，另一种是连续数据。那什么是离散数据呢？离散就是不连续的意思，有明确的边界，比如整数 1，2，3 就是离散数据，而 1 到 3 之间的任何数，就是连续数据，它可以取在这个区间里的任何数值。
我以下面的数据为例，这些是根据你之前的经验所获得的数据。然后给你一个新的数据：身高“高”、体重“中”，鞋码“中”，请问这个人是男还是女？

编号	身高	体重	鞋码	性别
1	高	重	大	男
2	高	重	大	男
3	中	中	大	男
4	中	中	中	男
5	矮	轻	小	女
6	矮	轻	小	女
7	矮	中	中	女
8	中	中	中	女

针对这个问题，我们先确定一共有 3 个属性，假设我们用 A 代表属性，用 A1, A2, A3 分别为身高 = 高、体重 = 中、鞋码 = 中。一共有两个类别，假设用 C 代表类别，那么 C1,C2 分别是：男、女，在未知的情况下我们用 Cj 表示。
那么我们想求在 A1、A2、A3 属性下，Cj 的概率，用条件概率表示就是 P(Cj|A1A2A3)。根据上面讲的贝叶斯的公式，我们可以得出：

$$P(C_j \mid A_1A_2A_3) = \frac{P(A_1A_2A_3 \mid C_j)P(C_j)}{P(A_1A_2A_3)}$$

因为一共有 2 个类别，所以我们只需要求得 P(C1|A1A2A3) 和 P(C2|A1A2A3) 的概率即可，然后比较下哪个分类的可能性大，就是哪个分类结果。
在这个公式里，因为 P(A1A2A3) 都是固定的，我们想要寻找使得 P(Cj|A1A2A3) 的最大值，就等价于求 P(A1A2A3|Cj)P(Cj) 最大值。
我们假定 Ai 之间是相互独立的，那么： P(A1A2A3|Cj)=P(A1|Cj)P(A2|Cj)P(A3|Cj)
然后我们需要从 Ai 和 Cj 中计算出 P(Ai|Cj) 的概率，带入到上面的公式得出 P(A1A2A3|Cj)，最后找到使得 P(A1A2A3|Cj) 最大的类别 Cj。
我分别求下这些条件下的概率：
P(A1|C1)=1/2, P(A2|C1)=1/2, P(A3|C1)=1/4，P(A1|C2)=0, P(A2|C2)=1/2, P(A3|C2)=1/2，所以 P(A1A2A3|C1)=1/16, P(A1A2A3|C2)=0。
因为 P(A1A2A3|C1)P(C1)>P(A1A2A3|C2)P(C2)，所以应该是 C1 类别，即男性。

连续数据案例
实际生活中我们得到的是连续的数值，比如下面这组数据：

编号	身高（CM）	体重（斤）	鞋码（欧码）	性别
1	183	164	45	男
2	182	170	44	男
3	178	160	43	男
4	175	140	40	男
5	160	88	35	女
6	165	100	37	女
7	163	110	38	女
8	168	120	39	女

那么如果给你一个新的数据，身高 180、体重 120，鞋码 41，请问该人是男是女呢?

公式还是上面的公式，这里的困难在于，由于身高、体重、鞋码都是连续变量，不能采用离散变量的方法计算概率。而且由于样本太少，所以也无法分成区间计算。怎么办呢?
这时，可以假设男性和女性的身高、体重、鞋码都是正态分布，通过样本计算出均值和方差，也就是得到正态分布的密度函数。有了密度函数，就可以把值代入，算出某一点的密度函数的值。比如，男性的身高是均值 179.5、标准差为 3.697 的正态分布。所以男性的身高为 180 的概率为 0.1069。怎么计算得出的呢？你可以使用 EXCEL 的 NORMDIST(x,mean,standard_dev,cumulative) 函数，一共有 4 个参数：
x：正态分布中，需要计算的数值；
Mean：正态分布的平均值；
Standard_dev：正态分布的标准差；
Cumulative：取值为逻辑值，即 False 或 True。它决定了函数的形式。当为 TRUE 时，函数结果为累积分布；为 False 时，函数结果为概率密度。
这里我们使用的是 NORMDIST(180,179.5,3.697,0)=0.1069。
同理我们可以计算得出男性体重为 120 的概率为 0.000382324，男性鞋码为 41 号的概率为 0.120304111。
所以我们可以计算得出：
P(A1A2A3|C1)=P(A1|C1)P(A2|C1)P(A3|C1)=0.1069*0.000382324* 0.120304111=4.9169e-6
同理我们也可以计算出来该人为女的可能性：
P(A1A2A3|C2)=P(A1|C2)P(A2|C2)P(A3|C2)=0.00000147489* 0.015354144* 0.120306074=2.7244e-9
很明显这组数据分类为男的概率大于分类为女的概率。

当然在 Python 中，有第三方库可以直接帮我们进行上面的操作，这个我们会在下文中介绍。这里主要是给你讲解下具体的运算原理。

三、朴素贝叶斯分类器工作流程

朴素贝叶斯分类常用于文本分类，尤其是对于英文等语言来说，分类效果很好。它常用于垃圾文本过滤、情感预测、推荐系统等。
朴素贝叶斯分类器需要三个流程，我来给你一一讲解下这几个流程。
第一阶段：准备阶段
在这个阶段我们需要确定特征属性，比如上面案例中的“身高”、“体重”、“鞋码”等，并对每个特征属性进行适当划分，然后由人工对一部分数据进行分类，形成训练样本。
这一阶段是整个朴素贝叶斯分类中唯一需要人工完成的阶段，其质量对整个过程将有重要影响，分类器的质量很大程度上由特征属性、特征属性划分及训练样本质量决定。
第二阶段：训练阶段
这个阶段就是生成分类器，主要工作是计算每个类别在训练样本中的出现频率及每个特征属性划分对每个类别的条件概率。
输入是特征属性和训练样本，输出是分类器。
第三阶段：应用阶段
这个阶段是使用分类器对新数据进行分类。输入是分类器和新数据，输出是新数据的分类结果。
好了，在这次课中你了解了概率论中的贝叶斯原理，朴素贝叶斯的工作原理和工作流程，也对朴素贝叶斯的强大和限制有了认识。下一节中，我将带你实战，亲自掌握 Python 中关于朴素贝叶斯分类器工具的使用。

四、 sklearn 机器学习包

接下来带你一起使用朴素贝叶斯做下文档分类的项目，最重要的工具就是 sklearn 这个机器学习神器。
sklearn 的全称叫 Scikit-learn，它给我们提供了 3 个朴素贝叶斯分类算法，分别是高斯朴素贝叶斯（GaussianNB）、多项式朴素贝叶斯（MultinomialNB）和伯努利朴素贝叶斯（BernoulliNB）。
这三种算法适合应用在不同的场景下，我们应该根据特征变量的不同选择不同的算法：
高斯朴素贝叶斯：特征变量是连续变量，符合高斯分布，比如说人的身高，物体的长度。
多项式朴素贝叶斯：特征变量是离散变量，符合多项分布，在文档分类中特征变量体现在一个单词出现的次数，或者是单词的 TF-IDF 值等。
伯努利朴素贝叶斯：特征变量是布尔变量，符合 0/1 分布，在文档分类中特征是单词是否出现。
伯努利朴素贝叶斯是以文件为粒度，如果该单词在某文件中出现了即为 1，否则为 0。而多项式朴素贝叶斯是以单词为粒度，会计算在某个文件中的具体次数。而高斯朴素贝叶斯适合处理特征变量是连续变量，且符合正态分布（高斯分布）的情况。比如身高、体重这种自然界的现象就比较适合用高斯朴素贝叶斯来处理。而文本分类是使用多项式朴素贝叶斯或者伯努利朴素贝叶斯。

什么是 TF-IDF 值呢?
我在多项式朴素贝叶斯中提到了“词的 TF-IDF 值”，如何理解这个概念呢?
TF-IDF 是一个统计方法，用来评估某个词语对于一个文件集或文档库中的其中一份文件的重要程度。
TF-IDF 实际上是两个词组 Term Frequency 和 Inverse Document Frequency 的总称，两者缩写为 TF 和 IDF，分别代表了词频和逆向文档频率。

词频 TF 计算了一个单词在文档中出现的次数，它认为一个单词的重要性和它在文档中出现的次数呈正比。
逆向文档频率 IDF ，是指一个单词在文档中的区分度。它认为一个单词出现在的文档数越少，就越能通过这个单词把该文档和其他文档区分开。IDF 越大就代表该单词的区分度越大。
所以 TF-IDF 实际上是词频 TF 和逆向文档频率 IDF 的乘积 。这样我们倾向于找到 TF 和 IDF 取值都高的单词作为区分，即这个单词在一个文档中出现的次数多，同时又很少出现在其他文档中。这样的单词适合用于分类。
TF-IDF 如何计算
首先我们看下词频 TF 和逆向文档概率 IDF 的公式。

$$\text{词频 TF} = \frac{\text{单词出现的次数}}{\text{该文档的总单词数}}$$

$$\text{逆向文档频率 IDF} = \log\frac{\text{文档总数}}{\text{该单词出现的文档数} + 1}$$

为什么 IDF 的分母中，单词出现的文档数要加 1 呢？因为有些单词可能不会存在文档中，为了避免分母为 0，统一给单词出现的文档数都加 1。
TF-IDF=TF*IDF。
你可以看到，TF-IDF 值就是 TF 与 IDF 的乘积，这样可以更准确地对文档进行分类。比如“我”这样的高频单词，虽然 TF 词频高，但是 IDF 值很低，整体的 TF-IDF 也不高。
我在这里举个例子。假设一个文件夹里一共有 10 篇文档，其中一篇文档有 1000 个单词，“this”这个单词出现 20 次，“bayes”出现了 5 次。“this”在所有文档中均出现过，而“bayes”只在 2 篇文档中出现过。我们来计算一下这两个词语的 TF-IDF 值。
针对“this”，计算 TF-IDF 值：

$$\text{词频 TF} = \frac{20}{1000} = 0.02$$

$$\text{逆向文档频率 IDF} = \log\frac{10}{10 + 1} = -0.0414$$

所以 TF-IDF=0.02*(-0.0414)=-8.28e-4。

针对“bayes”，计算 TF-IDF 值：

$$词频\ TF = \frac{5}{1000} = 0.005$$

$$逆向文档频率\ IDF = \log\frac{10}{2+1} = 0.5229$$

很明显“bayes”的 TF-IDF 值要大于“this”的 TF-IDF 值。这就说明用“bayes”这个单词做区分比单词“this”要好。
如何求 TF-IDF
在 sklearn 中我们直接使用 TfidfVectorizer 类，它可以帮我们计算单词 TF-IDF 向量的值。在这个类中，取 sklearn 计算的对数 log 时，底数是 e，不是 10。
下面我来讲下如何创建 TfidfVectorizer 类。
TfidfVectorizer 类的创建：
创建 TfidfVectorizer 的方法是：

```
1 TfidfVectorizer(stop_words=stop_words, token_pattern=token_pattern)
```

我们在创建的时候，有两个构造参数，可以自定义停用词 stop_words 和规律规则 token_pattern。需要注意的是传递的数据结构，停用词 stop_words 是一个列表 List 类型，而过滤规则 token_pattern 是正则表达式。
什么是停用词？停用词就是在分类中没有用的词，这些词一般词频 TF 高，但是 IDF 很低，起不到分类的作用。为了节省空间和计算时间，我们把这些词作为停用词 stop words，告诉机器这些词不需要帮我计算。

参数表	作用
stop_words	自定义停用词表，为列表List类型
token_pattern	过滤规则，正则表达式，如r"(?u)\b\w+\b"

当我们创建好 TF-IDF 向量类型时，可以用 fit_transform 帮我们计算，返回给我们文本矩阵，该矩阵表示了每个单词在每个文档中的 TF-IDF 值。

方法表	作用
fit_transform(X)	拟合模型，并返回文本矩阵

在我们进行 fit_transform 拟合模型后，我们可以得到更多的 TF-IDF 向量属性，比如，我们可以得到词汇的对应关系（字典类型）和向量的 IDF 值，当然也可以获取设置的停用词 stop_words。

属性表	作用
vocabulary_	词汇表；字典型
idf_	返回idf值
stop_words_	返回停用词表

举个例子，假设我们有 4 个文档：
文档 1：this is the bayes document；
文档 2：this is the second second document；
文档 3：and the third one；
文档 4：is this the document。
现在想要计算文档里都有哪些单词，这些单词在不同文档中的 TF-IDF 值是多少呢？
首先我们创建 TfidfVectorizer 类：

```
1 from sklearn.feature_extraction.text import TfidfVectorizer
2 tfidf_vec = TfidfVectorizer()
```

然后我们创建 4 个文档的列表 documents，并让创建好的 tfidf_vec 对 documents 进行拟合，得到 TF-IDF 矩阵：

```
1 documents = [
2     'this is the bayes document',
3     'this is the second second document',
4     'and the third one',
5     'is this the document'
6 ]
7 tfidf_matrix = tfidf_vec.fit_transform(documents)
```

输出文档中所有不重复的词：

```
1 print('不重复的词:', tfidf_vec.get_feature_names())
```

运行结果

```
1 不重复的词: ['and', 'bayes', 'document', 'is', 'one', 'second', 'the', 'third', 'this']
```

输出每个单词对应的 id 值：

```
1 print('每个单词的 ID:', tfidf_vec.vocabulary_)
```

运行结果

```
1 每个单词的 ID: {'this': 8, 'is': 3, 'the': 6, 'bayes': 1, 'document': 2, 'second': 5, 'and': 0, 'third': 7, 'one': 4}
```

输出每个单词在每个文档中的 TF-IDF 值，向量里的顺序是按照词语的 id 顺序来的：

```
1 print('每个单词的 tfidf 值:', tfidf_matrix.toarray())
```

运行结果：

```
每个单词的 tfidf 值: [[0.          0.63314609 0.40412895 0.40412895 0.          0.
  0.33040189 0.          0.40412895]
 [0.          0.          0.27230147 0.27230147 0.          0.85322574
  0.22262429 0.          0.27230147]
 [0.55280532 0.          0.          0.          0.55280532 0.
  0.28847675 0.55280532 0.        ]
 [0.          0.          0.52210862 0.52210862 0.          0.
  0.42685801 0.          0.52210862]]
```

五、 如何对文档进行分类

如果我们要对文档进行分类，有两个重要的阶段：
1.基于分词的数据准备 ，包括分词、单词权重计算、去掉停用词；
2.应用朴素贝叶斯分类进行分类 ，首先通过训练集得到朴素贝叶斯分类器，然后将分类器应用于测试集，并与实际结果做对比，最终得到测试集的分类准确率。
下面，我分别对这些模块进行介绍。

模块 1：对文档进行分词

在准备阶段里，最重要的就是分词。那么如果给文档进行分词呢？英文文档和中文文档所使用的分词工具不同。
在英文文档中，最常用的是 NTLK 包。NTLK 包中包含了英文的停用词 stop words、分词和标注方法。

```
import nltk
word_list = nltk.word_tokenize(text) # 分词
nltk.pos_tag(word_list) # 标注单词的词性
```

在中文文档中，最常用的是 jieba 包。jieba 包中包含了中文的停用词 stop words 和分词方法。

```
import jieba
word_list = jieba.cut (text) # 中文分词
```

模块 2：加载停用词表
我们需要自己读取停用词表文件，从网上可以找到中文常用的停用词保存在 stop_words.txt，然后利用 Python 的文件读取函数读取文件，保存在 stop_words 数组中。

```
stop_words = [line.strip().decode('utf-8') for line in io.open('stop_words.txt').readlines()]
```

模块 3：计算单词的权重
这里我们用到 sklearn 里的 TfidfVectorizer 类，上面我们介绍过它使用的方法。
直接创建 TfidfVectorizer 类，然后使用 fit_transform 方法进行拟合，得到 TF-IDF 特征空间 features，你可以理解为选出来的分词就是特征。我们计算这些特征在文档上的特征向量，得到特征空间 features。

```
tf = TfidfVectorizer(stop_words=stop_words, max_df=0.5)
features = tf.fit_transform(train_contents)
```

这里 max_df 参数用来描述单词在文档中的最高出现率。假设 max_df=0.5，代表一个单词在 50% 的文档中都出现过了，那么它只携带了非常少的信息，因此就不作为分词统计。
一般很少设置 min_df，因为 min_df 通常都会很小。
模块 4：生成朴素贝叶斯分类器
我们将特征训练集的特征空间 train_features，以及训练集对应的分类 train_labels 传递给贝叶斯分类器 clf，它会自动生成一个符合特征空间和对应分类的分类器。
这里我们采用的是多项式贝叶斯分类器，其中 alpha 为平滑参数。为什么要使用平滑呢？因为如果一个单词在训练样本中没有出现，这个单词的概率就会被计算为 0。但训练集样本只是整体的抽样情况，我们不能因为一个事件没有观察到，就认为整个事件的概率为 0。为了解决这个问题，我们需要做平滑处理。
当 alpha=1 时，使用的是 Laplace 平滑。Laplace 平滑就是采用加 1 的方式，来统计没有出现过的单词的概率。这样当训练样本很大的时候，加 1 得到的概率变化可以忽略不计，也同时避免了零概率的问题。
当 0<alpha<1 时，使用的是 Lidstone 平滑。对于 Lidstone 平滑来说，alpha 越小，迭代次数越多，精度越高。我们可以设置 alpha 为 0.001。

多项式贝叶斯分类器

```
1 from sklearn.naive_bayes import MultinomialNB
2 clf = MultinomialNB(alpha=0.001).fit(train_features, train_labels)
```

模块 5：使用生成的分类器做预测

首先我们需要得到测试集的特征矩阵。

方法是用训练集的分词创建一个 TfidfVectorizer 类，使用同样的 stop_words 和 max_df，然后用这个 TfidfVectorizer 类对测试集的内容进行 fit_transform 拟合，得到测试集的特征矩阵 test_features。

```
1 test_tf = TfidfVectorizer(stop_words=stop_words, max_df=0.5, vocabulary=train_vocabulary)
2 test_features=test_tf.fit_transform(test_contents)
```

然后我们用训练好的分类器对新数据做预测。

方法是使用 predict 函数，传入测试集的特征矩阵 test_features，得到分类结果 predicted_labels。predict 函数做的工作就是求解所有后验概率并找出最大的那个。

```
1 predicted_labels=clf.predict(test_features)
```

模块 6：计算准确率

计算准确率实际上是对分类模型的评估。我们可以调用 sklearn 中的 metrics 包，在 metrics 中提供了 accuracy_score 函数，方便我们对实际结果和预测的结果做对比，给出模型的准确率。

使用方法如下：

```
1 from sklearn import metrics
2 print metrics.accuracy_score(test_labels, predicted_labels)
```

三种贝叶斯分类

因变量是定量型的归纳学习称为回归，或者说是连续变量预测
因变量是定性型的归纳学习称为分类，或者说是离散变量预测

P（A B）= P（B | A）P（A）

P（A）叫做 A 事件的先验概率，就是一般情况下，认为 A 发生的概率。

P（B|A）叫做似然度，是 A 假设条件成立的情况下发生 B 的概率。

P（A|B）叫做后验概率，在 B 发生的情况下发生 A 的概率，也就是要计算的概率。

P（B）叫做标准化常量，和 A 的先验概率定义类似，就是一般情况下，B 的发生概率。

（1）高斯朴素贝叶斯（Gaussian Naive Bayes）；

（2）多项式朴素贝叶斯（Multinomial Naive Bayes）；

（3）伯努利朴素贝叶斯（Bernoulli Naive Bayes）。

其中，高斯朴素贝叶斯是利用高斯概率密度公式来进行分类拟合的。多项式朴素贝叶斯多用于高维度向量分类，最常用的场景是文章分类。伯努利朴素贝叶斯一般是针对布尔类型特征值的向量做分类的过程。

```
from sklearn.naive_bayes import GaussianNB    # 高斯贝叶斯分类
# 0: 晴 1: 阴 2: 降水 3: 多云
data_table = [["date", "weather"],
              [1, 0],
              [2, 1],
              [3, 2],
              [4, 1],
              [5, 2],
```

```
            [6, 0],
            [7, 0],
            [8, 3],
            [9, 1],
            [10, 1]]
# 当天的天气
X = [[0], [1], [2], [1], [2], [0], [0], [3], [1]]
# 当天的天气对应后一天的天气
y = [1, 2, 1, 2, 0, 0, 3, 1, 1]
# 现在把训练数据和对应的分类放入分类器中进行训练
clf = GaussianNB().fit(X, y)    # BernoulliNB() 伯努力  ComplementNB() 多项式  GaussianNB()  高斯
p = [[1]]
print(clf.predict(p))
```

结果为[2]。

一、高斯

```
>>> import numpy as np
    >>> X = np.array([[-1, -1], [-2, -1], [-3, -2], [1, 1], [2, 1], [3, 2]])
    >>> Y = np.array([1, 1, 1, 2, 2, 2])
    >>> from sklearn.naive_bayes import GaussianNB
    >>> clf = GaussianNB()
    >>> clf.fit(X, Y)
    GaussianNB(priors=None, var_smoothing=1e-09)
    >>> print(clf.predict([[-0.8, -1]]))
    [1]
    >>> clf_pf = GaussianNB()
    >>> clf_pf.partial_fit(X, Y, np.unique(Y))
    GaussianNB(priors=None, var_smoothing=1e-09)
    >>> print(clf_pf.predict([[-0.8, -1]]))
    [1]
```

二、多项式

```
>>> import numpy as np
    >>> X = np.random.randint(5, size=(6, 100))
    >>> y = np.array([1, 2, 3, 4, 5, 6])
    >>> from sklearn.naive_bayes import ComplementNB
```

```
>>> clf = ComplementNB()
>>> clf.fit(X, y)
ComplementNB(alpha=1.0, class_prior=None, fit_prior=True, norm=False)
>>> print(clf.predict(X[2:3]))
[3]
```

三、伯努力

```
>>> import numpy as np
>>> X = np.random.randint(2, size=(6, 100))
>>> Y = np.array([1, 2, 3, 4, 4, 5])
>>> from sklearn.naive_bayes import BernoulliNB
>>> clf = BernoulliNB()
>>> clf.fit(X, Y)
BernoulliNB(alpha=1.0, binarize=0.0, class_prior=None, fit_prior=True)
>>> print(clf.predict(X[2:3]))
[3]
```

Python 数据挖掘—分类—贝叶斯分类

pandas 之 get_dummies

方法：

pandas.get_dummies(data,prefix=None,prefix_sep="_",dummy_na=False,columns=None,sparse=False,drop_first=False)

该方法可以将类别变量转换成新增的虚拟变量/指示变量

参数说明：

- **data：array-like、Series 、 DataFrame ， 输入数据**

- **prefix：string、list of strings、dict of strings ， default 为 None，get_dummies 转换后，列名的前缀**
- **columns：list-like， default 为 False，指定需要实现类别转换的列名**
- **dummy_na：bool， default 为 False，增加一列表示空缺值，如果 False 就忽略空缺值**
- **drop_first：fool，default 为 False ，获取 K 中的 K-1 个类别之，去除第一个**

举例：

下面通过例子来进一步说明 get_dummies()

1、首先构造一个数据列

```
1 import pandas as pd

2 s=pd.Series(list('abca'))

3

4 s_1=pd.get_dummies(s)
```

1、两个变量：s 为 Series、s_1 为 DataFrame

Index	0
0	a
1	b
2	c
3	a

变成→

Index	a	b	c
0	1	0	0
1	0	1	0
2	0	0	1
3	1	0	0

2、去除第一列

```
1 b=pd.get_dummies(s,drop_first=True)
```

得到：

Index	b	c
0	0	0
1	1	0
2	0	1
3	0	0

3、查看 dummy_na 功能

创建数据

```
1 import numpy as np
2 s_2=["a","b",np.nan]
```

结果为：

Index	Type	Size	Value
0	str	1	a
1	str	1	b
2	float	1	nan

```
1 pd.get_dummies(s_2,dummy_na=True)
2 pd.get_dummies(s_2,dummy_na=False)
```

结果为：

Index	a	b	nan
0	1	0	0
1	0	1	0
2	0	0	1

、

Index	a	b
0	1	0
1	0	1
2	0	0

4、创建数据框

```
df=pd.DataFrame({
    "A":["a","b","a"],"B":["b","a","c"],
    "C":[1,2,3]})
pd.get_dummies(df,prefix=["col_1","col_2"])
```

结果为：

Index	A	B	C
0	a	b	1
1	b	a	2
2	a	c	3

变为→

Index	C	col1 a	col1 b	col2 a	col2 b	col2 c
0	1	1	0	0	1	0
1	2	0	1	1	0	0
2	3	1	0	0	0	1

实例：

```
import pandas

data=pandas.read_csv(
        "C:\\Users\\Jw\\Desktop\\python_work\\Python 数据挖掘实战课程课件\\5.2\\data1.csv",
        encoding='utf-8')


dummyColumns=['症状','职业']

for column in dummyColumns:
    data[column]=data[column].astype('category')
```

```
12
13
14 dummiesData=pandas.get_dummies(
15     data,
16     columns=dummyColumns,
17     prefix=dummyColumns,
18     prefix_sep=' ')
19
20 dummiesData=pandas.get_dummies(
21     data,
22     columns=dummyColumns,
23     prefix=dummyColumns,
24     prefix_sep=' ',
25     drop_first=True)
```

Index	症状	职业	疾病
0	打喷嚏	护士	感冒
1	打喷嚏	农夫	过敏
2	头痛	建筑工人	脑震荡
3	头痛	建筑工人	感冒
4	打喷嚏	教师	感冒
5	头痛	教师	脑震荡

构造虚拟变量

Index	疾病	症状 头痛	症状 打喷嚏	职业 农夫	职业 建筑工人	职业 护士	职业 教师
0	感冒	0	1	0	0	1	0
1	过敏	0	1	1	0	0	0
2	脑震荡	1	0	0	1	0	0
3	感冒	1	0	0	1	0	0
4	感冒	0	1	0	0	0	1
5	脑震荡	1	0	0	0	0	1

drop_first 后

Index	疾病	症状 打喷嚏	职业 建筑工人	职业 护士	职业 教师
0	感冒	1	0	1	0
1	过敏	1	0	0	0
2	脑震荡	0	1	0	0
3	感冒	0	1	0	0
4	感冒	1	0	0	1
5	脑震荡	0	0	0	1

实例

步骤：

- **导入数据，设置虚拟变量，将虚拟变量转变为 category 类，(category 变量)类别变量转换成新增的虚拟变量/指示变量**
- 通过 pandas 自带的 get_dummies 功能，将所有分类扁平化扩张以增加列的形式实现，离散数据按照[0，1]分布。

知识点：

Categorical Type：什么是 categorical Type?不知道确切的英文翻译,但是可以按照字面意思来也就是分类数据,比如皮肤的颜色,可以分为黄色,白色,黑色等等,但是这些数据的均值以及数值计算比如加减的结果是没有意义的;但是我们可以将不同的数据分为这几类,在比如人类的性别,男女也属于 categorical 类别; 英文中欧冠也可以称之为 Nominal Data.

```
1 import pandas;
2
3 data = pandas.read_csv(
4     "C:\\Users\\Jw\\Desktop\\python_work\\Python 数据挖掘实战课程课件\\5.2\\data1.csv",
5     encoding='utf8'
6 )
7
```

```
dummyColumns = ['症状', '职业']

for column in dummyColumns:
    data[column]=data[column].astype('category')

dummiesData = pandas.get_dummies(  #调用 get_dummyColumns 方法进行不可比较大小虚拟变量的转换
    data,
    columns=dummyColumns,
    prefix=dummyColumns,
    prefix_sep=" "
)

dummiesData = pandas.get_dummies(
    data,
    columns=dummyColumns,
    prefix=dummyColumns,
    prefix_sep=" ",
    drop_first=True
)
```

```
#伯努利贝叶斯
from sklearn.naive_bayes import BernoulliNB
BNBModel = BernoulliNB()

fNames = ['症状 打喷嚏', '职业 建筑工人', '职业 护士', '职业 教师']
tData = dummiesData['疾病']
fData = dummiesData[fNames]

BNBModel.fit(fData, tData)
```

上述代码：建模，构造伯努利方程，设置自变量和因变量，训练变量，得到训练集

```
#病症是打喷嚏的建筑工人
newData = pandas.DataFrame({
    '症状':['打喷嚏'],
    '职业':['建筑工人']
```

```
})

for column in dummyColumns:
    newData[column] = newData[column].astype(
        'category',
        categories=data[column].cat.categories
    )

dummiesNewData = pandas.get_dummies(
    newData,
    columns=dummyColumns,
    prefix=dummyColumns,
    prefix_sep=" ",
    drop_first=True
)

pData = dummiesNewData[fNames]
BNBModel.predict(pData)
```

训练：

```
#病症是打喷嚏的建筑工人
newData=pandas.DataFrame({
        '症状':['打喷嚏'],
        '职业':['建筑工人']})

for column in dummyColumns:
    newData[column]=newData[column].astype(
        'category',
        categories=data[column].cat.categories
)

dummiesNewData=pandas.get_dummies(
        newData,
        columns=dummyColumns,
```

```
        prefix=dummyColumns,
        prefix_sep=' ',
        drop_first=True)

pData=dummiesNewData[fNames]
BNBModel.predict(pData)
```

分类: Python 数据挖掘

机器学习实战笔记(Python 实现)-03-朴素贝叶斯

目录

正文

本系列文章为《机器学习实战》学习笔记，内容整理自书本，网络以及自己的理解，如有错误欢迎指正。

源码在 Python3.5 上测试均通过，代码及数据 --> https://github.com/Wellat/MLaction

---

1、算法概述

1.1 朴素贝叶斯

朴素贝叶斯是使用概率论来分类的算法。其中**朴素**：各特征条件独立；**贝叶斯**：根据贝叶斯定理。

根据贝叶斯定理，对一个分类问题，给定样本特征 x，样本属于类别 y 的概率是：

$$p(y|x) = \frac{p(x|y)p(y)}{p(x)} \quad \text{-------（1）}$$

在这里，x 是一个特征向量，设 x 维度为 M。因为朴素的假设，即特征条件独立，根据全概率公式展开，上式可以表达为：

$$p(y = c_k|x) = \frac{\prod_{i=1}^{M} p(x^i|y = c_k)p(y = c_k)}{\sum_k p(y = c_k) \prod_{i=1}^{M} P(x^i|y = c_k)}$$

这里，只要分别估计出，特征 X^i 在每一类的条件概率就可以了。类别 y 的先验概率可以通过训练集算出，同样通过训练集上的统计，可以得出对应每一类上的，条件独立的特征对应的条件概率向量。

1.2 算法特点

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。

2、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。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。

2.1 准备数据：从文本中构建词向量

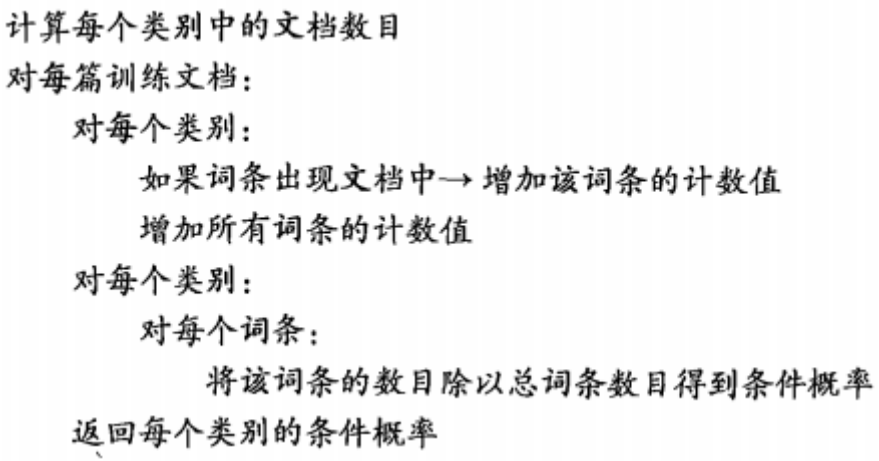
```
1 from numpy import *
2
3 def loadDataSet():
4     '''
5     postingList: 进行词条切分后的文档集合
6     classVec:类别标签
7     '''
8     postingList=[['my', 'dog', 'has', 'flea', 'problems', 'help', 'please'],
```

```
                ['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 代表侮辱性文字，0 代表正常言论
    return postingList,classVec

def createVocabList(dataSet):
    vocabSet = set([])#使用 set 创建不重复词表库
    for document in dataSet:
        vocabSet = vocabSet | set(document) #创建两个集合的并集
    return list(vocabSet)

def setOfWords2Vec(vocabList, inputSet):
    returnVec = [0]*len(vocabList)#创建一个所包含元素都为 0 的向量
    #遍历文档中的所有单词，如果出现了词汇表中的单词，则将输出的文档向量中的对应值设为 1
    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
'''
我们将每个词的出现与否作为一个特征，这可以被描述为词集模型(set-of-words model)。
如果一个词在文档中出现不止一次，这可能意味着包含该词是否出现在文档中所不能表达的某种信息，
这种方法被称为词袋模型(bag-of-words model)。
在词袋中，每个单词可以出现多次，而在词集中，每个词只能出现一次。
为适应词袋模型，需要对函数 setOfWords2Vec 稍加修改，修改后的函数称为 bagOfWords2VecMN
'''
def bagOfWords2VecMN(vocabList, inputSet):
    returnVec = [0]*len(vocabList)
    for word in inputSet:
        if word in vocabList:
            returnVec[vocabList.index(word)] += 1
```

```
43      return returnVec
```

2.2 训练算法：从词向量计算概率

计算每个类别的条件概率，伪代码：




```
1  def trainNB0(trainMatrix,trainCategory):
2      '''
3      朴素贝叶斯分类器训练函数(此处仅处理两类分类问题)
4      trainMatrix:文档矩阵
5      trainCategory:每篇文档类别标签
6      '''
7      numTrainDocs = len(trainMatrix)
8      numWords = len(trainMatrix[0])
9      pAbusive = sum(trainCategory)/float(numTrainDocs)
10     #初始化所有词出现数为1，并将分母初始化为2，避免某一个概率值为0
11     p0Num = ones(numWords); p1Num = ones(numWords)#
12     p0Denom = 2.0; p1Denom = 2.0 #
13     for i in range(numTrainDocs):
14         if trainCategory[i] == 1:
15             p1Num += trainMatrix[i]
16             p1Denom += sum(trainMatrix[i])
17         else:
```

```
            p0Num += trainMatrix[i]
            p0Denom += sum(trainMatrix[i])
    #将结果取自然对数，避免下溢出，即太多很小的数相乘造成的影响
    p1Vect = log(p1Num/p1Denom)#change to log()
    p0Vect = log(p0Num/p0Denom)#change to log()
    return p0Vect,p1Vect,pAbusive
```

2.3 测试算法

分类函数：

```
def classifyNB(vec2Classify, p0Vec, p1Vec, pClass1):
    '''
    分类函数
    vec2Classify:要分类的向量
    p0Vec, p1Vec, pClass1:分别对应 trainNB0 计算得到的 3 个概率
    '''
    p1 = sum(vec2Classify * p1Vec) + log(pClass1)
    p0 = sum(vec2Classify * p0Vec) + log(1.0 - pClass1)
    if p1 > p0:
        return 1
    else:
        return 0
```

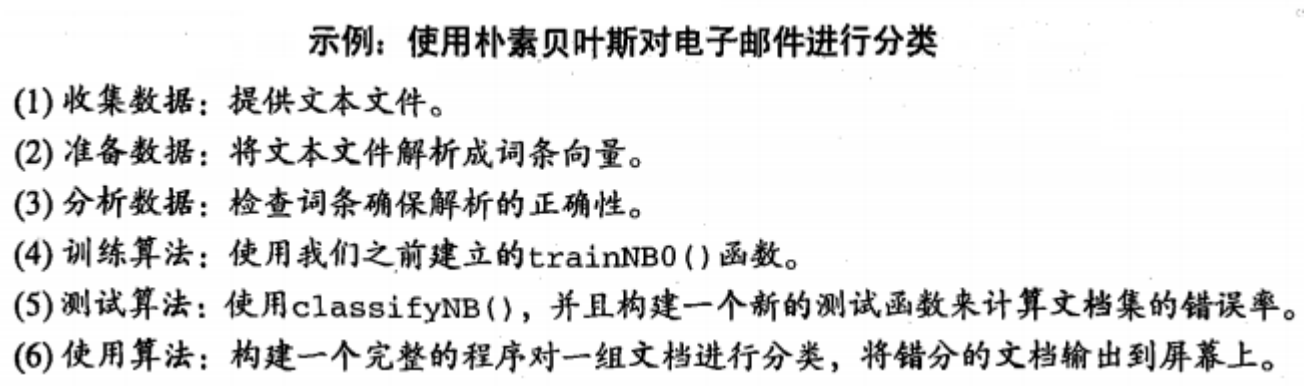
测试：

```
def testingNB():
    listOPosts,listClasses = loadDataSet()
    myVocabList = createVocabList(listOPosts)
    trainMat=[]
    for postinDoc in listOPosts:
        trainMat.append(setOfWords2Vec(myVocabList, postinDoc))
    #训练模型，注意此处使用 array
    p0V,p1V,pAb = trainNB0(array(trainMat),array(listClasses))
    testEntry = ['love', 'my', 'dalmation']
    thisDoc = array(setOfWords2Vec(myVocabList, testEntry))
    print(testEntry,'classified as: ',classifyNB(thisDoc,p0V,p1V,pAb))
```

```
    testEntry = ['stupid', 'garbage']
    thisDoc = array(setOfWords2Vec(myVocabList, testEntry))
    print(testEntry,'classified as: ',classifyNB(thisDoc,p0V,p1V,pAb))
```

3、实例：使用朴素贝叶斯过滤垃圾邮件

一般流程：



3.1 切分文本

将长字符串切分成词表，包括将大写字符转换成小写，并过滤字符长度小于 3 的字符。

```
def textParse(bigString):#
    '''
    文本切分
    输入文本字符串，输出词表
    '''
    import re
    listOfTokens = re.split(r'\W*', bigString)
    return [tok.lower() for tok in listOfTokens if len(tok) > 2]

```

3.2 使用朴素贝叶斯进行垃圾邮件分类

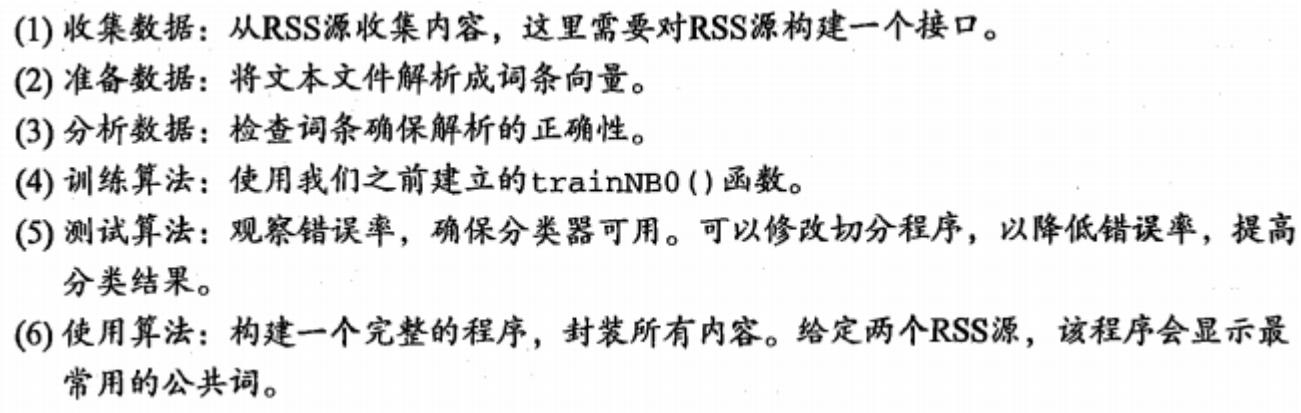
```
def spamTest():
    '''
```

```
    垃圾邮件测试函数
    '''
    docList=[]; classList = []; fullText =[]
    for i in range(1,26):
        #读取垃圾邮件
        wordList = textParse(open('email/spam/%d.txt' % i,'r',encoding= 'utf-8').read())
        docList.append(wordList)
        fullText.extend(wordList)
        #设置垃圾邮件类标签为 1
        classList.append(1)
        wordList = textParse(open('email/ham/%d.txt' % i,'r',encoding= 'utf-8').read())
        docList.append(wordList)
        fullText.extend(wordList)
        classList.append(0)
    vocabList = createVocabList(docList)#生成次表库
    trainingSet = list(range(50))
    testSet=[]           #
    #随机选 10 组做测试集
    for i in range(10):
        randIndex = int(random.uniform(0,len(trainingSet)))
        testSet.append(trainingSet[randIndex])
        del(trainingSet[randIndex])
    trainMat=[]; trainClasses = []
    for docIndex in trainingSet:#生成训练矩阵及标签
        trainMat.append(bagOfWords2VecMN(vocabList, docList[docIndex]))
        trainClasses.append(classList[docIndex])
    p0V,p1V,pSpam = trainNB0(array(trainMat),array(trainClasses))
    errorCount = 0
    #测试并计算错误率
    for docIndex in testSet:
        wordVector = bagOfWords2VecMN(vocabList, docList[docIndex])
        if classifyNB(array(wordVector),p0V,p1V,pSpam) != classList[docIndex]:
            errorCount += 1
            print("classification error",docList[docIndex])
```

```
    print('the error rate is: ',float(errorCount)/len(testSet))
    #return vocabList,fullText
```

4、实例：使用朴素贝叶斯分类器从个人广告中获取区域倾向

一般流程：


(1) 收集数据：从RSS源收集内容，这里需要对RSS源构建一个接口。
(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：观察错误率，确保分类器可用。可以修改切分程序，以降低错误率，提高分类结果。
(6) 使用算法：构建一个完整的程序，封装所有内容。给定两个RSS源，该程序会显示最常用的公共词。

在这个中，我们将分别从美国的两个城市中选取一些人，通过分析这些人发布的征婚广告信息，来比较这两个城市的人们在广告用词上是否不同 。

4.1 实现代码

```
'''
函数 localWords()与程序清单中的 spamTest()函数几乎相同，区别在于这里访问的是
RSS 源而不是文件。然后调用函数 calcMostFreq()来获得排序最高的 30 个单词并随后将它们移除
'''
def localWords(feed1,feed0):
    import feedparser
    docList=[]; classList = []; fullText =[]
    minLen = min(len(feed1['entries']),len(feed0['entries']))
    for i in range(minLen):
        wordList = textParse(feed1['entries'][i]['summary'])
        docList.append(wordList)
        fullText.extend(wordList)
        classList.append(1) #NY is class 1
        wordList = textParse(feed0['entries'][i]['summary'])
```

```
        docList.append(wordList)
        fullText.extend(wordList)
        classList.append(0)
    vocabList = createVocabList(docList)#create vocabulary
    top30Words = calcMostFreq(vocabList,fullText)   #remove top 30 words
    for pairW in top30Words:
        if pairW[0] in vocabList: vocabList.remove(pairW[0])
    trainingSet = list(range(2*minLen)); testSet=[]           #create test set
    for i in range(10):
        randIndex = int(random.uniform(0,len(trainingSet)))
        testSet.append(trainingSet[randIndex])
        del(trainingSet[randIndex])
    trainMat=[]; trainClasses = []
    for docIndex in trainingSet:#train the classifier (get probs) trainNB0
        trainMat.append(bagOfWords2VecMN(vocabList, docList[docIndex]))
        trainClasses.append(classList[docIndex])
    p0V,p1V,pSpam = trainNB0(array(trainMat),array(trainClasses))
    errorCount = 0
    for docIndex in testSet:        #classify the remaining items
        wordVector = bagOfWords2VecMN(vocabList, docList[docIndex])
        if classifyNB(array(wordVector),p0V,p1V,pSpam) != classList[docIndex]:
            errorCount += 1
    print('the error rate is: ',float(errorCount)/len(testSet))
    return vocabList,p0V,p1V

def calcMostFreq(vocabList,fullText):
    '''
    返回前 30 个高频词
    '''
    import operator
    freqDict = {}
    for token in vocabList:
        freqDict[token]=fullText.count(token)
    sortedFreq = sorted(freqDict.items(), key=operator.itemgetter(1), reverse=True)
```

```
49      return sortedFreq[:30]
50
51  if __name__ == "__main__":
52      #导入RSS数据源
53      import operator
54      ny=feedparser.parse('http://newyork.craigslist.org/stp/index.rss')
55      sf=feedparser.parse('http://sfbay.craigslist.org/stp/index.rss')
56      localWords(ny,sf)
```

分类: 机器学习