第六章：概率和朴素贝叶斯

原文：http://guidetodatamining.com/chapter-6

朴素贝叶斯

还是让我们回到运动员的例子。如果我问你Brittney Griner的运动项目是什么，她有6尺8寸高，207磅重，你会说“篮球”；我再问你对此分类的准确度有多少信心，你会回答“非常有信心”。

我再问你Heather Zurich，6尺1寸高，重176磅，你可能就不能确定地说她是打篮球的了，至少不会像之前判定Brittney那样肯定。因为从Heather的身高体重来看她也有可能是跑马拉松的。

最后，我再问你Yumiko Hara的运动项目，她5尺4寸高，95磅重，你也许会说她是跳体操的，但也不太敢肯定，因为有些马拉松运动员也是类似的身高体重。

使用近邻算法时，我们很难对分类结果的置信度进行量化。但如果使用的是基于概率的分类算法——贝叶斯算法——那就可以给出分类结果的可能性了：这名运动员有80%的几率是篮球运动员；这位病人有40%的几率患有糖尿病；拉斯克鲁塞斯24小时内有雨的概率是10%。

近邻算法又称为**被动学习**算法。这种算法只是将训练集的数据保存起来，在收到测试数据时才会进行计算。如果我们有10万首音乐，那每进行一次分类，都需要遍历这10万条记录才行。

贝叶斯算法则是一种**主动学习**算法。它会根据训练集构建起一个模型，并用这个模型来对新的记录进行分类，因此速度会快很多。

所以说，贝叶斯算法的两个优点即：能够给出分类结果的置信度；以及它是一种主动学习算法。

概率

相信大多数人对概率并不陌生。比如投掷一个硬币，正面出现的概率是50%；掷骰子，出现1点的概率是16.7%；从一群19岁的青少年中随机挑出一个，让你说出她是女生的可能性，你会回答50%。以上这些我们用符号P(h)来表示，即事件h发生的概率：

- 投掷硬币：P(正面) = 0.5
- 掷骰子：P(1) = 1/6
- 青少年：P(女生) = 0.5

如果我再告诉你一些额外的信息，比如这群19岁的青少年都是弗兰科学院建筑专业的学生，于是你到Google上搜索后发现这所大学的女生占86%，这时你就会改变你的答案——女生的可能性是86%。

这一情形我们用P(h|D)来表示，即D条件下事件h发生的概率。比如：

P(女生|弗兰克学院的学生) = 0.86

计算的公式是：

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)}$$

再举一个例子，下表是一些人使用笔记本电脑和手机的品牌：

name	laptop	phone
Kate	PC	Android
Tom	PC	Android
Harry	PC	Android
Annika	Mac	iPhone
Naomi	Mac	Android
Joe	Mac	iPhone
Chakotay	Mac	iPhone
Neelix	Mac	Android
Kes	PC	iPhone
B'Elanna	Mac	iPhone

使用iPhone的概率是多少？

$$P(iPhone) = \frac{5}{10} = 0.5$$

如果已知这个人使用的是Mac笔记本，那他使用iPhone的概率是？

$$P(iPhone \mid mac) = \frac{P(mac \cap iPhone)}{P(mac)}$$

首先计算出同时使用Mac和iPhone的概率：

$$P(mac \cap iPhone) = \frac{4}{10} = 0.4$$

使用Mac的概率则是：

$$P(mac) = \frac{6}{10} = 0.6$$

从而计算得到Mac用户中使用iPhone的概率：

$$P(iPhone \mid mac) = \frac{0.4}{0.6} = 0.667$$

以上是正规的解法，不过为了简单起见，我们可以直接通过计数得到：

$$P(\text{iPhone}|\text{mac}) = \frac{\text{同时使用 Mac 和 iPhone 的人数}}{\text{使用 Mac 的人数}}$$

$$P(\text{iPhone}|\text{mac}) = \frac{4}{6} = 0.667$$

练习

iPhone用户中使用Mac的概率是？

$$P(mac \mid iPhone) = \frac{P(iPhone \cap mac)}{P(iPhone)}$$

$$= \frac{0.4}{0.5} = 0.8$$

术语

P(h)表示事件h发生的概率，称为h的**先验概率**。在我们进行任何计算之前就已经得知人们使用Mac的概率是0.6。计算之后我们可能会得知使用Mac的人同时会使用iPhone。

P(h|d)称为**后验概率**，表示在观察了数据集d之后，h事件发生的概率是多少。比如说，我们在观察了使用iPhone的用户后可以得出他们使用Mac的概率是多少。后验概率又称为**条件概率**。

在构建一个贝叶斯分类器前，我们还需要两个概率：P(D)和P(D|h)，请看下面的示例。

微软购物车

你听说过微软的智能购物车吗？没错，他们真有这样的产品。这个产品是微软和一个名为Chaotic Moon的公司合作开发的。这家公司的标语是“我们比你聪明，我们比你有创造力。”你可以会觉得这样的标语有些狂妄自大，这里暂且不谈。这种购物车由以下几个部分组成：Windows 8平板电脑、Kinect体感设备、蓝牙耳机（购物车可以和你说话）、以及电动装置（购物车可以跟着你走）。

你走进一家超市，持有一张会员卡，智能购物车会识别出你，它会记录你的购物记录（当然也包括其他人的）。

智能购物车也会显示广告（比如日本的Sensha绿茶），不过它只会向那些有可能购买此物品的用户进行展示。

以下是一些数据示例：

客户 ID	邮编	是否购买有机食品	是否购买 Sencha 绿茶
1	88005	是	是
2	88001	否	否
3	88001	是	是
4	88005	否	否
5	88003	是	否
6	88005	否	是
7	88005	否	否
8	88001	否	否
9	88005	是	是
10	88003	是	是

P(D)表示从训练集数据中计算得到的概率，比如上表中邮编为88005的概率是：

P(88005) = 0.5

P(D|h)表示在一定条件下的观察结果。比如说购买过Sencha绿茶的人中邮编为88005的概率为：

$$P(88005 \mid SenchaTea) = \frac{3}{5} = 0.6$$

练习

没有买Sencha的人中邮编为88005的概率是？

$$P(88005 \mid \neg SenchaTea) = \frac{2}{5} = 0.4$$

上式中的"¬"表示取反

邮编为88001的概率是？

P(88001) = 0.3

购买了Sencha的人中邮编为88001的概率？

$$P(88001 \mid SenchaTea) = \frac{1}{5} = 0.2$$

没有购买Sencha的人中邮编为88001的概率？

$$P(88001 \mid \neg SenchaTea) = \frac{2}{5} = 0.4$$

贝叶斯法则

贝叶斯法则描述了P(h)、P(h|D)、P(D)、以及P(D|h)这四个概率之间的关系：

$$P(h \mid D) = \frac{P(D \mid h)P(h)}{P(D)}$$

这个公式是贝叶斯方法论的基石。在数据挖掘中，我们通常会使用这个公式去判别不同事件之间的关系。我们可以计算得到在某些条件下这位运动员是从事体操、马拉松、还是篮球项目的；也可以计算得到某些条件下这位客户是否会购买Sencha绿茶等。我们会通过计算不同事件的概率来得出结论。

比如说我们要决定是否给一位客户展示Sencha绿茶的广告，已知他所在的地区邮编是88005。我们有两个相反的假设：

- 这位用户会购买Sencha绿茶的概率，即：P(购买|88005)；
- 不会购买的概率：P(¬ 购买|88005)。

假设我们计算得到P(购买|88005) = 0.6，而P(¬ 购买|88005) = 0.4，则可以认为用户会购买，从而显示相应的广告。

再比如我们要为一家销售电子产品的公司发送宣传邮件，共有笔记本、台式机、平板电

脑三种产品。我们需要根据目标用户的类型来分别派送这三种宣传邮件。比如我们有一位居住在88005地区的女士，她的女儿在读大学，并居住在家中，而且她还会参加瑜伽课程。那我应该派发哪种邮件呢？

让我们用D来表示这位客户的特征：

- 居住在88005地区
- 有一个正在读大学的女儿
- 练习瑜伽

因此我们需要计算以下三个概率：

$$P(\text{笔记本}|D) = \frac{P(D|\text{笔记本})P(\text{笔记本})}{P(D)}$$

$$P(\text{台式机}|D) = \frac{P(D|\text{台式机})P(\text{台式机})}{P(D)}$$

$$P(\text{平板电脑}|D) = \frac{P(D|\text{平板电脑})P(\text{平板电脑})}{P(D)}$$

并选择概率最大的结果。

再抽象一点，如果我们有$h_1, h_2, \ldots h_n$等事件，它们就相当于不同的类别（篮球、体操、或是有没有患糖尿病等）。

$$P(h_1 \mid D) = \frac{P(D \mid h_1)P(h_1)}{P(D)} \quad , \quad P(h_2 \mid D) = \frac{P(D \mid h_2)P(h_2)}{P(D)}$$

$$\ldots \quad P(h_n \mid D) = \frac{P(D \mid h_n)P(h_n)}{P(D)}$$

在计算出以上这些概率后，选取最大的结果，就能用作分类了。这种方法叫最大后验估计，记为h_{MAP}。

我们可以用以下公式来表示最大后验估计：

$$h_{MAP} = \arg\max_{h \in H} P(h \mid D)$$

H表示所有的事件，所以h∈H表示“对于集合中的每一个事件”。整个公式的含义就是：对于集合中的每一个事件，计算出P(h|D)的值，并取最大的结果。使用贝叶斯法则进行替换：

$$h_{MAP} = \arg\max_{h \in H} \frac{P(D \mid h)P(h)}{P(D)}$$

所以我们需要计算的就是以下这个部分：

$$\frac{P(D \mid h)P(h)}{P(D)}$$

可以发现对于所有的事件，公式中的分母都是P(D)，因此即便只计算P(D|h)P(h)，也可以判断出最大的结果。那么这个公式就可以简化为：

$$h_{MAP} = \arg\max_{h \in H} P(D \mid h)P(h)$$

作为演示，我们选取Tom M. Mitchell《机器学习》中的例子。Tom是卡耐基梅隆大学机器学习部的首席，也是非常友好的一个人。这个例子是通过一次血液检验来判断某人是否患有某种癌症。已知这种癌症在美国的感染率是0.8%。血液检验的结果有阳性和阴性两种，且存在准确性的问题：如果这个人患有癌症，则有98%的几率测出阳性；如果他没有癌症，会有97%的几率测出阴性。

我们来尝试将这些描述语言用公式来表示：

- 美国有0.8%的人患有这种癌症：P(癌症) = 0.008
- 99.2%的人没有患有这种癌症：P(¬ 癌症) = 0.992
- 对于患有癌症的人，他的血液检测结果返回阳性的概率是98%：P(阳性|癌症) = 0.98
- 对于患有癌症的人，检测结果返回阴性的概率是2%：P(阴性|癌症) = 0.02
- 对于没有癌症的人，返回阴性的概率是97%：P(阴性|¬ 癌症) = 0.97
- 对于没有癌症的人，返回阳性的概率是3%：P(阳性|¬ 癌症) = 0.03

Ann到医院做了血液检测，呈阳性。初看结果并不乐观，毕竟这种血液检测的准确率高达98%。那让我们用贝叶斯法则来计算看看：

- P(阳性|癌症)P(癌症) = 0.98 * 0.008 = 0.0078
- P(阳性|¬ 癌症)P(¬ 癌症) = 0.03 * 0.992 = 0.0298

分类的结果是她不会患有癌症。

如果想得到确切的概率，我们可以使用标准化的方法：

$$P(\text{癌症}|\text{阳性}) = \frac{0.0078}{0.0078 + 0.0298} = 0.21$$

可以看到，血液检测为阳性的人患有这种癌症的概率是21%。

可能你会觉得这并不合情理，毕竟血液检测的准确率有98%，而结果却说Ann很可能并没有这种癌症。事实上，很多人都会有这样的 疑问。

我来说明一下为什么会是这样的结果。很多人只看到了血液检测的准确率是98%，但没有考虑到全美只有0.8%的人患有这种癌症。假设我们给一个有着一百万人口的城市做血

液检测，也就是说其中有8,000人患有癌症，992,000人没有。首先，对于那8,000个癌症病人，有7,840个人的血液检测结果会呈阳性，160人会呈阴性。对于992,000人，有962,240人会呈阴性，30,000人呈阳性。将这些数字总结到下表中：

	阳性	阴性
患有癌症	7,840	160
健康	30,000	962,240

Ann的测试结果呈阳性，从上表看阳性中有30,000人其实是健康的，只有7,840人确实患有癌症，所以我们才会认为Ann很有可能是健康的。

还是没弄明白？没关系，在接触了更多练习后就会慢慢理解了。

为什么我们需要贝叶斯法则？

首先回顾一下贝叶斯公式：

$$P(h \mid D) = \frac{P(D \mid h)P(h)}{P(D)}$$

再看看微软购物车的数据：

客户 ID	邮编	是否购买有机食品	是否购买 Sencha 绿茶
1	88005	是	是
2	88001	否	否
3	88001	是	是
4	88005	否	否
5	88003	是	否
6	88005	否	是
7	88005	否	否
8	88001	否	否
9	88005	是	是
10	88003	是	是

比如，我们为居住在邮编为88005地区的客户设置两个事件：买或不买Sencha绿茶，即：

$P(h_1|D)$ = P(买绿茶|88005)

$P(h_2|D)$ = P(¬ 买绿茶|88005)

你也许会问，这两个概率我们都可以直接从数据中计算得到，为什么还要计算下面这个公式呢？

$$\frac{P(88005|买绿茶)P(买绿茶)}{P(88005)}$$

那是因为在现实问题中要计算P(h|D)往往是很困难的。

以上一节中的医学示例来说，我们想要根据血液测试结果来判断该人是否患有癌症：

P(癌症|阳性) ≈ P(阳性|癌症)P(癌症)

P(┐癌症|阳性) ≈ P(阳性|┐癌症)P(┐癌症)

上面两个式子中，我们更容易计算约等号右边的结果。比如，要计算P(阳性|癌症)，我们可以对一部分已经确诊有癌症的人做血液测试；计算P(阳性|┐癌症)时则可对确定健康的人做测试。而P(癌症)则可直接从政府公布的官方数据中获得，P(┐癌症)更是简单的1 - P(癌症)。

但若要计算P(癌症|阳性）的话就非常有挑战性了，也就是计算出整个人群中血液测试结果呈阳性且确诊为癌症患者的概率。如果采用抽样法，对1000个人进行抽样测试，只有8个人患有这种癌症，这样得出的结果显然不具有代表性，除非进一步加大抽样数量。而贝叶斯法则提供了更为简便的方法。

朴素贝叶斯

很多时候我们会用到不止一个前提条件，比如判断一个人是否会购买Sencha绿茶时可以用到顾客所在地以及是否买过有机食品这两个条件。计算这样的概率时只需将各个条件概率相乘即可：

P(买绿茶|88005 & 买有机食品) = P(88005|买绿茶)P(买有机食品|买绿茶)P(买绿茶) = 0.6 * 0.8 * 0.5 = 0.24

P(┐买绿茶|88005 & 买有机食品) = P(88005|┐买绿茶)P(买有机食品|┐买绿茶)P(┐买绿茶) = 0.4 * 0.25 * 0.5 = 0.05

所以得到的结论是居住在88005地区且买过有机食品的客户更有可能购买Sencha绿茶。这样就让我们在智能购物车上显示广告吧！

以下是史提芬贝克对智能购物车的评价：

> 这种购物车的使用体验是：你取走一辆购物车，刷了会员卡，屏幕上会显示一份购物列表，里面的内容都是基于你平时的购物习惯进行推荐的，牛奶、鸡蛋、西葫芦等等。智能系统会提示出购买这些物品的最佳路径。另外，它还允许你修改列表中的商品，比如你可以让它不要再提示菜花和盐焗花生。Accenture的研究表明，人们在购物时会忘记11%的原本打算购买的商品，如果有了这样的智能购物车，就可以省去客户的来回路程，也能为超市增加销量。

i100、i500健康手环

现在我们要为iHealth公司销售健康手环产品，从而和Nike Fuel、Fitbit Flex竞争。iHealth新出产了两件商品：i100和i500：

iHealth 100

能够监测心率，使用GPS导航（从而计算每小时运动公里数等），带WiFi无线，可随时上传数据到iHealth网站上。

iHealth 500

除了提供i100的功能外，还能监测血液含氧量等指标，且提供免费的3G网络连接到iHealth网站。

这些产品通过网络平台销售，所以iHealth雇佣我们开发一套推荐系统。我通过让购买的用户填写调查问卷来收集数据，每个问题都对应一个特征。比如，我们会问客户为什么要开始运动，有三个选项：健康（health）、外表（appearance）、两者皆是（both）；我们会问他目前的运动水平：很少运动（sedentary）、一般（moderate）、经常运动（active）；我们会问他对健身的热情是高（aggressive）还是一般（moderate）；最后，我们会问他是否适应使用高科技产品。整理后的数据如下：

Main Interest	Current Exercise Level	How Motivated	Comfortable with tech. Devices?	Model #
both	sedentary	moderate	yes	i100
both	sedentary	moderate	no	i100
health	sedentary	moderate	yes	i500
appearance	active	moderate	yes	i500
appearance	moderate	aggressive	yes	i500
appearance	moderate	aggressive	no	i100
health	moderate	aggressive	no	i500
both	active	moderate	yes	i100
both	moderate	aggressive	yes	i500
appearance	active	aggressive	yes	i500
both	active	aggressive	no	i500
health	active	moderate	no	i500
health	sedentary	aggressive	yes	i500
appearance	active	moderate	no	i100
health	sedentary	moderate	no	i100

实践

已知一位客户的运动目的是健康、当前水平是中等、热情一般、能适应高科技产品，请用朴素贝叶斯来推荐手环型号。

我们需要计算以下两个概率，并选取较大的结果：

P(i100|健康，中等水平、热情一般，适应）
P(i500|健康，中等水平，热情一般，适应）

我们先来看第一个概率：

P(i100|健康，中等水平、热情一般，适应） = P(健康|i100)P(中等水平|i100)P

其中：

P(健康|i100) = 1/6
P(中等水平|i100) = 1/6
P(热情一般|i100) = 5/6
P(适应|i100) = 2/6

```
P(i100) = 6/15
```

因此：

```
P(i100|满足条件) = 0.167 * 0.167 * 0.833 * 0.333 * 0.4 = 0.00309
```

再计算另一个模型的概率：

```
P(i500|满足条件) = P(健康|i500)P(中等水平|i500)P(热情一般|i500)P(适应
    = 4/9 * 3/9 * 3/9 * 6/9 * 9/15
    = 0.444 * 0.333 * 0.333 * 0.667 * 0.6
    = 0.01975
```

使用Python编写朴素贝叶斯分类器

上例的数据格式如下：

```
both    sedentary   moderate    yes i100
both    sedentary   moderate    no  i100
health  sedentary   moderate    yes i500
appearance  active  moderate    yes i500
appearance  moderate    aggressive  yes i500
appearance  moderate    aggressive  no  i100
health  moderate    aggressive  no  i500
both    active  moderate    yes i100
both    moderate    aggressive  yes i500
appearance  active  aggressive  yes i500
both    active  aggressive  no  i500
health  active  moderate    no  i500
health  sedentary   aggressive  yes i500
appearance  active  moderate    no  i100
health  sedentary   moderate    no  i100
```

虽然这个这个例子中只有15条数据，但是我们还是保留十折交叉验证的过程，以便用于更大的数据集。十折交叉验证要求数据集等分成10份，这个例子中我们简单地将15条数据全部放到一个桶里，其它桶留空。

朴素贝叶斯分类器包含两个部分：训练和分类。

训练

训练的输出结果应该是：

- 先验概率，如P(i100) = 0.4；
- 条件概率，如P(健康|i100) = 0.167

我们使用如下代码表示先验概率：

```
self.prior = {'i500': 0.6, 'i100': 0.4}
```

条件概率的表示有些复杂，用嵌套的字典来实现：

```
{'i500': {1: {'appearance': 0.3333333333333333, 'health': 0.4444
              'both': 0.2222222222222222},
          2: {'active': 0.4444444444444444, 'sedentary': 0.22222
              'moderate': 0.3333333333333333},
          3: {'aggressive': 0.6666666666666666, 'moderate': 0.33
          4: {'yes': 0.6666666666666666, 'no': 0.333333333333333
 'i100': {1: {'both': 0.5, 'health': 0.16666666666666666,
              'appearance': 0.3333333333333333},
          2: {'active': 0.3333333333333333, 'sedentary': 0.5,
              'moderate': 0.16666666666666666},
          3: {'aggressive': 0.16666666666666666, 'moderate': 0.8
          4: {'yes': 0.3333333333333333, 'no': 0.666666666666666
```

1、2、3、4表示第几列，所以第一行可以解释为购买i500的顾客中运动目的是外表的概率是0.333。

首先我们要来进行计数，比如以下几行数据：

```
both    sedentary   moderate    yes i100
both    sedentary   moderate    no  i100
health  sedentary   moderate    yes i500
appearance  active  moderate    yes i500
```

我们用字典来统计每个模型的次数，变量名为classes，逐行扫描后的结果是：

```
# 第一行
{'i100': 1}

# 第二行
{'i100': 2}

# 第三行
{'i500': 1, 'i100': 2}

# 全部
{'i500': 9, 'i100': 6}
```

要获取模型的先验概率，只要将计数结果除以总数就可以了。

计算后验概率也需要计数，变量名为counts。这个字典较为复杂，如扫完第一行第一列的结果是：

```
{'i100': {1: {'both': 1}}
```

处理完所有数据后的计数结果是：

```
{'i500': {1: {'appearance': 3, 'health': 4, 'both': 2},
          2: {'active': 4, 'sedentary': 2, 'moderate': 3},
          3: {'aggressive': 6, 'moderate': 3},
          4: {'yes': 6, 'no': 3}},
 'i100': {1: {'both': 3, 'health': 1, 'appearance': 2},
          2: {'active': 2, 'sedentary': 3, 'moderate': 1},
          3: {'aggressive': 1, 'moderate': 5},
          4: {'yes': 2, 'no': 4}}}
```

计算概率时，只需将计数除以该模型的总数就可以了：

P(外表|i100) = 2 / 6 = 0.333

以下是训练用的Python代码：

```
class Classifier:

    def __init__(self, bucketPrefix, testBucketNumber, dataForma
        """bucketPrefix 分桶数据集文件前缀
        testBucketNumber 测试桶编号
        dataFormat 数据格式，形如：attr attr attr attr class
        """

        total = 0
        classes = {}
        counts = {}

        # 从文件中读取数据
        self.format = dataFormat.strip().split('\t')
        self.prior = {}
        self.conditional = {}
        # 遍历十个桶
        for i in range(1, 11):
            # 跳过测试桶
            if i != testBucketNumber:
                filename = "%s-%02i" % (bucketPrefix, i)
                f = open(filename)
                lines = f.readlines()
                f.close()
                for line in lines:
                    fields = line.strip().split('\t')
                    ignore = []
                    vector = []
                    for i in range(len(fields)):
                        if self.format[i] == 'num':
                            vector.append(float(fields[i]))
                        elif self.format[i] == 'attr':
                            vector.append(fields[i])
```

```
                    elif self.format[i] == 'comment':
                        ignore.append(fields[i])
                    elif self.format[i] == 'class':
                        category = fields[i]
                # 处理该条记录
                total += 1
                classes.setdefault(category, 0)
                counts.setdefault(category, {})
                classes[category] += 1
                # 处理各个属性
                col = 0
                for columnValue in vector:
                    col += 1
                    counts[category].setdefault(col, {})
                    counts[category][col].setdefault(columnV
                    counts[category][col][columnValue] += 1

        # 计数结束，开始计算概率

        # 计算先验概率P(h)
        for (category, count) in classes.items():
            self.prior[category] = count / total

        # 计算条件概率P(h|D)
        for (category, columns) in counts.items():
              self.conditional.setdefault(category, {})
              for (col, valueCounts) in columns.items():
                  self.conditional[category].setdefault(col, {})
                  for (attrValue, count) in valueCounts.items():
                      self.conditional[category][col][attrValue]
                          count / classes[category])
        self.tmp =  counts
```

分类

分类函数会这样使用：

```
c.classify(['health', 'moderate', 'moderate', 'yes'])
```

我们需要计算：

$$h_{MAP} = \arg\max_{h \in H} P(D \mid h)P(h)$$

```
    def classify(self, itemVector):
        """返回itemVector所属类别"""
        results = []
        for (category, prior) in self.prior.items():
            prob = prior
```

```
            col = 1
            for attrValue in itemVector:
                if not attrValue in self.conditional[category][c
                    # 属性不存在，返回0概率
                    prob = 0
                else:
                    prob = prob * self.conditional[category][col
                col += 1
            results.append((prob, category))
        # 返回概率最高的结果
        return(max(results)[1])
```

让我们测试一下：

```
>>> c = Classifier('iHealth/i', 10, 'attr\tattr\tattr\tattr\tcla
>>> c.classify(['health' 'moderate', 'moderate', 'yes'])
i500
```

共和党还是民主党

我们来看另一个数据集——美国国会投票数据，可以从 机器学习仓库 获得。每条记录代表一个选民，第一列是分类名称（democrat, republican），之后是16条法案，用y和n表示该人是否支持。

1. 残疾婴儿法案
2. 用水成本分摊
3. 预算改革
4. 医疗费用
5. 萨瓦尔多援助
6. 校园宗教组织
7. 反卫星武器
8. 尼加拉瓜援助
9. MX导弹

10. 移民法案
11. 合成燃料缩减
12. 教育支出法案
13. 有毒废物基金
14. 犯罪
15. 出口免税
16. 南非出口管控

文件格式如下：

```
democrat    y   n   y   n   n   y   y   y   y   y   n   n   y
democrat    y   y   y   n   n   y   y   y   y   n   n   n   n
republican  y   y   n   y   y   y   n   n   n   y   n   y   y
```

在调用上一节编写的朴素贝叶斯分类器时使用以下dataFormat参数就可以了：

```
"class\tattr\tattr\tattr\tattr\tattr\tattr\tattr\tattr\tattr\tat
```

十折交叉验证的结果是：

```
            Classified as:
               democrat   republican
               +-------+-------+
   democrat    |   111 |    13 |
               |-------+-------|
 republican    |     9 |    99 |
               +-------+-------+

90.517 percent correct
total of 232 instances
```

看起来不错。

但是，这个方法有一些问题。

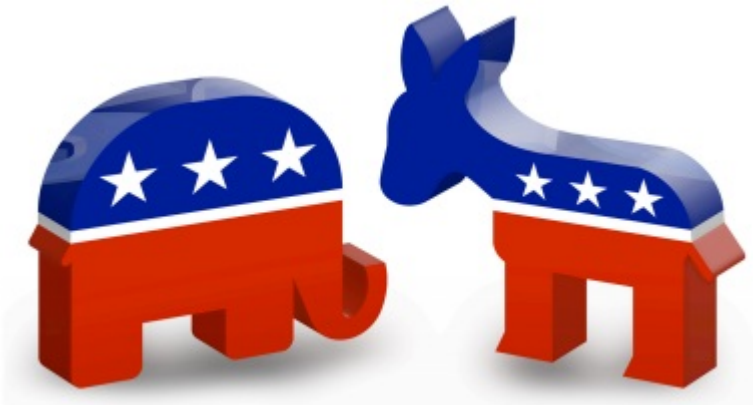
首先我们来模拟一个数据集，其中包含100个民主党派和100个共和党派。下表是他们对四个法案的投票情况：

	网络安全法案	读者隐私	网络营销税负	非法网络传播
共和党	0.99	0.01	0.99	0.5
民主党	0.01	0.99	0.01	1.0

从表中可以看到，共和党人中，99%赞成网络安全法案，只有1%的人赞成读者隐私。下面我们再选取一位议会代表——X先生，对他进行分类：

	网络安全法案	读者隐私	网络营销税负	非法网络传播
共和党	0.99	0.01	0.99	0.5
民主党	0.01	0.99	0.01	1.0
X 先生	否决	赞成	否决	否决

你觉得X先生是民主党还是共和党呢？



我会猜测民主党。我们用朴素贝叶斯来进行分类。首先，先验概率P(民主党)和P(共和党)都是0.5，因为样本中两党分别有100人。X先生对网络安全法案投了否决票，并且：

```
P(共和党|C=no) = 0.01
P(民主党|C=no) = 0.99
```

为了表示方便，我们用字母来表示这四个法案：

1. 网络安全：C
2. 读者隐私：R
3. 网销税负：T
4. 非法传播：S

记到表格里就是：

h=	P(h)	P(C=no|h)				P(h|D)
共和党	0.5	0.01				0.005
民主党	0.5	0.99				0.495

我们将X先生对读者隐私和网络营销税负法案的投票记到表格中：

h=	P(h)	P(C=no|h)	P(R=yes|h)	P(T=no|h)		P(h|D)
共和党	0.5	0.01	0.01	0.01		0.0000005
民主党	0.5	0.99	0.99	0.99		0.485

对这些概率进行标准化后可以得到：

$$P(\text{民主党}|D) = \frac{0.485}{0.485 + 0.0000005} = \frac{0.485}{0.4850005} = 0.99999$$

也就是说到目前为止我们有99.99%的信心认为X先生是民主党的。

最后，我们将X先生对非法传播法案的投票记入表格：

h=	P(h)	P(C=no|h)	P(R=yes|h)	P(T=no|h)	P(S=no|h)	P(h|D)
共和党	0.5	0.01	0.01	0.01	0.50	2.5E-07
民主党	0.5	0.99	0.99	0.99	0.00	0

你会惊讶地发现，X先生是民主党的可能性从99%降至0了！这是因为我们的数据集中没有一个民主党人对网络非法传播法案投了否决票。

概率估计

使用朴素贝叶斯计算得到的概率其实是真实概率的一种**估计**，而真实概率是对全量数据做统计得到的。比如说，我们需要对所有人都做血液测试，才能得到健康人返回阴性结果的真实概率。显然，对全量数据做统计是不现实的，所以我们会选取一个样本，如1000人，对他们进行测试并计算概率。大部分情况下，这种估计都是接近于真实概率

的。但当真实概率非常小时，这种抽样统计的做法就会有问题了。

比如说，民主党对网络非法传播法案的否决率是0.03，即P(S=no|民主党) = 0.03。如果我们分别选取十个民主党和共和党人，看他们对该法案的投票情况，你觉得得到的概率会是什么？答案很可能是0。

从上一节的例子中也看到了，在朴素贝叶斯中，概率为0的影响是很大的，甚至会不顾其他概率的大小。此外，抽样统计的另一个问题是会低估真实概率。

如何解决

我们计算P(S=no|民主党)的公式是这样的：

$$P(S=\text{no}|\text{民主党}) = \frac{\text{民主党中对网络非法传播法案投否决票的人数}}{\text{民主党总人数}}$$

为了表示方便，我们采用以下公式：

$$P(x \mid y) = \frac{n_c}{n}$$

其中，n表示训练集中y类别的记录数；n_c表示y类别中值为x的记录数。

我们的问题是上式中的n_c可能为0，解决方法是将公式变为以下形式：

$$P(x \mid y) = \frac{n_c + mp}{n + m}$$

这个公式摘自Tom Mitchell《机器学习》的第179页。

m是一个常数，表示等效样本大小。决定常数m的方法有很多，我们这里使用值的类别来作为m，比如投票有赞成和否决两种类别，所以m就为2。p则是相应的先验概率，比如说赞成和否决的概率分别是0.5，那p就是0.5。

我们回到上面的例子，看看要如何应用这个方法。下表是投票的情况：

共和党

	网络安全	读者隐私	网销税负	非法传播
赞成	99	1	99	50
否决	1	99	1	50

民主党

	网络安全	读者隐私	网销税负	非法传播
赞成	1	99	1	0
否决	99	1	99	100

X先生对网络安全法案投了否决票，我们先来计算共和党对安全法案投否决票的概率。

新的公式是：

$$P(x \mid y) = \frac{n_c + mp}{n + m}$$

其中n是共和党的人数100，n_c是共和党对安全法案投否决票的人数1，m是2，p是0.5，因此：

$$P(C = no|共和党) = \frac{1 + 2 \times 0.5}{100 + 2} = \frac{2}{102} = 0.01961$$

再计算民主党：

$$P(C = no|民主党) = \frac{99 + 2 \times 0.5}{100 + 2} = \frac{100}{102} = 0.9804$$

代入之前用到的表格中：

h=	P(h)	P(C=no|h)				P(h|D)
共和党	0.5	0.01961				0.0098
民主党	0.5	0.9804				0.4902

对剩余三条法案进行计算，得到的结果是：

h=	P(h)	P(C=no|h)	P(R=yes|h)	P(I=no|h)	P(S=no|h)	P(h|D)
共和党	0.5	0.01961	0.01961	0.01961	0.5	0.000002
民主党	0.5	0.9804	0.9804	0.9804	0.0098	0.004617

因此，X先生的党派是民主党，这和我们的直觉一致。

一点说明

在这个例子中，所有公式里的m都是2，但这并不表示其他数据集也是这样。比如我们之前做的健康手环问卷调查，运动目的有三个选项，是否适应高科技则有两个选项，所以在计算运动目的概率时m=3、p=1/3，代入公式即：

$$P(很少运动|i500) = \frac{n_c + mp}{n + m} = \frac{0 + 3 \times 0.333}{100 + 3} = \frac{1}{103} = 0.0097$$

数值型数据

你可能已经注意到，在讨论近邻算法时，我们使用的都是数值型的数据，而在学习朴素贝叶斯算法时，用的是分类型的数据。比如，人们对法案的投票有赞成和否决两类；音乐家可以用他们演奏的乐器来分类等等。这些分类之间是没有距离的，萨克斯手和钢琴家的距离并不会比鼓手近。而数值型数据则有这种远近之分。

在贝叶斯方法中，我们会对事物进行计数，这种计数则是可以度量的。对于数值型的数

据要如何计数呢？通常有两种做法：

方法一：区分类别

我们可以划定几个范围作为分类，如：

- 年龄
  - < 18
  - 18 - 22
  - 23 - 30
  - 31 - 40
  - 40
- 年薪
  - $200,000
  - 150,000 - 200,000
  - 100,000 - 150,000
  - 60,000 - 100,000
  - 40,000 - 60,000

划分类别后，进行可以应用朴素贝叶斯方法了。

方法二：高斯分布

我想将收入数据进行分类，然后应用朴素贝叶斯算法。

你的做法已经过时了，我会使用高斯分布和概率密度函数来做。

高斯分布和概率密度函数这两个词听起来很酷，他们的作用也非常大。下面我们将学习如何在朴素贝叶斯算法中使用高斯分布。

首先，我们为健康手环的例子增加一列收入属性：

Main Interest	Current Exercise Level	How Motivated	Comfortable with tech. Devices?	Income (in $1,000)	Model #
both	sedentary	moderate	yes	60	i100
both	sedentary	moderate	no	75	i100
health	sedentary	moderate	yes	90	i500
appearance	active	moderate	yes	125	i500
appearance	moderate	aggressive	yes	100	i500
appearance	moderate	aggressive	no	90	i100
health	moderate	aggressive	no	150	i500
both	active	moderate	yes	85	i100
both	moderate	aggressive	yes	100	i500
appearance	active	aggressive	yes	120	i500
both	active	aggressive	no	95	i500
health	active	moderate	no	90	i500
health	sedentary	aggressive	yes	85	i500
appearance	active	moderate	no	70	i100
health	sedentary	moderate	no	45	i100

我们来看购买i500的用户的收入情况，比如取平均值：

$$mean = \frac{90+125+100+150+100+120+95+90+85}{9} = \frac{955}{9} = 106.111$$

还可以求出标准差：

$$sd = \sqrt{\frac{\sum_i (x_i - \overline{x})^2}{card(x)}}$$

标准差是用来衡量数据的离散程度的，如果所有数据都接近于平均值，那标准差也会比较小。通过公式我们可以计算得到i500用户的收入标准差是20.108。

总体标准差和样本标准差

上面的标准差公式是总体标准差，我们需要对所有的数据进行统计才能得出，比如统计

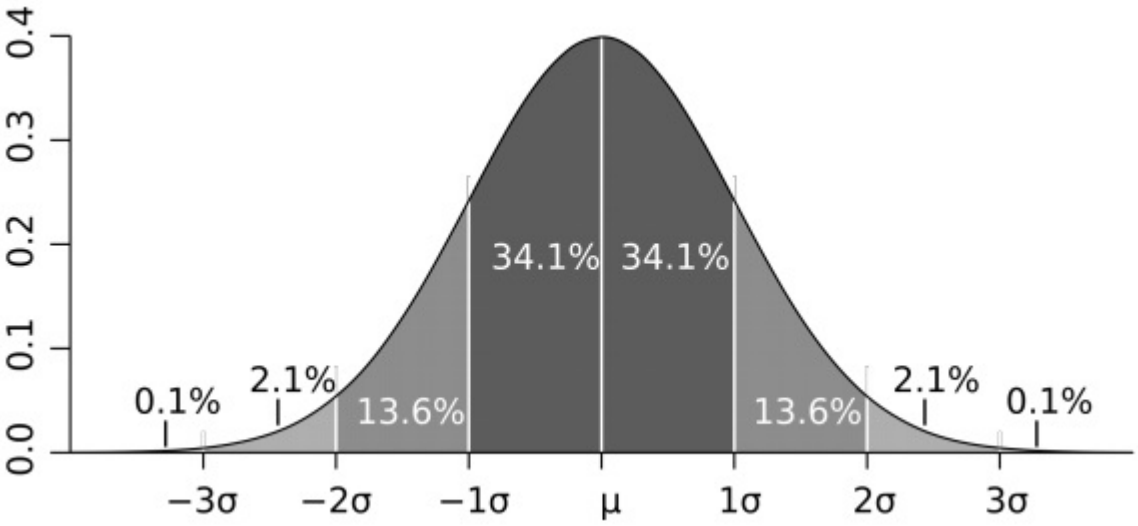
500名学生的成绩，就能计算出总体标准差。但通常我们无法获取总体的数据，比如要统计新墨西哥北部鹿鼠的重量，我们不可能对所有的鹿鼠进行称重，只能选取一部分样本，这时计算得到的就是样本标准差。

样本标准差的公式是：

$$sd = \sqrt{\frac{\sum_i (x_i - \bar{x})^2}{card(x) - 1}}$$

所以计算得到i500用户收入的样本标准差是21.327。下面的内容都会使用样本标准差。

你可能听说过正态分布、钟型曲线、高斯分布等术语，他们指的是同一件事：68%的数据会落在标准差为1的范围内，95%的数据会落在标准差为2的范围内：



在我们的示例中，平均值是106.111，样本标准差是21.327，因此购买i500的用户中有95%的人收入在42,660美元至149,770美元之间。如果我问你P(100k|i500)的概率有多大，你可以回答非常大；如果问你P(20k|i500)的概率有多大，你可以回答基本不可能。

用公式来表示是：

$$P(x_i \mid y_j) = \frac{1}{\sqrt{2\pi}\sigma_{ij}} e^{\frac{-(x_i-\mu_{ij})^2}{2\sigma_{ij}^2}}$$

每次出现这些公式时我都想提醒读者千万不要紧张，其实他们只是看起来比较复杂，只需一步一步拆解开就能理解了。数据挖掘学到后面会遇到各种复杂的公式，千万不要被他们的外表吓到。

让我们逐步拆解这个公式。假设我们要计算P(100k|i500)的概率，即购买i500的用户中收入是100,000美元的概率。之前我们计算过购买i500的用户平均收入以及样本标准差，我们用希腊字母μ（读“谬”）来表示平均值，σ（读“西格玛”）来表示标准差。

$$P(x_i \mid y_j) = \frac{1}{\sqrt{2\pi}\sigma_{ij}} e^{\frac{-(x_i-\mu_{ij})^2}{2\sigma_{ij}^2}}$$

$\mu_{ij} = 106.111$
$\sigma_{ij} = 21.327$
$x_i = 100$

$$P(x_i \mid y_j) = \frac{1}{\sqrt{2\pi}(21.327)} e^{\frac{-(100-106.111)^2}{2(21.327)^2}}$$

$$P(x_i \mid y_j) = \frac{1}{\sqrt{6.283}(21.327)} e^{\frac{-(37.344)}{909.68}}$$

$$P(x_i \mid y_j) = \frac{1}{53.458} e^{-0.0411}$$

e是自然常数，约等于2.718。

$$P(x_i \mid y_j) = \frac{1}{53.458}(2.718)^{-0.0411} = (0.0187)(0.960) = 0.0180$$

练习

下表中列出了加仑公里数为35的车型的马力（HP），现在我想知道同样是35加仑公里的Datsun 280z马力为132的概率。

car	HP
Datsun 210	65
Ford Fiesta	66
VW Jetta	74
Nissan Stanza	88
Ford Escort	65
Triumph tr7 coupe	88
Plymouth Horizon	70
Suburu DL	67

$\mu_{ij} = 72.875$

$\sigma_{ij} = 9.804$

$x_i = 132$

$$P(132hp \mid 35mpg) = \frac{1}{\sqrt{2\pi}(9.804)} e^{\frac{-(132-72.875)^2}{2(9.804)^2}}$$

$$= \frac{1}{\sqrt{6.283}(9.804)} e^{\frac{-3495.766}{192.237}} = \frac{1}{24.575} e^{-18.185}$$
$$= 0.0407(0.00000001266)$$
$$= 0.0000000005152$$

结果表明这个概率非常低，而事实上这辆车就是132马力。

代码实现提示

在训练朴素贝叶斯分类器时，可以讲所有属性的平均值和样本标准差计算出来，而分类阶段使用下面这段代码就能实现了：

```
import math

def pdf(mean, ssd, x):
    """概率密度函数，计算P(x|y)"""
    ePart = math.pow(math.e, -(x - mean) ** 2 / (2 * ssd ** 2))
    return (1.0 / (math.sqrt(2 * math.pi) * ssd)) * ePart
```

测试一下：

```
>>> pdf(106.111, 21.327, 100)
```

```
0.017953602706962717
>>> pdf(72.875, 9.804, 132)
5.152283971078022e-10
```

休息一下吧

使用Python实现

训练阶段

朴素贝叶斯需要用到先验概率和条件概率。让我们回顾一下民主党和共和党的例子：先验概率指的是我们已经掌握的概率，比如美国议会中有233名共和党人，200名民主党人，那共和党人出现的概率就是：

P(共和党) = 233 / 433 = 0.54

我们用P(h)来表示先验概率。而条件概率P(h|D)则表示在已知D的情况下，事件h出现的概率。比如说P(民主党|法案1=yes)。朴素贝叶斯公式中，我们计算的是P(D|h)，如P(法案1=yes|民主党)。

在之前的Python代码中，我们用字典来表示这些概率：

```
{'democrat': {'bill 1': {'yes': 0.333, 'no': 0.667},
              'bill 2': {'yes': 0.778, 'moderate': 0.222}},
 'republican': {'bill 1': {'yes': 0.811, 'no': 0.189},
                'bill 2': {'yes': 0.250, 'no': 0.750}}}
```

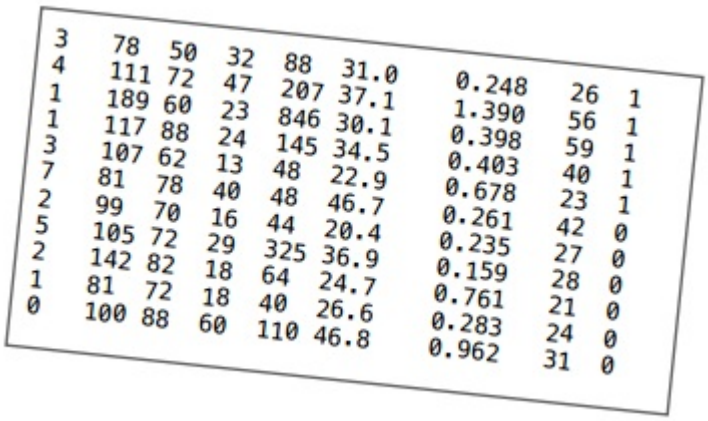
所以民主党中对法案1投赞成票的概率是：P(bill 1=yes|民主党) = 0.667。

对于分类型的数据，我们用上面的方法来保存概率，而对连续性的数据，我们要使用概率密度函数，因此需要保存平均值和样本标准差。如：

```
mean = {'democrat': {'age': 57, 'years served': 12},
        'republican': {'age': 53, 'years served': 7}}
ssd = {'democrat': {'age': 7, 'years served': 3},
```

```
        'republican': {'age': 5, 'years served': 5}}
```

和之前一样，数据文件中的每一行表示一条记录，不同的特征值使用制表符分隔，比如下面是比马印第安人糖尿病的数据：



```
3   78  50  32  88  31.0    0.248   26  1
4   111 72  47  207 37.1    1.390   56  1
1   189 60  23  846 30.1    0.398   59  1
1   117 88  24  145 34.5    0.403   40  1
3   107 62  13  48  22.9    0.678   23  1
7   81  78  40  48  46.7    0.261   42  0
2   99  70  16  44  20.4    0.235   27  0
5   105 72  29  325 36.9    0.159   28  0
2   142 82  18  64  24.7    0.761   21  0
1   81  72  18  40  26.6    0.283   24  0
0   100 88  60  110 46.8    0.962   31  0
```

前八列是特征，最后一列是分类（1-患病，0-健康）。

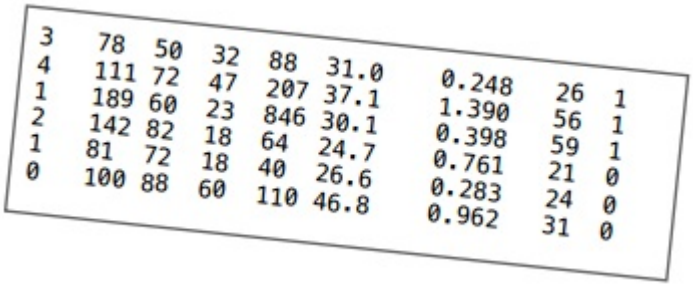
我们同样需要一个格式字符串来表示每一行记录：

- `attr`表示这一列是分类型的特征
- `num` 表示这一列是数值型的特征
- `class` 表示这一列是分类

对于比马数据集，格式化字符串是：

```
"num     num     num     num     num     num     num     num     class"
```

我们需要一个数据结构来存储平均值和样本标准差，看下面这几行数据：



```
3   78  50  32  88  31.0    0.248   26  1
4   111 72  47  207 37.1    1.390   56  1
1   189 60  23  846 30.1    0.398   59  1
2   142 82  18  64  24.7    0.761   21  0
1   81  72  18  40  26.6    0.283   24  0
0   100 88  60  110 46.8    0.962   31  0
```

为计算每一个分类的平均值，我们需要保存合计值，可以用字典来实现：

```
totals = {'1': {1: 8, 2: 378, 3: 182, 4: 102, 5: 1141,
                6: 98.2, 7: 2.036, 8: 141},
          '0': {1: 3, 2: 323, 3: 242, 4: 96, 5: 214,
                6: 98.1, 7: 2.006, 8: 76}}
```

对于分类1，第一列的合计是8（3 + 4 + 1），第二列的合计是378。

对于分类0，第一列的合计是3（2 + 1 + 0），第二列的合计是323，以此类推。

在计算标准差时，我们还需要保留原始的值：

```
numericValues = {'1': {1: [3, 4, 1], 2: [78, 111, 189], ...},
                 '0': {1: [2, 1, 0], 2: [142, 81, 100], ...}}
```

将这些逻辑添加到分类器的__init__方法中：

```
import math

class Classifier:
    def __init__(self, bucketPrefix, testBucketNumber, dataForma
        """bucketPrefix 分桶数据集文件前缀
        testBucketNumber 测试桶编号
        dataFormat 数据格式，形如：attr attr attr attr class
        """
        total = 0
        classes = {}
        # 对分类型数据进行计数
        counts = {}
        # 对数值型数据进行求和
        # 我们会使用下面两个变量来计算每个分类各个特征的平均值和样本标准差
        totals = {}
        numericValues = {}

        # 从文件中读取数据
        self.format = dataFormat.strip().split('\t')
        self.prior = {}
        self.conditional = {}

        # 遍历1-10号桶
        for i in range(1, 11):
            # 判断是否跳过
            if i != testBucketNumber:
                filename = "%s-%02i" % (bucketPrefix, i)
                f = open(filename)
                lines = f.readlines()
                f.close()
                for line in lines:
                    fields = line.strip().split('\t')
                    ignore = []
                    vector = []
                    nums = []
                    for i in range(len(fields)):
                        if self.format[i] == 'num':
                            nums.append(float(fields[i]))
                        elif self.format[i] == 'attr':
                            vector.append(fields[i])
```

```
                elif self.format[i] == 'comment':
                    ignore.append(fields[i])
                elif self.format[i] == 'class':
                    category = fields[i]
            # 处理这条记录
            total += 1
            classes.setdefault(category, 0)
            counts.setdefault(category, {})
            totals.setdefault(category, {})
            numericValues.setdefault(category, {})
            classes[category] += 1
            # 处理分类型数据
            col = 0
            for columnValue in vector:
                col += 1
                counts[category].setdefault(col, {})
                counts[category][col].setdefault(columnV
                counts[category][col][columnValue] += 1
            # 处理数值型数据
            col = 0
            for columnValue in nums:
                col += 1
                totals[category].setdefault(col, 0)
                #totals[category][col].setdefault(columnV
                totals[category][col] += columnValue
                numericValues[category].setdefault(col,
                numericValues[category][col].append(colu

        # 计算先验概率P(h)
        for (category, count) in classes.items():
            self.prior[category] = count / total

        # 计算条件概率P(h|D)
        for (category, columns) in counts.items():
              self.conditional.setdefault(category, {})
              for (col, valueCounts) in columns.items():
                  self.conditional[category].setdefault(col, {})
                  for (attrValue, count) in valueCounts.items():
                      self.conditional[category][col][attrValue]
                          count / classes[category])
        self.tmp =  counts

        # 计算平均值和样本标准差
        self.means = {}
        self.ssd = {}
        # 动手实践
```

动手实践[1]

为上述代码实现计算平均值和样本标准差的逻辑，输出的结果如下：

```
>>> c = Classifier('pimaSmall/pimaSmall', 1, 'num\tnum\tnum\tnum
>>> c.ssd
{'1': {1: 4.21137914295475, 2: 29.52281872377408, ...},
 '0': {1: 2.54694671925252, 2: 23.454755259159146, ...}}
>>> c.means
{'1': {1: 5.25, 2: 146.05555555555554, ...},
 '0': {1: 2.8867924528301887, 2: 111.90566037735849, ...}}
```

解答

```
        # 计算平均值和样本标准差
        self.means = {}
        self.ssd = {}

        for (category, columns) in totals.items():
            self.means.setdefault(category, {})
            for (col, cTotal) in columns.items():
                self.means[category][col] = cTotal / classes[cat

        for (category, columns) in numericValues.items():
            self.ssd.setdefault(category, {})
            for (col, values) in columns.items():
                SumOfSquareDifferences = 0
                theMean = self.means[category][col]
                for value in values:
                    SumOfSquareDifferences += (value - theMean)*
                columns[col] = 0
                self.ssd[category][col] = math.sqrt(SumOfSquareD
```

动手实践[2]

修改分类函数`classify()`，使其能够使用概率密度函数进行分类。

```
    def classify(self, itemVector, numVector):
        """返回itemVector所属分类"""
        results = []
        sqrt2pi = math.sqrt(2 * math.pi)
```

```
for (category, prior) in self.prior.items():
    prob = prior
    col = 1
    for attrValue in itemVector:
        if not attrValue in self.conditional[category][c
            # 该特征值没有出现过，因此概率给0
            prob = 0
        else:
            prob = prob * self.conditional[category][col
        col += 1
    col = 1
    for x in  numVector:
        mean = self.means[category][col]
        ssd = self.ssd[category][col]
        ePart = math.pow(math.e, -(x - mean)**2/(2*ssd**
        prob = prob * ((1.0 / (sqrt2pi*ssd)) * ePart)
        col += 1
    results.append((prob, category))
# 返回概率最高的分类
#print(results)
return(max(results)[1])
```

朴素贝叶斯的效果要比近邻算法好吗？

上一章中我们对近邻算法做了统计：

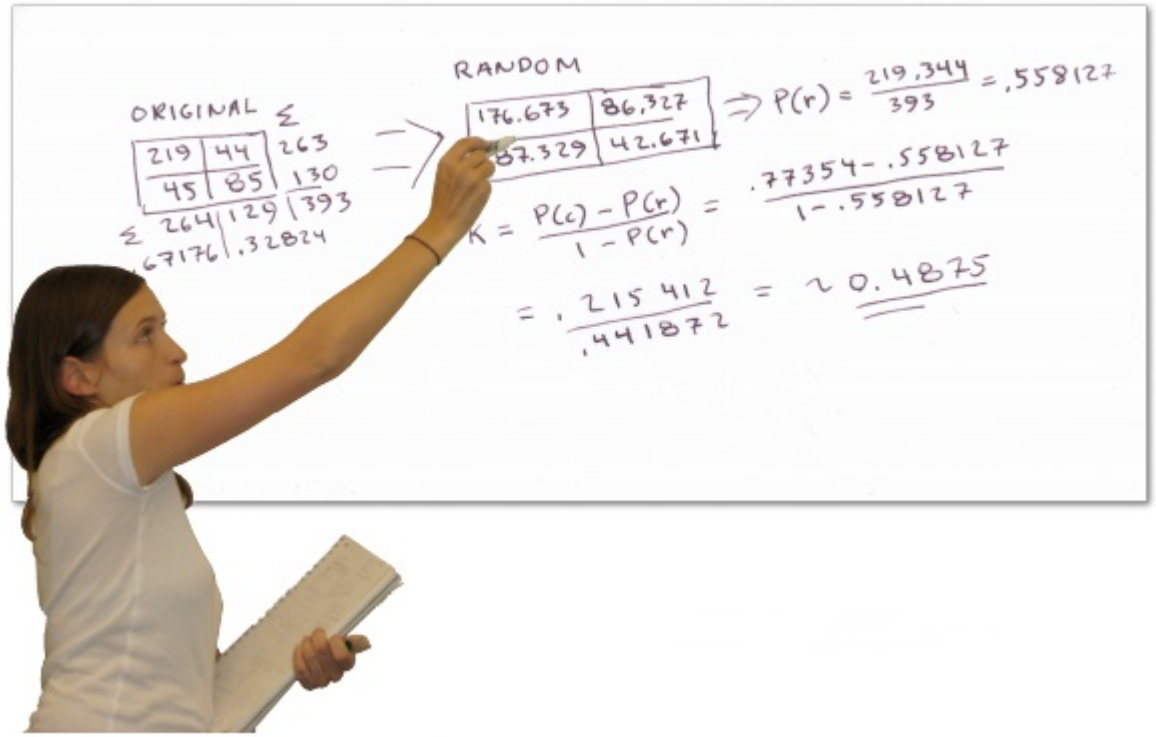
	pimaSmall	pima
k=1	59.00%	71.247%
k=3	61.00%	72.519%

以下是贝叶斯算法的结果：

	pimaSmall	pima
Bayes	72.000%	77.354%

哇，看来贝叶斯的效果要比近邻算法来得好呢！

kNN算法中，k=3时的Kappa指标是0.35415，效果一般。那朴素贝叶斯的Kappa指标是多少呢？



贝叶斯的Kappa指标是0.4875，符合期望。

因此在这个例子中，朴素贝叶斯的效果要比近邻算法好。

贝叶斯方法的优点：

- 实现简单（只需计数即可）
- 需要的训练集较少
- 运算效率高

贝叶斯方法的主要缺点是无法学习特征之间的相互影响。比如我喜欢奶酪，也喜欢米

饭，但是不喜欢两者一起吃。

kNN算法的优点：

- 实现也比较简单
- 不需要按特定形式准备数据
- 需要大量内存保存训练集数据

当我们的训练集较大时，kNN算法是一个不错的选择。这个算法的用途很广，包括推荐系统、蛋白质分析、图片分类等。

为什么要叫“朴素贝叶斯”呢？

我们之所以能将多个概率进行相乘是因为这些概率都是具有独立性的。比如说，有一个游戏是同时抛硬币和掷骰子，骰子的点数并不依赖于硬币是正面还是反面，所以在计算联合概率时可以直接相乘。如果我们要计算同时抛出正面（heads）以及掷出6点的概率：

$$P(heads \wedge 6) = P(heads) \times P(6) = 0.5 \times \frac{1}{6} = 0.08333$$

再比如我们有一副扑克牌，保留所有黑色牌（26张），以及红色牌中的人头牌（6张），一共32张。那么，选出一张人头牌（facecard）的概率就是：

$$P(facecard) = \frac{12}{32} = 0.375$$

选出红色牌（red）的概率是：

$$P(red) = \frac{6}{32} = 0.1875$$

那么，选出一张即是红色牌又是人头牌的概率是多少呢？直觉告诉我们**不能**这样计算：

$$P(red \wedge facecard) = P(red) \times P(facecard) = 0.375 \times 0.185 = 0.0703$$

因为红色牌的概率是0.1875，但这张红色牌100%是人头牌，所以红色人头牌的概率应该是0.1875。

这里不能做乘法就是因为这两个事件不是互相独立的，在选择红色牌时，人头牌的概率就变了，反之亦然。

在现实数据挖掘场景中，这种特征变量之间不独立的情况还是很多的。

- 运动员例子中，身高和体重不是互相独立的，因为高的人体重也会较高。
- 地区邮编、收入、年龄，这些特征也不完全独立，一些地区的房屋都很昂贵，一些地区则只有房车：加州帕罗奥图大多是20岁的年轻人，而亚利桑那州则多是退休人员。
- 在音乐基因工程中，很多特征也是不独立的，如果音乐中有很多变音吉他，那小提琴的概率就降低了。
- 血液检验的结果中，T4和TSH这两个指标通常是呈反比的。

再从你身边找找例子，比如你的车，它的各种特征之间有相关性吗？一部电影呢？亚马逊上的购买记录又如何？

所以，在使用贝叶斯方法时，我们需要互相独立的特征，但现实生活中很难找到这样的应用，因此我们只能假设他们是独立的了！我们完全忽略了这个问题，因此才称为“朴素的”（天真的）贝叶斯方法。不过事实证明朴素贝叶斯的效果还是很不错的。

动手实践

你能将朴素贝叶斯方法应用到其他数据集上吗？比如加仑公里数的例子，kNN算法的正确率是53%，尝试用贝叶斯方法来实现吧。

```
>>> tenfold('mpgData/mpgData', 'class\tattr\tnum\tnum\tnum\tnum\
```