【机器学习-2】似然与朴素贝叶斯 及实现

引言

这次文章灵感出现于我在学校听数学课的时候，因为神游我注意到了一些平时不会想的东西，而决定将贝叶斯统计作为机器学习的第二篇。

> 叛逆期的学生上课是这样的

我们将在这篇文章以朴素贝叶斯的知识为核心，了解基础的贝叶斯统计学概念，并编程来实现这些算法（如果会出贝叶斯的下一篇那应该会是贝叶斯神经网络），其中实战部分会同步更新到我的Haskell机器学习库（RHML，ROR Haskell Machine Learning）
本文有初中数学基础以及基本的编程逻辑即可（我们使用Haskell语言实现文章内容），但是我仍然建议你在阅读前能有关于导数、微积分和线性代数的一些基础概念，如果完全没有的话不妨先去补点：

线性代数随笔【1】
【微积分入门】从极限到黎曼积分的探索

建议每一个想研究机器学习（之后为了方便统一叫ML）的读者一定要把这些基本的数学概念给搞清楚，不然会哭的，相信我...

概率与似然

古典概率

概率论中有一种最基本的概率模型叫做古典概率，人们最早从抛硬币掷骰子什么的来研究概率，而它们的特点就是样本空间是有限的，例如硬币只有正和反两种可能，骰子有六面。古典概率假设所有可能的结果是等可能发生的，每个结果发生的概率相等。
我们可以很简单地基于事件发生的频率来计算古典概率，在互斥事件（即两个事件不会同时发生的情况下）中，若我们要计算事件 A 发生的概率，对于 A 有利事件的数量为 m，总样本空间的大小为 n，则：

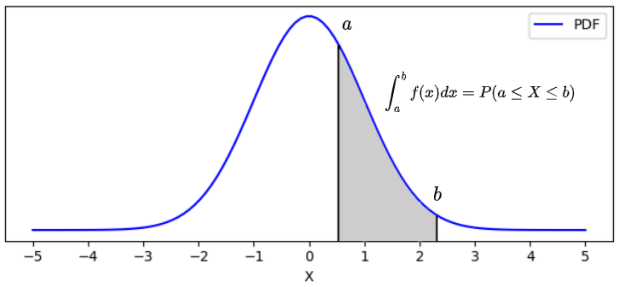
$$P(A) = \frac{m}{n}$$

这就是典型的先验概率，即在考虑任何观察或证据之前，根据先前的经验或基本原理来估计的概率。在古典概率中，概率分布是基于对随机实验或事件的理论性分析，而不考虑任何新的数据或观察结果。

概率分布函数

在说似然函数之前，我们可能需要简单了解下概率分布相关的一些函数。

我们可以用概率密度函数（Probability Density Function，之后为了方便简称PDF）来描述连续型随机变量的概率分布的函数。



如图是一个高斯分布的概率密度函数，如果 X 是一个连续型随机变量，考虑概率密度函数 $f(x)$ 则我们可以在其中选定某一区间来表示变量落在这个区间中的概率，我们用定积分来表示曲线下某区间的面积，即：

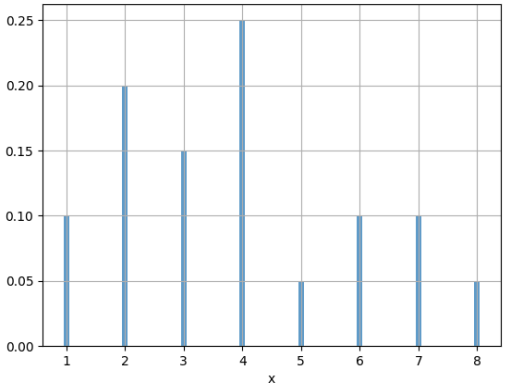
$$\int_a^b f(x)dx = P(a \le X \le b)$$

概率密度函数具备一些基本的性质，首先，它的整个定义域都是非负的。其次，它具有归一性，整个定义域的积分等于1，即 $\int f(x)dx = 1$，因为总概率是1。

而对于离散型随机变量，我们可以用概率质量函数（Probability Mass Function，之后简称PMF）来描述它的概率分布。相比概率密度函数，概率质量函数本身代表的是这个值的概率，而非PDF那样必须指定一个区间并积分才能得到概率（因为它的概率是曲线下的面积，若只取一个点就是0）

如图的PMF，每一个离散的变量x对应了一个概率，而它们总和为1，即：

$$\sum P(X = x) = 1$$



似然函数

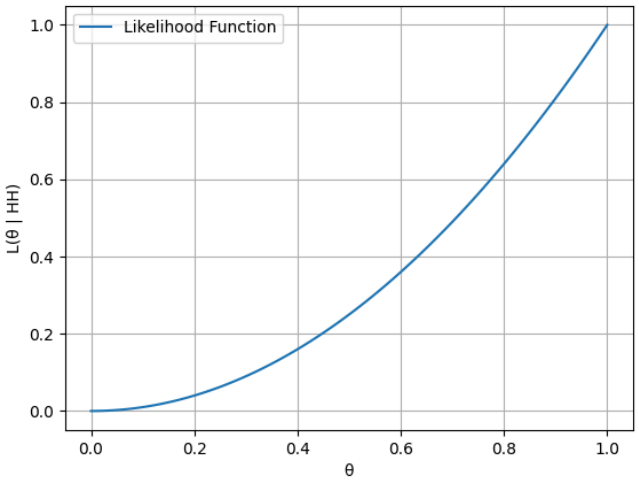
在比较不正式的情况下，似然（Likelihood）可能会和概率的概念混淆使用，但现在开始就不要这样想了，重新认识它就好。

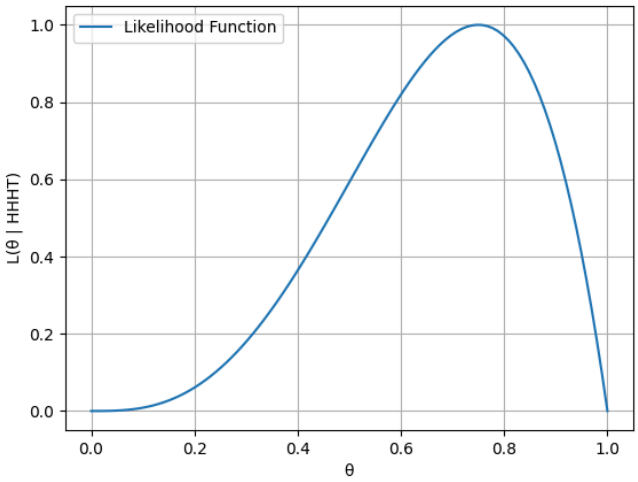
概率一般而言就只是在一组已知参数下，某事件发生的可能性。而似然是在已知一组数据的情况下，关于模型参数的可能性，就好比，一枚正反面的硬币，我扔出两次正面，然后我说『扔出正面的概率为50%』的可信度是多少？或者说『扔出正面的概率为70%』的可信度是多少？这个“两次正面”是一组已知的数据，“50%” 是被估计可能性的模型参数，似然就是这个“可信度”。

似然函数是在似然的概念上，对模型参数可能性的分布函数。通常我们用 $L(\theta|X)$ 表示，其中：

- L：表示似然函数
- θ：表示模型的参数（即示例中的50%）
- X：表示观测到的数据（即示例中的 “两次正面” ）

若我们用 H 表示硬币的正面，T 表示硬币的反面，观察下图，是分别以扔出两次正面 HH 和以扔出三次正面和一次反面 $HHHT$ 的估计：





我们可以看到，当我们扔出两次都是正面的时候，因为我们是以结果估计参数（也就是反过来的），所以在我们不知道硬币正反概率而得出这样结果的情况下，如果我说『硬币扔出正面的概率是1』可信度是1，而如果我说『硬币扔出正面的概率是0.5』，在这样的结果下说这句话的可信度会更低。而再看右边，当我们扔出三次正面一次反面时，函数就不一样了，这种时候再说『硬币扔出正面的概率是1』就毫无可信度，因为出现了一次反。

关于参数 θ 的似然函数是将条件概率 $f(x_i|\theta)$ 从1到 n 所有观测数据点连乘起来的结果：

$$L(\theta) = \prod_{i=1}^{n} f(x_i|\theta)$$

很多情况下，为了方便后续积分，我们可以多取个对数，这就是对数似然函数。因为对数可以满足 $log(ab) = log(a) + log(b)$，所以我们可以很容易地使用对数将乘法转为加法，又不影响函数极值位置，即：

$$\log(L(\theta)) = \log(\prod_{i=1}^{n} f(x_i|\theta)) = \sum_{i=1}^{n} \log(f(x_i|\theta))$$

将似然函数取对数通常会将累乘运算变为累加运算，这可以更容易地进行计算，尤其是在使用算法需要进行优化时。

最大似然估计

既然如此，似然的最大值意味着的就是事情发生最有可能的概率（注意不是概率本身）。我们要关注的就是函数图象的最大值，前面提到我们可以将似然函数变为对数似然函数，在最大似然估计我们就经常需要这样做。因为对数函数是单调递增的，其增长的速度会随 x 的增长而减缓，因此它不会影响到极值的位置且我们可以更方便地找到最大似然。

其实是很简单的，我们都知道当导数为0时，所在的地方就是函数局部的最大值，而对数似然函数是单调递增的，所以并不用考虑局部的问题，我们只需要将函数的导数置为0即 $log'(L(\theta)) = 0$ 即可。

但实际上导数置为0的做法，对于具有凸性的似然函数也是可行的，因为我们可以直接确定局部最大值。但是实际我们很少做还是因为对数似然函数具有的更多良好性质，包括是在计算机实际实现的数值稳定性。概念很简单，所以我们可以实践一下。

实现：二项分布参数估计

我们用于建模多次独立二元试验的概率分布，时常用二项分布表示。也就是，每次试验都有两种互斥的结果，我们当作『成功』和『失败』，而假设试验的总次数为 n，成功的概率为 p，成功的次数为 k，由于测试的结果是离散的，所以我们使用概率质量函数表示，若随机变量 x 服从参数为 n, p 的二项分布，则经过 n 次试验得到 k 次成功的概率为：

$$P(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$$

其中，这个组合数 $\binom{n}{k}$ 表示在 n 次试验中选择 k 次对象的方式的数量，就像从 n 本书中选其中的 k 本书，这表示了你有多少种方式来选择这 n 本书，它的计算公式为：

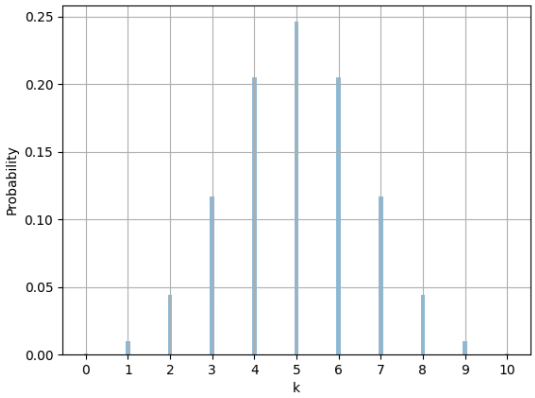
$$\binom{n}{k} = \frac{n!}{(k! * (n-k)!)}$$

我们本质关心的是成功的概率 p，$1-p$ 表示失败的概率，而 p^k 表示 k 次成功的概率，$p^k(1-p)^{n-k}$ 表示在剩余的 $n-k$ 次试验中每次都失败的概率。

例如，图例表示假设尝试10次，且成功的概率为0.5，其中五次成功的概率最高，但只是很接近0.25。可能有读者在此处就会察觉到，二项分布与伯努利分布的差别仅在于考虑到了多次试验，若 $n = 1$ 那就是伯努利分布。

这只是一个分布规律，而我们要做的当然没有那么简单——阿也没有多复杂，我们要做的无非是对于一组给定的数据和概率，来判断『这组数据服从二项分布的可能性』。

我们在代码中默认1为成功，0为失败，我们先定义函数签名、运算过程、阶乘和数值：




```
binomialLikelihood :: [Int] -> Double -> Double
  binomialLikelihood observations p = sum [log (choose n k) + fromIntegral k * log p + fromIntegral (n - k) * log (1 - p) | k <- observations]
  where
      choose :: Int -> Int -> Double
      choose n k = factorial n / (factorial k * factorial (n - k))
      n = fromIntegral $ length observations
      factorial 0 = 1
      factorial m = fromIntegral m * factorial (m - 1)
```

传入一个列表的整数，虽然我们并不对01还是其他数值进行检查，但是这会影响到结果，每个1都代表一次成功所以你也可以传入3或者4什么的Int类型。其中choose就是 $\binom{n}{k}$，并且我们在下方定义阶乘运算。

然后我们取对数似然，运算并求和（代码第二行），输出的数值就是似然。然后我们再定义一个函数FindMLE得到最大似然，它通过迭代从0.01到0.99等可能的概率值该找到似然函数的最大值，得到最可能的估计参数。此处我们通过maximumBy实现比较观测数据下的似然函数值：

```
findMLE :: [Int] -> Double
findMLE xs = maximumBy (\p1 p2 -> compare (likelihoodForP xs p1) (likelihoodForP xs p2)) [0.01, 0.02 .. 0.99]
```

写出来可以自己测试一下，我这里写了点简单的测试代码测试最简单的情况，结果是0.25

```
main :: IO ()
main = do
    let observations = [1, 0]
    let likelihood = BinomialDistribution.findMLE observations
    putStrLn $ "Likelihood: " ++ show likelihood
```

此处我们已经对之后会用到的概率和似然，以及贝叶斯定理等基础概念有了一定认识，后面我们会探讨贝叶斯统计在朴素贝叶斯的应用。

朴素贝叶斯分类

贝叶斯定理

先简单粗暴地提定理，在满足一个条件（例如另外一件随机事件的发生）的情况下发生一件随机事件的概率被称为条件概率。例如，有随机事件 A, B，那条件概率 $P(A|B)$ 就是在 B 发生时，A 发生的概率。然后，我们用 $P(A \cap B)$ 表示 A 和 B 同时发生的概率。则贝叶斯定理为：

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

当然，我们也有（很多地方这里会代入上式使用，记住概念就好）：

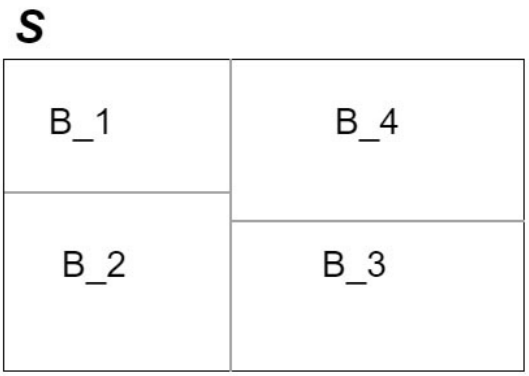
$$P(A \cap B) = P(A|B)P(B)$$

$$P(A \cap B) = P(B|A)P(A)$$

我曾经在网上看过很多乱七八糟的什么，反正就是一堆奇奇怪怪的例子来说明这个定理的意义。说白了关键思想是通过已知的信息（先验概率）来更新我们对事件的『信心』，以得到事件在新信息下的概率（后验概率）。就好比垃圾邮件，当邮件含有『只要』『低风险』等字眼时，它是或者不是垃圾邮件的概率是多少？亦或者反过来，垃圾邮件包含这些字眼的概率是多少？

另外要注意的是，条件概率和后验概率这两个概念看起来比较相似，表示方式一样，事实也是如此。但条件概率是单指在某一条件（例如一个事件发生）的前提下，另一个事件发生的概率。而后验概率是指在考虑新的观测信息后对某一事件概率的重新估计，也就是它是一种更新的概率估计。你可以理解成条件概率相对是一个更一般的概念，仅描述事件的关系，而后验概率是一种应用，将新的观测应用于更新概率上。

全概率公式，我们考虑一组互斥，也就是不会同时发生的事件 $\{B_1, B_2, \cdots, B_n\}$ 划分了样本空间 S，我们可以称其为完备事件组。我们要计算概率的目标事件是 A，但在这个样本空间有很多个『互不干涉的前提』，所以我们要用 $P(A|B_i)$ 表示在其中一个前提 B_i 的情况下，A 发生的概率。



由于还是一个样本空间，我们通过将事件 A 在不同前提下的条件概率和这些前提的概率相乘并求和，这是将每个前提概率都按照相应的条件事件的权重来加权，这样一来我们就可以更容易地计算出事件 A 的总体概率：

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

我们可以根据已知的先验概率 $P(A)$ 及在事件 A 发生的情况下，进行前提 B_i 发生的后验概率 $P(B_i|A)$ 计算。我们只需要将前面简单的贝叶斯定理代入到全概率公式即可：

$$P(B_i|A) = \frac{P(A|B_i)P(B_i)}{\sum_{j=1}^{n} P(A|B_j)P(B_j)}$$

在公式中，分子是条件概率 $P(A|B_i)P(B_i)$ 和先验概率 $P(B_i)$ 的积，实际上就是事件 A 在前提 B_i 下发生的似然度和事件 B_i 自身的先验概率。然后，全概率公式作为归一化因子出现在分母，之前说过它是不同情况下事件概率的加权求和，因此它的作用是将后验概率的分子部分 $P(A|B_i)P(B_i)$ 除以总概率，从而确保后验概率的和为1。

综上所述，全概率公式意义在于得到一个事件的概率分解为在不同条件下的条件概率的加权和，也就是计算事件的先验概率，而贝叶斯定理的意义在于对一个事件发生概率的『溯源』（这里指的是定义中的 A），已知这个事件发生，求另一个事件在此条件下发生的概率。由于这个计算基于贝叶斯定理，并且包括了事件自身的先验概率 $P(B_j)$，根据它们重新估计事件 A 前提下 B_i 发生的概率，所以它是一个后验概率。

朴素贝叶斯

朴素贝叶斯是贝叶斯统计的应用之一，作为一个机器学习算法用于文本分类和自然语言处理。

现实的情况总是很复杂，所以朴素贝叶斯有一个简单的假设就是“特征之间相互独立”，尽管这在现实不总是成立，但它的表现仍然相当出色，当然这也是“朴素”这个词的含义。我们假设 $x = \{x_1, x_2, \cdots, x_n\}$ 是一组样本，类别则用 c （class）表示，朴素贝叶斯独立性假设意味着：

$$P(x|c) = P(x_1|c)P(x_2|c)\cdots P(x_n|c) = \prod_{i=1}^{n} P(x_i|c)$$

如何我们代回去就能发现，这个表达式的一部分就是先验概率 $\frac{P(x|c)P(x)}{P(x)}$，其余部分就是特征条件概率的乘积，直接用连乘符号表示即可：

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

> 原谅我选择这里开始使用小写，因为一直大写AB什么的真的很不方便（而且我认为不好看x）

我们在这里要考虑的是没有任何特征信息的前提下，估计在分类器中每个类别 c 出现的先验概率。此处 $|D_c|$ 是属于类别 c 的训练集 D 中样本的数量，而 $|D|$ 是整个训练集，包括所有类别的集合的样本数量，我们通过简单粗暴的分数线即可得到类别 c 的先验概率。

$$P(c)=\frac{|D_c|}{|D|}$$

另一方面，朴素贝叶斯分类器也会涉及到似然概率的估计，$|D_{c,x_i}|$ 表示具备特征 x_i 且属于类别 c 的训练集 D 中的样本的数量，则似然概率 $P(x_i|c)$ 的估计公式为：

$$P(x_i|c)=\frac{|D_{c,x_i}|}{|D_c|}$$

> 我选择像大部分参考资料那样省略类别 c 的下标，这是因为在贝叶斯分类器中讨论单一类别的情况省略下标可以简化表达，实质上这也没有很大影响所以理解就行。
> 只要此处你应该搞清楚就是朴素贝叶斯往往类别不止一个，所以在实际操作中要区分多个类别（而理论我们大部分情况只是针对单一类别进行讨论）

拉普拉斯平滑

有一种很常见的方法可以优化我们上述先验概率的计算方法，拉普拉斯平滑或叫加一平滑。它的重点很简单，就算在概率估计的分子加一个1，确保所有事件的概率都非零，并且分母添加训练集 D 中可能的类别数 N 作为修正项。

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

$$P(x_i|c)=\frac{|D_{c,x_i}|+1}{|D_c|+N}$$

使用拉普拉斯平滑很大程度是为了在数据稀疏的情况下防止某些事件的概率为零，因为这可能导致模型预测的偏差很极端（例如事件不可能出现）。另一方面能提高模型的鲁棒性，使其更稳定而不会对一些并不大的变化过于敏感，减少模型对数据的噪声过度拟合等。

独依赖估计

在理论阶段，朴素贝叶斯分类器选择了简单的独立来概括各种情况，事实的情况经常做不到，尽管在大部分情况下照用这样的前提进行分类也不影响它出色的效果，但凡事还是不能那么随变，所以我们要考虑一些相对复杂但能考虑到依赖关系的变体，这种我们一般称为半朴素贝叶斯。

独依赖估计是在每个属性分类的情况下允许至多依赖一个其他属性的变体，我们在原本的基础上多加一个 x_i 依赖的属性 pa_i ，叫 x_i 的父属性，我们应该改写之前关于 x 的概率公式为：

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

而对于给定观测数据 x 的情况下，类别 c_i 的后验概率表示为：

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

$\propto$ 在此处表示『正比于』，这意味着我们只需要关注它相对的数值，并且我们可以省略原本公式 $\frac{P(c)}{P(x)}$ 中的 $P(x)$ 并将其作为一个归一化因子。

考虑到父属性 pa_i 的条件概率可以这样表示：

$$P(x_i|c, pa_i) = \frac{|D_{c,x_i,pa_i}|}{|D_{c_i,pa_i}|}$$

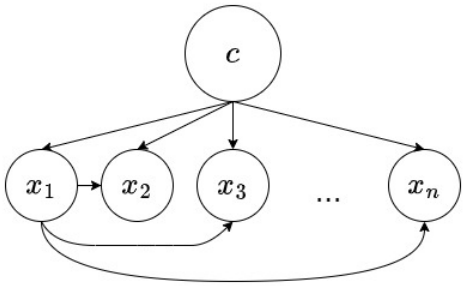
对于独依赖估计的做法，还有细分为几种有不同依赖关系的算法。

SPODE算法

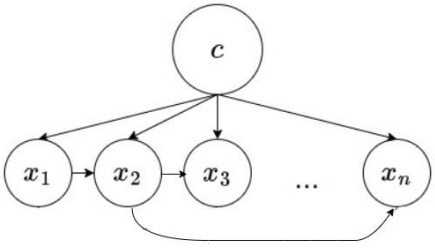
最简单的一种独依赖估计的算法叫SPODE算法，它假设所有属性依赖的都是同一个属性，这个被依赖的属性被称为超父。

SPODE的依赖关系很简单，如图所示，其中属性 x_1 就是超父。

> 这个图的结构参考自周自华的西瓜书，后来看过蛮多各种各样的博客几乎都是用这个类似的图。



TAN算法



TAN算法做的是选择一个特征作为根节点，并以树状结构使用最小数目的边连接所有节点，在不形成回路的情况下使得边长最大，这会使得父节点成为条件独立假设最接近成立的节点，同时还能表示特征之间的依赖关系。

因此TAN算法的重点是考虑如何生成最大带权生成树以建立依赖关系，我们可以通过这个公式计算任意两个属性之间的条件互信息。

$$I(x_i, x_j|c) = \sum_{x_i,x_j,c \in c} log \frac{P(x_i, x_j|c)}{P(x_i, c)P(x_j|c)}$$

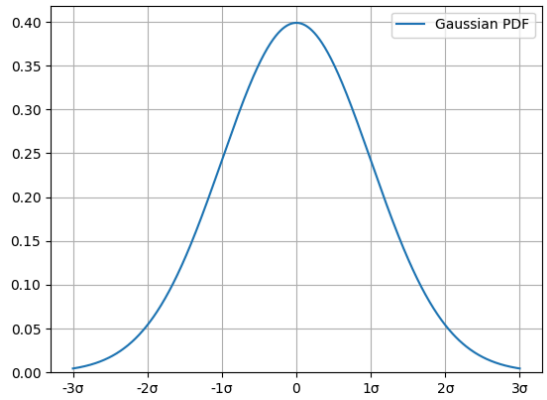
通过计算两个属性 x_i 和 x_j 在给定类别 c 的条件下的联合分布与它们各自在类别 c 下的边际分布之间的比值的对数，来度量 x_i 和 x_j 之间的条件互信息。条件互信息越大，表示 x_i 和 x_j 之间的相关性越高。

高斯朴素贝叶斯

高斯朴素贝叶斯是朴素贝叶斯的一种变体，一般而言的朴素贝叶斯适用的是离散特征的概率计算，不同的在于它的前提是特征值的概率呈高斯分布（要求是连续的概率分布）。我们都知道，均值 μ 是高斯分布的中心或期望值，表示数据的平均值、标准差 σ 是数据分散度的参数，表示数据点相对均值的离散程度，高斯分布的概率密度函数为：

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} exp(-\frac{(x-\mu)^2}{2\sigma^2})$$

进一步地，我们将 $\frac{1}{\sigma^2\sqrt{2\pi}}$ 作为归一化因子保证概率密度函数的积分为1。用 $exp(-\frac{(x_i-\mu_c)^2}{2\sigma_c^2})$ 表示高斯分布的概率密度函数，其中自然指数函数 exp 中的项，分子 $(x_i-\mu_c)^2$ 是特征值与均值的差值的平方，很显然就是表示差值的，而分母是标准差平方的两倍 $2\sigma_c^2$ ，它作为一个度量数据分散程度来控制曲线形状的值。



所以一元高斯分布的模型中，我们计算特征 x_i 属于类别 c 的条件概率的公式为：

$$P(x_i|c) = \frac{1}{\sigma\sqrt{2\pi}} exp(-\frac{(x_i-\mu_c)^2}{2\sigma^2})$$

为了之后的实战支持对于多维特征向量的估计，我们需要知道多元高斯分布的概率密度函数，以及特征向量对于类别的条件概率同理如下：

$$f(\mathbf{x};\mu,\sigma) = \frac{1}{2\pi^{n/2}|\sigma|^{1/2}} exp(-\frac{1}{2}(\mathbf{x}-\mu)^\top\sigma^{-1}(\mathbf{x}-\mu))$$

高斯朴素贝叶斯模型中的参数相对较少，通常只需要估计每个特征在每个类别下的均值和方差，这使得它不会对噪声和极端数据过于敏感，对于小数据集可以有效防止过拟合的模型，另外这也是一种处理零频率问题的方法。

多项式朴素贝叶斯

多项式朴素贝叶斯是另外一种针对于离散特征概率计算的朴素贝叶斯算法变体，它可能更多被用于文档分类。先从多项式分布说起，这是之前二项分布的推广，如果我们用扔硬币来作为二项分布的例子，那可能可以用骰子来作为多项式分布的例子。

在我的代码仓库内，高斯概率的计算是一个独立的子模块，所以我们在这里也先完成相关的函数。

> 没怎么看过组合数学？没关系 Makiror 会出手

上一章的内容，若随机变量 x 服从参数为 n, p 的二项分布，则一件事情经过 n 次试验得到 k 次成功的概率为：

$$P(X=k)=\binom{n}{k}p^k(1-p)^{n-k}$$

进一步地我们可以把这个公式推广到多项式分布，若可能的结果有 $x_1, x_2, \cdots, x_k$ ，进行 n 次试验，每种结果发生的概率为 $p_1, p_2, \cdots, p_k$ 则多项分布的联合概率函数是：

$$P(X_1=x_1, X_2=x_2, \cdots, X_k=x_k)=\frac{n!}{x_1!x_2!\cdots x_k!}p_1^{x_1}p_2^{x_2}\cdots p_k^{x_k}$$

稍微解释一下，$\frac{n!}{x_1!x_2!\cdots x_k!}$ 这个组合系数就类似于之前那个 $\binom{n}{k}$ 推广到多项分布的情况，用于计算有多少种不同的排列方式可以满足各个类别出现次数的要求。然后我们将每个类别出现的概率乘起来即可就是 $p_1^{x_1}, p_2^{x_2}, \ldots, p_k^{x_k}$ 。

其实这东西的优势很大程度就在于文本分类，至于我对它印象很深刻的原因，就是初一时的人工智能课老师把这玩意吹上了天，不懂的人吹不懂的东西让我觉得很蠢。很多时候自己把事情做一遍会发现没那么难所以——我们的实验a就是实现一个进行词频分析和文本分类的多项式朴素贝叶斯。

> 写于排版文章的晚上：我想我当时可能是疯了（）因为接下来的 Haskell 编程逻辑几乎都是在神游的几分钟脑补的，下课后简单的打了点伪代码草稿但是丢了，凭记忆写出来的。我觉得有些时候我的编程逻辑乱七八糟，以至于相比实现本身，我花了更多时间改缩进以提高代码的可读性。原谅我讨厌写注释的毛病（尤其是对FP语言），所以我像以往一样只是对代码的逻辑直接进行描述和解释，但貌似对于读者理解仍具有一定难度，所以之后我闲下来会另外发个PDF给代码详细的嵌套调用一一对应解释。

实现a：词频分析和文本分类

我们将会实现一个文章积极性的分类器，为这个分类器提供一些样本，例如一个文段和对应的性质（积极的、中性的、消极的），通过学习使得程序可以为一个句子估计它的积极性。这个分析是基于样本的词频的，所以它可以同理地用于文章分类等方面，只要你能完成词频统计的部分的话，它也不限语言。

我们考虑先定义数据类型，即分类的类别和作为训练文档的数据类型：

```
data DocumentClass = Positive | Negative | Neutral deriving (Eq, Show)
type Document = (DocumentClass, [String])
```

我们可以先写一个函数，通过一组训练数据（即文档）来计算目标类别的概率，这里很简单地将指定类别文档的数量除以总文档即可。直接用filter进行筛选得到目标类别的数量，直接除就行。

```
calculateCategoryProbability :: [Document] -> DocumentClass -> Double
```

```
calculateCategoryProbability documents category =
    let totalDocs = length documents
        categoryDocs = length (filter (\(docClass, _) -> docClass == category) documents)
    in fromIntegral categoryDocs / fromIntegral totalDocs
```

然后我们需要一个函数计算给定训练数据中每个单词在给定文档中的概率。

```
calculateWordProbabilities :: [Document] -> DocumentClass -> [(String, Double)]
calculateWordProbabilities documents category =
    let categoryDocs = filter (\(docClass, _) -> docClass == category) documents
        categoryTokens = concatMap snd categoryDocs
        wordCounts = fromListWith (+) [(word, 1) | word <- categoryTokens]
        totalWords = length categoryTokens
    in [(word, fromIntegral count / fromIntegral totalWords) | (word, count) <- toList wordCounts]
```

第四行的 categoryTokens 我们通过 snd 函数从每个文档中提取单词列表，并用 concatMap 函数将其应用于列表的每个元素，将结果连成一个新的列表，然后 wordCounts 是由 fromListWith (+) 生成的关联列表，后面的参数以生成一个包含单词和计数为1的元组列表作为列表推导式，函数通过将相同单词的计数相加得到每个单词在文档的总计数。

最后就是我们进行文本分类的函数，它会使用到之前的两个函数，接收一个训练集和目标文本并返回目标文本对应每个类别的概率，我们分开从抽象的部分写。重点在于这个嵌套有点多的列表推导式，具体而言里面的每个元素是文档对应每个类别的后验概率，其中 documentTokens 是不重复的单词列表，在列表推导式中，我们先计算文档属于该类别的概率，再对其中每个单词调用 lookupWordProbability（暂未定义） 函数并相乘，得到的就是文档在该类别的条件概率，最后与文档与类别的条件概率相乘并组成一个元组存放在列表中。

```
classifyDocument :: [Document] -> [String] -> [(DocumentClass, Double)]
classifyDocument documents document =
    let categories = [Positive, Negative, Neutral]
        documentTokens = nub document
    in [ (category,
        calculateCategoryProbability documents category *
        product [lookupWordProbability word category | word <- documentTokens])
        | category <- categories]
```

这里实质就是应用贝叶斯定理，公式应该为：

$$P(\text{category}|\text{document}) = \frac{P(\text{category})P(\text{document}|\text{category})}{P(\text{document})}$$

然后我们定义 lookupWordProbability 函数，它将调用前面计算单词在文档中概率的函数，若有概率则返回，概率为零则使用拉普拉斯平滑值返回，这一般会发生在文档中未出现的单词。

```
    where
        lookupWordProbability word category =
            case lookup word (calculateWordProbabilities documents category) of
                Just prob -> prob
                Nothing -> 1.0 / (fromIntegral (length (nub (map fst documents))) + 1.0)
```

由于它针对的是文档未出现相应单词的情况，所以实质上将拉普拉斯平滑的公式以如下形式应用即可，其中修正项 N 为 1，作为伪计数使用：

$$P(\text{word}|\text{category}) = \frac{0 + 1}{|\text{Documents}_{\text{category}}| + 1}$$

最后的最后我们可以写点什么测试一下，这里提供一个我的测试代码，并在得到结果后比较哪个概率更大以直接判断内容的积极性。后半段是将概率列表取最大概率的类别，并根据此判断和输出这句话的积极性。

```
main :: IO ()
main = do
    let trainingData = [(Positive, ["love", "happy", "joy"]),
                        (Negative, ["hate", "sad", "angry"]),
                        (Neutral, ["neutral", "indifferent", "meh"])]
    let testSentence = words "I feel sad and angry"
    let classified = classifyDocument trainingData testSentence
    let result = maximumBy (compare `on` snd) classified
    putStrLn $ "The sentence belongs to class: " ++ show (fst result)
```

实现b：高斯朴素贝叶斯分类器

刚才我们趁热打铁地实现了一个基于多项式朴素贝叶斯的文本分类器，而现在我们再实现一个基于高斯数据分布的高斯朴素贝叶斯算法来对观察值进行分类。根据之前高斯概率密度函数，我们写出高斯概率的计算函数，参数是目标的值、包含高斯分布均值 μ (mean) 和标准差 σ (variance) 的元组，之后我们称其为统计特征：

```
gaussianProbability :: Double -> (Double, Double) -> Double
gaussianProbability x (mean, variance) =
    (1.0 / (sqrt (2.0 * pi * variance))) * exp (-((x - mean) ** (2.0 :: Double)) / (2.0 * variance))
```

注意，这里是一元高斯概率的计算，即对应公式：

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} exp(-\frac{(x-\mu)^2}{2\sigma^2})$$

另一方面就是以高斯分布为前提，给定一组数据计算它们的统计特征。

$$\mu = \frac{1}{N}\sum_{i=1}^{N} x_i$$

$$\sigma = \frac{1}{N}\sum_{i=1}^{N} (x_i - \mu)^2$$

```
calculateClassStats :: [Double] -> (Double, Double)
```

```
calculateClassStats values =
    let mean = sum values / fromIntegral (length values)
        variance = sum [(x - mean) ** (2.0 :: Double) | x <- values] / fromIntegral (length values)
    in (mean, variance)
```

好，然后我们定义分类器的一些数据结构，分别是数据特征集、类别名 Class 和表示一个类别对应的统计特征的 ClassStats。注意，这里的数据特征集意味着我们将实现的算法支持多维高斯分布，你看，是不是很酷？

```
type Document = [Double]
type Class = String
type ClassStats = [(Double, Double)]
```

定义一个函数，很直接地通过一组类别，计算目标类别出现的先验概率：

```
priorProbability :: [Class] -> Class -> Double
priorProbability categories category =
    let totalDocs = length categories
        categoryCount = length (filter (== category) categories)
    in fromIntegral categoryCount / fromIntegral totalDocs
```

然后我们再定于训练模型的函数，用于根据给定的训练集和指定一个类别，训练一个高斯朴素贝叶斯的模型。先定义 documents 作为训练集中属于指定类别的所有数据（相比文档我还是更喜欢叫数据）的特征值，并计算特征值的数量 numFeatures，然后通过对每个特征进行迭代的列表推导式，通过调用之前写的 calculateClassStats 函数来计算该特征值目标类别下的参数，也就是我们要的统计特征。

```
trainModel :: [(Class, [Document])] -> Class -> ClassStats
trainModel trainingData category =
    let documents = concatMap snd $ filter (\(c, _) -> c == category) trainingData
        numFeatures = length (head documents)
    in [calculateClassStats [feature !! i | feature <- documents] | i <- [0..numFeatures - 1]]
```

这样一来，我们就可以通过提供训练集来计算一组数据在该高斯朴素贝叶斯模型下最大可能的所属类别。其中，我们先计算先验概率，对应的是 priorProbability (map fst classStats) category 这段，得到的是数据属于该目标类别的先验概率；然后在另一边，通过 map 函数对训练数据集的内容进行迭代，即对列表中每一个元组计算其在对应高斯概率和先验概率下的后验概率，具体的操作就是用 map fst classStats 提取所有类别的列表，然后将列表中每个数据的特征值在对应类别的统计特征下的高斯概率连乘，得到的就是所有特征的概率密度的乘积，将这两个东西相乘就算我们要的后验概率。

得到这组数据对于每个类别的后验概率后，我们简单地在里面选取最大值作为它最可能属于的类别并返回。

```
predictCategory :: [Double] -> [(Class, ClassStats)] -> Class
predictCategory document classStats =
    let posteriorProbs = map (\(category, stats) ->
                        (category, priorProbability (map fst classStats) category *
                            product (zipWith (\feature stat ->
                                gaussianProbability feature stat) document stats)))
                        classStats
        (maxCategory, _) = maximumBy (\(_, p1) (_, p2) -> compare p1 p2) posteriorProbs
    in maxCategory
```

这里可以提提理论，之前我们在高斯朴素贝叶斯中提及的一元高斯分布和多元高斯分布的概率密度函数，在这里我们没直接应用多元高斯分布的公式，但我们通过将多维特征向量的每个分量看作是一元高斯分布的变量，在假设各个特征之间独立的前提下，计算联合概率密度函数就是各个维度上概率密度函数的乘积，一句话概括就是 **将多维特征向量的处理简化为各个维度上一元高斯分布的处理**，这其实是我神游时的设想，加以证实后确实可行。

好，最后我们可以写点什么测试一下，先训练每个类别对应的模型，也就是得到统计特征，然后定义测试文档，扔给函数 predictCategory 进行判断，我们先试试最简单的一元高斯分布测试，当然因为迁就一下支持多维高斯分布的数据结构，这里训练集的特征值写法可能比较难看（如果需求简单的话可以适当调整数据结构），反正，下述测试代码得到的结果是 Class1。

```
main :: IO ()
main = do
    let trainingData = [("Class1", [[1.2], [1.4], [1.5], [1.3]]),
                        ("Class2", [[2.0], [2.1], [2.2], [2.3]]),
                        ("Class3", [[3.0], [3.1], [3.2], [3.3]])]

    let classStats = map (\(category, _) -> (category, trainModel trainingData category)) trainingData
    let testDocument = [1.34]
    let predictedCategory = predictCategory testDocument classStats
    putStrLn $ "Predicted Category: " ++ predictedCategory
```

下面测试代码是一个二元高斯分布，输出结果是 Class 3，另外如果我将数据改成 [2.2, 4.2]，输出结果会是 Class 2，这是符合直觉的，反正自己写点试试就好。

```
main :: IO ()
main = do
    let trainingData = [("Class1", [[1.2, 2.3], [1.4, 2.9], [1.5, 2.7], [1.3, 2.6]]),
                        ("Class2", [[2.0, 3.1], [2.1, 3.5], [2.2, 3.3], [2.3, 3.4]]),
                        ("Class3", [[3.0, 4.1], [3.1, 4.5], [3.2, 4.3], [3.3, 4.4]])]

    let classStats = map (\(category, _) -> (category, trainModel trainingData category)) trainingData
    let testDocument = [3.2, 4.2]
    let predictedCategory = predictCategory testDocument classStats
    putStrLn $ "Predicted Category: " ++ predictedCategory
```

结尾

差不多就是这样了，其实我最早想写的单纯就只是贝叶斯统计学基础本身，但是我觉得那么简单地放过读者（？）不值，所以选择了专门针对朴素贝叶斯展开内容，要抱歉的就是这个学期太紧张，空闲非常分散导致我很难顺着思路组织文章（况且这次文章的思路，尤其是实战的部分几乎都是即兴发挥）。如果有疑惑或者有错误的地方非常欢迎指正，私信发邮件评论都是可以的。

另外我现在摸数学和计算机的时间少了很多，主要是因为想多学一两门外语和发展点其他兴趣爱好，以及要中考和顾及重要程度相当的事情，所以之后不会固定写文章的时间和内容了，如果有什么疑问或者想交流的也是，可以直接联系我。

一些推荐资料和链接

我这次没有因写文章针对性地参考资料，所以这里就放点ML和统计学有用的链接好了。

Introduction to Machine Learning：https://books.google.com/books/about/Introduction_to_Machine_Learning.html?id=SY789Y7Fc14C

Bayesian Data Analysis：https://books.google.com/books/about/Bayesian_Data_Analysis_Second_Edition.html?id=TNYhnkXQSjAC

我的Github主页：https://github.com/MAKIROR

随性写写的RHML：https://github.com/MAKIROR/RHML

Makiror Ouyang gzanan@gmail.com 01:45 29/10/2023