《机器学习》课程系列

朴素贝叶斯*

武汉纺织大学数学与计算机学院

杜小勤

2020/05/01

Contents

1 分类模型

朴素贝叶斯 (Naive Bayes) 是一种简单而有效的分类方法，发源于古典数学理论，有着坚实的数学基础，属于概率生成式模型。朴素贝叶斯主要基于 1 个定理与 1 个假设——使用贝叶斯定理，且假设输入特征向量 $\boldsymbol{x}$ 在给定分类时，它的每

*本系列文档属于讲义性质，仅用于学习目的。Last updated on: June 20, 2020。

个特征分量 x_i 之间具有条件独立性。朴素贝叶斯所需估计的参数较少，对缺失的数据也不太敏感，算法实现也比较简单。

虽然特征独立性是一种非常强的假设，不一定符合实际应用，但是它的分类效果却相对较好。一种解释是，它的模型参数个数较少[1]，模型具有相对的、抑制过拟合的能力。另外一个优点是，学习算法只需要少量的训练数据就可以对参数进行估计。

具体而言，对于给定的数据集，朴素贝叶斯模型基于条件独立性假设，学习输入输出的联合概率分布；然后，对于新输入 $\boldsymbol{x}$，再利用贝叶斯定理计算后验概率，将后验概率最大的 y 作为 $\boldsymbol{x}$ 的分类。

设 $\boldsymbol{X}$ 是定义在输入空间 $\mathcal{X} \subseteq \mathbf{R}^d$ 上的随机向量，Y 是定义在输出空间 $\mathcal{Y} = \{c_1, c_2, \ldots, c_K\}$($K$ 表示类别个数) 上的随机变量；输入特征向量 $\boldsymbol{x} \in \mathcal{X}$，输出标记 $y \in \mathcal{Y}$，它们分别是随机向量 $\boldsymbol{X}$ 和随机变量 Y 的具体取值。对于 $\boldsymbol{X} = \boldsymbol{x}$，可以表示成等价的分量形式 $(X^{(1)} = x^{(1)}, X^{(2)} = x^{(2)}, \ldots, X^{(d)} = x^{(d)})$。

给定数据集 $T = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_N, y_N)\}$，其中 $\boldsymbol{x}_i \in \mathcal{X}$，$y_i \in \mathcal{Y}$，现假设该数据集依 $P(\boldsymbol{X}, Y)$ 独立同分布产生。朴素贝叶斯的目标是，从数据集 T 中学习联合概率分布 $P(\boldsymbol{X}, Y)$。

首先，将联合概率分布 $P(\boldsymbol{X} = \boldsymbol{x}, Y = y)$ 写成等价的分量形式：

$$P(\boldsymbol{X} = \boldsymbol{x}, Y = y) = P(Y = y, X^{(1)} = x^{(1)}, X^{(2)} = x^{(2)}, \ldots, X^{(d)} = x^{(d)}) \tag{1}$$

继续变形：

$$\begin{aligned}
&P(\boldsymbol{X} = \boldsymbol{x}, Y = y) \\
&= P(Y = y, X^{(1)} = x^{(1)}, X^{(2)} = x^{(2)}, \ldots, X^{(d)} = x^{(d)}) \\
&= P(Y = y) P(X^{(1)} = x^{(1)}, X^{(2)} = x^{(2)}, \ldots, X^{(d)} = x^{(d)} | Y = y) \\
&= P(Y = y) P(X^{(1)} = x^{(1)} | Y = y) P(X^{(2)} = x^{(2)}, \ldots, X^{(d)} = x^{(d)} | X^{(1)} = x^{(1)}, Y = y) \\
&= \ldots \\
&= P(Y = y) \prod_{j=1}^{d} P(X^{(j)} = x^{(j)} | X^{(1)} = x^{(1)}, X^{(2)} = x^{(2)}, \ldots, X^{(j-1)} = x^{(j-1)}, Y = y) \\
&= P(Y = y) \prod_{j=1}^{d} P(X^{(j)} = x^{(j)} | Y = y)
\end{aligned} \tag{2}$$

[1] 参数个数服从线性关系 $O(cd)$(其中，c 表示类别个数，d 表示特征个数)，而不是指数级关系。

其中，最后一步的推导利用了朴素贝叶斯模型的“特征条件独立性”假设。上式中，$P(Y=y)$ 是类别的先验概率，$P(X^{(j)}=x^{(j)}|Y=y)$ 是特征分量的条件概率。这 2 类参数是朴素贝叶斯模型需要从数据集 T 中学习得到的，下文将讨论参数的估计方法。

为了方便表达，将 $P(Y=y)$ 简写为 $P(y)$，将 $P(X^{(j)}=x^{(j)}|Y=y)$ 简写为 $P(x^{(j)}|y)$。于是，$P(\boldsymbol{X}=\boldsymbol{x},Y=y)$ 表示为：

$$P(\boldsymbol{x},y)=P(y)\prod_{j=1}^{d}P(x^{(j)}|y) \tag{3}$$

假设已经从数据集 T 中学习到了模型参数 $P(y)$ 和 $P(x^{(j)}|y)$(其中 $j=1,2,\ldots,d$, $y\in\mathcal{Y}$)，那么利用贝叶斯定理就可以计算后验概率 $P(Y=y|\boldsymbol{X}=\boldsymbol{x})$(简记为 $P(y|\boldsymbol{x})$)，且将后验概率最大的类别作为 $\boldsymbol{x}$ 的类输出：

$$\begin{aligned}
P(y|\boldsymbol{x})&=\frac{P(\boldsymbol{x},y)}{\sum\limits_{y}P(\boldsymbol{x},y)}=\frac{P(y)\prod\limits_{j=1}^{d}P(x^{(j)}|y)}{\sum\limits_{y}P(y)\prod\limits_{j=1}^{d}P(x^{(j)}|y)}\\
y^{*}&=\arg\max_{y}P(y|\boldsymbol{x})\\
&=\arg\max_{y}\frac{P(y)\prod\limits_{j=1}^{d}P(x^{(j)}|y)}{\sum\limits_{y}P(y)\prod\limits_{j=1}^{d}P(x^{(j)}|y)}=\arg\max_{y}P(y)\prod_{j=1}^{d}P(x^{(j)}|y)
\end{aligned} \tag{4}$$

上式中，最后一行中的分母为归一化项，对所有的类别而言，它是相同的常数，因而可以从优化公式中去掉。

为方便，下面给出朴素贝叶斯模型的完整定义。

定义 1.1（朴素贝叶斯模型）设 $\mathcal{X}\subseteq\mathbf{R}^{d}$($d$ 表示特征分量个数)，$\mathcal{Y}=\{c_1,c_2,\ldots,c_K\}$($K$ 表示类别个数)，输入特征向量 $\boldsymbol{x}\in\mathcal{X}$，输出标记 $y\in\mathcal{Y}$，则模型学习联合概率分布：

$$P(\boldsymbol{x},y)=P(y)\prod_{j=1}^{d}P(x^{(j)}|y) \tag{5}$$

其中，模型参数为 $P(y)(\forall y\in\mathcal{Y})$ 和 $P(x^{(j)}|y)(j\in\{1,2,\ldots,d\})$，它们都满足常规的概率约束，即 $\forall y\ P(y)\geqslant 0$ 且 $\sum\limits_{y}P(y)=1$、$\forall j\ P(x^{(j)}|y)\geqslant 0$ 且 $\sum\limits_{x^{(j)}}P(x^{(j)}|y)=1$。

从上面的讨论可以看出，朴素贝叶斯模型利用特征分量独立性假设，将模型参数个数从指数级别简化为线性级别，有助于减轻“维数灾难”问题；同时，利用贝叶斯定理进行类别推理，将最大后验概率的类别作为当前样本点的分类输出。在后续的章节中，将利用最大似然方法估计模型参数，同时也将利用 EM(Expectation Maximization) 算法估计类别不可见的朴素贝叶斯模型的参数。

2 类别推理的依据——期望风险最小化

为什么朴素贝叶斯模型将后验概率最大的类别作为样本点的输出类别？下面来分析一下其中的原理。

对于分类问题而言，从损失函数的角度，如果模型不使用数值迭代优化方法，那么选取 0-1 损失函数是合适的。现假设损失函数为 0-1 损失函数：

$$L\left(Y, f(\boldsymbol{X})\right) = \begin{cases} 1 & Y \neq f(\boldsymbol{X}) \\ 0 & Y = f(\boldsymbol{X}) \end{cases} \tag{6}$$

其中，$f(\boldsymbol{X})$ 为分类决策函数。分类的期望风险函数为：

$$\begin{aligned} &R_{exp}(f) = \mathbb{E}_{P(\boldsymbol{X},Y)}\left[L(Y, f(\boldsymbol{X}))\right] \quad \Rightarrow \\ &R_{exp}(f) = \mathbb{E}_{P(\boldsymbol{X})} \sum_{y} L(y, f(\boldsymbol{X}))P(y|\boldsymbol{X}) \end{aligned} \tag{7}$$

利用数据样本点 $(\boldsymbol{X}, Y)$ 之间的独立同分布假设：

$$\begin{aligned} &\min_{f} R_{exp}(f) = \min_{f} \mathbb{E}_{P(\boldsymbol{X})} \sum_{y} L(y, f(\boldsymbol{X}))P(y|\boldsymbol{X}) \quad \Leftrightarrow \\ &\min_{f} \sum_{y} L(y, f(\boldsymbol{x}))P(y|\boldsymbol{x}) \quad \forall \boldsymbol{x} \in \mathcal{X} \end{aligned} \tag{8}$$

上式表明，为了使期望风险最小化，只需对 $\boldsymbol{X} = \boldsymbol{x}$ 逐个最小化，于是得到：

$$\begin{aligned} &f^*(\boldsymbol{x}) = \arg\min_{f} \sum_{y} L(y, f(\boldsymbol{x}))P(y|\boldsymbol{x}) \quad \Rightarrow \\ &f^*(\boldsymbol{x}) = \arg\min_{f} \sum_{y} I\{y \neq f(\boldsymbol{x})\}P(y|\boldsymbol{x}) \quad \Rightarrow \\ &f^*(\boldsymbol{x}) = \arg\min_{f} \sum_{y \neq f(\boldsymbol{x})} P(y|\boldsymbol{x}) \quad \Rightarrow \\ &f^*(\boldsymbol{x}) = \arg\min_{f} 1 - P_{y=f(\boldsymbol{x})}(y|\boldsymbol{x}) \quad \Rightarrow \\ &f^*(\boldsymbol{x}) = \arg\max_{f} P(y = f(\boldsymbol{x})|\boldsymbol{x}) \end{aligned} \tag{9}$$

其中，$I\{y \neq f(\boldsymbol{x})\}$ 表示指示函数，当花括号中的条件成立时，其值为 1；否则，值为 0。上式的最后一行表明，当 $y = f^*(\boldsymbol{x}) = \arg\max_f P(y|\boldsymbol{x})$——取后验概率最大的类别作为 $\boldsymbol{x}$ 的类别时，期望风险取得最小值。

3 参数估计

在给出参数估计方法之前，需要将类别 y 表示成 One-Hot 向量形式，即 $(y^{(1)}, y^{(2)}, \ldots, y^{(K)})$[2]，其中 $y^{(k)}$ 取值 0 或 1，且 $\sum_{k=1}^{K} y^{(k)} = 1$。

给定数据集 $T = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_N, y_N)\}$，其中 $\boldsymbol{x}_i \in \mathcal{X}$，$y_i \in \mathcal{Y}$，对于由定义1.1给出的模型，可以使用最大似然方法估计模型的参数 $P(y^{(k)})$ 和 $P(x^{(j)}|y^{(k)})$：

$$\mathcal{L}(\boldsymbol{\theta}) = \prod_{(\boldsymbol{x},y)} P(\boldsymbol{x}, y) = \prod_{(\boldsymbol{x},y)} P(y) \prod_{j=1}^{d} P(x^{(j)}|y) \tag{10}$$

将 y 转换成 One-Hot 向量形式，得到：

$$\begin{aligned}
\mathcal{L}(\boldsymbol{\theta}) &= \prod_{(\boldsymbol{x},y)} P(y) \prod_{j=1}^{d} P(x^{(j)}|y) \\
&= \prod_{(\boldsymbol{x},y)} \prod_{k=1}^{K} \left(P(y^{(k)})^{y^{(k)}} \prod_{j=1}^{d} P(x^{(j)}|y^{(k)})^{I\{\boldsymbol{x}^{(j)}=x^{(j)}, y^{(k)}\}} \right) \\
&= \prod_{(\boldsymbol{x},y)} \prod_{k=1}^{K} \left(P(y^{(k)})^{I\{y=c_k\}} \prod_{j=1}^{d} P(x^{(j)}|y^{(k)})^{I\{\boldsymbol{x}^{(j)}=x^{(j)}, y=c_k\}} \right) \\
&= \prod_{k=1}^{K} \left(P(y^{(k)})^{\sum\limits_{(\boldsymbol{x},y)} I\{y=c_k\}} \prod_{j=1}^{d} P(x^{(j)}|y^{(k)})^{\sum\limits_{(\boldsymbol{x},y)} I\{\boldsymbol{x}^{(j)}=x^{(j)}, y=c_k\}} \right) \\
&= \prod_{k=1}^{K} \left(P(y^{(k)})^{n_k} \prod_{j=1}^{d} P(x^{(j)}|y^{(k)})^{n_{jk}} \right)
\end{aligned} \tag{11}$$

其中，$\boldsymbol{\theta}$ 表示模型参数 $P(y^{(k)})(k \in \{1, 2, \ldots, K\})$ 和 $P(x^{(j)}|y^{(k)})(j \in \{1, 2, \ldots, d\})$；$I\{\cdot\}$ 表示指示函数，当花括号中的条件成立时，值为 1，否则，值为 0；$n_k = \sum\limits_{(\boldsymbol{x},y)} I\{y = c_k\}$，$n_{jk} = \sum\limits_{(\boldsymbol{x},y)} I\{\boldsymbol{x}^{(j)} = x^{(j)}, y = c_k\}$。

两端取对数，得到对数似然函数：

$$\log \mathcal{L}(\boldsymbol{\theta}) = \sum_{k=1}^{K} n_k \log P(y^{(k)}) + \sum_{k=1}^{K} \sum_{j=1}^{d} n_{jk} \log P(x^{(j)}|y^{(k)}) \tag{12}$$

[2]注意，本节之前的 y 应理解成 K 个类别 $\{c_1, c_2, \ldots, c_K\}$ 中的某个取值。本节的 y 均使用 One-Hot 向量形式表示。

该对数似然函数的最优化，需要满足如下的约束条件：

$$\begin{cases} y^{(k)} \in \{0,1\} \\ \sum\limits_{k=1}^{K} y^{(k)} = 1 \\ P(y^{(k)}) \geqslant 0 \\ \sum\limits_{k=1}^{K} P(y^{(k)}) = 1 \qquad\qquad j = 1,2,\ldots,d \\ P(x^{(j)}|y^{(k)}) \geqslant 0 \\ \sum\limits_{x^{(j)}} P(x^{(j)}|y^{(k)}) = 1 \end{cases} \tag{13}$$

这是一个带约束的最优化问题，使用拉格朗日乘数法求解，写出相应的拉格朗日优化函数为[3]：

$$\begin{aligned} L(\boldsymbol{\theta}) = &-\sum_{k=1}^{K} n_k \log P(y^{(k)}) - \sum_{k=1}^{K}\sum_{j=1}^{d} n_{jk} \log P(x^{(j)}|y^{(k)}) + \\ &\alpha\left(1 - \sum_{k=1}^{K} y^{(k)}\right) + \beta\left(1 - \sum_{k=1}^{K} P(y^{(k)})\right) + \sum_{k=1}^{K}\sum_{j=1}^{d} \gamma_{kj}\left(1 - \sum_{x^{(j)}} P(x^{(j)}|y^{(k)})\right) \end{aligned} \tag{14}$$

注意，为便于求解，没有将 2 个不等式约束置入拉格朗日优化函数中。求解后，我们将验证不等式约束是否满足即可。首先，求解偏导数 $\frac{\partial L(\boldsymbol{\theta})}{\partial P(y^{(k)})}$：

$$\frac{\partial L(\boldsymbol{\theta})}{\partial P(y^{(k)})} = -\frac{n_k}{P(y^{(k)})} - \beta \tag{15}$$

令 $\frac{\partial L(\boldsymbol{\theta})}{\partial P(y^{(k)})} = 0$，得到：

$$\frac{n_k}{P(y^{(k)})} = -\beta \quad \Rightarrow \quad \beta P(y^{(k)}) = -n_k \tag{16}$$

上式两端对 k 求和，得到：

$$\beta = -\sum_{k=1}^{K} n_k \tag{17}$$

上式利用了公式 $\sum\limits_{k=1}^{K} P(y^{(k)}) = 1$。于是，得到：

$$P(y^{(k)}) = -\frac{n_k}{\beta} = \frac{n_k}{\sum\limits_{k=1}^{K} n_k} = \frac{\sum\limits_{(\boldsymbol{x},y)} I\{y = c_k\}}{N} \tag{18}$$

[3]按照惯例，将最大化问题转化为最小化问题。

其中 $\sum_{k=1}^{K} n_k = N$ 表示数据集中样本的个数。容易验证，约束条件 $P(y^{(k)}) \geqslant 0$ 得到满足。

然后，求解偏导数 $\frac{\partial L(\boldsymbol{\theta})}{\partial P(x^{(j)}|y^{(k)})}$：

$$\frac{\partial L(\boldsymbol{\theta})}{\partial P(x^{(j)}|y^{(k)})} = -\frac{n_{jk}}{P(x^{(j)}|y^{(k)})} - \gamma_{kj} \tag{19}$$

令 $\frac{\partial L(\boldsymbol{\theta})}{\partial P(x^{(j)}|y^{(k)})} = 0$，得到：

$$\frac{n_{jk}}{P(x^{(j)}|y^{(k)})} = -\gamma_{kj} \quad \Rightarrow \quad P(x^{(j)}|y^{(k)})\gamma_{kj} = -n_{jk} \tag{20}$$

上式两端对 $x^{(j)}$ 求和，利用公式 $\sum_{x^{(j)}} P(x^{(j)}|y^{(k)}) = 1$，得到：

$$\gamma_{kj} = -\sum_{x^{(j)}} n_{jk} \tag{21}$$

于是，得到：

$$\begin{aligned} P(x^{(j)}|y^{(k)}) = -\frac{n_{jk}}{\gamma_{kj}} = \frac{n_{jk}}{\sum\limits_{x^{(j)}} n_{jk}} &= \frac{\sum\limits_{(\boldsymbol{x},y)} I\{\boldsymbol{x}^{(j)} = x^{(j)}, y = c_k\}}{\sum\limits_{x^{(j)}} \sum\limits_{(\boldsymbol{x},y)} I\{\boldsymbol{x}^{(j)} = x^{(j)}, y = c_k\}} \\ &= \frac{\sum\limits_{(\boldsymbol{x},y)} I\{\boldsymbol{x}^{(j)} = x^{(j)}, y = c_k\}}{\sum\limits_{(\boldsymbol{x},y)} I\{y = c_k\}} \end{aligned} \tag{22}$$

容易验证，约束条件 $P(x^{(j)}|y^{(k)}) \geqslant 0$ 得到满足[4]。

需要注意的是，在推导 $P(x^{(j)}|y^{(k)})$ 的过程中，利用了公式 $\sum_{x^{(j)}} P(x^{(j)}|y^{(k)}) = 1$，即假设 $P(x^{(j)}|y^{(k)})$ 服从离散分布。如果 $P(x^{(j)}|y^{(k)})$ 服从连续分布，例如高斯分布，那么就直接使用数据集估算均值与方差参数，然后利用公式 (4) 计算后验概率求取最佳类别即可。

在实际应用中，如果估计的概率值 (公式 (18) 和公式 (22)) 出现 0 值，那么就会影响到后验概率的计算，从而使得分类产生偏差。一个可行的解决办法是，在分子分母上增加平滑项：

$$\begin{aligned} P(y^{(k)}) &= \frac{\sum\limits_{(\boldsymbol{x},y)} I\{y = c_k\} + \lambda}{N + K\lambda} \\ P(x^{(j)}|y^{(k)}) &= \frac{\sum\limits_{(\boldsymbol{x},y)} I\{\boldsymbol{x}^{(j)} = x^{(j)}, y = c_k\} + \lambda}{\sum\limits_{(\boldsymbol{x},y)} I\{y = c_k\} + M\lambda} \end{aligned} \tag{23}$$

[4]这意味着 $\sum_{(\boldsymbol{x},y)} I\{y = c_k\} > 0$，即给定的数据集 T 至少分配给每个类别一个样本点。显然，这一条件相当弱。

其中，K 表示类别个数，M 表示 $x^{(j)}$ 的取值个数，$\lambda \geqslant 0$。当 $\lambda = 1$ 时，被称为拉普拉斯平滑 (Laplacian Smoothing)。显然，平滑项的添加并没有改变上述公式的概率性质。

4 学习算法

在实际应用中，有如下三种常见的朴素贝叶斯模型：

- 高斯朴素贝叶斯 (Gaussian Naive Bayes)；
- 伯努利朴素贝叶斯 (Bernoulli Naive Bayes)；
- 多项朴素贝叶斯 (Multinomial Naive Bayes)；

对它们而言，类别先验概率 $P(y^{(k)})$ 的估计都是一样的，都使用了频率学派的最大似然估计，求解的结果也很直观——类别出现的概率近似为数据集中该类别出现的比例。

条件概率 $P(x^{(j)}|y^{(k)})$ 的估计则形成了不同的朴素贝叶斯模型。如果 $P(x^{(j)}|y^{(k)})$ 是离散分布，例如 (多元) 伯努利分布 (使用离散值 $\{0, 1\}$，表示特征出现与否) 和多项式分布 (使用离散整数值，表示特征出现的次数)，则其参数的估计就可以采用前述的参数估计方法，它们分别形成了伯努利朴素贝叶斯和多项朴素贝叶斯模型。如果 $P(x^{(j)}|y^{(k)})$ 是连续分布，例如高斯分布，那么参数 $P(x^{(j)}|y^{(k)})$ 的估计实际上就转变为对均值与方差的估计[5]，并形成了高斯朴素贝叶斯模型。实际上，根据问题中特征分量的离散或连续特征，这几种朴素贝叶斯可以混合使用。

对于特征分量 $x^{(j)}$ 为连续值情形，可以假设它服从高斯分布：

$$P(x^{(j)}|y^{(k)}) = \frac{1}{\sqrt{2\pi\sigma_{jk}^2}} \exp\left(-\frac{(x^{(j)} - \mu_{jk})^2}{2\sigma_{jk}^2}\right) \tag{24}$$

[5]由于朴素贝叶斯模型的特征分量独立性假设，多元高斯分布可以被当作多个一元高斯分布而分别处理。当然，参数的估算方法仍然使用最大似然估计，具体方法请参考《机器学习》课程系列之基础知识，Chapter1-CN.pdf。

它的 2 个模型参数均使用最大似然方法从数据集中进行估计：

$$\begin{aligned} \mu_{jk} &= \frac{\sum_{i=1}^{N} x_i^{(j)}}{N} \\ \sigma_{jk}^2 &= \frac{1}{N}\sum_{i=1}^{N}\left(x_i^{(j)} - \mu_{jk}\right)^2 \end{aligned} \tag{25}$$

可以看出，由于朴素贝叶斯模型的特征分量独立性假设，所以可以把每个连续特征分量当作一元高斯分布处理。

对于特征分量 $x^{(j)}$ 为离散值情形，如果只考虑特征出现与否 (二值型)，那么可以假设它服从伯努利分布。例如，在文本分类应用中，可以使用单词出现向量 (Word Occurrence Vector) 来表示输入特征。如果需要考虑特征的出现次数 (多值型)，那么可以假设它服从多项分布。例如，在文本分类应用中，也可以使用单词计数向量 (Word Count Vector) 来表示输入特征。其参数估算方法已经在上一节进行了讨论。

下面给出朴素贝叶斯模型的学习与预测算法。

算法 **4.1**（朴素贝叶斯模型的学习与预测算法）

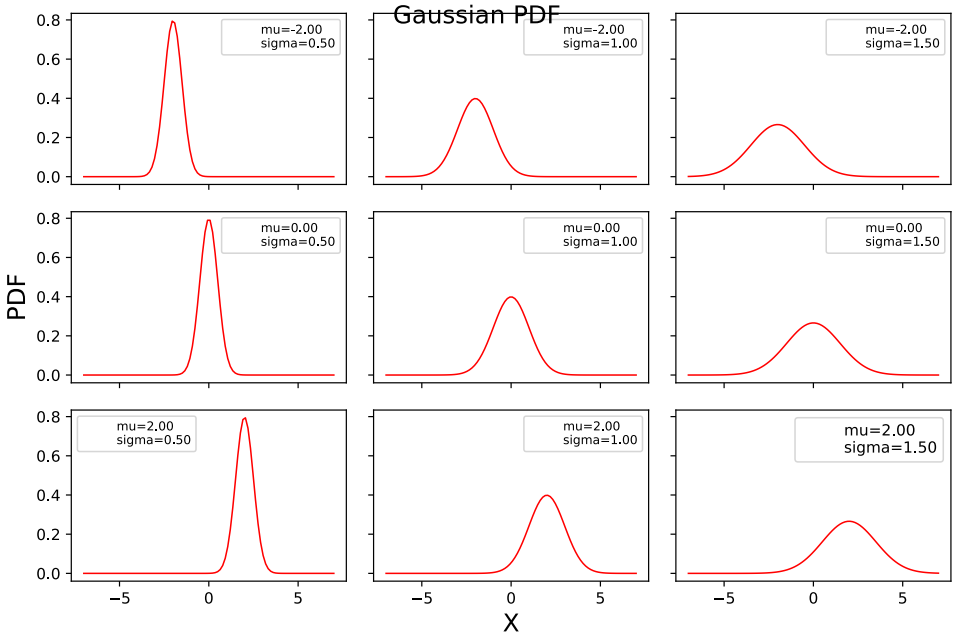
```
Input:
      Dataset: T
      New Sample: x
Output:
      Best Class: Ck
def NB_train_predict(T, x):
    Calculate P(Yk) in each class from T
    Repeat:
        if Xj is discrete feature:
            Calculate P(Xj|Yk) using Multinomial
        elif Xj is continuous feature:
            Calculate mean and sigma using Gaussian
    Until the probabilities of all features in each class have been
    calculated.
    Calculate posteriors for x, Ck, k=1,2,...
    return argmax(Ck)
```

5 朴素贝叶斯实验

绘制高斯概率密度函数曲线

```
In [1]: import numpy as np
        from scipy import stats
        import matplotlib.pyplot as plt
        %matplotlib inline
        %config InlineBackend.figure_format = 'svg'
```

```
In [2]: MU = [-2, 0, 2]
        STD = [0.5, 1, 1.5]
        X = np.linspace(-7, 7, 100)
        fig, AX = plt.subplots(len(MU), len(STD), sharex = True, sharey = True,
         figsize = (9, 6))
        for row in range(len(MU)):
          for col in range(len(STD)):
            mu = MU[row]
            std = STD[col]
            Y = stats.norm(mu, std).pdf(X) # 高斯概率密度函数
            AX[row, col].plot(X, Y, 'r', linewidth = 1)
            AX[row, col].plot(0, 0, label='mu={:3.2f}\nsigma={:3.2f}'.format(mu, std),
             alpha = 0)
            AX[row, col].legend(fontsize = 8, loc = 'best')
        AX[2, 1].set_xlabel('X', fontsize = 16)
        AX[1, 0].set_ylabel('PDF', fontsize = 16)
        plt.suptitle('Gaussian PDF', fontsize = 16)
        plt.tight_layout()
        plt.legend()
        plt.savefig('NB_OUTPUT1.pdf', bbox_inches='tight')
```



定义高斯朴素贝叶斯模型

```
In [3]: from collections import Counter
```

```
In [4]: class GaussianNaiveBayes:
            def __init__(self):
                self.model = None

            # 计算每列特征数据的高斯分布参数：均值与标准差
            def CalculateGaussian(self, data):
                return [(np.mean(feature), np.std(feature)) for feature in
                 zip(*data)]

            # 使用数据集对模型进行训练
            def fit(self, X, Y):
                labels_counter = Counter(Y) # 提取出类标签及相应的样本个数
                self.labels_prob = {label:counter/len(Y) for label, counter
                 in labels_counter.items()} # 生成类标签概率
                # 生成 dataset 字典数据，依类别存放各自的数据
                dataset = {label:[] for label in labels_counter.keys()}
                for x, y in zip(X, Y): # 分类别重新存放训练数据
                    dataset[y].append(x)
                # 依标签存放条件高斯概率分布 P(X|C) 的参数
                self.features_prob = {label:self.CalculateGaussian(data)
                 for label, data in dataset.items()}
                print('Gaussian Naive Bayes training done!')
                print('Label probability:', self.labels_prob)
                print('The parameters of features\' conditional probability:',
                 self.features_prob)

            # 给定 X，预测类标签
            def predict(self, X):
                results = []
                for x in X:
                    posterior = {}
                    for label in self.labels_prob.keys():
                        posterior[label] = self.labels_prob[label]
                        for i_feature, mustd in enumerate(self.features_prob[label]):
                            posterior[label] *= stats.norm(mustd[0], mustd[1]).
                             pdf(x[i_feature])
                    results.append(sorted(posterior.items(),
                     key = lambda x:x[-1])[-1][0]) # 取出后验概率最大的类别
                return results
```

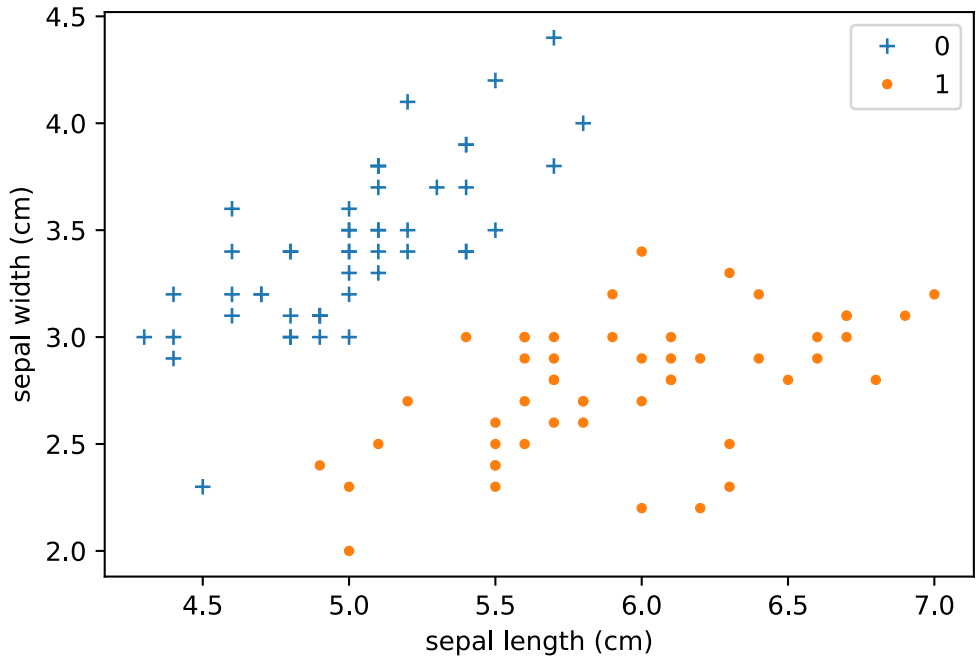
载入预存的鸢尾花数据集

```
In [5]: iris_npz = np.load('iris.npz')
        data = iris_npz['data']
```

```
X = iris_npz['X']
Y = iris_npz['Y']
```

绘制数据集中的数据

```
In [6]: plt.plot(data[:50, 0], data[:50, 1], '+', label='0')
        plt.plot(data[50:100, 0], data[50:100, 1], '.', label='1')
        plt.xlabel('sepal length (cm)')
        plt.ylabel('sepal width (cm)')
        plt.legend()
        plt.savefig('NB_OUTPUT2.pdf', bbox_inches='tight')
```



使用鸢尾花数据集训练高斯朴素贝叶斯模型

```
In [7]: model = GaussianNaiveBayes()
        model.fit(X, Y)
```

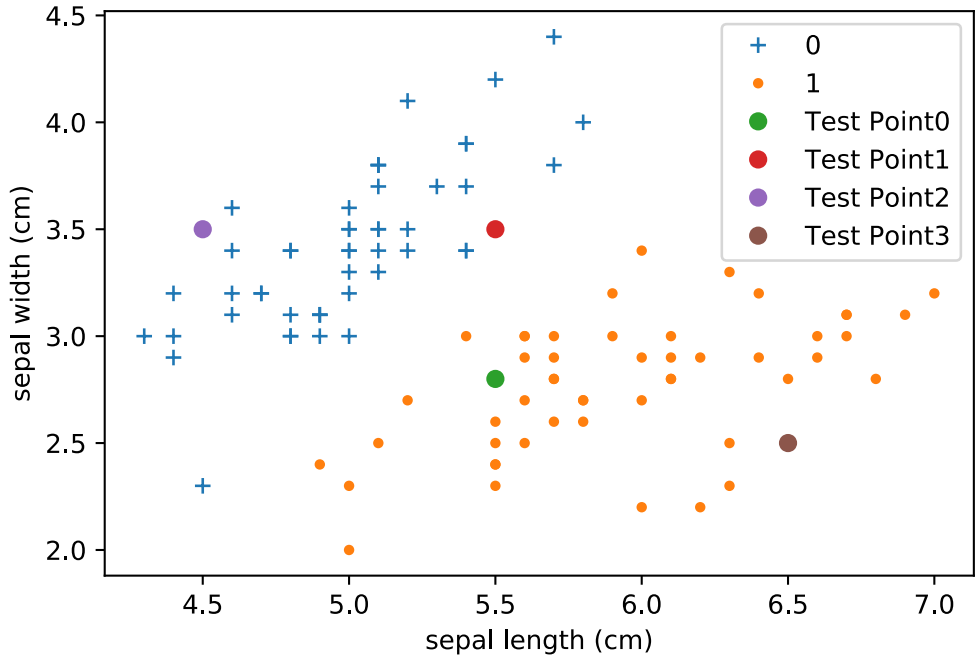
```
Gaussian Naive Bayes training done!
Label probability: {0.0: 0.5, 1.0: 0.5}
The parameters of features' conditional probability:
{0.0: [(5.0060000000000002, 0.3489469873777391),
(3.4180000000000001, 0.37719490982779713)],
 1.0: [(5.9359999999999999, 0.5109833656783751),
(2.7700000000000005, 0.31064449134018135)]}
```

预测测试

```
In [8]: XTEST = [(5.5, 2.8), (5.5, 3.5), (4.5, 3.5), (6.5, 2.5)]
        results = model.predict(XTEST)
        print(results)
```

```
[1.0, 0.0, 0.0, 1.0]
```

```
In [9]: plt.plot(data[:50, 0], data[:50, 1], '+', label='0')
        plt.plot(data[50:100, 0], data[50:100, 1], '.', label='1')
        for i, point in enumerate(XTEST):
            plt.plot(point[0], point[1], 'o', label='Test Point{0}'.format(i))
        plt.xlabel('sepal length (cm)')
        plt.ylabel('sepal width (cm)')
        plt.legend()
        plt.savefig('NB_OUTPUT3.pdf', bbox_inches='tight')
```



利用测试数据集进行测试，并计算正确率（Accuracy）

```
In [10]: from sklearn.model_selection import train_test_split
```

```
In [11]: np.random.seed()
         XTRAIN, XTEST, YTRAIN, YTEST = train_test_split(X, Y, test_size = 0.3)
         model = GaussianNaiveBayes()
         model.fit(XTRAIN, YTRAIN)
         results = model.predict(XTEST)
         scores = (results==YTEST)
         print('Accuracy = {:3.2f}%'.format(Counter(scores)[True]/len(YTEST) * 100))
```

```
Gaussian Naive Bayes training done!
Label probability: {0.0: 0.5285714285714286, 1.0: 0.4714285714285714}
The parameters of features' conditional probability:
{0.0: [(5.0162162162162165, 0.30180852486797044),
(3.4027027027027028, 0.29269159720391097)],
1.0: [(5.8878787878787877, 0.48788819866700073),
(2.7454545454545451, 0.32760925458762619)]}
Accuracy = 96.67%
```

使用鸢尾花数据集的所有数据，进行测试

```
In [12]: import pandas as pd
         from sklearn.datasets import load_iris

In [13]: iris = load_iris()
         iris_df = pd.DataFrame(iris.data, columns = iris.feature_names)
         iris_df['label'] = iris.target
         data = np.array(iris_df.iloc[:, :])
         XALL, YALL = data[:, :-1], data[:, -1]
         print(data.shape)
         print(XALL.shape)
         print(YALL.shape)
```

```
(150, 5)
(150, 4)
(150,)
```

保存数据集，以后可以直接使用

```
In [14]: np.savez_compressed('iris_full.npz', data = data, X = XALL, Y = YALL)
```

利用测试数据集进行测试，并计算正确率（Accuracy）

```
In [15]: np.random.seed()
         XTRAIN, XTEST, YTRAIN, YTEST = train_test_split(XALL, YALL,
          test_size = 0.3)
         model = GaussianNaiveBayes()
         model.fit(XTRAIN, YTRAIN)
         results = model.predict(XTEST)
         scores = (results==YTEST)
         print('Number of mislabeled points out of a total %d points : %d' %
          (len(YTEST), len(YTEST) - scores.sum()))
         print('Accuracy = {:3.2f}%'.format(Counter(scores)[True]/len(YTEST) * 100))
```

```
Gaussian Naive Bayes training done!
Label probability:
{0.0: 0.34285714285714286, 1.0: 0.3142857142857143, 2.0: 0.34285714285714286}
```

```
The parameters of features' conditional probability:
{0.0: [(4.9972222222222236, 0.32273665831027615),
(3.4027777777777781, 0.37229476904964481), (1.4833333333333334, 0.17078251276599329),
(0.21944444444444444, 0.077529365205293688)], 1.0: [(5.9212121212121209,
 0.54537036387437676), (2.7393939393939397, 0.34810255710968196),
 (4.2363636363636354, 0.51157403079194352), (1.3181818181818179, 0.21805551900007297)],
 2.0: [(6.6527777777777786, 0.60712716547831103), (3.0194444444444439,
 0.32302342918505017), (5.6055555555555552, 0.55474941512384401), (2.0500000000000003,
 0.26718699236468996)]}
Number of mislabeled points out of a total 45 points : 3
Accuracy = 93.33%
```

使用 sklearn.naive_bayes 库

```
In [16]: from sklearn.naive_bayes import GaussianNB

In [17]: clf = GaussianNB()
         clf.fit(XTRAIN, YTRAIN)

Out[17]: GaussianNB(priors=None)

In [18]: print('Accuracy = {:3.2f}%'.format(clf.score(XTEST, YTEST) * 100))

Accuracy = 93.33%
```

随机生成训练数据集与测试数据集进行伯努利朴素贝叶斯模型的测试

```
In [19]: from sklearn.naive_bayes import BernoulliNB

In [20]: X = np.random.randint(2, size=(6, 100))
         Y = np.array([1, 2, 3, 4, 5, 6])
         clf = BernoulliNB()
         clf.fit(X, Y)

Out[20]: BernoulliNB(alpha=1.0, binarize=0.0, class_prior=None, fit_prior=True)

In [21]: XTEST = np.random.randint(2, size=(4, 100))
         print(clf.predict(XTEST))

[1 3 2 3]
```

使用随机数据集进行多项式朴素贝叶斯模型的测试

```
In [22]: from sklearn.naive_bayes import MultinomialNB

In [23]: X = np.random.randint(5, size=(6, 100))
         Y = np.array([1, 2, 3, 4, 5, 6])
         clf = MultinomialNB()
         clf.fit(X, Y)
```

```
Out[23]: MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)

In [24]: XTEST = np.random.randint(5, size=(2, 100))
         print(clf.predict(XTEST))

[5 4]
```

6 参考文献

1. 周志华。《机器学习》，清华大学出版社，2016 年 1 月第 1 版。
2. Christopher M. Bishop. Pattern Recognition and Machine Learning. Springer, 2006.
3. Kevin P. Murphy. Machine Learning: A Probabilistic Perspective, The MIT Press, 2012.
4. 李航。《统计学习方法》，清华大学出版社，2019 年 5 月第 2 版。
5. Michael Collins. The Naive Bayes Model, Maximum-Likelihood Estimation, and the EM Algorithm. http://www.cs.columbia.edu/ mcollins/em.pdf.
6. https://github.com/wzyonggege/statistical-learning-method.
7. http://www.cnblogs.com/pinard/p/6069267.html.
8. https://machinelearningmastery.com/naive-bayes-classifier-scratch-python/.
9. https://scikit-learn.org/stable/modules/naive_bayes.html.
10. https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes. MultinomialNB.html.
11. https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.BernoulliNB.html.