1 机器学习引论

1.1 机器学习的定义

通俗而言, 机器学习是告诉机器一些已知的信息, 机器通过一定方式寻找其内在的规律, 然后对另一些未知的情形给出预测的过程. 严格一些的定义可以参考如下:

定义 1.1 机器学习的定义 这里给出两种比较通用的定义.

1. Herbert Simon 的定义: 如果一个系统, 能够通过执行某个过程, 就此改进了它的性能, 那么这个过程就是学习.
2. Tom Mitchell 的定义: A computer program is said to learn from experience **E** with respect to some task **T** and some performance measure **P**, if its performance on **T**, as measured by **P**, improves with experience **E**.

适合用机器学习去解决的问题, 一般满足如下条件: 有内在规律可以学习; 规律复杂, 很难通过解析或穷举的方法列清楚规则; 有足够多能够学习到规律的数据. 它尤其适用于那种机制不清 (如图像识别), 或者机制清但计算量太大 (如围棋, 量子力学计算), 能够接受近似 (误差) 以换取速度的问题.

1.2 机器学习的分类

机器学习大致可以分成三大类: **监督学习 (supervised learning)**, **无监督学习 (unsupervised learning)**, **强化学习 (reinforcement learning)**. 我们现在分别给出其定义和示例.

1.2.1 监督学习

定义 1.2 监督学习 监督学习是根据已经标注的数据集建立模型 (或函数), 并以此模式推测新的实例的学习过程.

记号 1.3 数据集 通常将监督学习的数据集记作 $\{\mathbf{x}_n, y_n\}(n = 1, \cdots, N)$, 其中输入量 $\mathbf{x}_i$ 是具有 d 个维度的矢量, 即

$$\mathbf{x}_i = \begin{bmatrix} x_{i1} \\ \vdots \\ x_{id} \end{bmatrix}$$

y_i 则为 $\mathbf{x}_i$ 对应的输出量. 输出有时也是一个向量, 此时也应当采取相应的表示.

根据数据输出的不同, 可以将监督学习分成两类: **回归 (Regression)**, **分类 (Classification)**.

定义 1.4 回归 如果数据集的输出 y 是连续的, 那么这一监督学习被称作**回归**.

回归的典型例子就是直线或曲线的拟合. 在下一讲中就主要讨论线性回归.

定义 1.5 分类 如果数据集的输出 y 是离散的, 那么这一监督学习被称作**分类**.

分类的典型例子就是数字的识别.

为了让机器能正确分析并处理样本, 我们需要将样本的性质进行量化, 即**特征**.

定义 1.6 特征 特征是样本在特征空间中的坐标分量, 是输入向量的基本组成部分, 每个特征对应样本的一项可观测或可计算的属性.

例如, 在预测反应的选择性时, 样本为反应体系, 特征则为底物的取代基, 溶剂, 温度, 反应时间等数据. 不同特征在模型中可能具有不同的重要性; 特征的质量和选择往往决定模型性能的上限.

1.2.2 无监督学习

定义 1.7 无监督学习 无监督学习是给定未标记的数据集 $\{\mathbf{x}_n\}$, 建立描述其内在关系的模型的学习过程.

无监督学习的常见例子包括聚类分析, 关联规则等.

1.2.3 强化学习

定义 1.8 强化学习 强化学习是让智能体通过与环境交互, 根据奖励反馈不断调整行为策略, 以最大化长期累计奖励的学习方法.

强化学习的常见例子就是各种棋类游戏的 AI.

1.3 机器学习的一般过程

定理 1.9 机器学习的一般过程 机器学习一般需要经过以下过程:

1. **训练**: 利用部分已知数据 (即**训练集, Training Set**) 进行拟合.
2. **测试**: 利用另一部分已知数据 (即**测试机, Test Set**) 检验训练结果的准确性.
3. **预测**: 利用前面得出的结论对未知情况进行预测.

2 线性回归

2.1 单变量线性拟合

2.1.1 简单多项式拟合与误差函数

实际的数据集通常没有明显的规律, 或者内含的规律被随机分布的噪声所隐藏. 对于回归问题, 最简单的方式就是通过多项式进行拟合.

假定训练集为包含 N 组数据的集合 $\{(x_1, y_1), \cdots, (x_N, y_N)\}$, 其中各 x_i 为自变量, 各 y_i 为真实值. 我们考虑形式如下的多项式函数以预测自变量 x 对应的值 (其中 $\hat{y}$ 表示 y 的预测值):

$$\hat{y} = f(x; \boldsymbol{w}) = w_0 + w_1 x + \cdots + w_M x^M = \sum_{j=0}^{M} w_j x^j$$

定义 2.1 拟合多项式的阶数 上述拟合多项式中最高次项的次数 M 被称作拟合多项式的**阶数**.

定义 2.2 权重矢量 权重矢量 $\boldsymbol{w}$ 是拟合多项式的参数构成的矢量. 例如, 上述多项式的权重矢量即为

$$\boldsymbol{w} = (w_0, w_1, \cdots, w_M)$$

给定多项式的阶数, 我们能写出很多可能的拟合多项式. 为了得出与数据集相差最小的多项式, 我们需要定义**误差函数**以衡量预测值与实际值之间的差距.

定义 2.3 误差函数 **误差函数 (Error Function)**(或称**代价函数 (Cost Function)**) 是衡量模型预测值 $\hat{y}$ 与实际值 y 之间的差距的函数.

机器学习中常用的误差函数为**平方和误差函数 (Sum-of-Square Error Function)**, 即**二乘**[1]**误差函数**:

$$E(\boldsymbol{w}) = \frac{1}{2} \sum_{i=1}^{N} (\hat{y}_i - y_i)^2 = \frac{1}{2} \sum_{i=1}^{N} (f(x_i; \boldsymbol{w}) - y_i)^2$$

定义 2.4 最小二乘多项式拟合 使用二乘误差函数的多项式拟合称作最小二乘多项式拟合.

容易看出这与统计学中最小二乘法的关系. 显然, 误差函数越小, 模型给出的预测值 $\hat{y}$ 整体而言与目标值 y 越接近.

2.1.2 拟合多项式的求法: 以线性回归为例

线性拟合是最简单的多项式拟合的方法. 使用一次函数模型

$$f(x) = ax + b$$

[1]这里的二乘即平方. 另外, 函数前的系数 1/2 是为了方便计算而采取的取法.

进行拟合, 那么权重矢量 $\boldsymbol{w}=(a,b)$. 我们有如下结论:

定理 2.5 简单线性回归的参数 假定训练集为包含 N 组数据的集合 $\{(x_1,y_1),\cdots,(x_N,y_N)\}$, 采取模型 $f(x)=ax+b$ 进行拟合, 误差函数为二乘误差函数, 那么得到的结果应为

$$a=\frac{\overline{xy}-\bar{x}\bar{y}}{\overline{x^2}-\bar{x}^2} \qquad b=\frac{\overline{x^2}\bar{y}-\bar{x}\cdot\overline{xy}}{\overline{x^2}-\bar{x}^2}$$

其中 $\bar{*}$ 代表基于训练集数据得到的平均值.

证明. 误差函数 $E(\boldsymbol{w})$ 的自变量为 $\boldsymbol{w}$, 具有两个独立的参数 a,b. 为了让 $E(\boldsymbol{w})$ 最小, 我们不妨对 a,b 分别求偏导, 并令偏导数为 0.

$$\begin{cases}\dfrac{\partial E}{\partial a}=\displaystyle\sum_{i=1}^{N}(ax_i+b-y_i)\,x_i=0\\ \dfrac{\partial E}{\partial b}=\displaystyle\sum_{i=1}^{N}(ax_i+b-y_i)=0\end{cases}$$

整理可得

$$\begin{cases}a\displaystyle\sum_{i=1}^{N}x_i^2+b\sum_{i=1}^{N}x_i-\sum_{i=1}^{N}x_iy_i=0\\ a\displaystyle\sum_{i=1}^{N}x_i+bN-\sum_{i=1}^{N}y_i=0\end{cases}$$

解这个方程组即可得到定理中的结论. □

对于一般的简单多项式拟合问题, 都可以用求偏导数的方法进行求解.

2.1.3 过拟合与正则化

用不同阶数的多项式对数据进行拟合, 可以看到, 线性拟合的效果并不好, 而阶数为 3 的多项式拟合则相对而言比较接近实际值. 随着阶数的增加, 拟合效果变得越来越好. 然而, 当阶数过高时 (例如这里采取的阶数为 10), 拟合曲线开始出现剧烈的振荡, 并且在数据点之间的区域偏离了真实函数. 可以想见, 在一般的测试集上, 高阶多项式的拟合效果会变得很差.

定义 2.6 过拟合 当模型在训练数据上表现良好, 但在未见过的数据上表现较差时, 我们称模型发生了**过拟合 (Overfitting)** 现象.

过拟合的原因在于, 复杂的函数具有更强的表达能力, 能够更好地拟合训练数据 (上面的 f_{10} 对每个数据点都没有偏差), 但同时也会受到数据中的噪声影响, 导致模型在训练数据之外的表现变差.

除了在测试集中较差的表现之外, 过拟合的另一个特征是模型的参数值变得非常大. 这和曲线的明显振荡是对应的.

解决过拟合问题的一个简单办法是增加测试集的规模, 或减小模型的复杂度. 另一种常用的方法是**正则化 (Regularization)**.

定义 2.7 正则化 **正则化 (Regularization)** 是通过在误差函数中加入对模型复杂度的惩罚项, 以防止模型过拟合的方法.

例如, 我们可以使用如下的正则化误差函数:

$$\tilde{E}(\boldsymbol{w}) = \frac{1}{2}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2 + \frac{\lambda}{2}||\boldsymbol{w}||^2$$

代替原来的误差函数进行拟合. 其中 λ 为正则化参数, 控制正则化项的权重. 如果 $\lambda = 0$, 那么没有惩罚;λ 越大, 惩罚越强, 此时模型倾向于用更小的参数进行拟合.λ 极大时,$\boldsymbol{w}$ 的各参数接近 0, 是一条接近横轴的平坦的线. 通过调整 λ, 我们可以在拟合训练数据和符合测试数据之间找到一个平衡点.

定义 2.8 岭回归 上述采用二乘误差函数与参数范数的平方作为正则化项的回归方法, 称为**岭回归 (Ridge Regression)**.

同样地, 采取不同的正则化项也可以得到不同的正则化方法.

定义 2.9 Lasso 回归 如果正则化项采用参数范数的绝对值, 即

$$\tilde{E}(\boldsymbol{w}) = \frac{1}{2}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2 + \lambda||\boldsymbol{w}||_1 = \frac{1}{2}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2 + \lambda\sum_{j=0}^{M}|w_j|$$

这种回归方法称为 **Lasso 回归 (Least Absolute Shrinkage and Selection Operator)**.

2.1.4 维度灾难

当输入数据 $\boldsymbol{x}$ 的分量较多时, 我们称其为高维数据. 例如, 一张 28×28 的灰度图像可以看作是一个 784 维的向量.

高维数据的一个重要问题是**维度灾难 (Curse of Dimensionality)**.

定义 2.10 维度灾难 维度灾难是指随着数据维度的增加, 数据量需求呈指数级增长, 从而导致计算和存储的巨大开销, 以及模型难以捕捉数据的真实分布等问题.

前面所举的例子中的 $\boldsymbol{x}$ 是一维的, 这是很容易用多项式拟合的. 但是, 如果类似的方法用到高维数据上就有很大的问题了. 对于一张分辨率为 640×480 的图像, 每个像素采用 RGB 表示, 那么一张图像就可以看作是一个 $640 \times 480 \times 3 = 921600$ 维的向量. 如果仍然采用多项式拟合, 那么需要的各阶参数数量为

$$N_0 = 1 \qquad N_1 = 921600 \qquad N_2 = C^2_{921600} = 424354112000 \qquad N_3 = C^3_{921600} = 1.303 \times 10^{17} \cdots$$

很难找到足够的训练数据来拟合这些参数.

简单而言, 维度的增加对数据量的要求是指数增长的, 而数据量本身却是有限的. 这就导致了维度灾难. 此

外, 高维空间中的点往往是稀疏分布的; 高维空间的球绝大部分分布于球面附近, 这些反直觉的性质使得模型难以捕捉数据的真实分布.

例如, 采用监督学习的方式让程序分辨猫和狗, 很有可能程序采用与猫和狗无关的特征来进行分类, 例如图像的亮度, 特定像素点的数目等, 从而导致模型在测试集上表现不佳.

2.2 从概率论的角度看待线性回归

2.2.1 正态分布与最小二乘法

通常, 我们假定观测值 y 可以写成下面的形式:

$$y = g(x; \boldsymbol{w}) + \epsilon$$

其中 $g(x; \boldsymbol{w})$ 是一个确定的函数, 显示了数据的内在规律; ϵ 为噪声. 大部分时候, 我们都假设噪声满足均值为 0, 方差为 σ 的正态分布.

定义 2.11 正态分布 随机变量 X 服从均值为 μ, 方差为 σ^2 的**正态分布 (Normal Distribution)**(或称**高斯分布 (Gaussian Distribution)**), 如果它的概率密度函数为

$$p(X = x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \equiv \mathcal{N}\left(x|\mu, \sigma^2\right)$$

这样, 对于给定的自变量 x_i 和观测值 t_i, 其概率为

$$P\left(t_i|x_i\right) = P\left(\varepsilon = t_i - g\left(x_i; \boldsymbol{w}\right)\right) = \mathcal{N}\left(t_i - g\left(x_i; \boldsymbol{w}\right)|0, \sigma^2\right) = \mathcal{N}\left(t_i|g\left(x_i; \boldsymbol{w}\right), \sigma^2\right)$$

假定各组数据是相互独立的, 并且各处的噪声满足同一分布 (即方差 σ^2 在各处相同) 那么整个数据集的概率函数为

$$P\left(\{t_i\}|\{x_i\}\right) = \prod_{i=1}^{N} P\left(t_i|x_i\right) = \prod_{i=1}^{N} \mathcal{N}\left(t_i|g\left(x_i; \boldsymbol{w}\right), \sigma^2\right)$$

在不引起混淆的情况下, 数据集也可以用粗体字母表示, 例如 $\mathbf{t} = \{t_i\}, \mathbf{x} = \{x_i\}$. 另外按机器学习领域的惯例, 记 $\beta = \dfrac{1}{\sigma^2}$, 于是上述概率函数可以写成

$$P\left(\mathbf{t}|\mathbf{x}, \boldsymbol{w}, \beta\right) = \prod_{i=1}^{N} \mathcal{N}\left(t_i|g\left(x_i, \boldsymbol{w}\right), \beta^{-1}\right)$$

在训练过程中, 我们只知道训练集 $\mathbf{x}$ 与 $\mathbf{t}$, 不知道参数 $\boldsymbol{w}$ 与 β. 直观而言[2], 我们希望选择一组参数 $\boldsymbol{w}, \beta$, 使得在这组参数下, 观测到训练集的概率最大. 这就是**极大似然估计 (Maximum Likelihood Estimation)** 的思想.

[2]我们将在后面详细讨论贝叶斯公式以完善对这种直观的严谨叙述.

对于连乘函数的最值估计, 通常先取对数后再考虑. 于是上述概率函数取对数后得到

$$\ln P\left(\mathbf{t}|\mathbf{x}, \boldsymbol{w}, \beta\right)=\sum_{i=1}^{N} \ln \mathcal{N}\left(t_i|g\left(x_i; \boldsymbol{w}\right), \beta^{-1}\right)=-\frac{\beta}{2} \sum_{i=1}^{N}\left(g\left(x_i; \boldsymbol{w}\right)-t_i\right)^2+\frac{N}{2} \ln \beta-\frac{N}{2} \ln (2 \pi)$$

与训练集有关的项即前面的求和项. 回顾二乘误差函数的定义:

$$E(\boldsymbol{w})=\frac{1}{2} \sum_{i=1}^{N}\left(g(x_i; \boldsymbol{w})-t_i\right)^2$$

仍然采用偏导等于 $\mathbf{0}$ 的办法, 即

$$\frac{\partial}{\partial \boldsymbol{w}} \ln P\left(\mathbf{t}|\mathbf{x}, \boldsymbol{w}, \beta\right)=-\beta \frac{\partial E(\boldsymbol{w})}{\partial \boldsymbol{w}}=\mathbf{0}$$

我们就可以得到与最小二乘法相同的结果, 即拟合函数为

$$f\left(x; \boldsymbol{w}_{\mathrm{ML}}\right)=g\left(x; \boldsymbol{w}_{\mathrm{ML}}\right)$$

这里的 $\boldsymbol{w}_{\mathrm{ML}}$ 表示极大似然估计得到的参数, 与前面推导中的一般权重矢量 $\boldsymbol{w}$ 不同.

这说明在噪声服从正态分布的假设下, 最小二乘法实际上是极大似然估计的一种实现方式. 另外, 我们还可以根据前面的推导求出噪声的表达形式:

$$\frac{1}{\beta}=\sigma^2=\frac{1}{N} \sum_{i=1}^{N}\left(f\left(x_i; \boldsymbol{w}_{\mathrm{ML}}\right)-t_i\right)^2$$

2.2.2 Bayes 定理

定理 2.12 Bayes 定理 设 A, B 为两个事件, 且 $P(B)>0$, 那么

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

其中 $P(A)$ 与 $P(B)$ 分别为事件 A 与事件 B 的概率, 又称为**先验概率**; $P(A|B)$ 与 $P(B|A)$ 分别为在事件 B 发生的条件下事件 A 发生的概率, 又称为**后验概率**与**似然函数**.

证明. 由条件概率的定义, 我们有

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

整理可得 Bayes 定理. □

例题 2.1 如果你的邻居购买了 n 张彩票, 并且其中有 m 张是中奖的, 请估计你购买一张彩票中奖的概率.

注意: 在你没有购买彩票之前, 你对中奖概率一无所知, 因此只能假定中奖概率是均匀分布的, 即它是 $[0,1]$ 上等概率取的一个数.

解. 由于本题涉及的中奖概率是连续变量, 因此下面的概率均为概率密度函数. 设中奖概率为 x, 那么买 n 张彩票而中 m 张的概率密度函数为

$$P(m,n|x)=C_n^m x^m(1-x)^{n-m}$$

根据 Bayes 定理, 我们有

$$P(x|m,n)=\frac{P(m,n|x)P(x)}{P(m,n)}$$

而

$$P(m,n)=\int_0^1 P(m,n|x)P(x)\mathrm{d}x$$

由于 x 在 $[0,1]$ 上均匀分布, 因此 $P(x)=1$. 于是

$$P(x|m,n)=\frac{P(m,n|x)}{\int_0^1 P(m,n|x)\mathrm{d}x}=\frac{C_n^m x^m(1-x)^{n-m}}{\int_0^1 C_n^m x^m(1-x)^{n-m}\mathrm{d}x}=\frac{x^m(1-x)^{n-m}}{\int_0^1 x^m(1-x)^{n-m}\mathrm{d}x}=\frac{x^m(1-x)^{n-m}}{\mathrm{B}(m+1,n-m+1)}$$

这里的 B 表示 Beta 函数. 对于连续变量 x 而言, 我们最好用期望来表示 x 的估计值. 因此

$$\hat{x}=\int_0^1 xP(x|m,n)\mathrm{d}x=\frac{\int_0^1 x^{m+1}(1-x)^{n-m}\mathrm{d}x}{\mathrm{B}(m+1,n-m+1)}=\frac{\mathrm{B}(m+2,n-m+1)}{\mathrm{B}(m+1,n-m+1)}=\frac{m+1}{n+2}$$

也即, 我们估计买彩票中奖的概率为 $\dfrac{m+1}{n+2}$. 这是符合直觉的, 因为当 $m=0$ 时, 我们也不应估计中奖概率为 0; 同理, 当 $m=n$ 时, 我们也不应估计中奖概率为 1. 买的越多, 估计的结果就越接近 m/n, 也就越准确.

2.2.3 Bayes 定理在机器学习中的应用

现在, 我们把目光重新放回机器学习上. 把 Bayes 定理写成如下形式:

$$p(\boldsymbol{w}|\mathcal{D})=\frac{p(\mathcal{D}|\boldsymbol{w})p(\boldsymbol{w})}{p(\mathcal{D})}$$

这里的 $\mathcal{D}$ 表示训练集, 而 $\boldsymbol{w}$ 表示模型参数.

定义 2.13 先验分布, 后验分布和似然函数 在上述 Bayes 定理中,$p(\boldsymbol{w})$ 称为**先验分布 (Prior Distribution)**, 表示在给定训练集之前对参数 $\boldsymbol{w}$ 的认识; $p(\boldsymbol{w}|\mathcal{D})$ 称为**后验分布 (Posterior Distribution)**, 表示在给定训练集之后对参数 $\boldsymbol{w}$ 的认识; $p(\mathcal{D}|\boldsymbol{w})$ 称为**似然函数 (Likelihood Function)**, 表示在给定参数 $\boldsymbol{w}$ 的条件下观测到训练集的概率.

公式中的 $p(\mathcal{D})$ 是一个归一化常数, 与参数 $\boldsymbol{w}$ 无关, 通常就不去特意考虑. 因此, 为了获取后验分布, 我们只需要计算似然函数与先验分布的乘积.

频率学派的处理方法 频率学派认为 $\boldsymbol{w}$ 是固定不变的, 不随数据变化, 因此拟合的目的是求出最优的 $\boldsymbol{w}$.

最简单的处理方法, 就是假定先验分布是均匀分布, 即对所有可能的参数 $\boldsymbol{w}$ 都一视同仁. 此时, 后验分布与似然函数成正比, 如果我们取概率最大的 $\boldsymbol{w}$ 作为拟合问题的解, 这时的似然函数就最大. 这就是**极大似然估计**.

定义 2.14 极大似然估计 通过选择使似然函数 $p(\mathcal{D}|\boldsymbol{w})$ 最大的参数 $\boldsymbol{w}$ 来拟合数据的方式, 称为**极大似然估计 (Maximum Likelihood Estimation)**.

有时, 我们已经对先验分布 $p(\boldsymbol{w})$ 有一定的估计. 综合考虑 $p(\mathcal{D}|\boldsymbol{w})$ 与 $p(\boldsymbol{w})$ 的乘积, 选择使得后验分布最大的参数 $\boldsymbol{w}$ 作为拟合问题的解, 这就是**极大后验估计**.

定义 2.15 极大后验估计 通过选择使后验分布 $p(\boldsymbol{w}|\mathcal{D})$ 最大的参数 $\boldsymbol{w}$ 来拟合数据的方式, 称为**极大后验估计 (Maximum A Posteriori Estimation)**.

下面给出了极大后验估计的一个例子.

例题 2.2 在过拟合一节中, 我们讲到参数越大越容易过拟合的情形, 即认为 $||\boldsymbol{w}||$ 越大越不可能, 这是与 $\mathcal{D}$ 无关的先验知识.
现在, 假定 $\boldsymbol{w}$ 的范数满足正态分布:

$$p(\boldsymbol{w}|\alpha) = \mathcal{N}\left(||\boldsymbol{w}|| \,\middle|\, 0, \frac{1}{\alpha}\right) = \left(\frac{\alpha}{2\pi}\right)^{\frac{M+1}{2}} \exp\left(-\frac{\alpha||\boldsymbol{w}||^2}{2}\right)$$

求极大后验估计的结果.

解. 由 Bayes 定理, 我们有

$$p(\boldsymbol{w}|\mathcal{D}) = \frac{p(\mathcal{D}|\boldsymbol{w})p(\boldsymbol{w}|\alpha)}{p(\mathcal{D})}$$

由于 $p(\mathcal{D})$ 与 $\boldsymbol{w}$ 无关, 因此我们只需考虑分子部分. 取对数后得到

$$\ln p(\boldsymbol{w}|\mathcal{D}) = \ln p(\mathcal{D}|\boldsymbol{w}) + \ln p(\boldsymbol{w}|\alpha) + \text{const}$$

假定似然函数仍然为

$$P(\mathcal{D}|\boldsymbol{w}, \beta) = \prod_{i=1}^{N} \mathcal{N}\left(t_i | g(x_i, \boldsymbol{w}), \beta^{-1}\right)$$

忽略与 $\boldsymbol{w}$ 无关的项可得

$$\ln p(\boldsymbol{w}|\mathcal{D}) = -\frac{\beta}{2}\sum_{i=1}^{N}\left(g(x_i;\boldsymbol{w}) - t_i\right)^2 - \frac{\alpha}{2}||\boldsymbol{w}||^2 + \text{const}$$

这与前述正则化误差函数

$$\tilde{E}(\boldsymbol{w}) = \frac{1}{2}\sum_{i=1}^{N}\left(g(x_i;\boldsymbol{w}) - t_i\right)^2 + \frac{\lambda}{2}||\boldsymbol{w}||^2$$

在形式上一致. 事实上, 如果取 $\lambda = \dfrac{\alpha}{\beta}$, 那么极大后验估计与正则化误差函数的结果是一样的.

同样不难想到, 对 $p(\boldsymbol{w})$ 的不同估计对应不同的正则化误差函数.

Bayes 学派的处理方法 Bayes 学派与频率学派观点相左. 他们认为数据才是固定的, 模型的参数则是随机的, 并且服从某种分布[3]. 我们只能对模型的参数有一个最初的估计[4], 这一估计会随着数据的增加而改变, 最终得到比较准确的结论.

定理 2.16 Bayes 回归 设训练集为 $\mathcal{D}$, 根据 Bayes 回归可以得到任意输入 x 的预测值 t 的概率密度函数

$$p(t|x,\mathcal{D}) = \int p(t|x,\boldsymbol{w})p(\boldsymbol{w}|\mathcal{D})\mathrm{d}\boldsymbol{w} = \mathcal{N}\left(t|m(x), s^2(x)\right)$$

其中

$$\boldsymbol{\phi}(x) = \left(1, x, \cdots, x^M\right)^{\mathrm{t}}$$

$$m(x) = \beta\,\boldsymbol{\phi}(x)^{\mathrm{t}}\,\boldsymbol{S}\sum_{i=1}^{N} t_i\,\boldsymbol{\phi}(x_i)$$

$$s^2(x) = \beta^{-1} + \boldsymbol{\phi}(x)^{\mathrm{t}}\,\boldsymbol{S}\boldsymbol{\phi}(x)$$

$$\boldsymbol{S}^{-1} = \alpha\boldsymbol{I} + \beta\sum_{i=1}^{N}\boldsymbol{\phi}(x_i)\,\boldsymbol{\phi}(x_i)^{\mathrm{t}}$$

这里的 α, β 为先验分布与似然函数的参数, $\boldsymbol{I}$ 为单位矩阵.

证明. 我们仍然假定观测值 y 可以写成下面的形式:

$$y = g(\boldsymbol{x};\boldsymbol{w}) + \epsilon$$

其中 ε 服从均值为 0, 方差为 β^{-1} 的正态分布. 假定 $\boldsymbol{w}$ 的先验也是多元高斯分布[5], 即:

$$p(\boldsymbol{w}) = \mathcal{N}\left(\boldsymbol{w}|0, \alpha^{-1}\boldsymbol{I}\right)$$

即各分量 w_i 满足均值为 0, 方差为 α^{-1} 的正态分布, 并且各分量相互独立. 由于噪声是独立同分布, 因此有

$$p\left(\mathcal{D}|\boldsymbol{w}\right) = \prod_{i=1}^{N}\mathcal{N}\left(t_i|g\left(\boldsymbol{x}_i;\boldsymbol{w}\right), \beta^{-1}\right)$$

取对数可得

$$\ln p\left(\mathcal{D}|\boldsymbol{w}\right) = -\frac{\beta}{2}\sum_{i=1}^{N}\left(t_i - g\left(\boldsymbol{x}_i;\boldsymbol{w}\right)\right)^2 + \mathrm{const}$$

将上式中的平方项展开并忽略与 $\boldsymbol{w}$ 无关的项, 可得

$$\ln p\left(\mathcal{D}|\boldsymbol{w}\right) = -\frac{\beta}{2}\left(\boldsymbol{w}^{\mathrm{t}}\sum_{i=1}^{N}\boldsymbol{\phi}\left(\boldsymbol{x}_i\right)\boldsymbol{\phi}\left(\boldsymbol{x}_i\right)^{\mathrm{t}}\boldsymbol{w} - 2\sum_{i=1}^{N}t_i\,\boldsymbol{\phi}\left(\boldsymbol{x}_i\right)^{\mathrm{t}}\boldsymbol{w}\right) + \mathrm{const}$$

[3]例如, 抛一枚硬币, 如果已经抛出了连续 100 次正面, 那么我们就有理由怀疑这枚硬币不是均匀的, 而是正面朝上的概率更大.

[4]这和极大后验估计是一样的, 但 Bayes 回归会

[5]多元高斯分布的概率密度函数为

$$p(\boldsymbol{x}) = \frac{1}{(2\pi)^{n/2}|\boldsymbol{\Sigma}|^{1/2}}\exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^{\mathrm{t}}\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right)$$

其中 $\boldsymbol{\mu}$ 为均值矢量, $\boldsymbol{\Sigma}$ 为协方差矩阵. 对于这里的先验分布, 均值矢量为 $\boldsymbol{0}$, 协方差矩阵为 $\alpha^{-1}\boldsymbol{I}$. 事实上, 多元高斯分布等价于每个分量满足独立的一元高斯分布.

先验概率 $p(\boldsymbol{w})$ 的对数为

$$\ln p(\boldsymbol{w}) = -\frac{\alpha}{2}\boldsymbol{w}^{\mathrm{t}}\boldsymbol{w} + \text{const}$$

由 Bayes 定理, 我们有

$$\begin{aligned}
\ln p(\boldsymbol{w}|\mathcal{D}) &= \ln p\left(\mathcal{D}|\boldsymbol{w}\right) + \ln p(\boldsymbol{w}) + \text{const} \\
&= -\frac{1}{2}\left(\beta\boldsymbol{w}^{\mathrm{t}}\sum_{i=1}^{N}\boldsymbol{\phi}\left(\boldsymbol{x}_i\right)\boldsymbol{\phi}\left(\boldsymbol{x}_i\right)^{\mathrm{t}}\boldsymbol{w} + \alpha\boldsymbol{w}^{\mathrm{t}}\boldsymbol{w}\right) + \beta\sum_{i=1}^{N}t_i\,\boldsymbol{\phi}\left(\boldsymbol{x}_i\right)^{\mathrm{t}}\boldsymbol{w} + \text{const} \\
&= -\frac{1}{2}\boldsymbol{w}^{\mathrm{t}}\left(\beta\sum_{i=1}^{N}\boldsymbol{\phi}\left(\boldsymbol{x}_i\right)\boldsymbol{\phi}\left(\boldsymbol{x}_i\right)^{\mathrm{t}} + \alpha\boldsymbol{I}\right)\boldsymbol{w} + \left(\beta\sum_{i=1}^{N}t_i\boldsymbol{\phi}\left(\boldsymbol{x}_i\right)^{\mathrm{t}}\right)\boldsymbol{w} + \text{const}
\end{aligned}$$

令

$$\boldsymbol{S}^{-1} = \alpha\boldsymbol{I} + \beta\sum_{i=1}^{N}\boldsymbol{\phi}\left(\boldsymbol{x}_i\right)\boldsymbol{\phi}\left(\boldsymbol{x}_i\right)^{\mathrm{t}} \qquad \boldsymbol{b} = \beta\sum_{i=1}^{N}t_i\boldsymbol{\phi}\left(\boldsymbol{x}_i\right)^{\mathrm{t}}$$

则有

$$\ln p(\boldsymbol{w}|\mathcal{D}) = -\frac{1}{2}\boldsymbol{w}^{\mathrm{t}}\boldsymbol{S}^{-1}\boldsymbol{w} + \boldsymbol{b}^{\mathrm{t}}\boldsymbol{w} + \text{const}$$

我们可以发现, 上述式子恰好可以写作平方展开的形式. 令 $\boldsymbol{S}^{-1}\boldsymbol{m} = \boldsymbol{b}$, 则有

$$\ln p(\boldsymbol{w}|\mathcal{D}) = -\frac{1}{2}\left(\boldsymbol{w}-\boldsymbol{m}\right)^{\mathrm{t}}\boldsymbol{S}^{-1}\left(\boldsymbol{w}-\boldsymbol{m}\right) + \text{const}$$

这说明后验分布 $p(\boldsymbol{w}|\mathcal{D})$ 也是一个高斯分布, 其均值为 $\boldsymbol{m}$, 协方差矩阵为 $\boldsymbol{S}$:

$$p(\boldsymbol{w}|\mathcal{D}) = \mathcal{N}\left(\boldsymbol{w}|\boldsymbol{m}, \boldsymbol{S}\right)$$

最后用 Bayes 定理计算预测值 t 的概率密度函数:

$$\begin{aligned}
p(t|x,\mathcal{D}) &= \int p(t|x,\boldsymbol{w})p(\boldsymbol{w}|\mathcal{D})\mathrm{d}\boldsymbol{w} \\
&= \int \mathcal{N}\left(t|\boldsymbol{w}^{\mathrm{t}}\boldsymbol{\phi}\left(x\right), \beta^{-1}\right)\mathcal{N}\left(\boldsymbol{w}|\boldsymbol{m}, \boldsymbol{S}\right)\mathrm{d}\boldsymbol{w} \\
&= \mathcal{N}\left(t|m(x), s^2(x)\right)
\end{aligned}$$

□

2.3 基函数

2.3.1 基函数的概念

定义 2.17 基函数 回归模型的函数 $f\left(\boldsymbol{x};\boldsymbol{w}\right)$ 可以表示为一组函数的线性组合:

$$f\left(\boldsymbol{x};\boldsymbol{w}\right) = w_0\phi_0\left(\boldsymbol{x}\right) + \cdots + w_M\phi_M\left(\boldsymbol{x}\right) = \sum_{j=0}^{M}w_j\phi_j\left(\boldsymbol{x}\right) = \boldsymbol{w}^{\mathrm{t}}\boldsymbol{\phi}\left(\boldsymbol{x}\right)$$

这组函数 $\phi_0\left(\boldsymbol{x}\right), \cdots, \phi_M\left(\boldsymbol{x}\right)$ 称为**基函数 (Basis Functions)**.

2.3.2 常见的基函数模型

多项式基函数 多项式基函数的形式为

$$\phi_j(x) = x^j (j = 0, 1, \cdots, M)$$

多项式基函数具有全局性, 一处的改变会影响所有函数的改变.

高斯基函数 高斯基函数的形式为

$$\phi_j(x) = \exp\left(-\frac{(x-\mu_j)^2}{2\sigma^2}\right)$$

其中 μ_j 为高斯基函数的中心, σ 为宽度. 高斯基函数具有局部性, 一处的改变只会影响与其中心相近的函数.

Sigmoid 基函数 Sigmoid 基函数的形式为

$$\phi_j(x) = \frac{1}{1+\exp(-\beta(x-\mu_j))}$$

其中 μ_j 为 Sigmoid 基函数的中心, β 为宽度. Sigmoid 基函数具有平滑性, 在中心附近变化较大, 而在两侧变化较小.

Sigmoid 基函数常用于神经网络中, 作为激活函数.

2.3.3 基函数用于回归

如果仍然采用前面的假设, 即观测值 y 可以写成下面的形式:

$$y = f(\boldsymbol{x}; \boldsymbol{w}) + \epsilon$$

其中 $f(\boldsymbol{x}; \boldsymbol{w})$ 是由基函数构成的线性组合:

$$f(\boldsymbol{x}; \boldsymbol{w}) = \sum_{j=0}^{M} w_j \phi_j(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{t}} \boldsymbol{\phi}(\boldsymbol{x})$$

那么通过极大似然估计可以得到

$$\boldsymbol{w}_{ML} = \left(\boldsymbol{\Phi}^{\mathrm{t}}\boldsymbol{\Phi}\right)^{-1}\boldsymbol{\Phi}^{\mathrm{t}}\boldsymbol{y} = \boldsymbol{\Phi}^{\dagger}\boldsymbol{y}$$

其中**设计矩阵 $\boldsymbol{\Phi}$** 为

$$\boldsymbol{\Phi} = \begin{bmatrix} \phi_0(\boldsymbol{x}_1) & \cdots & \phi_M(\boldsymbol{x}_1) \\ \vdots & \ddots & \vdots \\ \phi_0(\boldsymbol{x}_N) & \cdots & \phi_M(\boldsymbol{x}_N) \end{bmatrix}$$

当数据太多时, 可以采用循序学习的方式, 即每次只用一组数据进行更新.

定义 2.18 循序学习 循序学习 (Sequential Learning) 是指每次只用一组数据进行更新的学习方式. 对于给定的一组数据 $(\boldsymbol{x}_n, t_n)$, 我们有

$$\boldsymbol{w}^{(n)} = \boldsymbol{w}^{(n-1)} + \eta \left(t_n - \left(\boldsymbol{w}^{(n-1)} \right)^{\mathrm{t}} \boldsymbol{\phi} \left(\boldsymbol{x}_n \right) \right) \boldsymbol{\phi} \left(\boldsymbol{x}_n \right)$$

这里的 η 为学习率, 控制每次更新的幅度.

循序学习也称为在线学习, 在神经网络与深度学习中被称作随机梯度下降.

另外, 如果采用正则化误差函数, 例如岭回归, 那么结果为

$$\boldsymbol{w}_{ML} = \left(\lambda \boldsymbol{I} + \boldsymbol{\Phi}^{\mathrm{t}} \boldsymbol{\Phi} \right)^{-1} \boldsymbol{\Phi}^{\mathrm{t}} \boldsymbol{y}$$

3 分类的线性方法

3.1 线性拟合与感知器

既然分类的输出是离散值, 它当然也属于连续值. 于是, 我们可以简单地沿用线性回归方法, 即

$$y = g(\boldsymbol{x}; \boldsymbol{w}) = w_0 x_0 + \cdots + w_M x_M = \boldsymbol{w}^{\mathrm{t}} \boldsymbol{x}$$

回归模型输出的是连续值, 因此需要离散化. 例如, 当输出值为 $\{0, 1\}$ 时, 可以使用

$$g(\boldsymbol{x}) = \begin{cases} 1, & f(\boldsymbol{x}; \boldsymbol{w}) \geqslant \dfrac{1}{2} \\ 0, & f(\boldsymbol{x}; \boldsymbol{w}) < \dfrac{1}{2} \end{cases}$$

将结果离散化. 然而, 如果出现远离数据聚集处的点, 那么线性回归可能会因为迎合这些点而产生较大偏差. 因此, 我们可以直接用阶跃函数进行拟合, 即

$$y = g(\boldsymbol{x}; \boldsymbol{w}) = f\left(\boldsymbol{w}^{\mathrm{t}} \boldsymbol{x}\right) = H\left(\boldsymbol{w}^{\mathrm{t}} \boldsymbol{x}\right), \quad \text{where } H(x) = \begin{cases} 1, & x \geqslant 0 \\ 0, & x < 0 \end{cases}$$

其中 $f(z)$ 为激活函数, 对输入的 z 进行非线性变换得到结果. 这里采用的激活函数为阶跃函数 $H(z)$. 直接用上述定义的 $g(\boldsymbol{x}; \boldsymbol{w})$ 进行拟合 (例如进行最大似然法估计等), 这就是**感知器 (perceptron)** 模型.

定义 3.1 感知器 使用阶跃函数 $H(z)$ 作为激活函数的线性分类模型, 即

$$y = g(\boldsymbol{x}; \boldsymbol{w}) = H\left(\boldsymbol{w}^{\mathrm{t}} \boldsymbol{x}\right)$$

称为**感知器 (perceptron)** 模型.

3.2 逻辑回归

3.2.1 逻辑回归的模型

感知器模型的激活函数是阶跃函数, 它在 $x = 0$ 处不可导, 因此无法使用梯度下降法等涉及导数的方法进行优化. 为了替换成光滑的函数以便优化, 我们可以用 sigmoid 函数 $\sigma(z)$ 替换前面的阶跃函数:

$$y = g\left(\boldsymbol{x}; \boldsymbol{w}\right) = \sigma\left(\boldsymbol{w}^{\mathrm{t}} \boldsymbol{x}\right), \quad \text{where } \sigma(z) = \frac{1}{1 + e^{-z}}$$

这就是**逻辑回归 (logistic regression)** 模型.

定义 3.2 逻辑回归 使用 sigmoid 函数 $\sigma(z)$ 作为激活函数的线性分类模型, 即

$$y = g\left(\boldsymbol{x}; \boldsymbol{w}\right) = \sigma\left(\boldsymbol{w}^{\mathrm{t}} \boldsymbol{x}\right), \quad \text{where } \sigma(z) = \frac{1}{1 + e^{-z}}$$

称为**逻辑回归 (logistic regression)** 模型.

上述函数返回一个 $(0,1)$ 间的连续值. 因此, 我们需要将结果解读为样本属于某一类别的概率.

当 $\sigma\left(\boldsymbol{w}^{\mathrm{t}}\boldsymbol{x}\right)>0.5$ 时, 我们预测样本属于类别 1 的概率更大; 否则预测样本属于类别 0 的概率更大. 所预测的两种类别的边界称作**决策面 (Decision Boundary)**, 由下式描述:

$$\boldsymbol{w}^{\mathrm{t}}\boldsymbol{x}=0$$

这是 $\boldsymbol{x}$ 的空间中的一个线性平面.

3.2.2 逻辑回归的求解

与前面一样, 逻辑回归也可以归结为一个概率优化问题, 使用极大似然法即可. 仍然假定数据集为 $\mathcal{D}=\{\boldsymbol{x}_n,t_n\}_{n=1}^N$. 当 $t_n=1$ 时, 模型预测结果与 t_n 一致的概率时 $y_n=\sigma(\boldsymbol{w}^{\mathrm{t}}\boldsymbol{x}_n)$, 反之则为 $1-y_n$. 综合而言, 模型预测正确的概率为

$$p(t_n|\boldsymbol{w})=y_n^{t_n}(1-y_n)^{1-t_n}$$

假定数据点之间是独立的, 似然函数可以写成

$$p(\boldsymbol{t}|\boldsymbol{w})=\prod_{n=1}^{N}y_n^{t_n}(1-y_n)^{1-t_n}$$

定义交叉熵误差函数

$$E(\boldsymbol{w})=-\ln p(\boldsymbol{t}|\boldsymbol{w})=-\sum_{n=1}^{N}\left[t_n\ln y_n+(1-t_n)\ln(1-y_n)\right]$$

这样做是因为取对数后的函数是凸函数, 可以更好地进行求解. 为了利用梯度下降法求解 $\boldsymbol{w}$ 的最佳值, 我们先来推导误差函数的梯度 $\nabla_{\boldsymbol{w}}E(\boldsymbol{w})$. 首先有

$$\frac{\mathrm{d}\sigma(z)}{\mathrm{d}z}=\frac{-\mathrm{e}^{-z}}{-(1+\mathrm{e}^{-z})^2}=\sigma(z)[1-\sigma(z)]$$

于是

$$\begin{aligned}\nabla_{\boldsymbol{w}}E(\boldsymbol{w})&=-\sum_{n=1}^{N}\left(\frac{t_n}{y_n}-\frac{1-t_n}{1-y_n}\right)\frac{\mathrm{d}y_n}{\mathrm{d}\boldsymbol{w}}\\&=-\sum_{n=1}^{N}\left(\frac{t_n}{y_n}-\frac{1-t_n}{1-y_n}\right)\frac{\mathrm{d}\sigma(\boldsymbol{w}^{\mathrm{t}}\boldsymbol{x}_n)}{\mathrm{d}\boldsymbol{w}}\\&=-\sum_{n=1}^{N}\left(\frac{t_n}{y_n}-\frac{1-t_n}{1-y_n}\right)y_n(1-y_n)\boldsymbol{x}_n\\&=\sum_{n=1}^{N}(y_n-t_n)\boldsymbol{x}_n\end{aligned}$$

这一矢量给出了 $E(\boldsymbol{w})$ 随 $\boldsymbol{w}$ 增加最快的方向和速度. 因此, 梯度下降的迭代求解可以写为

$$\boldsymbol{w}^{(\tau+1)}=\boldsymbol{w}^{(\tau)}-\eta\nabla_{\boldsymbol{w}}E(\boldsymbol{w})$$

当数据个数太多时, 梯度计算公式中的求和将耗费大量时间. 考虑到迭代过程本身就是一个逐步逼近最优解的过程, 每次的移动方向不一定要很精确, 因此可以利用部分数据而非全部数据计算梯度. 在极端条件下, 每次只利用一个训练数据点, 即

$$\boldsymbol{w}^{(\tau+1)} = \boldsymbol{w}^{(\tau)} - \eta \nabla_{\boldsymbol{w}} E_n(\boldsymbol{w}), \quad E_n(\boldsymbol{w}) = (y_n - t_n)\boldsymbol{x}_n$$

3.2.3 基函数和正则化的使用

基函数和正则化的使用与在线性回归中是类似的, 这里不多赘述. 引入基函数可以更好地描述决策面的形状, 它在基函数的线性空间中是一个平面, 但在样本数据空间中则可以表现地更为复杂.

3.2.4 多分类问题

处理多分类问题常有以下两种方法.

定义 3.3 一对余方法 **一对余方法 (one-versus-the-rest)** 的基本想法是每次考虑将样本分为一个类别和其余类别, 从而将多分类问题转化为多个二分类问题.

一对余方法适用于分类不互斥的方法.

定义 3.4 softmax 方法 **softmax** 方法对每个类别 i 定义

$$z_i(\boldsymbol{x}) = \boldsymbol{w}_{(i)}^{\mathrm{t}} \boldsymbol{x}$$

然后利用如下 softmax 函数计算 $\boldsymbol{x}$ 属于类别 i 的概率

$$y^{(i)}(\boldsymbol{x}) = \frac{\mathrm{e}^{z_i}}{\sum\limits_i \mathrm{e}^{z_i}} = \frac{\mathrm{e}^{\boldsymbol{w}_{(i)}^{\mathrm{t}} \boldsymbol{x}}}{\sum\limits_i \mathrm{e}^{\boldsymbol{w}_{(i)}^{\mathrm{t}} \boldsymbol{x}}}$$

softmax 方法满足各类别概率之和等于 1 的要求, 因此特别适用于类别互斥的情形.

4 模型选择与评估

4.1 模型的训练, 选择与交叉检验

4.1.1 训练集, 验证集, 测试集

对于同一问题, 我们可以采取不同的模型来进行训练. 即使模型相同, 其中的超参数也可以不同. 如何选择最优的模型就是本节需要解决的问题.

我们在前面主要通过模型在测试集上的表现来评价模型. 然而, 如果用模型在测试集的误差来选择模型, 相当于把测试集的数据也作为训练过程的一部分, 这就导致了测试集的**数据泄露 (data leakage)**. 这会导致模型泛化能力的下降.

因此, 较好的做法如下:

定义 4.1 训练集, 验证集, 测试集 为了选择更优的模型的同时防止数据泄露, 可以将数据集划分为三部分: **训练集 (Training Set)**, **验证集 (Validation Set)** 和**测试集 (Test Set)**. 其中训练集用于训练模型, 验证集用于选择模型, 测试集用于评估模型的最终性能.

以监督学习为例, 记已知数据集 $\mathcal{D}:\{\boldsymbol{x}_n,t_n\}$. 假定模型给出的预测函数为 $\hat{f}(\boldsymbol{x};\boldsymbol{w})$, 单个样本的损失函数为 $L(t,\hat{f}(\boldsymbol{x};\boldsymbol{w}))$, 则任意数据集 $\mathcal{D}$ 上的误差可以表示为:

$$E(\boldsymbol{w},\mathcal{D})=\sum_{(\boldsymbol{x},t)\in\mathcal{D}}L(t,\hat{f}(\boldsymbol{x};\boldsymbol{w}))$$

在使用训练集 $\mathcal{D}_{\text{train}}$ 训练时, 我们一般还考虑正则化项, 因此训练的目标为:

$$\boldsymbol{w}(\lambda)=\arg\min_{\boldsymbol{w}}\tilde{E}\left(\boldsymbol{w},\mathcal{D}_{\text{train}}\right),\quad \tilde{E}\left(\boldsymbol{w},\mathcal{D}_{\text{train}}\right)=E\left(\boldsymbol{w},\mathcal{D}_{\text{train}}\right)+\frac{\lambda}{2}||\boldsymbol{w}||^2$$

此时模型在验证集 $\mathcal{D}_{\text{val}}$ 上的误差称作验证误差, 即 (注意这里不包含正则化项)

$$E_{\text{val}}(\lambda)=E\left(\boldsymbol{w},\mathcal{D}_{\text{val}}\right)$$

选择模型, 也就是选择使得验证误差最小的超参数 λ:

$$\lambda^*=\arg\min_{\lambda}E_{\text{val}}(\lambda)$$

最后, 用选择出的超参数 λ 和对应的模型参数 $\boldsymbol{w}$ 在测试集 $\mathcal{D}_{\text{test}}$ 上评估其性能, 即计算测试误差:

$$E_{\text{test}}=E\left(\boldsymbol{w}(\lambda^*),\mathcal{D}_{\text{test}}\right)$$

这就可以衡量模型的最终效果, 即泛化能力.

关于如何划分训练集, 验证集和测试集, 一般的做法是将数据集随机划分为 50% 的训练集,25% 的验证集和 25% 的测试集 (也有人建议为 60%,20%,20%). 如果数据量较少, 也可以考虑使用交叉检验的方法来更有效地利用数据.

4.1.2 K 折交叉检验

在数据不足的情况下, 如果还要划分出前述三个数据集, 每个数据集的样本量都会较少, 从而影响模型的训练和评估. 同时, 数据集的划分本身也是随机的, 这也可能导致结果的不准确.

为了减少数据浪费和数据拆分的随机性, 这时可以考虑使用 **K 折交叉检验 (K-Fold Cross Validation)** 的方法来更有效地利用数据.

定义 4.2 K 折交叉检验 **K 折交叉检验**将数据集平均划分为 K 个子集, 每次用其中的 $K-1$ 个子集作为训练集, 剩下的一个子集作为验证集, 重复 K 次, 每个子集都作为一次验证集. 最后将 K 次的验证误差取平均作为最终的验证误差.

在最极端的情形下, 还可以采用留一法.

定义 4.3 留一法 **留一法 (Leave-One-Out, LOO)** 是 K 折交叉检验的特例, 即 $K=N$, 每次用 $N-1$ 个样本作为训练集, 剩下的一个样本作为验证集, 重复 N 次, 每个样本都作为一次验证集. 最后将 N 次的验证误差取平均作为最终的验证误差.

交叉检验虽然计算代价很高, 但它对数据的使用更充分, 在数据较少的情况下具有不错的效果.

4.1.3 误差的偏差-方差分解

我们在前面讲到模型过于简单和过于复杂分别会导致欠拟合和过拟合, 都会使得模型在测试集上的误差较大. 下面我们通过对**误差的偏差-方差分解**来解释之.

设观察值 t_n 可以表示成下面的形式:

$$t_n = f(\boldsymbol{x}_n) + \varepsilon$$

其中 f 是内禀的真实函数, ε 是均值为 0 的噪声. 基于训练集 $\mathcal{D}_{\text{train}}$ 和验证集 $\mathcal{D}_{\text{val}}$(将两者合并记作 $\mathcal{D}$) 训练并选择得到的机器学习函数为 $\hat{f}(\boldsymbol{x};\boldsymbol{w}|\mathcal{D})$, 于是在测试集上某个样本 $(\boldsymbol{x}_m, t_m)$ 上的误差 (这里采取二乘误差) 为:

$$E_{\text{test}} = \left(f(\boldsymbol{x}_m) + \varepsilon - \hat{f}(\boldsymbol{x}_m;\boldsymbol{w}|\mathcal{D})\right)^2$$

由于数据选择的随机性, 以及一些模型 (比如随机梯度下降) 本身的随机性, 我们可以把 $\hat{f}(\boldsymbol{x}_m;\boldsymbol{w}|\mathcal{D})$ 视作一个随机变量, 其期望记作 $\left\langle \hat{f}(\boldsymbol{x}_m;\boldsymbol{w}|\mathcal{D})\right\rangle$. 将上式稍作整理可得

$$E_{\text{test}} = \left[\left(f(\boldsymbol{x}_m) - \left\langle \hat{f}(\boldsymbol{x}_m;\boldsymbol{w}|\mathcal{D})\right\rangle\right) - \left(\hat{f}(\boldsymbol{x}_m;\boldsymbol{w}|\mathcal{D}) - \left\langle \hat{f}(\boldsymbol{x}_m;\boldsymbol{w}|\mathcal{D})\right\rangle\right) + \varepsilon\right]^2$$

上述三项中, 第一项是无随机性的定值, 于是 E_{test} 的期望为:

$$\begin{aligned}\left\langle E_{\text{test}}\right\rangle = &\left[f(\boldsymbol{x}_m) - \left\langle \hat{f}(\boldsymbol{x}_m;\boldsymbol{w}|\mathcal{D})\right\rangle\right]^2 + \left\langle\left[\hat{f}(\boldsymbol{x}_m;\boldsymbol{w}|\mathcal{D}) - \left\langle \hat{f}(\boldsymbol{x}_m;\boldsymbol{w}|\mathcal{D})\right\rangle\right]^2\right\rangle \\ &- 2\left\langle \varepsilon\left(\hat{f}(\boldsymbol{x}_m;\boldsymbol{w}|\mathcal{D}) - \left\langle \hat{f}(\boldsymbol{x}_m;\boldsymbol{w}|\mathcal{D})\right\rangle\right)\right\rangle + \left\langle \varepsilon^2\right\rangle\end{aligned}$$

由于交叉项 $2\left\langle \varepsilon \left(\hat{f}\left(\boldsymbol{x}_m;\boldsymbol{w}|\mathcal{D}\right) - \left\langle \hat{f}\left(\boldsymbol{x}_m;\boldsymbol{w}|\mathcal{D}\right)\right\rangle \right)\right\rangle$ 中的两个随机变量 ε 和 $\hat{f}\left(\boldsymbol{x}_m;\boldsymbol{w}|\mathcal{D}\right) - \left\langle \hat{f}\left(\boldsymbol{x}_m;\boldsymbol{w}|\mathcal{D}\right)\right\rangle$ 分别来源于训练集和测试集, 因此它们是独立的, 所以该项为零. 于是

$$\left\langle E_{\text{test}}\right\rangle = \underbrace{\left[f\left(\boldsymbol{x}_m\right) - \left\langle \hat{f}\left(\boldsymbol{x}_m;\boldsymbol{w}|\mathcal{D}\right)\right\rangle\right]^2}_{\text{偏差}^2} + \underbrace{\left\langle \left[\hat{f}\left(\boldsymbol{x}_m;\boldsymbol{w}|\mathcal{D}\right) - \left\langle \hat{f}\left(\boldsymbol{x}_m;\boldsymbol{w}|\mathcal{D}\right)\right\rangle\right]^2\right\rangle}_{\text{方差}} + \underbrace{\left\langle \varepsilon^2\right\rangle}_{\text{噪声}}$$

上式即为误差的偏差-方差分解, 亦可以简记为

$$\left\langle E_{\text{test}}\right\rangle = \left\langle \varepsilon^2\right\rangle + B^2 + V$$

其中, **偏差 (Bias)** 衡量了模型预测值的期望与真实值之间的差距, 反映了模型的拟合能力; **方差 (Variance)** 衡量了模型预测值的波动性, 反映了模型对训练数据的敏感程度; **噪声 (Noise)** 是数据本身固有的不可避免的误差.

如果选取过于简单的模型, 那么 $\hat{f}(\boldsymbol{x};\boldsymbol{w})$ 不太受到训练集的影响, 其方差较小, 但由于模型能力有限, 其偏差较大, 从而导致较大的误差. 这就是**欠拟合**.

如果选取过于复杂的模型, 那么 $\hat{f}(\boldsymbol{x};\boldsymbol{w})$ 会过度拟合训练集, 即使由于模型能力强, 其偏差较小, 也会因为对训练数据的敏感性较高而导致较大的方差, 从而导致较大的误差. 这就是**过拟合**.

4.1.4 学习曲线

为了更直观地理解偏差-方差分解, 我们可以通过绘制**学习曲线 (Learning Curve)** 来观察模型在训练集和验证集上的误差随训练样本数量的变化情况.

一般来说, 随着训练样本数量的增加, 模型在训练集上的误差会逐渐增大, 而在验证集上的误差会逐渐减小, 最终两者趋于一致.

对于欠拟合的情况, 两者很早就达到饱和并且相差较小. 这表明模型能力有限, 无法很好地拟合数据, 也就对应高偏差的情况.

对于过拟合的情况, 训练集上的误差很低, 而验证集上的误差较高, 并且两者相差较大. 这表明模型过度拟合了训练数据, 也就对应高方差的情况.

4.2 数据的预处理

4.2.1 数据标准化

数据的输入量 $\boldsymbol{x}$ 的各个分量可能具有不同的变化范围, 甚至可能具有不同的量纲. 例如, 在预测房价时, 房间楼层是一个整数, 但房屋面积是一个带面积单位的实数, 甚至其取值也会随着单位的变化而变化. 这样的情况会导致某些分量对模型的影响过大, 从而影响模型的训练效果.

为了避免这种情况, 我们通常会对数据进行标准化处理, 使得各个分量具有相似的变化范围和量纲. 常见的标准化方法如下.

定义 4.4 最小-最大缩放 最小-最大缩放 (Min-Max Scaling) 将数据对应到 $[0,1]$ 内, 具体做法是对每个分量 x_i 进行如下变换:

$$x_i \leftarrow \frac{x_i - \min(x_i)}{\max(x_i) - \min(x_i)}$$

定义 4.5 Z-Score 标准化 Z-Score 标准化 (Z-Score Normalization) 将数据变换为均值为 0, 标准差为 1 的分布, 具体做法是对每个分量 x_i 进行如下变换:

$$x_i \leftarrow \frac{x_i - \mu_i}{\sigma_i}$$

其中 μ_i 和 σ_i 分别是 $\{x_i\}$ 的均值和标准差.

Z-Score 标准化对于大多数机器学习算法来说是较好的选择, 因为它能够更好地处理异常值和不同分布的数据.

概率模型一般不需要进行数据标准化, 因为它主要关心变量的分布和变量之间的条件概率, 决策树方法就属于此类. 除此之外, 线性回归, 逻辑回归, 神经网络, 支持向量机等模型一般都要进行数据标准化.

4.2.2 异常值检测

4.3 数据不平衡及分类模型的评价指标

在分类问题中, 有时某些类别的样本数量远少于其他类别, 这就导致了**数据不平衡 (Data Imbalance)** 的问题. 例如, 在样品合格检测中, 不合格的样本数量通常远少于合格的样本数量. 如果直接使用准确率作为评价指标, 可能会导致模型偏向于将所有样本都预测为合格, 从而不能达到预期.

在分类问题中, 可以用**混淆矩阵 (Confusion Matrix)** 来表示模型的预测结果.

定义 4.6 混淆矩阵 对于一个 n 个类别的分类问题, 模型的**混淆矩阵**是一个 $n \times n$ 的矩阵 C, 其中 C_{ij} 表示真实类别为 i 并且被模型预测为 j 的样本数量.

对于常见的二分类问题, 混淆矩阵可以表示为:

$$\begin{bmatrix} TP & FN \\ FP & TN \end{bmatrix}$$

其中 TP 表示真阳性 (True Positive), 即真实类别为阳性且被模型预测为阳性的样本数量;TN 表示真阴性 (True Negative), 即真实类别为阴性且被模型预测为阴性的样本数量;FP 表示假阳性 (False Positive), 即真实类别为阴性但被模型预测为阳性的样本数量;FN 表示假阴性 (False Negative), 即真实类别为阳性但被模型预测为阴性的样本数量.

基于混淆矩阵, 我们可以定义一些常用的评价指标.

定义 4.7 基于混淆矩阵的评价指标

1. **准确率 (Accuracy)**: 表示模型预测正确的样本数量占总样本数量的比例, 定义为:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

准确率表征了一个模型的整体性能, 但这里的性能没有偏向于某一种类别 (更多时候, 它被占多数的类别所主导), 因此在数据不平衡的情况下, 准确率可能会误导我们对模型性能的判断.

2. **精度 (Precision)**(又称为**查准率**): 表示被模型预测为阳性的样本中真实为阳性的比例, 定义为:

$$\text{Precision} = \frac{TP}{TP + FP}, \quad \text{Precision} = \frac{TN}{TN + FN}$$

精度反映了模型预测一个点属于某一类时, 该点真正属于此类的概率. 对某一类别的精度越高, 说明模型在确定某一点数以该类时越可靠.

在推荐系统中, 精度是一个重要的指标. 模型主要关心推荐的是否是用户真正感兴趣的内容, 而不太关心没有推荐的内容中是否有用户感兴趣的.

3. **召回率 (Recall)**(又称为**查全率**): 表示真实为某一类的样本中被模型预测为该类的比例, 定义为:

$$\text{Recall} = \frac{TP}{TP + FN}, \quad \text{Recall} = \frac{TN}{TN + FP}$$

召回率反映了模型对某一类的覆盖能力. 对某一类别的召回率越高, 说明模型能够识别出更多属于该类的样本.

在疾病检测中, 召回率是一个重要的指标. 由于患病而漏诊的代价十分严重, 因此模型主要关心尽可能多地识别出患病的患者, 即使会误诊一些健康的患者.

4. **F_1 分数 (F_1 Score)**: 是精度和召回率的调和平均数, 定义为:

$$F_1 = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

F_1 分数综合考虑了精度和召回率, 在数据不平衡的情况下, 比单独使用精度或召回率更能反映模型的性能.

5 神经网络

5.1 神经网络概述

5.1.1 适应数据的可变基

前面章节中的回归方法在数学上可以描述为

$$y(\boldsymbol{x}) = f\left(\sum_{j=1}^{M} w_j \phi_j(\boldsymbol{x})\right)$$

基函数 $\phi_j(x)$ 的选择是固定的, 当数据维度较高时会导致基函数的数目过于庞大, 导致维度灾难.

为了解决这一问题, 可以引入适应基的概念, 即固定基函数的数目, 但允许基函数本身随训练数据进行调整.

定义 5.1 适应基 形如 $\phi_j(\boldsymbol{x}, \boldsymbol{p}_j)$ 的基函数被称作**适应基 (Adaptive Basis Functions)**, 它不仅依赖于输入变量 $\boldsymbol{x}$, 还依赖于一组可调参数 $\boldsymbol{p}_j$. 通过调整这些参数, 可以使基函数更好地适应训练数据.

5.1.2 神经网络的数学模型

神经网络可以被描述为一个 $N+1$ 层的网络, 每层包含若干个节点 (神经元). $n=1$ 的层表示输入层, 节点数目即输入 $\boldsymbol{x}$ 的分量数目 (有时会额外增加一个为 1 的分量表示截距项), $n=N+1$ 的层表示输出层, 节点数目即输出 $\boldsymbol{y}$ 的分量数目; 中间的层称为隐藏层. 每个节点包含刺激值 $z_i^{(k)}$ 和响应值 $a_i^{(k)}$.

神经网络的迭代方式如下: 对于第 $k+1$ 层的每个节点 j, 它的刺激值 $z_j^{(k+1)}$ 由上一层的所有节点的响应值 $a_i^{(k)}$ 加权得到, 然后通过一个激活函数 $h(\cdot)$ 得到响应值 $a_j^{(k+1)}$, 即

$$z_j^{(k+1)} = \sum_i w_{ij}^{(k)} a_i^{(k)}, \quad a_j^{(k+1)} = h\left(z_j^{(k+1)}\right)$$

其中 $w_{ij}^{(k)}$ 表示第 k 层的节点 i 到第 $k+1$ 层的节点 j 的连接权重.

神经网络的功能依赖于激活函数 $h(\cdot)$ 的选择, 常用的激活函数包括 Sigmoid 函数, ReLU 函数和 tanh 函数等. 通过一定的办法求出连接权重 $w_{ij}^{(k)}$ 后, 神经网络就可以对输入数据进行映射, 完成各种任务.

可以看出, 神经网络的每一层都相当于一次逻辑回归, 因此神经网络可以看作是多层串联的逻辑回归模型. 大多数神经网络都具有多个隐藏层. 深度学习主要指的就是较多隐藏层的神经网络, 称深度神经网络.

5.1.3 万能近似定理

神经网络具有很强的拟合能力. 事实上, 数学上已经证明下面的定理:

定理 5.2 万能近似定理 设 $h: \mathbb{R} \to \mathbb{R}$ 是一个非恒等的、有界的、单调递增的连续函数. 令 $C(\mathbb{R}^n)$ 表示从 $\mathbb{R}^n$ 到 $\mathbb{R}$ 的所有连续函数构成的空间. 对于任意 $f \in C(\mathbb{R}^n)$ 和 $\varepsilon > 0$, 存在一个整数 M, 常数 $v_i, b_i \in \mathbb{R}$ 和向

量 $\boldsymbol{w}_i \in \mathbb{R}^n$ 使得

$$\left| f(\boldsymbol{x}) - \sum_{i=1}^{M} v_i h\left(\boldsymbol{w}_i^T \boldsymbol{x} + b_i\right) \right| < \varepsilon, \quad \forall \boldsymbol{x} \in \mathbb{R}^n$$

翻译成神经网络的语言即: 一个具有单隐藏层的神经网络, 只要隐藏层节点数目足够多, 就可以以任意精度逼近任意连续函数.

5.2 神经网络的训练

5.2.1 误差函数

与线性回归类似, 神经网络的训练目标也是最小化误差函数. 对于回归问题, 常用的误差函数也是均方误差函数:

$$E(\boldsymbol{w}) = \frac{1}{2} \sum_{n=1}^{N} \|\boldsymbol{y}(\boldsymbol{x}_n, \boldsymbol{w}) - \boldsymbol{t}_n\|^2$$

对于分类问题, 常用的误差函数是交叉熵误差函数:

$$E(\boldsymbol{w}) = -\sum_{n=1}^{N} \sum_{k=1}^{K} t_{nk} \ln y_k(\boldsymbol{x}_n, \boldsymbol{w})$$

5.2.2 神经网络的梯度: 误差反向传播算法

由前面递推定义的神经网络模型是一个嵌套函数, 因此误差函数关于连接权重 $w_{ij}^{(k)}$ 的梯度可以通过链式法则进行计算. 定义辅助变量

$$\delta_i^{(k)} = \frac{\partial E}{\partial z_i^{(k)}}$$

则

$$\delta_i^{(k)} = \sum_j \frac{\partial E}{\partial z_j^{(k+1)}} \frac{\partial z_j^{(k+1)}}{\partial a_i^{(k)}} \frac{\partial a_i^{(k)}}{\partial z_i^{(k)}} = h'\left(z_i^{(k)}\right) \sum_j \delta_j^{(k)} w_{ij}^{(k)}$$

$$\frac{\partial E}{\partial w_{ij}^{(k)}} = \frac{\partial E}{\partial z_j^{(k+1)}} \frac{\partial z_j^{(k+1)}}{\partial w_{ij}^{(k)}} = \delta_j^{(k+1)} a_i^{(k)}$$

对于回归问题和分类问题, 输出层 (假定其为 $N+1$ 层) 的 δ 值可由 E 的形式统一得出简单的表达式:

$$\delta_i^{(N+1)} = y_i - t_i = a_j^{(N+1)} - t_n$$

这样, 每次计算先正向逐层迭代计算激活信号 $a_i^{(k)}$, 然后反向逐层迭代计算 $\delta_i^{(k)}$, 最后计算梯度 $\dfrac{\partial E}{\partial w_{ij}^{(k)}}$. 这一过程称为**误差反向传播 (Backpropagation)** 算法.

6 核方法: 近邻法与支持向量机

6.1 密度估计的非参数法

对于随机变量 $\boldsymbol{x}$, 如果知道其概率密度函数 $p(\boldsymbol{x})$, 就能够计算出其在某一区域 R 内取值的概率. 现在, 我们需要根据 $\boldsymbol{x}$ 的一组采样点 $\mathcal{D}:\{\boldsymbol{x}_n\}_{n=1}^N$ 估计概率密度函数 $p(\boldsymbol{x})$. 这就是**密度估计 (density estimation)** 问题.

定义 6.1 密度估计 给定随机变量 $\boldsymbol{x}$ 的一组采样点 $\mathcal{D}:\{\boldsymbol{x}_n\}_{n=1}^N$, 密度估计是根据 $\mathcal{D}$ 估计随机变量 $\boldsymbol{x}$ 的概率密度函数 $p(\boldsymbol{x})$ 的过程.

密度估计有两类办法:

(1) 参数法: 假定 $p(\boldsymbol{x})$ 具有已知的带参数形式, 那么问题转化为应用最大似然法确定参数的值.

(2) 非参数法: 不对 $p(\boldsymbol{x})$ 作任何假设, 而是直接根据样本 $\mathcal{D}$ 估计 $p(\boldsymbol{x})$.

本节介绍的直方图方法, 核密度估计法和近邻法都属于非参数法.

6.1.1 直方图方法

直方图是我们熟知的表示随机变量分布的方法. 将数据空间划分为若干个小区域, 称为**区间 (bins)** 或**箱 (buckets)**, 然后统计每个区间内的样本点数目, 最后通过归一化得到概率密度函数的估计. 具体地, 设数据空间被划分为 M 个区间 $\{\mathcal{R}_j\}_{j=1}^M$, 每个区间的体积为 V, 则在区间 $\mathcal{R}_j$ 内的概率密度函数估计为

$$p_{\mathcal{R}_j}(\boldsymbol{x})=\frac{1}{NV}\sum_{n=1}^{N}\mathbb{I}(x_n\in\mathcal{R}_j)=\frac{n_j}{NV}$$

其中 n_j 即 $\mathcal{R}_j$ 中的样本点的数目.

直方图的数学原理可以推导如下.

证明. 根据概率密度函数的定义, 随机变量 $\boldsymbol{x}$ 落在某一区域 $\mathcal{R}$ 内的概率为

$$P=\int_{\mathcal{R}}p(\boldsymbol{x})\mathrm{d}\boldsymbol{x}$$

对 $\boldsymbol{x}$ 随机采样 N 次, 有 K 个点落入 $\mathcal{R}$ 的概率服从二项分布:

$$p\left(K|N,P\right)=C_N^KP^K(1-P)^{N-K}$$

如果 N 和 K 都很大, 那么上述二项分布是一个窄峰, 我们就可以近似地认为 $P\approx\dfrac{K}{N}$. 另一方面, 如果区域 $\mathcal{R}$ 足够小, 那么可以认为在该区域内 $p(\boldsymbol{x})$ 近似为常数, 即 $P=p(\boldsymbol{x})V$. 结合上述两式即可得

$$p(\boldsymbol{x})=\frac{K}{NV}$$

这就说明了直方图方法的合理性. □

直方图方法的优点是简单直观, 但缺点也很明显: 结果曲线不光滑, 并且高维空间下将因维度灾难而效果变差.

6.1.2 核密度估计法

直方图对 $p(\boldsymbol{x}) = \dfrac{K}{NV}$ 的处理方法是固定区域 $\mathcal{R}$ 的体积 V, 然后从已知数据中估计 K. 类似地, 为了计算某一 $\boldsymbol{x}$ 处的概率密度 $p(\boldsymbol{x})$, 我们可以将 $\mathcal{R}$ 选择为中心位于 $\boldsymbol{x}$, 边长为 h 的小立方体, 然后定义如下核函数

$$k\left(\frac{\boldsymbol{x}-\boldsymbol{x}_n}{h}\right) = \begin{cases} 1, & \text{if } \forall 1 \leqslant i \leqslant D, \left|\dfrac{x_i - x_{n,i}}{h}\right| < \dfrac{1}{2} \\ 0, & \text{otherwise} \end{cases}$$

其中 x_i 是 $\boldsymbol{x}$ 的第 i 个分量. 则

$$K = \sum_n k\left(\frac{\boldsymbol{x}-\boldsymbol{x}_n}{h}\right)$$

将其代入概率密度函数的估计公式就有

$$p(\boldsymbol{x}) = \frac{1}{N}\sum_n \frac{1}{h^D} k\left(\frac{\boldsymbol{x}-\boldsymbol{x}_n}{h}\right)$$

这样的方法被称作**核密度估计法 (Kernal Density Estimation)**.

我们既可以将上式理解为有多少数据点 $\boldsymbol{x}_n$ 落到以 $\boldsymbol{x}$ 为中心的立方体内, 也可以理解为 $\boldsymbol{x}$ 落到多少个以 $\boldsymbol{x}_n$ 为中心的小立方体里.

与直方图类似, 采用上述核函数给出的 $p(\boldsymbol{x})$ 是不光滑的. 我们可以用光滑的核函数 $k(\boldsymbol{x}, \boldsymbol{x}_n)$, 例如高斯核函数, 来解决这一问题:

$$k(\boldsymbol{x}, \boldsymbol{x}_n) = \frac{1}{(2\pi h^2)^{D/2}} \exp\left(-\frac{|\boldsymbol{x}-\boldsymbol{x}_n|^2}{2h^2}\right)$$

此时估计的 $p(\boldsymbol{x})$ 可以写为

$$p(\boldsymbol{x}) = \frac{1}{N}\sum_n k(\boldsymbol{x}, \boldsymbol{x}_n)$$

6.1.3 近邻法

在数据分布不均匀时, 固定体积 V 的做法可能导致性能下降. 因此, 我们可以采用固定 K 而改变 V 的方法. 这就需要用到近邻法.

尽管近邻法可以用于密度估计, 这时属于无监督学习, 但更多的时候近邻法被用于分类的监督学习, 即 **K-近邻算法 (K-Nearest Neighbor, KNN)**.

K-近邻算法的原理基于 Bayes 公式. 记需要预测类别的点为 $\boldsymbol{x}$, 对于 $\boldsymbol{x}$ 的 K 个近邻数据点 (来自训练集, 因此类别已知), 属于第 k 类的数目记为 K_k, 则第 k 类在 $\boldsymbol{x}$ 处的分布的概率密度为

$$p(\boldsymbol{x}|\mathcal{C}_k) = \frac{K_k}{N_k V}$$

其中 N_k 是训练集中属于第 k 类的数据总数. 属于第 k 类的先验概率为

$$p(\mathcal{C}_k) = \frac{N_k}{N}$$

利用 Bayes 公式得到后验概率

$$p(\mathcal{C}_k|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|\mathcal{C}_k)p(\mathcal{C}_k)}{p(\boldsymbol{x})} = \frac{\frac{K_k}{N_k V}\cdot\frac{N_k}{N}}{\frac{K}{NV}} = \frac{K_k}{K}$$

对于此式的直观理解即在 $\boldsymbol{x}$ 的 K 个临近的样本中, 如果有 K_k 个属于第 k 类, 那么 $\boldsymbol{x}$ 属于第 k 类的概率自然为 $\frac{K_k}{K}$.

K-近邻算法只有一个超参数 K. K 越大, 偏差越大, 方差越小. 它还可以用于回归, 简单的做法是将 $\boldsymbol{x}$ 的 K 个近邻的某一属性的平均值 (或者使用核函数进行加权) 作为对 $\boldsymbol{x}$ 的属性的预测.

6.1.4 非参数法的优缺点

非参数法的最大优点是它不需要假设分布函数的形式, 而是通过数据推断分布函数并进行预测, 保证了估计的无偏性和一致性. 然而, 非参数法也需要大量的数据才能保证良好的效果.

6.2 核方法的主要思想

6.3 支持向量机

支持向量机通过核方法进行非线性分类. 它的主要想法是: 在所有可能的分类超平面中, 选择一个使得分类间隔最大的超平面作为最终的分类超平面.

6.3.1 支持向量机的数学原理

我们用更严谨的语言描述上述问题. 考虑线性分类模型

$$y(\boldsymbol{x}) = \boldsymbol{w}\cdot\boldsymbol{x} + b$$

或使用基函数的模型

$$y(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{t}}\boldsymbol{\phi}(\boldsymbol{x}) + b$$

决策面为 $y(\boldsymbol{x}) = 0$. 对于前一种情况, 可以看出 $\boldsymbol{w}$ 是垂直于决策面的向量, 单位化后即为 $\hat{\boldsymbol{w}} = \frac{\boldsymbol{w}}{||\boldsymbol{w}||}$. 任意一点 $\boldsymbol{x}$ 在 $\hat{\boldsymbol{w}}$ 上的投影长度为 $\hat{\boldsymbol{w}}\cdot\boldsymbol{x}$. 于是 $\boldsymbol{x}$ 与决策面的距离为

$$d(\boldsymbol{x}) = \hat{\boldsymbol{w}}\cdot\boldsymbol{x} - d_0$$

其中 d_0 是决策面上一点 $\boldsymbol{x}$ 对应的 $d(\boldsymbol{x})$ 值, 也即原点到决策面的距离. 于是有

$$d(\boldsymbol{x}) = \frac{\boldsymbol{w}\cdot\boldsymbol{x}}{||\boldsymbol{w}||} + \frac{b}{||\boldsymbol{w}||} = \frac{y(\boldsymbol{x})}{||w||}$$

记对象的类别标签为 $t_n \in \{-1, 1\}$, 则对于训练样本 $\{\boldsymbol{x}_n, t_n\}_{n=1}^N$, 我们希望所有样本点都被正确分类, 即满足

$$\frac{t_n y(\boldsymbol{x})}{||\boldsymbol{w}||} > 0, \quad n = 1, 2, \ldots, N$$

使用基函数也是类似. 对于线性可分体系, 训练集中的一类数据全部在决策面的一边, 而另一类数据全部在决策面的另一边. 因此对于训练集中任一数据 $\boldsymbol{x}_n$, 其与决策面的距离可以写成

$$\frac{t_n y(\boldsymbol{x})}{||\boldsymbol{w}||} = \frac{t_n\left[\boldsymbol{w}^{\mathrm{t}}\boldsymbol{\phi}(\boldsymbol{x}_n)+b\right]}{||\boldsymbol{w}||}$$

训练集中所有数据点离决策面的最小距离由下式给出:

$$\frac{1}{||\boldsymbol{w}||}\min_n\{t_n[\boldsymbol{w}^{\mathrm{t}}\boldsymbol{\phi}(\boldsymbol{x}_n)+b]\}$$

支持向量机的目标是使得这一间隔最大化, 即

$$\arg\max_{\boldsymbol{w},b}\left\{\frac{1}{||\boldsymbol{w}||}\min_n\{t_n[\boldsymbol{w}^{\mathrm{t}}\boldsymbol{\phi}(\boldsymbol{x}_n)+b]\}\right\}$$

上式既要求最小值又要求最大值, 比较复杂, 因此要进一步化简. 注意到当 $\boldsymbol{w}$ 和 b 成比例改变时, 距离公式 $\frac{t_n\left[\boldsymbol{w}^{\mathrm{t}}\boldsymbol{\phi}(\boldsymbol{x}_n)+b\right]}{||\boldsymbol{w}||}$ 不会改变, 因此不失一般性地总是可以选取 $\boldsymbol{w}, b$ 使得距离决策面最近的数据点 n 满足 $t_n[\boldsymbol{w}^{\mathrm{t}}\boldsymbol{\phi}(\boldsymbol{x}_n)+b]=1$, 于是对于任何数据点都有

$$t_n[\boldsymbol{w}^{\mathrm{t}}\boldsymbol{\phi}(\boldsymbol{x}_n)+b]\geqslant 1$$

这被称作决策平面的**规范表示 (Canonical Representation)**. 此时目标简化为

$$\arg\max_{\boldsymbol{w},b}\left\{\frac{1}{||\boldsymbol{w}||}\right\}$$

或者写成更方便的形式:

$$\arg\min_{\boldsymbol{w},b}\left\{\frac{1}{2}||\boldsymbol{w}||^2\right\}$$

这样, 支持向量机的目标就是在 $t_n[\boldsymbol{w}^{\mathrm{t}}\boldsymbol{\phi}(\boldsymbol{x}_n)+b]\geqslant 1$ 的约束下求解上述最小化问题. 这里约束条件是线性的, 目标函数是二次函数, 因此在数学上属于凸优化问题, 具有良好的性质.

不等式约束条件下的函数极值求解可以利用拉格朗日方法和 KKT 条件. 为每个训练集数据点 $\boldsymbol{x}_n$ 的约束条件引入拉格朗日因子 λ_n, 定义如下函数

$$L(\boldsymbol{w},b,\lambda)=\frac{1}{2}||\boldsymbol{w}||^2-\sum_n\lambda_n\left\{t_n[\boldsymbol{w}^{\mathrm{t}}\boldsymbol{\phi}(\boldsymbol{x}_n)+b]-1\right\}$$

则上述极值问题的 KKT 条件给出

$$\frac{\partial}{\partial(\boldsymbol{w},b)}L(\boldsymbol{w},b,\lambda)=0,\quad \lambda_n\left\{t_n[\boldsymbol{w}^{\mathrm{t}}\boldsymbol{\phi}(\boldsymbol{x}_n)+b]-1\right\}=0$$

对 $\boldsymbol{w}$ 的偏导数得出

$$\boldsymbol{w}=\sum_n\lambda_n t_n\boldsymbol{\phi}(\boldsymbol{x}_n)$$

将它代回 $y(\boldsymbol{x})=\boldsymbol{w}^{\mathrm{t}}\boldsymbol{\phi}(\boldsymbol{x})+b$ 得到

$$y(vecx)=\sum_{n,i}\lambda_n t_n\phi_i(\boldsymbol{x}_n)\phi_i(\boldsymbol{x})+b=\sum_n\lambda_n t_n k(\boldsymbol{x},\boldsymbol{x}_n)+b$$

于是模型的结果与核函数 $k(\boldsymbol{x}, \boldsymbol{x}_n) = \sum_i \phi_i(\boldsymbol{x}_n)\phi_i(\boldsymbol{x}) = \boldsymbol{\phi}^{\mathrm{t}}(\boldsymbol{x}_n)\boldsymbol{\phi}(\boldsymbol{x})$ 有关. 可以看到, 不处于间隔边缘上的点总有 $\lambda_n = 0$, 也即只有间隔边缘的数据点 (被称作**支持向量**) 对 $y(\boldsymbol{x})$ 的计算有贡献.

现在计算 b. 间隔边缘 $\mathcal{S}$ 上的数据点 $\boldsymbol{x}_m$ 满足 $t_m y(\boldsymbol{x}_m) = 1$, 根据前面得出的 $y(\boldsymbol{x})$ 的表达式可得

$$t_m \left[\sum_{n \in \mathcal{S}} \lambda_n t_n k(\boldsymbol{x}_m, \boldsymbol{x}_n) + b \right] = 1$$

于是

$$b = \frac{1}{N_{\mathcal{S}}} \sum_{m \in \mathcal{S}} \left[t_m - \sum_{n \in \mathcal{S}} \lambda_n t_n k(\boldsymbol{x}_m, \boldsymbol{x}_n) \right]$$

其中 $N_{\mathcal{S}}$ 表示落在间隔边缘上的点的数目.

6.3.2 软间隔的使用

在线性不可分体系中, 严格的边界条件就不再适用了. 为此, 我们可以引入**松弛变量 (Slack Variables)**, 记作 $\xi_n \geqslant 0$, 使得约束条件放宽为

$$t_n y(\boldsymbol{x}_n) \geqslant 1 - \xi_n$$

ξ_n 可以看作是数据点越过间隔边界的距离, 即它造成了某种误差, 需要给予相应的惩罚. 由此, 模型的误差函数可以改写成

$$E(\boldsymbol{w}, b, \xi_n; C) = C \sum_n \xi_n + \frac{1}{2}||\boldsymbol{w}||^2$$

其中 C 是超参数, 用于调整对于越界的惩罚力度. 同样可以用拉格朗日方法和 KKT 条件求解上述问题.

7 概率图模型

7.1 有向图: 贝叶斯网络

定义 7.1 贝叶斯网络 贝叶斯网络 **(Bayesian network)**, 又称信念网络 **(Belief Network)**, 是一种概率图模型, 其网络拓朴结构是一个有向无环图, 边的终点所指的事件依赖于边的起点所指的事件.

7.1.1 贝叶斯网络的三种情形

讨论贝叶斯网络中事件的条件独立性时, 总会遇到以下三种情形:

1. **尾对尾 (tail-to-tail)**: 两根箭头的尾部相连:

$$a \leftarrow c \rightarrow b$$

可以简单地理解为 a, b 有共同的起因 c(尽管并不一定是因果关系). 根据贝叶斯网络的性质, 此时的联合概率为

$$p(a,b,c) = p(a|c)p(b|c)p(c)$$

于是

$$p(a,b) = \sum_c p(a|c)p(b|c)p(c)$$

一般而言, 上式并不等于

$$p(a)p(b) = \left(\sum_c p(a|c)p(c)\right)(p(b|c)p(c))$$

因此 a 与 b 并不独立, 记作 $a \not\perp\!\!\!\perp b \mid \varnothing$. 这里 $\varnothing$ 表示空集, 即没有前提条件.

当固定 (已知)c 时则有

$$p(a,b|c) = \frac{p(a,b,c)}{p(c)} = p(a|c)p(b|c)$$

即 a 与 b 在 c 固定的条件下是独立的:

$$a \perp\!\!\!\perp b \mid c$$

可以举一个简单的例子: 假定 a, b, c 分别是一所小学中学生的鞋子尺码, 阅读能力和年龄. 当 c 不固定时, 将会得出阅读能力和鞋子尺码正相关的结论, 但是当固定年龄 c 时就会发现这两者是独立的.

2. **头对尾 (head-to-tail)**: 两根箭头头尾相连:

$$a \rightarrow c \rightarrow b$$

此时 a, b, c 的联合概率为

$$p(a,b,c) = p(a)p(c|a)p(b|c)$$

如果 c 不固定, 则

$$p(a,b) = p(a)\sum_c p(c|a)p(b|c)$$

一般而言, 上式并不等于

$$p(a)p(b) = p(a)\sum_a \left(p(a)\sum_c p(c|a)p(b|c) \right)$$

因此 $a \not\perp\!\!\!\perp b \mid \varnothing$. 当固定 c 时

$$p(a,b|c) = \frac{p(a,b,c)}{p(c)} = \frac{p(a)p(c|a)p(b|c)}{p(c)} = \frac{p(a,c)p(b|c)}{p(c)} = p(a|c)p(b|c)$$

即 $a \perp\!\!\!\perp b \mid c$. 也即, 固定 c 之后切断了 a 对 b 的影响.

3. **头对头 (head-to-head)**: 两根箭头的头部相连:

$$a \to c \leftarrow b$$

此时 a, b, c 的联合概率为

$$p(a,b,c) = p(a)p(b)p(c|a,b)$$

如果 c 不固定, 则有

$$p(a,b) = p(a)p(b)\sum_c p(c|a,b) = p(a)p(b)$$

即 $a \perp\!\!\!\perp b \mid \varnothing$. 然而, 如果 c 固定, 则有

$$p(a,b|c) = \frac{p(a,b,c)}{p(c)} = \frac{p(a)p(b)p(c|a,b)}{\sum_{a,b} p(a)p(b)p(c|a,b)}$$

而

$$p(a|c)p(b|c) = \frac{p(a)p(b)\left(\sum_a p(a)p(c|a,b)\right)\left(\sum_b p(b)p(c|a,b)\right)}{\left(\sum_{a,b} p(a)p(b)p(c|a,b)\right)^2}$$

一般情况下两者并不相等, 即 $a \not\perp\!\!\!\perp b \mid c$. 这表明在两个事件共同的结果确定时, 这两者具有一定的相关关系.

可以举一个简单的例子: 假定 a, b, c 分别是文化课成绩好, 体育成绩好和进入某大学. 如果文化课成绩好和体育成绩好都能进入大学, 那么观察大学里的学生就很可能发现文化课成绩好的体育成绩差, 反之亦然. 实际上两者并没有负相关关系, 但是固定 c 之后就导致了表面上的负相关.

7.2 朴素贝叶斯

7.2.1 朴素贝叶斯分类器

贝叶斯分类器是一种用于分类任务的特殊的贝叶斯网络。它假设在类别 $\mathcal{C}$ 固定时, 输入 $\boldsymbol{x}$ 的各个分量 (特征) 的分布之间是独立的. 朴素指的就是这种算法假定各个分量的分布独立. 尽管真实数据可能不满足这个简化假设, 但这一方法对很多复杂的问题还是很有效果的.

在朴素贝叶斯模型中, 类别 $\mathcal{C}$ 是父节点, 输入 $\boldsymbol{x}$ 的各个分量 $x_1, \cdots, x_D$ 是其子节点, 子节点之间没有连线. 于是联合概率分布为

$$p(\boldsymbol{x}|\mathcal{C}) = p(\mathcal{C}) \prod_{i=1}^{D} p_i(x_i|\mathcal{C})$$

于是就把分布处理成多个一元分布, 然后根据数据对每个维度进行训练. 这很好地避免了维度灾难的问题.

朴素贝叶斯的推断可以利用贝叶斯公式, 即

$$p(\mathcal{C}|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|\mathcal{C})p(\mathcal{C})}{p(\boldsymbol{x})} = \frac{p(\mathcal{C})}{p(\boldsymbol{x})} \prod_{i=1}^{D} p_i(x_i|\mathcal{C})$$

7.3 马尔科夫随机场与玻尔兹曼机

7.3.1 马尔科夫随机场

定义 7.2 马尔科夫随机场 **马尔科夫随机场 (Markov random field)**, 又称**马尔科夫网络 (Markov network)**, 也是一类概率图模型, 属于无向图模型 (undirected graphical model), 连接结点的边是没有方向的.

7.3.2 玻尔兹曼机与受限玻尔兹曼机

8 混合模型

8.1 K 均值法

8.1.1 K 均值法的原理

假设我们需要把数据集 $\mathcal{D}:\{\boldsymbol{x}_n\}$ 中的数据点分到 K 个组. 直观地讲, 我们希望决定一个点分到某个组中时, 该点与该组中其他点的距离尽可能近, 而与其他组中点的距离尽可能远. 于是我们需要找到能衡量这一距离的指标.

聚类分析的一种思路是寻找一些**原型点 (Prototype points)**$\{\boldsymbol{\mu}_k\}$ 代表每个组. 计算任一数据点 $\boldsymbol{x}_n$ 与第 k 个组的距离时, 只需计算 $||\boldsymbol{x}_n-\boldsymbol{\mu}_k||$ 即可. K 均值法就采用这样的思路, 用二乘误差衡量模型的误差, 就得到下面的**畸变函数 (Distorition function)**:

$$J(\boldsymbol{R},\{\boldsymbol{\mu}_k\})=\sum_{n=1}^{N}\sum_{k=1}^{K}\boldsymbol{R}_{nk}||\boldsymbol{x}_n-\boldsymbol{\mu}_k||^2$$

其中 $\boldsymbol{R}_{nk}$ 是第 n 个数据点分组结果的**独热编码**, 如果 $\boldsymbol{x}_n$ 被分到第 k 组, 则 $\boldsymbol{R}_{nk}=1$, 否则 $\boldsymbol{R}_{nk}=0$. 于是, 聚类问题就转化为如下最小化问题:

$$\arg\min_{\boldsymbol{R},\boldsymbol{\mu}_k}J(\boldsymbol{R},\{\boldsymbol{\mu}_k\})$$

畸变函数 J 类似于监督学习中的误差函数.

求解上述最小化问题时, 我们可以采用**交替优化 (Alternating optimization)** 的方法, 具体步骤为:

1. 固定 $\{\boldsymbol{\mu}_k\}$, 求 $\boldsymbol{R}$ 使得 $J(\boldsymbol{R},\{\boldsymbol{\mu}_k\})$ 最小化. 不难看出, 只需对于每个数据点 $\boldsymbol{x}_n$ 找到与其距离最近的原型点 $\boldsymbol{\mu}_k$, 并将 $\boldsymbol{x}_n$ 分到对应的组中 (即令 $\boldsymbol{R}_{nk}=1$) 即可.
2. 固定 $\boldsymbol{R}$, 求 $\{\boldsymbol{\mu}_k\}$ 使得 $J(\boldsymbol{R},\{\boldsymbol{\mu}_k\})$ 最小化. 对每个原型点 $\boldsymbol{\mu}_k$, 不难看出当其取聚类中所有点的均值时, 畸变函数 J 取得最小值. 即:
$$\boldsymbol{\mu}_k=\frac{\sum_{n=1}^{N}\boldsymbol{R}_{nk}\boldsymbol{x}_n}{\sum_{n=1}^{N}\boldsymbol{R}_{nk}}$$
这也是 K 均值法名称的由来.
3. 不断重复步骤 1 和 2, 直到畸变函数 J 收敛 (即不再发生变化). 可以证明 J 在此过程中单调下降, 但不一定收敛到全局最小值. 因此, 实际应用中通常需要多次运行 K 均值算法, 每次重新随机初始化, 并选择畸变函数 J 最小的结果作为最终结果.

对于在线学习的情形, 我们可以采用**在线 K 均值算法 (Online K-means algorithm)**. 每次只处理一个数据点 $\boldsymbol{x}_n$, 并根据该点更新对应的原型点 $\boldsymbol{\mu}_k$. 具体地, 对于每个数据点 $\boldsymbol{x}_n$, 我们首先找到与其距离最近的原型点 $\boldsymbol{\mu}_k$, 然后根据如下公式更新该原型点:

$$\boldsymbol{\mu}_k\leftarrow\boldsymbol{\mu}_k+\eta_n(\boldsymbol{x}_n-\boldsymbol{\mu}_k)$$

其中 η_n 是学习率, 通常取 $\eta_n = \dfrac{1}{s_k}$, 其中 s_k 是到目前为止被分到第 k 组的数据点数量. 这种更新方式等价于将 $\boldsymbol{\mu}_k$ 更新为当前所有被分到第 k 组的数据点的均值.

8.1.2 超参数 K 的选取

不难看出 K 均值法只有一个超参数 K. 聚类作为非监督学习不能通过已知的标签辅助选择 K. 除去通过实际问题的情况选择 K 外, 可以使用一些辅助指标选择 K. 一种常用的方法是**肘部法则 (Elbow method)**. 具体地, 我们可以计算不同 K 值下畸变函数 J 的值, 并绘制 J 随 K 变化的曲线. 通常情况下, 随着 K 的增加, 畸变函数 J 会减小, 但减小的幅度会逐渐变小. 当曲线出现明显的“肘部”时, 对应的 K 值就是一个较好的选择.

8.2 高斯混合模型

8.2.1 高斯混合模型的原理

K 均值法的优势在于算法简单易行, 执行高效, 但它的主要缺点是它对于聚类中心平均值的使用太单一, 倾向于认为组内数据的分布呈球形. 这可能在某些数据分布复杂的情况下表现不佳. 为了解决这个问题, 我们可以使用更复杂的模型来描述数据的分布, 例如**高斯混合模型 (Gaussian Mixture Model,GMM)**.

高斯混合模型假设 $\boldsymbol{x}$ 的分布是多个高斯分布的加权和. 具体而言假设有 K 个高斯分布, 每个高斯分布有其均值 $\boldsymbol{\mu}_k$ 和协方差矩阵 $\boldsymbol{\Sigma}_k$, 以及对应的混合系数 π_k, 满足 $\sum\limits_{k=1}^{K} \pi_k = 1$, 则高斯混合模型的概率密度函数为:

$$p(\boldsymbol{x}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

每个高斯分布代表聚类的一个组别. 引入隐藏变量 $\boldsymbol{z}$, 其第 k 个分量 $\boldsymbol{z}_k$ 取 1 时表示 $\boldsymbol{x}$ 属于第 k 个组别 (对于给定的数据点而言就是独热编码). 于是可知

$$p(\boldsymbol{z}_k = 1) = \pi_k, \quad p(\boldsymbol{x}|\boldsymbol{z}_k = 1) = \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

于是根据 Bayes 公式可知:

$$p(\boldsymbol{x}) = p(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})$$

现在我们考虑如何求解高斯混合模型的参数 $\{\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\}$. 如果 $\boldsymbol{z}$ 是已知的, 那么可以很容易地根据多元高斯分布的性质求解 $\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k$ 的值. 然而实际上 $\boldsymbol{z}$ 是未知的 (这正是聚类的结果), 因此我们需要使用**期望最大化 (Expectation-Maximization,EM)** 算法来估计参数.

简单而言, 首先对于每个 $\boldsymbol{x}_n$ 估计 $\boldsymbol{z}_n$ 的值, 然后基于 $\boldsymbol{x}_n, \boldsymbol{z}_n$ 求解模型参数; 然后用求出的模型参数重新推断 $\boldsymbol{z}_n$, 如此迭代直到收敛为止.

1. 初始化参数 $\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k$.

2. E 步骤: 根据猜测的 $\boldsymbol{z}_n$, 记 $\gamma_k(\boldsymbol{z}_n)$ 为数据点 $\boldsymbol{x}_n$ 属于第 k 个聚类的概率, 则属于第 k 个聚类的数据数目为

$$N_k = \sum_{n=1}^{N} \gamma_k(\boldsymbol{z}_n)$$

基于多元高斯分布的性质可得

$$\boldsymbol{\mu}_k = \frac{\sum\limits_{n=1}^{N} \gamma_k(\boldsymbol{z}_n)\boldsymbol{x}_k}{\sum\limits_{n=1}^{N} \gamma_k(\boldsymbol{z}_n)} = \frac{1}{N_k}\sum_{n=1}^{N} \gamma_k(\boldsymbol{z}_n)\boldsymbol{x}_n$$

$$\boldsymbol{\Sigma}_k = \frac{\sum\limits_{n=1}^{N} \gamma_k(\boldsymbol{z}_n)(\boldsymbol{x}_n - \boldsymbol{\mu}_k)(\boldsymbol{x}_n - \boldsymbol{\mu}_k)^{\mathrm{t}}}{\sum\limits_{n=1}^{N} \gamma_k(\boldsymbol{z}_n)} = \frac{1}{N_k}\sum_{n=1}^{N} \gamma_k(\boldsymbol{z}_n)(\boldsymbol{x}_n - \boldsymbol{\mu}_k)(\boldsymbol{x}_n - \boldsymbol{\mu}_k)^{\mathrm{t}}$$

$$\pi_k = \frac{N_k}{N}$$

3. M 步骤: 基于更新后的参数, 重新计算 $\gamma_k(\boldsymbol{z}_n)$. 根据 Bayes 公式可得

$$\gamma_k(\boldsymbol{z}_n) = p(\boldsymbol{z}_k = 1|\boldsymbol{x}_n) = \frac{p(\boldsymbol{x}_n|\boldsymbol{z}_k = 1)p(\boldsymbol{z}_k = 1)}{\sum\limits_{j=1}^{K} \pi_j \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)} = \frac{\pi_k \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum\limits_{j=1}^{K} \pi_j \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}$$

4. 重复 E 步骤和 M 步骤, 直到参数收敛.

9 隐藏连续变量: 降维

9.1 主成分分析

定义 9.1 主成分分析 **主成分分析 (Principal Component Analysis, PCA)** 是一种统计方法, 用于通过线性变换将数据从高维空间映射到低维空间, 以保留数据的主要特征和结构.

一般而言, 数据的分布在空间中是各向异性的, 即在某些方向上数据的变化更显著. 通过 PCA 可以找到点云数据的主要方向 (即主轴), 舍弃变化较小的方向, 从而实现数据降维.

首先, 考虑给定的一组点 $\boldsymbol{x}_1, \cdots, \boldsymbol{x}_N$ 以及中心坐标

$$\bar{\boldsymbol{x}} = \frac{1}{N} \sum_{i=1}^{N} \boldsymbol{x}_i$$

首先, 为了防止平移对 PCA 结果的影响, 我们需要将点云数据进行中心化处理, 即将每个点减去中心坐标:

$$\boldsymbol{x}'_n = \boldsymbol{x}_n - \bar{\boldsymbol{x}}, \quad n = 1, \cdots, N$$

构造矩阵

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}'_1 \\ \vdots \\ \boldsymbol{x}'_N \end{bmatrix}$$

然后计算 $\boldsymbol{X}$ 的协方差矩阵

$$\boldsymbol{\Sigma}_{\boldsymbol{X}} = \boldsymbol{X}\boldsymbol{X}^{\mathrm{t}}$$

二维情况下的协方差矩阵的两个特征向量 $\boldsymbol{v}_1, \boldsymbol{v}_2$ 表示数据的两条轴线. 对应于更大的特征值的特征向量 $\boldsymbol{v}_1$ 表示数据变化更显著的方向, 即主成分方向, 而 $\boldsymbol{v}_2$ 则表示数据变化最小的方向对于更高维的情况也是同理.

对协方差矩阵 $\boldsymbol{\Sigma}_{\boldsymbol{X}}$ 进行特征值分解, 可得:

$$\boldsymbol{\Sigma}_{\boldsymbol{P}} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^{\mathrm{t}}$$

其中矩阵 $\boldsymbol{V}$ 的列向量即为 $\boldsymbol{\Sigma}_{\boldsymbol{X}}$ 的特征向量, 它们也就确定了点云的主轴方向. 矩阵 $\boldsymbol{\Lambda}$ 为对角矩阵, 其对角线上的元素为对应的特征值, 这些特征值表示了数据在各个主轴方向上的方差大小.

通过选择前 k 个最大的特征值所对应的特征向量, 我们可以构造一个降维映射矩阵 $\boldsymbol{W}$:

$$\boldsymbol{W} = [\boldsymbol{v}_1, \boldsymbol{v}_2, \cdots, \boldsymbol{v}_k]$$

然后, 我们可以将原始数据投影到这个低维空间中, 得到降维后的数据表示:

$$\boldsymbol{Y} = \boldsymbol{X}\boldsymbol{W}$$

这就完成了数据的降维.

10 组合模型

在实际使用中, 将多个模型结合起来往往能进一步提高总体性能, 这种方法被称作**组合模型 (Combination Models)** 或**集成学习 (Ensemble Learning)**. 在组合模型中, 我们先训练多个模型, 称为**弱学习器**, 然后再将它们按照一定的方式结合起来.

在大多数情况下, 我们会使用单一的基础学习算法来训练多个弱学习器, 这样组成的模型被称作**同质的**组合模型. 如果使用不同的基础学习算法来训练弱学习器, 则称为**异质的**组合模型.

对弱学习器的选择应当与组合方法相匹配. 对于低偏差高方差的基础模型, 应当使用能减小方差的组合方法; 对于低方差高偏差的基础模型, 应当使用能减少偏差的组合方式.

10.1 委员会方法与自抽样

定义 10.1 委员会方法 训练多个不同模型, 取其均值作为结果的方法称为**委员会方法 (Committees)**.

委员会方法能有效减少模型的方差. 下面从数学上证明这一点.

证明. 假定我们有 M 个模型 $y_1(\boldsymbol{x}), \cdots, y_M(\boldsymbol{x})$, 则委员会的预测结果 y_{com} 为

$$y_{\text{com}}(\boldsymbol{x}) = \frac{1}{M}\sum_{m=1}^{M} y_m(\boldsymbol{x})$$

设第 m 个模型的预测结果 $y_m(\boldsymbol{x})$ 与真值 $h(\boldsymbol{x})$ 的关系为

$$y_m(\boldsymbol{x}) = h(\boldsymbol{x}) + \varepsilon_m(\boldsymbol{x})$$

于是该模型的均方误差为

$$\mathbb{E}_m = \left\langle [y_m(\boldsymbol{x}) - h(\boldsymbol{x})]^2 \right\rangle = \left\langle \varepsilon_m(\boldsymbol{x})^2 \right\rangle$$

于是委员会的均方误差为

$$\mathbb{E}_{\text{com}} = \left\langle \left[\frac{1}{M}\sum_{m=1}^{M} y_m(\boldsymbol{x}) - h(\boldsymbol{x}) \right]^2 \right\rangle = \left\langle \left[\frac{1}{M}\sum_{m=1}^{M} \varepsilon_m(\boldsymbol{x}) \right]^2 \right\rangle$$

我们假定每个模型都是无偏的, 即 $\langle \varepsilon_m(\boldsymbol{x}) \rangle = 0$, 且各模型之间的误差相互独立, 即 $\langle \varepsilon_m(\boldsymbol{x})\varepsilon_n(\boldsymbol{x}) \rangle = 0 (m \neq n)$. 于是

$$\mathbb{E}_{\text{com}} = \frac{1}{M^2}\sum_{m=1}^{M} \left\langle \varepsilon_m(\boldsymbol{x})^2 \right\rangle = \frac{1}{M^2}\sum_{m=1}^{M} \mathbb{E}_m = \frac{1}{M} \left\langle \mathbb{E}_m \right\rangle$$

也即委员会预测的均方误差只有单个模型平均均方误差的 $1/M$. 当然, 实际情况中个模型的误差并非完全独立, 但只要相关性不强, 委员会方法仍然能显著降低方差. □

委员会方法使用的不同模型通常是用单一基础学习算法在不同的数据集进行训练得到的. 如果简单地把原式数据集划分为 M 份, 可能导致每一份的数据量过少而产生偏差. 这就要用到自抽样法.

定义 10.2 自抽样法 **自抽样法 (Bootstrap Sampling)** 是从原始数据集中有放回地随机抽取样本, 组成新的训练集的方法. 具体而言, 对一个样本数为 N 的数据集, 按有放回随机抽样的方法抽取 N 次形成一个新的样本数为 N 的数据集 $\mathcal{D}_m$, 重复 M 次用作训练每个模型的数据集.

自抽样法在统计学上属于非参数的蒙特卡洛方法. 当自抽样法和委员会方法结合使用时, 被称为**袋装法 (Bagging)** 或**自举汇聚法 (Bootstrap Aggregating)**.

10.2 提升法和自适应增强法

前述的委员会方法使用的是并行方式组合模型, 而下面介绍的提升法使用的是串行方式组合模型.

定义 10.3 提升法 **提升法 (Boosting)** 是通过串行地训练多个弱学习器, 并将它们结合成一个强学习器的方法.

提升法的朴素思想类似于错题本, 即每次训练完成之后将预测错误的样本加大权重, 使得下一个弱学习器更关注这些难以预测的样本. 这样经过多次训练后, 最终的强学习器能更好地处理各种样本. 下面以**自适应增强法 (AdaBoost)** 为例介绍提升法的具体过程, 这是一种用于二分类问题的提升算法.

1. 初始化数据集的样本权重:

$$w_n^{(1)} = \frac{1}{N}, \quad n = 1, 2, \cdots, N$$

2. 依次对模型 $m = 1, 2, \cdots, M$ 执行以下步骤:

(a) 采用如下误差函数对模型 m 进行训练:

$$J_m = \sum_{n=1}^{N} w_n^{(m)} I(y_m(\boldsymbol{x}_n) \neq t_n)$$

其中 $I(y_m(\boldsymbol{x}_n) \neq t_n)$ 在预测预测错误时返回 1, 否则返回 0.

(b) 计算模型 m 的加权错误率:

$$\varepsilon_m = \frac{J_m}{\sum_{n=1}^{N} w_n^{(m)}}, \quad \alpha_m = \ln \frac{1 - \varepsilon_m}{\varepsilon_m}$$

这里 ε_m 即为模型 m 预测错误的比例, $\varepsilon_m = 0.5$ 对应于随机猜测. 如果 $\varepsilon_m \geqslant 0.5$, 只需把模型 m 的预测结果取反即可.

(c) 更新样本权重:

$$w_n^{(m+1)} = w_n^{(m)} \exp[\alpha_m I(y_m(\boldsymbol{x}_n) \neq t_n)]$$

即预测正确的样本权重不变, 预测错误的样本权重乘以一个大于 1 的因子 $\exp(\alpha_m)$.

3. 训练完成后, 可以用下式对新数据点进行预测:

$$y_{\text{ada}}(\boldsymbol{x}) = \sum_{m=1}^{M} \alpha_m y_m(\boldsymbol{x})$$

对于二分类问题, 通常取符号函数 $\text{sgn}(y_{\text{ada}}(\boldsymbol{x}))$ 作为最终预测结果.

10.3 决策树和随机森林

10.3.1 决策树

决策树是机器学习中的一种重要模型, 主要用于分类问题, 但也可以用于回归问题. 决策树与人类思考过程有很好的吻合程度, 具有优秀的可解释性.

决策树的主要思想是分治, 即将输入 $\boldsymbol{x}$ 在空间上划分成一些越来越小的区域, 在不同的小区域使用不同的简单模型进行预测.

通常, 决策树的节点通常是基于输入特征的某个阈值进行划分的. 例如, 对于一个二维输入 (x_1, x_2), 可以在 x_1 轴上选择一个阈值 x_1^*, 将数据划分为 $x_1 \leqslant x_1^*$ 和 $x_1 > x_1^*$ 两部分. 然后对每一部分继续选择另一个特征和阈值进行划分, 直到满足某个停止条件 (如达到最大深度或节点中的样本数过少).

决策树的训练过程通常使用贪心算法, 即在每一步选择当前最优的划分方式. 从根节点到叶节点的路径来进行预测.

10.3.2 随机森林

随机森林是基于决策树的一种集成学习方法, 即使用单颗决策树作为弱学习器的 bagging 方法.

11 强化学习与多臂老虎机

11.1 强化学习引言

人类是通过与环境互动来学习的, 而**强化学习 (Reinforcement Learning, RL)**，就是通过计算来实现从互动中学习. 它是一种目标导向的互动学习, *互动*性是它与监督学习的重要区别, 而有无明确的目标则是它与无监督学习的主要区别. 强化学习需要学习的主要内容, 就是怎样做才能使回报最大化. 它有两个显著特征: *试错*与*延迟回报*.

强化学习的主体通常称作**智能体 (Agent)**, 也通常称作代理. 我们将在下一节介绍智能体的数学框架, 即感知 (Sensation, S), 行动 (Action, A) 和目标 (Goal ,G).

11.2 多臂老虎机: 探索与利用的平衡

11.2.1 多臂老虎机问题的定义

有关多臂老虎机的现实背景不再赘述. 这里用数学语言描述多臂老虎机问题:

智能体在每一步都需要从 K 个可能的行动 $\{a_k\}_{k=1}^K$ 中选择一个行动, 当选择行动 a_k 后将得到回报 R_k, R_k 的静态分布 $p(R_k)$ 仅取决于行动 a_k. 智能体应当怎样逐步选择 N 个行动 $A_t(t=1,\cdots,N)$ 使得获得的总回报 $\sum_{t=1}^{N} R_t$ 最大?

定义行动 a 的价值函数为

$$q_*(a) := \mathbb{E}[R_t|A_t = a]$$

如果 $q_*(a)$ 已知, 那么上述问题就很简单, 只需在每步选择使得 $q_*(a)$ 最大的 a 即可. 然而, 智能体实际上不能获知 $q_*(a)$, 只能在第 t 步做选择时根据之前的结果计算 $q_*(a)$ 的估计值 $Q_t(a)$, 并据此选择合适的行动.

如果选择 $Q_t(a)$ 更大的行动, 那么主要是利用当前对行动价值函数的了解进行回报最大化, 但是容易陷入局部最优中; 如果选择其它行动, 那么更有利于精确地估计行动价值函数 $Q_t(a)$, 但是容易损失回报. 这两者经常是矛盾的, 倾向一者就会远离另一者, 这就是多臂老虎机中探索-利用权衡的难题.

针对上述问题, 人们提出了许多算法.

11.2.2 多臂老虎机的若干算法

多臂老虎机的算法大都基于对行动价值函数的估计, 即估计行动 a 的价值 $Q_t(a)$ 并在此基础上采取行动. 行动价值函数的一种简单的估计方法是统计此前所有采取 a 行动后得到回报的均值, 即

$$Q_t(a) = \frac{\sum_{i=1}^{t-1} R_i I(A_i = a)}{\sum_{i=1}^{t-1} I(A_i = a)}$$

其中 I 是示性函数. 当然, 采用贝叶斯方法通过后验概率估计也是可行的.

贪心算法 在智能体采取行动时可以采用**贪心算法 (Greedy Algorithm)**, 即选取使得 $Q_t(a)$ 取最大值的行动, 即

$$A_t = \arg\max_a Q_t(a)$$

为了防止贪心算法陷入局部最优, 可以强制加入探索的成分, 典型的是 **ε-贪心算法**, 即有小概率 ε 随机选择所有可能的行动, 大概率 $1-\varepsilon$ 遵循贪心算法选择回报期望最高的行动.

然而, 在 K 个选择的 ε-贪心算法中, 在足够长的时间后, 尽管 $Q_t(a)$ 已经足够接近 $q_*(a)$, 算法仍然有 $\dfrac{K-1}{K}\varepsilon$ 的概率选择非最优的步骤. 这启示我们也许可以在不同的阶段采取不同的策略.

UCB 算法 多臂老虎机问题的著名算法, **置信区间上界算法 (Upper Confidence Bound Algorithm, UCB)** 在一定程度上解决了上述问题, 即随着行动数目 t 的增大而逐步减小探索的概率.

UCB 算法分析了 $Q_t(a)$ 估计 $q_*(a)$ 的误差. 通常, 对随机变量 x 进行 n 次测量, 结果为 $X_1, \cdots, X_n$, 那么通常把结果写成

$$x = \bar{X} \pm \frac{\sigma}{\sqrt{n}}, \quad \bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i, \sigma = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(X_i - \bar{X})^2}$$

根据这一分析结果, 我们认为 $q_*(a)$ 有很大概率落在以下区间 (即置信区间) 内:

$$\left[Q_t(a) - \frac{\sigma_t(a)}{\sqrt{N_t(a)}}, Q_t(a) + \frac{\sigma_t(a)}{\sqrt{N_t(a)}}\right]$$

其中 $N_t(a) = \sum_{i=1}^{t-1} I(A_i = a)$ 是行动 a 被采纳的次数. UCB 算法采用了一种面对不确定性时的乐观想法, 将上述置信区间的上界作为 $q_*(a)$ 的估计, 给出如下的行动结果:

$$A_t = \arg\max_a \left[Q_t(a) + c\sigma_t(a)\sqrt{\frac{\ln t}{N_t(a)}}\right]$$

引入因子 $\ln t$ 和常数 c 可以更好地控制误差. 通过选择具有最高置信上界的行动 a, 就会倾向于选择那些既有较高期望奖励又较少被探索的行动, 从而获得更好的探索-利用平衡效果.

12 马尔科夫决策过程

对于强化学习问题, 常使用马尔科夫决策过程的数学框架来进行描述. 我们在这一章里对这一框架进行简单介绍, 并引出核心的贝尔曼最优方程.

12.1 有限马尔科夫决策过程

定义 12.1 随机事件的马尔科夫性质 给定一个随机过程的当前状态 s_t 和历史的所有状态 $s_1, \cdots, s_{t-1}$, 如果未来时刻的状态仅依赖于当前状态 s_t 而与历史状态无关, 即

$$p(s_{t+1}|s_1, \cdots, s_t) = p(s_{t+1}|s_t)$$

则称该随机过程具有**马尔科夫性质**

定义 12.2 马尔科夫过程 具有马尔科夫性质的随机过程称作**马尔科夫过程**.

马尔科夫过程具有广泛的应用. 在物理和化学中, 马尔科夫过程可用于对动力学过程进行建模. 例如, 化学反应就可以看成马尔科夫过程. 氧分子和氢分子能否碰撞生成水分子, 具有一定的随机性与概率, 但与氧分子和氢分子的历史位置和来源无关.

定义 12.3 有限马尔科夫过程 状态的可能取值的数目有限的马尔科夫过程称作**有限马尔科夫过程**. 此时的条件概率可以用状态转移矩阵描述:

$$p(s_{t+1}|s_t) = p(s'|s) = \begin{bmatrix} p_{11} & \cdots & p_{1n} \\ \vdots & \ddots & \vdots \\ p_{n1} & \cdots & p_{nn} \end{bmatrix}$$

其中矩阵的行号代表转移前的状态, 列号代表转移后的状态.

定义 12.4 马尔可夫决策过程 在马尔科夫过程的基础上对每个状态增加可自由控制的行动 a 和回报 r 就是**马尔科夫决策过程 (Markov Decision Process, MDP)**. 此时, 随机过程的行为由概率 $p(s', r|s, a)$ 刻画, 即体系处于状态 s 并采取行动 a 后状态变成 s' 并获得回报 r 的概率.

结合上述概念就是有限马尔科夫决策过程. 基于 $p(s', r|s, a)$ 可以计算一些其它的重要的概率, 例如:

1. 状态转移概率

$$p(s'|s, a) = \sum_{r \in \mathcal{R}} p(s', r|s, a)$$

2. 状态-行动的回报期望

$$r(s, a) = \sum_{r \in \mathcal{R}, s' \in \mathcal{S}} r p(s', r|s, a)$$

3. 状态-行动-后继状态的回报期望

$$r(s,a,s') = \sum_{r\in\mathcal{R}} rp(s',r|s,a) = \sum_{r\in\mathcal{R}} r\frac{p(s',r|s,a)}{p(s'|s,a)}$$

MDP 将马尔可夫性质应用于时序决策建模. 对于强化学习问题, 如果智能体的状态转移和回报符合马尔可夫性, 并进一步假设智能体的策略也具有马尔可夫性, 则强化学习问题就可以描述成马尔可夫决策过程, 进而可以利用动态规划, 随机采样等方法来求解使回报最大化的智能体策略. 具体地, MDP 框架把目标导向行为的强化学习概括成智能体与环境之间来回传递的三种信号, 即行动 a, 状态 s 和回报 r.

12.2 贝尔曼方程

12.2.1 目标函数

在马尔科夫决策过程的普适框架下, 强化学习在 t 时刻的决策目标, 是使之后接收的累积回报最大化. 定义目标函数

$$G_t = R_{t+1} + \cdots + R_t = \sum_{k=t+1}^{T} R_k$$

其中 T 是终止时刻. 这种定义适合**分幕制 (回合制)** 任务, 即具有明确开始和结束状态的任务, 例如棋类游戏等.

强化学习有时也涉及持续性任务, 时间无穷长, 没有明确终止状态, 例如恒温器调节温度等任务. 此时, 为防止目标函数发散, 一般引入折扣项 γ, 即

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

折扣项的引入能让算法在短期收益和长期收益达成平衡. 事实上, 上面两种目标函数可以统一地写成

$$G_t = \sum_{k=0}^{T} \gamma^k R_{t+k+1}$$

$\gamma = 1$ 和 $T \to \infty$ 分别代表了两种情形. 从上述式子中我们可以得到目标函数 G_t 的一个重要性质:

$$G_t = R_{t+1} + \gamma G_{t+1}$$

12.2.2 策略与价值函数

策略是指智能体怎样根据状态采取行动的规则.

定义 12.5 策略函数 在马尔科夫决策过程的框架下, **策略函数** π 给出了当状态为 s 时采取行动 a 的概率为 $\pi(a|s)$.

$$\pi(a|s)$$

这指导算法以概率分布的形式来选择行动, 因此是随机性策略.

定义 12.6 状态价值函数 状态 s 在策略 π 下的状态价值函数则是从 s 出发根据策略采取行动所能取得回报的期望:

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s]$$

价值函数衡量了状态 s 下策略 π 的优劣程度.

定义 12.7 行动价值函数 状态 s 在策略 π 下采取行动 a 的价值函数则是从 s 出发采取行动 a 所能取得回报的期望:

$$q_\pi(s,a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a]$$

行动价值函数衡量了在状态 s 下采取行动 a 的优劣程度.

状态价值函数和行动价值函数满足如下关系:

$$v_\pi(s) = \sum_{a\in\mathcal{A}} q_\pi(s,a)\pi(a|s)$$

12.2.3 贝尔曼方程

强化学习的贝尔曼方程是描述价值函数迭代关系的核心公式. 我们结合前面两小节的公式可得

$$\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi[G_t|S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s] \\
&= \sum_{a\in\mathcal{A}} \pi(a|s) \sum_{s'\in\mathcal{S}, r\in\mathcal{R}} p(s', r|s, a)\left(r + \gamma\mathbb{E}_\pi[G_{t+1}|S_{t+1} = s']\right) \\
&= \sum_{a\in\mathcal{A}} \pi(a|s) \sum_{s'\in\mathcal{S}, r\in\mathcal{R}} p(s', r|s, a)\left(r + \gamma v_\pi(s')\right)
\end{aligned}$$

类似地可得

$$q_\pi(s,a) = \sum_{s'\in\mathcal{S}, r\in\mathcal{R}} p(s', r|s, a)\left[r + \gamma \sum_{a'\in\mathcal{A}} q_\pi(s', a')\pi(a'|s')\right]$$

定义 12.8 贝尔曼方程 上述关于状态价值函数和行动价值函数的迭代方程称作**贝尔曼方程**.

贝尔曼方程将复杂的长期决策问题分解为当前行动与未来状态的迭代关系, 从而将问题转化为类似递归的形式.

12.2.4 最优策略与贝尔曼最优方程

在所有策略中, 总存在一个 (或一组) 最优策略 π^* 使得智能体在与环境交互时获得的回报的期望 $\mathbb{E}_\pi[G_t]$. 记最优策略下的状态价值函数为 $v_*(s)$, 行动价值函数为 $q_*(s,a)$, 则有

$$v_*(s) = \max_\pi v_\pi(s), \quad q_{*(s,a)=\max_\pi q_p i(s,a)}$$

$v_*(s)$ 就是从状态 s 出发能得到的最大的平均累计回报. 如果我们先采取行动 a, 然后遵循策略 π, 那么得到的累计回报事实上就是行动价值函数 $q_\pi(s,a)$. 为了使这一值最大, 我们可以改变 a 和 π, 以得到最大的取值作为 $v_*(s)$ 的结果, 即

$$v_*(s) = \max_{a,\pi} q_\pi(s,a) = \max_a q_*(s,a)$$

即, 最优策略下的某一状态 s 的价值等于该状态下最优行动的价值. 利用马尔可夫决策过程的特性将上式展开可得

$$\begin{aligned} v_*(s) &= \max_a q_*(s,a) \\ &= \max \mathbb{E}_{\pi^*}[G_t|S_t=s, A_t=a] \\ &= \max \mathbb{E}_{\pi^*}[R_{t+1}+\gamma G_{t+1}|S_t=s, A_t=a] \\ &= \max \mathbb{E}_{\pi^*}[R_{t+1}+\gamma v_*(S_{t+1})|S_t=s, A_t=a] \\ &= \max_a \sum_{s'\in\mathcal{S}, r\in\mathcal{R}} p(s',r|s,a)\left(r+\gamma v_*(s')\right) \end{aligned}$$

类似地可得

$$q_*(s,a) = \sum_{s'\in\mathcal{S}, r\in\mathcal{R}} p(s',r|s,a)\left(r+\gamma \max_{a'\in\mathcal{A}} q_*(s',a')\right)$$

定义 12.9 贝尔曼最优方程 上述关于最优策略下状态价值函数和行动价值函数的迭代方程称作**贝尔曼最优方程**.

原则上可以基于贝尔曼最优方程求解 $v_*(s)$ 与 $q_*(s,a)$. 不过由于 max 的存在, 方程组是非线性的, 实际求解比较困难.

如果 $v_*(s)$ 与 $q_*(s,a)$ 已经求得, 那么最优策略就变得非常容易, 只需利用 $v_*(s)$ 向前搜索一步

$$a^* = \arg\min_a \mathbb{E}_{\pi^*}[R_{t+1}+\gamma v_*(S_{t+1})|S_t=s, A_t=a] = \arg\min_a q_*(s,a)$$

即可得出最优行动 a^*. 我们将在下一节介绍求解这两个函数的具体办法.

13 强化学习的三种算法

13.1 动态规划

动态规划 (Dynamic Programming, DP) 是运筹学的一个分支, 是求解决策过程最优化的一种数学方法. 动态规划可用于模型 $p(s', r|s, a)$ 已知的马尔科夫决策过程的最优策略的求解, 因此可以作为一种强化学习的方法使用.

13.1.1 策略评估

根据前一节的介绍, 状态价值函数 $v_\pi(s)$ 满足贝尔曼方程:

$$v_\pi(s) = \sum_{a\in\mathcal{A}} \pi(a|s) \sum_{s'\in\mathcal{S}, r\in\mathcal{R}} p(s', r|s, a)\,(r + \gamma v_\pi(s'))$$

这是以 $v_\pi(s)$ 为变量的线性方程组, 因此可以用线性方程组的解法直接求解. 但是, 这样做的效率往往比较低, 因此动态规划通常将公式看作对 $v_\pi(s)$ 的迭代公式, 利用如下迭代法来进行求解:

$$v_{k+1}(s) = \sum_{a\in\mathcal{A}} \pi(a|s) \sum_{s'\in\mathcal{S}, r\in\mathcal{R}} p(s', r|s, a)\,(r + \gamma v_k(s'))$$

数学上可以证明

$$\lim_{k\to\infty} v_k(s) = v_\pi(s)$$

因此, 实际求解时只需将 $v_k(s)$ 初始化, 例如赋值为 0, 然后迭代直至收敛即可.

13.1.2 策略改进

利用策略评估得到某个策略 $\pi(a|s)$ 下的价值函数 $v_\pi(s)$ 以后, 就可以据此进行**策略改进 (Policy Improvement)**, 即通过优化当前策略以获得更优策略, 其核心目标是让新策略的价值函数不低于原策略, 以便最终逐步逼近最优策略.

当智能体面对状态 s 时, 如果一直遵循策略 π, 则回报是 $v_\pi(s)$. 如果智能体不遵循策略 π 而采取行动 a, 之后再遵循 π, 此时的回报就是策略 π 下的行动价值函数

$$q_\pi(s, a) = \sum_{s'\in\mathcal{S}, r\in\mathcal{R}} p(s', r|s, a)\,(r + \gamma v_\pi(s'))$$

如果 $q_\pi(s, a) > v_\pi(s)$, 那么这样的选择就比一直遵循 π 更好. 于是下次处于状态 s 时就仍应选择行动 a. 这样得到的新策略 π' 就应当比 π 更优. 一般的, 我们有如下定理.

定理 13.1 策略改进定理 对于确定性的策略 π 和 π', 如果对于任一状态 $s \in \mathcal{S}$ 都有

$$q_\pi(s, \pi'(s)) \geqslant v_\pi(s)$$

则 π' 不劣于 π, 即 $v_{\pi'}(s) \geqslant v_\pi(s)$.
如果是随机性策略, 则上述式子应改写为

$$\sum_{a\in\mathcal{A}} \pi'(a|s) \sum_{s'\in\mathcal{S},r\in\mathcal{R}} p(s',r|s,a)\left(r+\gamma v_\pi(s')\right) \geqslant v_\pi(s)$$

证明. □

由此, 我们可以采用前面所述的贪心的方法更新策略. 对于基于策略 π 的价值函数 $v_\pi(s)$, 可以做前向单步搜索, 构造如下贪心策略:

$$\pi'(s) = \arg\max_a q_\pi(s,a) = \arg\max_a \sum_{s'\in\mathcal{S},r\in\mathcal{R}} p(s',r|s,a)\left(r+\gamma v_\pi(s')\right)$$

由上式的最大化结果可知 $q_\pi(s,\pi'(s)) = \max_a q_\pi(s,a)$, 是所有行动 a 中使得 $q_\pi(s,a)$ 中最大的. 另一方面又有 $v_\pi(s) = \sum_{a\in\mathcal{A}} \pi(a|s) q_\pi(s,a)$ 是所有 $q_\pi(s,a)$ 的加权平均, 因此总有 $q_\pi(s,\pi'(s)) \geqslant v_\pi(s)$, 满足策略改进定理, 因此 π' 总是比 π 更好的策略 (除非 π 已经是在 s 下最优的策略).

总之, 我们证明了使用单步的贪心算法不断迭代就能使策略最优化. 这是动态规划求解强化学习问题的核心机制之一.

13.1.3 策略迭代与价值迭代

按照上述过程, 我们可以把策略迭代过程描述如下:

$$\pi_0 \xrightarrow{\text{E}} v_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} v_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} v_{\pi_*}$$

其中 E 为策略评估步骤, I 为策略改进步骤. 由于每次策略改进都使策略更优, 因此算法能保证收敛到最优解. 这样的 EI 循环与聚类问题中的 EM 循环是非常相似的.

在策略评估 E 步骤中, 通常需要反复迭代才能得到策略 π_k 的价值函数 $v_{\pi_k}(s)$. 由于 π_k 本来就作为需要改进的量, 因此精确计算并无必要, 我们可以只进行一次迭代. 于是就有以下的**价值迭代算法**:

$$v_{k+1}(s) = \max_a \sum_{s'\in\mathcal{S},r\in\mathcal{R}} p(s',r|s,a)\left(r+\gamma v_k(s')\right)$$

这就把 E 和 I 步骤合为一步. 在收敛后即可导出最优策略 π_*.

13.2 蒙特卡罗方法

在强化学习中, **蒙特卡罗 (Monte Carlo, MC) 方法**是一类基于采样轨迹来学习价值函数或策略的方法. 它不需要关于环境的完整信息 (即 $p(s',r|s,a)$), 而是直接从经验中学习, 即利用来自真实或模拟的环境交互中采样得到的状态, 行动, 回报的序列数据. 蒙特卡罗方法的核心思想是通过多次试验 (采样) 的平均结果来逼近真实的期望价值.

13.2.1 价值估计与策略改进

状态 s 在策略 π 下的价值函数就是 G_t 的期望值, 即 $v_\pi(s)=\mathbb{E}[G_t|S_t=s]$. 从一条轨迹可以直接得到

$$G_t=\sum_{k=0}^{T}\gamma^k R_{k+t+1}$$

蒙特卡罗方法就是通过在策略 π 下多次采样包含状态 s 的完整轨迹, 计算每次轨迹中从 s 开始的 G_t, 取其均值作为 $v_\pi(s)$ 的估计.

常用的**首次访问型 MC 算法 (First-visit MC)** 的流程如下:

1. 输入待评估的策略 π.

2. 对于所有状态 s, 初始化 s 的采样列表 $R(s)$.

3. 根据 π 生成一幕序列

$$S_0,A_0,R_1,S_1,A_1,R_1,\cdots,S_{T-1},A_{T-1},R_T$$

对本幕中的每一步循环, 分别令 $t=T-1,T-2,\cdots,0$, 根据 $G_t=\gamma G_{t+1}+R_{t+1}$ 计算 G_t, 然后如果 S_t 没有在 $S_0,\cdots,S_{t-1}$ 中出现过, 就将 G_t 加入 S_t 的采样列表 $R(S_t)$ 中.

4. 重复上述采样操作, 最后根据 S_t 的采样列表 $R(S_t)$ 的均值计算 $v_\pi(S_t)$.

即对每条轨迹中首次出现的 s 进行记录和平均. 另外, 还有**每次访问型 MC 算法 (Every-visit MC)**, 即对每条轨迹中所有出现的 s 都进行记录和平均.

当采样的轨迹足够多时, 两种算法都能收敛到 $v_\pi(s)$. 当轨迹有限时, 首次访问型 MC 算法是无偏的, 而每次访问型 MC 算法则是有偏的. 当轨迹较少时, 每次访问型 MC 算法效果更好, 因为它能提取更多的数据进行计算; 当轨迹较多时, 首次访问型 MC 算法收敛更快. 另外, 对动作价值函数 $q_\pi(s,a)$ 的计算也是类似的.

13.2.2 基于重要性采样的离轨策略

所有的学习控制方法都面临一个困境: 希望学到的策略可以使随后的行为是最优的 (此时往往只有一个最优行动), 但是为了搜索所有的行动 (以保证找到最优行动与策略), 又需要采取非最优行动. 我们可以采用两个策略, 一个用于学习并成为最优策略, 而另一个则更加具有探索性, 用于形成采样的样本. 普适地讲, 它们分属于以下两种类型:

同轨策略: 用于生成采样数据序列的策略 b 和用于实际决策的待评估和改进的策略 π 是相同的.

离轨策略: 用于生成采样数据序列的策略 b 和用于实际决策的待评估和改进的策略 π 是不同的.

通俗而言, 同轨策略可以类比为自己下棋并提高水平, 而离轨策略可以类比为看别人下棋提高水平. 我们现在来证明离轨策略的有效性.

证明. 从 S_t 出发, 轨迹 $S_t,A_t,S_{t+1},A_{t+1},\cdots,S_T$ 在策略 π 下发生的概率为

$$p(A_t,S_{t+1},\cdots,S_T|S_t,\pi)=\prod_{k=t}^{T-1}\pi(A_k|S_k)p(S_{k+1}|S_k,A_k)$$

类似地可得出同样的轨迹在策略 b 下的概率. 因此, 同一条轨迹在两种策略下的概率之比为

$$\rho_{t:T-1}=\frac{\prod_{k=t}^{T-1}\pi(A_k|S_k)p(S_{k+1}|S_k,A_k)}{\prod_{k=t}^{T-1}b(A_k|S_k)p(S_{k+1}|S_k,A_k)}=\frac{\prod_{k=t}^{T-1}\pi(A_k|S_k)}{\prod_{k=t}^{T-1}b(A_k|S_k)}$$

因此, 注意到这概率与环境的响应机制 $p(S_{k+1}|S_k,A_k)$ 无关, 只与两个策略有关.

根据定义, 策略 b 下的价值函数为

$$v_b(s)=\mathbb{E}_b[G_t|S_t=s]=\sum_{A_t,S_{t+1}\cdots,S_T}p(A_t,S_{t+1},\cdots,S_T|S_t,b)G_t$$

而策略 π 下的价值函数可以写成

$$\begin{aligned}v_\pi(s)&=\sum_{A_t,S_{t+1}\cdots,S_T}p(A_t,S_{t+1},\cdots,S_T|S_t,\pi)G_t\\&=\rho_{t:T-1}\sum_{A_t,S_{t+1}\cdots,S_T}p(A_t,S_{t+1},\cdots,S_T|S_t,b)G_t\\&=\mathbb{E}_b[\rho_{t:T-1}G_t|S_t=s]\end{aligned}$$

因此, 根据策略 b 下产生的蒙特卡罗模拟轨迹可以调整计算得到策略 π 下的结果:

$$v_\pi(s)=\frac{\sum_{\text{traj}}\rho_{t:T-1}G_t}{\sum_{\text{traj}}1}=\frac{1}{N}\sum_{\text{traj}}\rho_{t:T-1}G_t$$

其中 traj 是满足 $S_t=s$ 的采样轨迹,N 是采样的轨迹数目.

另外, 我们知道轨迹概率满足如下归一性质:

$$\sum_{A_t,S_{t+1}\cdots,S_T}p(A_t,S_{t+1},\cdots,S_T|S_t,\pi)=1$$

于是我们也可以把 $v_\pi(s)$ 改写为

$$v_\pi(s)=\frac{\sum_{A_t,S_{t+1},\cdots,S_T}p(A_t,S_{t+1},\cdots,S_T|S_t,\pi)G_t}{\sum_{A_t,S_{t+1},\cdots,S_T}p(A_t,S_{t+1},\cdots,S_T|S_t,\pi)}$$

然后把 $\rho_{t:T-1}$ 代入上式可得

$$v_\pi(s)=\frac{\mathbb{E}_b(\rho_{t:T-1}G_t|S_t=s)}{\mathbb{E}_b(\rho_{t:T-1}|S_t=s)}=\frac{\sum_{\text{traj}}\rho_{t:T-1}G_t}{\sum_{\text{traj}}\rho_{t:T-1}}$$

□

于是我们得到了两种计算 $v_\pi(s)$ 的公式. 它们在采样轨迹数目足够多时都收敛到真值; 在轨迹有限时, 前者是无偏的, 但方差较大; 后者是有偏的, 但方差较小. 实际应用中一般采用后者.

13.3 时序差分算法

13.3.1 时序差分预测

时序差分算法 (Temporal Difference, TD) 是强化学习中一种结合了动态规划和蒙特卡罗方法优点的重要算法, 核心是通过 bootstrapping(利用估计值更新估计值) 和在线学习 (无需等待完整轨迹结束) 来更新价值估计.

在蒙特卡罗模拟的在线学习中, 根据 $v_\pi(s)=\mathbb{E}_\pi[G_t|S_t=s]$ 可得如下更新公式

$$V(S_t)\leftarrow V(S_t)+\eta(G_t-V(S_t))$$

其中 $V(S_t)$ 是 $v_\pi(S_t)$ 的估计值, η 是学习率, G_t 需要等待一条轨迹结束之后才能运算得到.

现在, 根据 G_t 的性质可知 $v_\pi(s)=\mathbb{E}_\pi[R_{t+1}+\gamma v_\pi(S_{t+1})|S_t=s]$, 于是可以把更新公式改写为

$$V(S_t)\leftarrow V(S_t)+\eta(R_{t+1}+\gamma V(S_{t+1})-V(S_t))$$

其中 R_{t+1} 和 S_{t+1} 在单步完成后即可得到. 利用上式即可实现简单的时序差分算法, 称作单步 TD 法.

13.3.2 时序差分控制与 Q 学习

在决策优化问题中更重要的是动作价值函数. 类似地, 我们可以写出 $q_\pi(s,a)$ 的估计 $Q(S_t,A_t)$ 的更新公式:

$$Q(S_t,A_t)\leftarrow Q(S_t,A_t)+\eta(R_{t+1}+\gamma Q(S_{t+1},A_{t+1})-Q(S_t,A_t))$$

这里需要使用轨迹数据的五元组 $(S_t,A_t,R_{t+1},S_{t+1},A_{t+1})$, 因此算法称作 **SARSA 算法**. 这本质上是一种同轨策略. 这一算法的具体步骤为:

(i) 初始化 $Q(s,a)$ 表, 并且选定初始状态 S.

(ii) 根据当前状态 S_t 和当前策略选择行动 A_t.

(iii) 执行动作 A_t, 获得回报 R_{t+1}, 到达下一个状态 S_{t+1}.

(iv) 根据状态 S_{t+1} 和当前策略选择行动 A_{t+1}.

(v) 使用上述公式

$$Q(S_t,A_t)\leftarrow Q(S_t,A_t)+\eta(R_{t+1}+\gamma Q(S_{t+1},A_{t+1})-Q(S_t,A_t))$$

更新 $Q(S_t,A_t)$.

(vi) 重复上述 **(ii)** 到 **(v)** 步骤直到终止状态, 然后结束当前回合, 进入下一回合学习.

当然, 也可以使用离轨策略下的时序差分控制. 根据贝尔曼最优方程

$$q_*(s,a)=\sum_{s'\in\mathcal{S},r\in\mathcal{R}}p(s',r|s,a)\left(r+\gamma\max_{a'\in\mathcal{A}}q_*(s',a')\right)$$

并且根据时序差分预测中的更新公式可得

$$Q(S_t,A_t)\leftarrow Q(S_t,A_t)+\eta\left[R_{t+1}+\gamma\max_{a\in\mathcal{A}}Q(S_{t+1},a)-Q(S_t,A_t)\right]$$

这就是 **Q-学习 (Q-Learning)** 的基本思想. 这是一种离轨策略: 学习的是最优策略 (通过 max 操作实现), 而实际模拟轨迹的是探索策略 (例如 ε-贪心算法). 与 SARSA 相比, Q-学习不需要 A_{t+1}, 而是直接取可能的最大价值, 因此能更加稳定地逼近最优的 Q 函数.

14 强化学习的三种算法

14.1 动态规划

动态规划 (Dynamic Programming, DP) 是运筹学的一个分支, 是求解决策过程最优化的一种数学方法. 动态规划可用于模型 $p(s', r|s, a)$ 已知的马尔科夫决策过程的最优策略的求解, 因此可以作为一种强化学习的方法使用.

14.1.1 策略评估

根据前一节的介绍, 状态价值函数 $v_\pi(s)$ 满足贝尔曼方程:

$$v_\pi(s) = \sum_{a\in\mathcal{A}} \pi(a|s) \sum_{s'\in\mathcal{S}, r\in\mathcal{R}} p(s', r|s, a)\,(r + \gamma v_\pi(s'))$$

这是以 $v_\pi(s)$ 为变量的线性方程组, 因此可以用线性方程组的解法直接求解. 但是, 这样做的效率往往比较低, 因此动态规划通常将公式看作对 $v_\pi(s)$ 的迭代公式, 利用如下迭代法来进行求解:

$$v_{k+1}(s) = \sum_{a\in\mathcal{A}} \pi(a|s) \sum_{s'\in\mathcal{S}, r\in\mathcal{R}} p(s', r|s, a)\,(r + \gamma v_k(s'))$$

数学上可以证明

$$\lim_{k\to\infty} v_k(s) = v_\pi(s)$$

因此, 实际求解时只需将 $v_k(s)$ 初始化, 例如赋值为 0, 然后迭代直至收敛即可.

14.1.2 策略改进

利用策略评估得到某个策略 $\pi(a|s)$ 下的价值函数 $v_\pi(s)$ 以后, 就可以据此进行**策略改进 (Policy Improvement)**, 即通过优化当前策略以获得更优策略, 其核心目标是让新策略的价值函数不低于原策略, 以便最终逐步逼近最优策略.

当智能体面对状态 s 时, 如果一直遵循策略 π, 则回报是 $v_\pi(s)$. 如果智能体不遵循策略 π 而采取行动 a, 之后再遵循 π, 此时的回报就是策略 π 下的行动价值函数

$$q_\pi(s, a) = \sum_{s'\in\mathcal{S}, r\in\mathcal{R}} p(s', r|s, a)\,(r + \gamma v_\pi(s'))$$

如果 $q_\pi(s, a) > v_\pi(s)$, 那么这样的选择就比一直遵循 π 更好. 于是下次处于状态 s 时就仍应选择行动 a. 这样得到的新策略 π' 就应当比 π 更优. 一般的, 我们有如下定理.

定理 14.1 策略改进定理 对于确定性的策略 π 和 π', 如果对于任一状态 $s \in \mathcal{S}$ 都有

$$q_\pi(s, \pi'(s)) \geqslant v_\pi(s)$$

则 π' 不劣于 π, 即 $v_{\pi'}(s) \geqslant v_\pi(s)$.
如果是随机性策略, 则上述式子应改写为

$$\sum_{a\in\mathcal{A}} \pi'(a|s) \sum_{s'\in\mathcal{S},r\in\mathcal{R}} p(s',r|s,a)\left(r+\gamma v_\pi(s')\right) \geqslant v_\pi(s)$$

证明. □

由此, 我们可以采用前面所述的贪心的方法更新策略. 对于基于策略 π 的价值函数 $v_\pi(s)$, 可以做前向单步搜索, 构造如下贪心策略:

$$\pi'(s) = \arg\max_a q_\pi(s,a) = \arg\max_a \sum_{s'\in\mathcal{S},r\in\mathcal{R}} p(s',r|s,a)\left(r+\gamma v_\pi(s')\right)$$

由上式的最大化结果可知 $q_\pi(s,\pi'(s)) = \max_a q_\pi(s,a)$, 是所有行动 a 中使得 $q_\pi(s,a)$ 中最大的. 另一方面又有 $v_\pi(s) = \sum_{a\in\mathcal{A}} \pi(a|s) q_\pi(s,a)$ 是所有 $q_\pi(s,a)$ 的加权平均, 因此总有 $q_\pi(s,\pi'(s)) \geqslant v_\pi(s)$, 满足策略改进定理, 因此 π' 总是比 π 更好的策略 (除非 π 已经是在 s 下最优的策略).

总之, 我们证明了使用单步的贪心算法不断迭代就能使策略最优化. 这是动态规划求解强化学习问题的核心机制之一.

14.1.3 策略迭代与价值迭代

按照上述过程, 我们可以把策略迭代过程描述如下:

$$\pi_0 \xrightarrow{\text{E}} v_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} v_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} v_{\pi_*}$$

其中 E 为策略评估步骤, I 为策略改进步骤. 由于每次策略改进都使策略更优, 因此算法能保证收敛到最优解. 这样的 EI 循环与聚类问题中的 EM 循环是非常相似的.

在策略评估 E 步骤中, 通常需要反复迭代才能得到策略 π_k 的价值函数 $v_{\pi_k}(s)$. 由于 π_k 本来就作为需要改进的量, 因此精确计算并无必要, 我们可以只进行一次迭代. 于是就有以下的**价值迭代算法**:

$$v_{k+1}(s) = \max_a \sum_{s'\in\mathcal{S},r\in\mathcal{R}} p(s',r|s,a)\left(r+\gamma v_k(s')\right)$$

这就把 E 和 I 步骤合为一步. 在收敛后即可导出最优策略 π_*.

14.2 蒙特卡罗方法

在强化学习中, **蒙特卡罗 (Monte Carlo, MC) 方法**是一类基于采样轨迹来学习价值函数或策略的方法. 它不需要关于环境的完整信息 (即 $p(s',r|s,a)$), 而是直接从经验中学习, 即利用来自真实或模拟的环境交互中采样得到的状态, 行动, 回报的序列数据. 蒙特卡罗方法的核心思想是通过多次试验 (采样) 的平均结果来逼近真实的期望价值.

14.2.1 价值估计与策略改进

状态 s 在策略 π 下的价值函数就是 G_t 的期望值, 即 $v_\pi(s)=\mathbb{E}[G_t|S_t=s]$. 从一条轨迹可以直接得到

$$G_t=\sum_{k=0}^{T}\gamma^k R_{k+t+1}$$

蒙特卡罗方法就是通过在策略 π 下多次采样包含状态 s 的完整轨迹, 计算每次轨迹中从 s 开始的 G_t, 取其均值作为 $v_\pi(s)$ 的估计.

常用的**首次访问型 MC 算法 (First-visit MC)** 的流程如下:

1. 输入待评估的策略 π.

2. 对于所有状态 s, 初始化 s 的采样列表 $R(s)$.

3. 根据 π 生成一幕序列

$$S_0,A_0,R_1,S_1,A_1,R_1,\cdots,S_{T-1},A_{T-1},R_T$$

对本幕中的每一步循环, 分别令 $t=T-1,T-2,\cdots,0$, 根据 $G_t=\gamma G_{t+1}+R_{t+1}$ 计算 G_t, 然后如果 S_t 没有在 $S_0,\cdots,S_{t-1}$ 中出现过, 就将 G_t 加入 S_t 的采样列表 $R(S_t)$ 中.

4. 重复上述采样操作, 最后根据 S_t 的采样列表 $R(S_t)$ 的均值计算 $v_\pi(S_t)$.

即对每条轨迹中首次出现的 s 进行记录和平均. 另外, 还有**每次访问型 MC 算法 (Every-visit MC)**, 即对每条轨迹中所有出现的 s 都进行记录和平均.

当采样的轨迹足够多时, 两种算法都能收敛到 $v_\pi(s)$. 当轨迹有限时, 首次访问型 MC 算法是无偏的, 而每次访问型 MC 算法则是有偏的. 当轨迹较少时, 每次访问型 MC 算法效果更好, 因为它能提取更多的数据进行计算; 当轨迹较多时, 首次访问型 MC 算法收敛更快. 另外, 对动作价值函数 $q_\pi(s,a)$ 的计算也是类似的.

14.2.2 基于重要性采样的离轨策略

所有的学习控制方法都面临一个困境: 希望学到的策略可以使随后的行为是最优的 (此时往往只有一个最优行动), 但是为了搜索所有的行动 (以保证找到最优行动与策略), 又需要采取非最优行动. 我们可以采用两个策略, 一个用于学习并成为最优策略, 而另一个则更加具有探索性, 用于形成采样的样本. 普适地讲, 它们分属于以下两种类型:

同轨策略: 用于生成采样数据序列的策略 b 和用于实际决策的待评估和改进的策略 π 是相同的.

离轨策略: 用于生成采样数据序列的策略 b 和用于实际决策的待评估和改进的策略 π 是不同的.

通俗而言, 同轨策略可以类比为自己下棋并提高水平, 而离轨策略可以类比为看别人下棋提高水平. 我们现在来证明离轨策略的有效性.

证明. 从 S_t 出发, 轨迹 $S_t,A_t,S_{t+1},A_{t+1},\cdots,S_T$ 在策略 π 下发生的概率为

$$p(A_t,S_{t+1},\cdots,S_T|S_t,\pi)=\prod_{k=t}^{T-1}\pi(A_k|S_k)p(S_{k+1}|S_k,A_k)$$

类似地可得出同样的轨迹在策略 b 下的概率. 因此, 同一条轨迹在两种策略下的概率之比为

$$\rho_{t:T-1}=\frac{\prod_{k=t}^{T-1}\pi(A_k|S_k)p(S_{k+1}|S_k,A_k)}{\prod_{k=t}^{T-1}b(A_k|S_k)p(S_{k+1}|S_k,A_k)}=\frac{\prod_{k=t}^{T-1}\pi(A_k|S_k)}{\prod_{k=t}^{T-1}b(A_k|S_k)}$$

因此, 注意到这概率与环境的响应机制 $p(S_{k+1}|S_k,A_k)$ 无关, 只与两个策略有关.

根据定义, 策略 b 下的价值函数为

$$v_b(s)=\mathbb{E}_b[G_t|S_t=s]=\sum_{A_t,S_{t+1}\cdots,S_T}p(A_t,S_{t+1},\cdots,S_T|S_t,b)G_t$$

而策略 π 下的价值函数可以写成

$$\begin{aligned}v_\pi(s)&=\sum_{A_t,S_{t+1}\cdots,S_T}p(A_t,S_{t+1},\cdots,S_T|S_t,\pi)G_t\\&=\rho_{t:T-1}\sum_{A_t,S_{t+1}\cdots,S_T}p(A_t,S_{t+1},\cdots,S_T|S_t,b)G_t\\&=\mathbb{E}_b[\rho_{t:T-1}G_t|S_t=s]\end{aligned}$$

因此, 根据策略 b 下产生的蒙特卡罗模拟轨迹可以调整计算得到策略 π 下的结果:

$$v_\pi(s)=\frac{\sum_{\text{traj}}\rho_{t:T-1}G_t}{\sum_{\text{traj}}1}=\frac{1}{N}\sum_{\text{traj}}\rho_{t:T-1}G_t$$

其中 traj 是满足 $S_t=s$ 的采样轨迹,N 是采样的轨迹数目.

另外, 我们知道轨迹概率满足如下归一性质:

$$\sum_{A_t,S_{t+1}\cdots,S_T}p(A_t,S_{t+1},\cdots,S_T|S_t,\pi)=1$$

于是我们也可以把 $v_\pi(s)$ 改写为

$$v_\pi(s)=\frac{\sum_{A_t,S_{t+1},\cdots,S_T}p(A_t,S_{t+1},\cdots,S_T|S_t,\pi)G_t}{\sum_{A_t,S_{t+1},\cdots,S_T}p(A_t,S_{t+1},\cdots,S_T|S_t,\pi)}$$

然后把 $\rho_{t:T-1}$ 代入上式可得

$$v_\pi(s)=\frac{\mathbb{E}_b(\rho_{t:T-1}G_t|S_t=s)}{\mathbb{E}_b(\rho_{t:T-1}|S_t=s)}=\frac{\sum_{\text{traj}}\rho_{t:T-1}G_t}{\sum_{\text{traj}}\rho_{t:T-1}}$$

□

于是我们得到了两种计算 $v_\pi(s)$ 的公式. 它们在采样轨迹数目足够多时都收敛到真值; 在轨迹有限时, 前者是无偏的, 但方差较大; 后者是有偏的, 但方差较小. 实际应用中一般采用后者.

14.3 时序差分算法

14.3.1 时序差分预测

时序差分算法 (Temporal Difference, TD) 是强化学习中一种结合了动态规划和蒙特卡罗方法优点的重要算法, 核心是通过 bootstrapping(利用估计值更新估计值) 和在线学习 (无需等待完整轨迹结束) 来更新价值估计.

在蒙特卡罗模拟的在线学习中, 根据 $v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s]$ 可得如下更新公式

$$V(S_t) \leftarrow V(S_t) + \eta(G_t - V(S_t))$$

其中 $V(S_t)$ 是 $v_\pi(S_t)$ 的估计值, η 是学习率, G_t 需要等待一条轨迹结束之后才能运算得到.

现在, 根据 G_t 的性质可知 $v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s]$, 于是可以把更新公式改写为

$$V(S_t) \leftarrow V(S_t) + \eta(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$$

其中 R_{t+1} 和 S_{t+1} 在单步完成后即可得到. 利用上式即可实现简单的时序差分算法, 称作单步 TD 法.

14.3.2 时序差分控制与 Q 学习

在决策优化问题中更重要的是动作价值函数. 类似地, 我们可以写出 $q_\pi(s, a)$ 的估计 $Q(S_t, A_t)$ 的更新公式:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \eta(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t))$$

这里需要使用轨迹数据的五元组 $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$, 因此算法称作 **SARSA 算法**. 这本质上是一种同轨策略.

15 深度学习

15.1 引言

15.1.1 深度学习的定义和特征

根据数据形态与反馈信息的不同, 机器学习大致可以分为监督学习, 无监督学习和强化学习三种类型. 深度学习则是一种方法与体系架构, 可以用于这些任务中. 作为神经网络的一种, 深度学习的深度体现在其神经网络包含多个隐藏层, 从而逐层提取数据特征[6].

深度学习是当前人工智能繁荣时期的主角. 互联网的发展创造了大量可用的训练数据, 而芯片技术的发展大大提高了计算能力, 这两者使得深度神经网络的计算与训练成为可能.

15.1.2 深度学习的实例

深度学习的一个热门应用就是语言模型, 通常由于模型的规模很大, 称作**大语言模型 (Large Language Model, LLM)**.

在著名的 Attention is all you need 这篇论文中, 人们提出了注意力机制以及被称作 Transformer 的通用架构网络, 是近几年最重要, 应用最广泛的深度学习架构.

15.2 深度神经网络的训练

由于层数的增加, 深度神经网络的训练异常困难 (这也是深度神经网络在早期不被看好的原因之一). 为此, 人们发展了各种适用于深度神经网络训练的方法.

15.2.1 ADAM 与小批量算法

在梯度下降法中, 我们采用下式来更新优化模型的权重 $\boldsymbol{w}$:

$$\boldsymbol{w}^{(\tau+1)} = \boldsymbol{w}^{(\tau)} - \eta \nabla_{\boldsymbol{w}} E(\boldsymbol{w}^{(\tau)})$$

其中 $E(\boldsymbol{w})$ 是误差函数, η 是学习率. 在深度神经网络中, 由于参数众多, $E(\boldsymbol{w})$ 是高维空间中的复杂曲面, 使用梯度下降法容易陷入到局部极小值点或鞍点中.

如果把 $E(\boldsymbol{w})$ 想象成能量, $\boldsymbol{w}$ 想象成位置, 则 $-\nabla_{\boldsymbol{w}} E(\boldsymbol{w})$ 相当于是力, 于是上式是让位置的变化与力成正比, 换言之就是沿着能量下降最快的方向前进. 这种想法在遇到病态曲率的情况会出现问题, 因此人们提出了一种更加符合物理思维的想法, 即将力转换为加速度, 先影响速度, 再由速度影响位置. 这就是 ADAM 算法, 其具体过程如下:

i. 计算梯度:

$$\boldsymbol{g}_t \leftarrow \nabla_{\boldsymbol{w}} E(\boldsymbol{w}_{t-1})$$

[6] 例如识别图像时, 先识别边缘和颜色, 再到纹理和形状, 最后到完整的物体.

ii. 计算速度:

$$\boldsymbol{v}_t \leftarrow \beta_1 \boldsymbol{v}_{t-1} + (1-\beta_1)\boldsymbol{g}_t$$

iii. 计算梯度平方:

$$r_t \leftarrow \beta_2 r_{t-1} + (1-\beta_2)||\boldsymbol{g}_t||^2$$

iv. 矫正偏差:

$$\hat{\boldsymbol{v}}_t \leftarrow \frac{1}{1-\beta_1^t}\boldsymbol{v}_t, \quad \hat{r}_t = \frac{1}{1-\beta_2^t} r_t$$

v. 更新参数:

$$\boldsymbol{w}_t = \boldsymbol{w}_{t-1} - \frac{\eta}{\sqrt{\hat{r}_t}+\varepsilon}\hat{\boldsymbol{v}}_t$$

ADAM 收敛速度快, 训练稳定, 对超参数的敏感度较低, 调参难度小, 广泛应用于各类深度学习任务, 是目前最流行的优化算法之一.

15.2.2 梯度消失与残差网络

深度神经网络训练过程所面临的另外一个挑战就是**梯度消失问题 (Vanishing Gradient Problem)**, 过多的隐藏层往往会使模型丧失学习到先前信息的能力, 从而让优化变得极其困难, 模型难以训练, 甚至无法收敛.

梯度消失的核心原因是在反向传播算法中, 参数的更新依赖于梯度的计算, 而梯度需要从网络的输出层反向传播到输入层. 当层数较深时, 梯度在传播的过程中可能会因为不断地乘以小于 1 的数而减小, 最终逐渐趋于 0, 这就是梯度消失现象.

解决梯度消失问题的办法很多, 列举如下:

批归一化 批归一化 **(Batch Normalization)** 可以稳定各层输入分布, 减少梯度波动. 其主要想法是在神经网络训练时, 随着参数更新, 各层输入的分布会不断变化, 这会导致在后续层需要不断适应新的分布, 增加训练难度.

残差网络 何恺明等人提出的**残差网络 (Residual Network, ResNet)** 是解决梯度消失问题的重要方法. 从理论上讲, 增加隐藏层只会使模型更好, 不会更差. 基于这种朴素思想, 可以通过残差方式, 将输入直接跳过部分网络层传递到输出, 形成短路连接:

$$\boldsymbol{y} = F(\boldsymbol{x}, \boldsymbol{w}_i) + \boldsymbol{x}$$

其中 $F(\boldsymbol{x}, \boldsymbol{w}_i)$ 是原来的变换模块; 这被称作一个残差模块. 通过这样的残差连接, 输入中的信息总是可以被传递到任意一层, 不容易发生信息的丢失. 通过残差网络, 我们更容易构建多层的网络, 从而得到性能更好的模型.

15.2.3 过拟合

与其它的机器学习方法类似, 深度神经网络也会发生过拟合. 除了一些常见的防止过拟合的方法 (例如增加数据量, 正则化) 以外, 在深度学习中还发展了一些特有的方法.

早停法 早停法 **(Early Stopping)** 是一种简单有效的防止过拟合的策略. 其核心思想是监控模型在验证集上的性能, 提前终止训练过程, 避免出现对训练集的过拟合.

在训练的初期, 训练集和验证集的误差都随着训练进行而下降, 表明模型正在学习数据的普遍规律; 当训练到一定程度时, 可能出现训练集的误差逐渐下降, 但验证集的误差停止下降甚至上升的现象, 这表明模型开始对训练集产生过拟合, 此时就应当停止训练了.

丢弃法 丢弃法 **(Dropout)** 也是一种常见的防止过拟合的策略. 其核心思想是通过随机屏蔽部分神经元, 减少模型对特定神经元的依赖, 增强其泛化能力. 从组合模型的角度考虑, 丢弃法提供了一种简单易行的 Bagging 方法.

实际操作中, 只需在每轮训练时按照对于神经网络的某一层按照一定概率随机地将部分神经元的输出设为 0, 达到丢弃的效果, 使得每次训练的网络结构都不同. 通常在输入层或隐藏层进行如此处理, 输出层则一般不使用, 避免破坏最终预测结果.

15.3 生成对抗模型

15.3.1 生成对抗模型的基本原理

利用概率分布讨论体系性质是机器学习的一种重要思路. 在分类任务中, 如果能从已知数据集中推断出每个类别 $\mathcal{C}$ 的分布曲线 $p(\boldsymbol{x},\mathcal{C})$, 就可以据此进行预测, 也可以据此用来产生类别 $\mathcal{C}$ 的新样本. 后者被称作样本生成任务, 属于无监督学习的范畴.

概率函数 $p(\boldsymbol{x},\mathcal{C})$ 和 $p(\boldsymbol{x})$ 可以用马尔可夫网络或神经网络描述, 但因为归一化因子 (换言之, 通常的神经网络并不是归一化的) 的存在而不容易训练. **生成对抗网络 (Generative Adversarial Nets, GAN)** 提供了在不需要显式计算 $p(\boldsymbol{x})$ 的情况下产生新样本的有效方法.

GAN 由两个网络组成: 生成器网络 $\boldsymbol{x}=G(z)$ 和判别器网络 $y=D(\boldsymbol{x})$. D 用于判断一个输入 $\boldsymbol{x}$ 是来源于原有数据集还是由 G 生成的, 而 G 用于生成尽量使 D 判断错误的 $\boldsymbol{x}$. 两者对抗优化, 最后收敛于稳定的解. 在训练时, G 不接触训练集数据, 只接受来自 D 的反馈.

GAN 的误差函数为

$$V(D,G)=\mathbb{E}_{\boldsymbol{x}\sim p_{\mathcal{D}}}[\log D(\boldsymbol{x})]+\mathbb{E}_z[\log(1-D(G(z)))]$$

其中 z 是生成器 G 的输入, 一般服从简单的分布. D 的目标是使得上式尽可能大, 而 G 的目标是使得上式尽可能小, 即

$$\min_G\max_D V(D,G)$$

数学上可以证明, 当极值收敛是, $\boldsymbol{x}=G(z)$ 的分布将与原有数据集的概率分布 $p(\boldsymbol{x})$ 一致, 且 $D(\boldsymbol{x})=\dfrac{1}{2}$.

15.4 扩散模型

在 GAN 之后, 另一类深度学习生成模型快速崛起, 即**扩散模型 (Diffusion Models)**.

扩散模型采用了非平衡统计的思想, 目的是从一些样本中来学习其分布概率 $p(\boldsymbol{x})$ 并据此产生新的采样. 它的主要想法大致理解如下: 对于某个分布, 可以定义带随机性的变换使得分布随时间发生变化, 在时间较长时变换到某一简单分布.

例如, 对于外力 $\boldsymbol{f}(\boldsymbol{x})$ 下的扩散方程, 粒子流为

$$\boldsymbol{j} = -D\nabla p(\boldsymbol{x}) + \frac{D}{k_{\mathrm{B}}T} p(\boldsymbol{x})\boldsymbol{f}(\boldsymbol{x})$$

其中 D 为扩散系数. 如果将外力替换为

$$\boldsymbol{f}'(\boldsymbol{x}) = -\boldsymbol{f}(\boldsymbol{x}) + 2k_{\mathrm{B}}T\nabla \ln p(\boldsymbol{x})$$

则粒子流将分享, 末态会随时间演化成初态.

类似地, 扩散模型的工作过程分为两个阶段:

i. 前向扩散: 将 $p_0(\boldsymbol{x})$ 用训练数据集 $\{\boldsymbol{x}_n\}$ 所组成的 δ 函数之和来近似:

$$p_0(\boldsymbol{x}) = \frac{1}{N}\sum_n \delta(\boldsymbol{x} - \boldsymbol{x}_n)$$

在扩散过程中, δ 函数有简单的解析解 $\mathcal{N}(\boldsymbol{x}|\boldsymbol{x}_n, \sigma^2)$(其中 $\sigma^2 = 2Dt$ 是扩散所导致的方差, D 是 $\boldsymbol{x}$ 的维度), 其对逆过程外力的贡献也可以解析地给出. 因此, 我们可以通过对 $\{\boldsymbol{x}_n\}$ 采样, 即从真实数据 $\boldsymbol{x}_n$ 开始逐步添加高斯噪声模拟扩散过程, 计算其在任一时刻对外力的贡献, 并训练一个深度学习网络来近似总外力, 即

$$\boldsymbol{f}_{\mathrm{ext}}(\boldsymbol{x}, t) = 2k_{\mathrm{B}}T\nabla \ln \frac{1}{N}\sum_n \mathcal{N}(\boldsymbol{x}|\boldsymbol{x}_n, \sigma_t^2) = -\frac{2k_{\mathrm{B}}T}{\sigma_t^2}\frac{\sum_n (\boldsymbol{x} - \boldsymbol{x}_n)\mathcal{N}(\boldsymbol{x}|\boldsymbol{x}_n, \sigma_t^2)}{\sum_n \mathcal{N}(\boldsymbol{x}|\boldsymbol{x}_n, \sigma_t^2)}$$

经过足够长时间 ($t \gg 1$) 的扩散后, $p_t(\boldsymbol{x})$ 将变为完全随机的高斯分布 $\mathcal{N}(\boldsymbol{x}|\boldsymbol{0}, \sigma_T^2)$.

ii. 反向扩散: 利用前一阶段得到的外力 $\boldsymbol{f}_{\mathrm{ext}}(\boldsymbol{x}, t)$ 从随机的高斯分布 $p_T(\boldsymbol{x})$ 中选取一个初值 $\boldsymbol{x}(T)$, 利用外力下的扩散公式还原至初态, 生成与真实数据分布一致的样本.

扩散模型的典型例子就是图像生成与编辑.