

西瓜书笔记

黎雷蕾

2018 年 1 月 25 日

摘要

学医三年，自谓天下无不治之症；
行医三年，始信世间无可用之方。——孙思邈

纸上得来终觉浅，绝知此事要躬行。——陆游

绳锯木断，水滴石穿。——罗大经

目录

1 代数基础	8
1.1 损失函数	8
1.1.1 0-1 损失 (Zero-one Loss)	8
1.1.2 感知损失 (Perceptron Loss)	8
1.1.3 Hinge Loss	8
1.1.4 绝对值误差	9
1.1.5 均方误差	9
1.1.6 交叉熵	9
1.1.7 指数误差 (Exponential)	9
1.2 数值优化算法	9
1.2.1 牛顿法 (Newton's method)	9
1.2.2 拟牛顿法 (Quasi-Newton Methods)	10
1.2.3 梯度下降 (Gradient descent)	13
1.2.4 Momentum	15
1.2.5 Nesterov	15
1.2.6 Adagrad	16
1.2.7 Adadelta	16
1.2.8 RMSprop	17
1.2.9 Adam	17
1.2.10 Adamax	17
1.2.11 Nadam	18
1.3 拉格朗日乘子法	18

1.4	最小二乘法	19
2	线性模型 (linear model)	20
2.1	基本形式	20
2.2	线性回归 (linear regression)	20
2.3	多元线性回归 (multivariate linear regression)	22
2.4	对数线性回归 (log-linear regression)	23
2.5	对数几率回归/逻辑回归 (logistic regression)	23
2.6	线性判别分析 (linear discriminant analysis, LDA)	25
2.7	多元 LDA	26
2.8	多分类学习	27
2.9	类别不平衡问题 (class imbalance)	27
2.10	小结	28
3	决策树 (decision tree)	29
3.1	基本流程	29
3.2	划分选择	29
3.2.1	信息增益 (information gain)	29
3.2.2	信息增益率 (information gain ratio)	30
3.3	基尼指数 (Gini index)	30
3.4	剪枝处理 (pruning)	31
3.4.1	预剪枝 (prepruning)	31
3.4.2	后剪枝 (postpruning)	31
3.5	连续之和缺失值	32
3.5.1	连续值	32
3.5.2	缺失值	32
3.6	多变量决策树 (multivariate decision tree)	33
3.7	随机森林 (Random Forest)	33
3.8	迭代决策树 (Gradient Boost Decision Tree, GBDT)	35
3.9	小结	35

4 神经网络 (neural networks)	36
4.1 神经元模型	36
4.2 感知机 (Perceptron) 和多层网络	36
4.3 误差逆传播算法 (BackPropagation, BP)	37
4.4 全局最小与局部极小	40
4.5 其他常见神经网络	41
4.5.1 径向基函数网络, RBF	41
4.5.2 ART 网络	41
4.5.3 SOM 网络	41
4.5.4 级联相关网络	41
4.5.5 Elman 网络	42
4.5.6 Boltzmann 机	42
4.6 深度学习	42
4.7 小结	42
5 支持向量机 (Support Vector Machine,SVM)	43
5.1 间隔与支持向量	43
5.2 对偶问题	44
5.3 核函数	46
5.4 软间隔与正则化	48
5.5 支持向量回归 (Support Vector Regression,SVR)	50
5.6 核方法	52
5.7 小结	54
6 贝叶斯分类器 (bayes classifier)	55
6.1 贝叶斯公式	55
6.2 贝叶斯决策论 (bayesian decision theory)	55
6.3 极大似然估计 (Maximum Likelihood Estimation,MLE)	56
6.4 朴素贝叶斯分类器 (Naïve Bayes Classifier)	56
6.5 半朴素贝叶斯分类器 (semi-Naïve Bayes Classifier)	57
6.6 贝叶斯网 (Bayesian network)	57

6.6.1	贝叶斯网 (Bayesian network)-学习	58
6.6.2	贝叶斯网 (Bayesian network)-推断	59
6.7	EM(Expectation-Maximization) 算法	59
6.8	小结	60
7	集成学习 (ensemble learning)	61
7.1	个体与集成	61
7.2	Boosting	62
7.3	Bagging 与随机森林	65
7.3.1	Bagging	65
7.3.2	随机森林 (Random Forest,RF)	66
7.4	结合策略	67
7.4.1	平均法 (averaging)	67
7.4.2	投票法 (voting)	68
7.4.3	学习法	68
7.5	多样性	69
7.5.1	误差-分歧分解	69
7.5.2	多样性的度量 (diversity measure)	70
7.5.3	多样性增强	71
7.6	小结	72
8	聚类 (clustering)	73
8.1	聚类任务	73
8.2	性能度量	73
8.3	距离计算	75
8.4	原型聚类	76
8.4.1	k 均值算法 (k-means)	77
8.4.2	学习向量量化	78
8.4.3	高斯混合聚类	79
8.5	密度聚类 (density-based clustering)	82
8.6	层次聚类 (hierarchical clustering)	84

8.7 小结	86
9 降维与度量学习	87
9.1 k 近邻学习	87
9.2 低维嵌入	87
9.3 主成分分析 (Principal Component Analusis,PCA)	89
9.4 核化线性降维	91
9.5 流形学习 (manifold learning)	92
9.5.1 等度量映射 (Isometric Mapping,Isomap)	92
9.5.2 局部线性嵌入 (Locally Linear Embedding,LLE)	93
9.6 度量学习	94
9.7 小结	95
10 计算学习理论 (Computational Learning Theory)	96
10.1 基础知识	96
10.2 PAC 学习	97
10.3 有限假设空间	99
10.3.1 可分情形	99
10.3.2 不可分情形	100
10.4 VC 维	101
10.5 Rademacher 复杂度	102
10.6 稳定性	104
10.7 小结	105
11 特征选择与稀疏学习	106
11.1 子集搜索与评价	106
11.2 过滤式选择	107
11.3 包裹式选择	108
11.4 嵌入式选择与 L_1 正则化	110
11.5 稀疏表示与字典学习	112
11.6 压缩感知 (compressed sensing)	113
11.7 小结	115

12 半监督学习 (semi-supervised learning)	116
12.1 未标记样本	116
12.2 生成式方法 (generative methods)	117
12.3 半监督 SVM	118
12.4 图半监督学习	119
12.5 基于分歧的方法 (disagreement-based methods)	122
12.6 半监督聚类 (semi-supervised clustering)	125
12.7 小结	126
13 概率图模型 (probabilistic graphical model)	127
13.1 隐马尔可夫模型	127
13.2 马尔可夫随机场 (Markov Random Field)	129
13.3 条件随机场 (Conditional Random Field,CRF)	131
13.4 学习与推断	131
13.4.1 变量消去	132
13.4.2 信念传播 (Belief Propagation)	132
13.5 近似判断	133
13.5.1 MCMC 采样	133
13.5.2 变分推断	135
13.6 话题模型 (topic model)	137
13.7 小结	138
14 规则学习	139
14.1 基本概念	139
14.2 序贯覆盖	139
14.3 剪枝优化	139
14.4 一阶规则学习	140
14.5 归纳逻辑程序设计	140
14.5.1 最小一般泛化	140
14.6 小结	140

15 强化学习 (Reinforcement Learning)	141
15.1 任务与奖赏	141
15.2 K-摇臂赌博机	141
15.2.1 探索与利用	141
15.2.2 ϵ -贪心	142
15.2.3 softmax	143
15.3 有模型学习	144
15.3.1 策略评估	144
15.3.2 策略改进	146
15.3.3 策略迭代与值迭代	147
15.4 免模型学习	148
15.4.1 蒙特卡洛强化学习	148
15.4.2 时序差分学习	152
15.5 值函数近似	154
15.6 模仿学习	155
15.6.1 直接模仿学习	155
15.6.2 逆强化学习	155
15.7 小结	156

Chapter 1

代数基础

1.1 损失函数

1.1.1 0-1 损失 (Zero-one Loss)

最简单的损失函数，如果预测值 \hat{y}_i 与目标值 y_i 不相等，那么为 1，否则为 0.

$$\ell(y_i, \hat{y}_i) = \begin{cases} 1, & y_i \neq \hat{y}_i; \\ 0, & y_i = \hat{y}_i. \end{cases} \quad (1.1)$$

1.1.2 感知损失 (Perceptron Loss)

用来改进 0-1 损失中判定较为严格的问题：

$$\ell(y_i, \hat{y}_i) = \begin{cases} 1, & |y_i - \hat{y}_i| > t \\ 0, & |y_i - \hat{y}_i| \leq t. \end{cases} \quad (1.2)$$

1.1.3 Hinge Loss

Hinge Loss 可以用来解决 SVM 只能够的间隔最大化问题。

$$\begin{aligned} \ell(y_i, \hat{y}_i) &= \max\{0, 1 - y_i \cdot \hat{y}_i\}, \\ y_i \in \{-1, +1\}, \quad \hat{y}_i &\in [-1, +1]. \end{aligned} \quad (1.3)$$

1.1.4 绝对值误差

常用回归中:

$$\ell(y_i, \hat{y}_i) = |y_i - \hat{y}_i| \quad (1.4)$$

1.1.5 均方误差

常用于回归中:

$$\ell(y_i, \hat{y}_i) = \frac{1}{n} \sum_{i=0}^n (y_i - \hat{y}_i)^2, \quad (1.5)$$

1.1.6 交叉熵

神经网络、逻辑回归中常用的损失函数，二分类问题可写作：

$$\begin{aligned} \ell(y_i, \hat{y}_i) &= y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i) \\ y_i &\in \{0, 1\} \end{aligned} \quad (1.6)$$

多分类问题可写作：

$$\ell(y_i, \hat{y}_i) = - \sum_{i=0}^n y_i \log(\hat{y}_i) \quad (1.7)$$

1.1.7 指数误差 (Exponential)

常用于 boosting 算法：

$$\ell(y_i, \hat{y}_i) = \exp(-y_i \cdot \hat{y}_i) \quad (1.8)$$

1.2 数值优化算法

1.2.1 牛顿法 (Newton's method)

牛顿法是一种在实数域和复数域上近似求解方程的方法。方法使用函数 $f(x)$ 的泰勒级数的前面几项来寻找方程 $f(x) = 0$ 的根。也被称为切线法。

将 $f(x) = 0$ 在 x_0 处展开成泰勒级数:

$$f(x_0) \rightarrow \sum_{n=0}^{\infty} \frac{f^{(n)}(x_0)}{n!} (x - x_0)^n \quad (1.9)$$

我们只取其线性部分, 作为非线性方程的近似方程:

$$f(x_0) + (x - x_0)f'(x_0) = 0 \quad (1.10)$$

设 $f'(x_0) \neq 0$, 则其解为:

$$x = x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} \quad (1.11)$$

这个公式说明 $f(x_1)$ 的值将会比 $f(x_0)$ 更加接近 $f(x) = 0$, 我们就可以用迭代法逼近:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \quad (1.12)$$

最好是用二阶导数:

$$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)} \quad (1.13)$$

通过迭代, 上式必然会在 $f(x) = 0$ 处收敛。

1.2.2 拟牛顿法 (Quasi-Newton Methods)

拟牛顿法 (Quasi-Newton Methods) 是求解非线性优化问题最有效的方法之一。

海森矩阵 (Hessian Matrix)

海森矩阵是一个多元函数的二阶偏导数构成的方阵, 描述了函数的局部曲率。假设二元函数 $f(x_1, x_2)$ 在 $\mathbf{X}^{(0)}(x_1^{(0)}, x_2^{(0)})$ 点处的泰勒展开式为:

$$\begin{aligned} f(x_1, x_2) = & f(x_1^{(0)}, x_2^{(0)}) + \left. \frac{\partial f}{\partial x_1} \right|_{\mathbf{X}^{(0)}} \Delta x_1 + \left. \frac{\partial f}{\partial x_2} \right|_{\mathbf{X}^{(0)}} \Delta x_2 + \\ & \frac{1}{2} \left[\left. \frac{\partial^2 f}{\partial x_1^2} \right|_{\mathbf{X}^{(0)}} \Delta x_1^2 + 2 \left. \frac{\partial^2 f}{\partial x_1 \partial x_2} \right|_{\mathbf{X}^{(0)}} \Delta x_1 \Delta x_2 + \left. \frac{\partial^2 f}{\partial x_2^2} \right|_{\mathbf{X}^{(0)}} \Delta x_2^2 \right] + \dots \end{aligned} \quad (1.14)$$

其中, $\Delta x_1 = x_1 - x_1^{(0)}$, $\Delta x_2 = x_2 - x_2^{(0)}$.

将上式写成矩阵形式:

$$\begin{aligned} f(\mathbf{X}) &= \\ f(\mathbf{X}^{(0)}) &+ \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2} \right)_{\mathbf{X}^{(0)}} \begin{pmatrix} \Delta x_1 \\ \Delta x_2 \end{pmatrix} + \\ \frac{1}{2} (\Delta x_1, \Delta x_2) &\left. \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} \end{pmatrix} \right|_{\mathbf{X}^{(0)}} \begin{pmatrix} \Delta x_1 \\ \Delta x_2 \end{pmatrix} + \dots \end{aligned} \quad (1.15)$$

即:

$$f(\mathbf{X}) = f(\mathbf{X}^{(0)}) + \nabla f(\mathbf{X}^{(0)})^T \Delta \mathbf{X} + \frac{1}{2} \Delta \mathbf{X}^T H(\mathbf{X}^{(0)}) \Delta \mathbf{X} + \dots \quad (1.16)$$

其中:

$$H(\mathbf{X}^{(0)}) = \left. \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} \end{pmatrix} \right|_{\mathbf{X}^{(0)}}, \quad \Delta \mathbf{X} = \begin{pmatrix} \Delta x_1 \\ \Delta x_2 \end{pmatrix} \quad (1.17)$$

$H(\mathbf{X}^{(0)})$ 是 $f(x_1, x_2)$ 在 $\mathbf{X}^{(0)}$ 处的海森矩阵, 由 $f(x_1, x_2)$ 在 $\mathbf{X}^{(0)}$ 处的二阶偏导数组成。

将海森矩阵扩展到 n 元函数, 对应的梯度 $\nabla f(\mathbf{X}^{(0)})$ 和海森矩阵可以写作 $H(\mathbf{X}^{(0)})$:

$$\nabla f(\mathbf{X}^{(0)}) = \left[\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots, \frac{\partial f}{\partial x_n} \right]_{\mathbf{X}^{(0)}}^T \quad (1.18)$$

$$H(\mathbf{X}^{(0)}) = \left[\begin{array}{cccc} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \dots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \dots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \dots & \frac{\partial^2 f}{\partial x_n^2} \end{array} \right]_{\mathbf{X}^{(0)}} \quad (1.19)$$

拟牛顿法思想

在牛顿法的迭代中, 需要计算海森矩阵的逆矩阵 H^{-1} , 这个计算十分复杂, 所以考虑到用一个 n 阶矩阵 $G_k = G(x^{(k)})$ 来近似代替 $H_k^{-1} = H^{-1}(x^{(k)})$, 这就是拟牛顿法的基本思想了。

假设 $g_k = g(x^{(k)}) = \nabla f(x^k)$ 是 $f(x)$ 的梯度向量在点 $x^{(k)}$ 的值。那么牛顿法的更新公式就可以写作：

$$x^{(k+1)} = x^{(k)} - H_k^{-1} g_k \quad (1.20)$$

拟牛顿法的公式可以写作：

$$x^{(k+1)} = x^{(k)} - G_k g_k \quad (1.21)$$

Davidon Fletcher Powell, DFP 算法

DFP 选择 G_{k+1} 的方法是假设每一步迭代中矩阵 G_{k+1} 是由 G_k 加上两个附加项构成的：

$$G_{k+1} = G_k + P_k + Q_k \quad (1.22)$$

设 $y_k = g_{k+1} - g_k$, $\delta_k = x^{(k+1)} - x^{(k)}$, 那么可以求出 P_k, Q_k :

$$\begin{aligned} P_k &= \frac{\delta_k \delta_k^T}{\delta_k^T y_k}, \\ Q_k &= -\frac{G_k y_k y_k^T G_k^T}{y_k^T G_k y_k} \end{aligned} \quad (1.23)$$

那么 DFP 算法中 G_{k+1} 的迭代公式可以写作：

$$G_{k+1} = G_k + \frac{\delta_k \delta_k^T}{\delta_k^T y_k} - \frac{G_k y_k y_k^T G_k^T}{y_k^T G_k y_k} \quad (1.24)$$

那么优化迭代公式可以写作：

$$\begin{aligned} x^{(k+2)} &= x^{(k+1)} - G_{k+1} g_{k+1} \\ &= x^{(k+1)} - \left(G_k + \frac{\delta_k \delta_k^T}{\delta_k^T y_k} - \frac{G_k y_k y_k^T G_k^T}{y_k^T G_k y_k} \right) g_{k+1} \end{aligned} \quad (1.25)$$

Broyden Fletcher Goldfarb Shanno, BFGS 算法

与 DFP 算法不同的是，BFGS 采用一个容易求解逆矩阵的 B_k 去逼近海森矩阵本身 H , 而不是海森矩阵逆矩阵 H^{-1} 。

那么 BFGS 的迭代公式可以写为:

$$\begin{aligned} x^{(k+2)} &= x^{(k+1)} - B_{k+1}^{-1} g_{k+1} \\ &= x^{(k+1)} - \left(B_k + \frac{y_k y_k^T}{y_k^T \delta_k} - \frac{B_k \delta_k \delta_k^T B_k^T}{\delta_k^T B_k \delta_k} \right)^{-1} g_{k+1} \end{aligned} \quad (1.26)$$

1.2.3 梯度下降 (Gradient descent)

梯度

所谓梯度，就是指函数变化最快的地方。对于一个函数 $f(x, y)$ ，分别对于 x, y 求偏导，获得的梯度向量为 $(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y})^T$ ，简称为 $\text{grad } f(x, y)$ 或者 $\nabla f(x, y)$ 。梯度方向指的是沿着梯度向量 $\nabla f(x, y)$ 的方向。

梯度下降和梯度上升

两者的实质是一样的，梯度下降取相反数就是梯度上升了。

梯度下降的代数方法表示

假设一个线性回归函数的表示公式为:

$$h_\theta(x) = \sum_{i=0}^n \theta_i x_i \quad (1.27)$$

损失函数取均方误差:

$$J(\theta) = \frac{1}{n} \sum_{i=0}^n (h_\theta(x) - y_i)^2 \quad (1.28)$$

求出 $J(\theta)$ 的梯度:

$$\frac{\partial J(\theta)}{\partial \theta_i} = \frac{2}{n} \left(\sum_{i=0}^n \frac{\partial h_\theta(x)}{\partial \theta_i} - y_i \right) x_i^{(j)} \quad (1.29)$$

那么更新的梯度表达式可以写作:

$$\theta'_i = \theta_i - \alpha \frac{\partial J(\theta)}{\partial \theta_i} \quad (1.30)$$

其中 $\alpha \in [0, 1]$ 称为学习步长，取值过大造成震荡，太小会造成收敛过慢。

梯度下降的矩阵方法表示

假设一个线性回归函数的矩阵表示公式为：

$$h_{\theta}(\mathbf{x}) = \mathbf{X}\theta \quad (1.31)$$

那么损失函数可以表示为：

$$J(\theta) = \frac{1}{2}(\mathbf{X}\theta - \mathbf{Y})^T(\mathbf{X}\theta - \mathbf{Y}) \quad (1.32)$$

求出 $J(\theta)$ 的梯度：

$$\begin{aligned} \frac{\partial}{\partial \mathbf{X}}(\mathbf{X}\mathbf{X}^T) &= 2\mathbf{X} \\ \frac{\partial}{\partial \theta}(\mathbf{X}\theta) &= \mathbf{X}^T \\ \frac{\partial}{\partial \theta}J(\theta) &= \mathbf{X}^T(\mathbf{X}\theta - \mathbf{Y}) \end{aligned} \quad (1.33)$$

那么更新的梯度表达式可以写作：

$$\theta'_i = \theta_i - \alpha \frac{\partial}{\partial \theta_i} J(\theta) \quad (1.34)$$

梯度下降参数调优

1. 步长 α : 太大会造成震荡从而错过最优解, 太小会造成迭代速度太慢。
2. 参数初始值选择, 梯度下降求出的是局部最优解, 所以参数初始值不同求出的解也会不同, 最好是正态分布随机生成。
3. 归一化, 可以加快迭代速度。

与其它优化算法比较

梯度下降法和最小二乘法相比, 梯度下降法需要选择步长, 而最小二乘法不需要。梯度下降法是迭代求解, 最小二乘法是计算解析解。如果样本量不算很大, 且存在解析解, 最小二乘法比起梯度下降法要有优势, 计算速度很快。但是如果样本量很大, 用最小二乘法由于需要求一个超级大的逆矩

阵，这时就很难或者很慢才能求解解析解了，使用迭代的梯度下降法比较有优势。

梯度下降法和牛顿法/拟牛顿法相比，两者都是迭代求解，不过梯度下降法是梯度求解，而牛顿法/拟牛顿法是用二阶的海森矩阵的逆矩阵或伪逆矩阵求解。相对而言，使用牛顿法/拟牛顿法收敛更快。但是每次迭代的时间比梯度下降法长。

1.2.4 Momentum

Momentum 借鉴了物理学中动量的思想，通过积累之前的动量 m_{t-1} 来加速当前的梯度。设 μ 是动量因子，通常设为 0.9 或其近似值：

$$\begin{aligned} m_t &= \mu \cdot m_{t-1} + \alpha \nabla J(\theta) \\ \theta'_t &= \theta_t - m_t \end{aligned} \tag{1.35}$$

特点：

- 接近局部最优解时， μ 的存在可以使更新幅度变大，从而跳出局部最优；
- 当前后两次梯度方向一致时，可以加速收敛，当前后两次梯度方向不一致时，可以减少震荡。

1.2.5 Nesterov

Nesterov 是基于 Momentum 的算法，可以在梯度更新时对当前梯度进行一个矫正，避免前进太快，同时提高灵敏度：

$$\begin{aligned} m_t &= \mu \cdot m_{t-1} + \alpha \nabla J(\theta - \mu \cdot m_{t-1}) \\ \theta'_t &= \theta_t - m_t \end{aligned} \tag{1.36}$$

1.2.6 Adagrad

Adagrad 对学习速率进行了一个约束。设梯度 $g_t = \nabla_{\theta} J(\theta)$, 那么:

$$\begin{aligned} n_t &= n_{t-1} + (g_t)^2 \\ \theta_t &= \theta_{t-1} - \frac{\eta}{\sqrt{n_t + \epsilon}} \cdot g_t = \theta_{t-1} - \frac{\eta}{\sqrt{\sum_{i=1}^t g_r^2 + \epsilon}} \cdot g_t \end{aligned} \quad (1.37)$$

其中 η 是一个全局学习率, ϵ 是一个常数来保证分母不为 0。

- 优点:
 - 前期 n_t 较小的时候, 梯度的系数较大, 能够放大梯度;
 - 后期 n_t 较大的时候, 梯度的系数较小, 能够缩小梯度;
- 缺点:
 - 中后期梯度系数会逐渐趋于 0, 可以使得训练提前结束, 这可能导致无法取得最优值。

1.2.7 Adadelta

针对 Adagrad 会提前结束的问题, Adadelta 只累计固定大小的项, 并且也不直接存储这些项, 仅仅计算对应的近似平均值。

$$\begin{aligned} g_t &= \nabla_{\theta} J(\theta) \\ n_t &= v \cdot n_{t-1} + (1 - v) \cdot g_t^2 \\ \theta_t &= \theta_{t-1} - \frac{\eta}{\sqrt{n_t + \epsilon}} \cdot g_t \\ E[g^2]_t &= \rho \cdot E[g^2]_{t-1} + (1 - \rho) \cdot g_t^2 \end{aligned} \quad (1.38)$$

综上, 系数项可以写为:

$$-\frac{\sum_{i=1}^{t-1} \Delta \theta_i}{\sqrt{E[g^2]_t + \epsilon}} \quad (1.39)$$

其中 E 代表期望, v, ρ 是常数。

特点:

- 训练初期, 加速效果很好;
- 训练后期, 反复在局部最优解附近震荡, 但是不会停止。

1.2.8 RMSprop

RMSprop 是 Adadelta 的一个特例，即 $\rho = 0.5$ 。

1.2.9 Adam

Adam 是一个非常好用的优化器，基本可以当成很多算法的初始优化器了。Adam 本质是带有动量项的 Adadelta。

$$\begin{aligned} m_t &= \mu \cdot m_{t-1} + (1 - \mu) \cdot g_t \\ n_t &= v \cdot n_{t-1} + (1 - v) \cdot g_t^2 \\ \hat{m}_t &= \frac{m_t}{1 - \mu^t} \\ \hat{n}_t &= \frac{n_t}{1 - v^t} \\ \Delta\theta &= -\frac{\hat{m}_t}{\sqrt{\hat{n}_t} + \epsilon} \cdot \eta \end{aligned} \tag{1.40}$$

其中的参数初始设置： $\mu = 0.9, v = 0.999, \epsilon = 10^{-8}$ 。该算法特点：

- Adma 参数比较平稳。
- 善于处理非平稳目标。
- 学习速率是自动计算出来的，不需要自己设置。
- 适用于大多数非凸优化问题。

1.2.10 Adamax

Adamax 是 Adam 的一种变化，使得 Adma 的学习率边界范围更简单。

$$\begin{aligned} n_t &= \max(v \cdot n_{t-1}, |g_t|) \\ \Delta\theta_t &= -\frac{\hat{m}_t}{n_t + \epsilon} \cdot \eta \end{aligned} \tag{1.41}$$

1.2.11 Nadam

Nadam 类似于带有 Nesterov 动量项的 Adam 算法.

$$\begin{aligned}
 \hat{g}_t &= \frac{g_t}{1 - \prod_{i=1}^t \mu_i} \\
 m_t &= \mu \cdot m_{t-1} + (1 - \mu) \cdot g_t \\
 \hat{m}_t &= \frac{m_t}{1 - \prod_{i=1}^{t+1} \mu_i} \\
 n_t &= v \cdot n_{t-1} + (1 - v) \cdot g_t^2 \\
 \hat{n}_t &= \frac{n_t}{1 - v^t} \hat{m}_t = (1 - \mu_t) \cdot \hat{g}_t + \mu_{t+1} \cdot \hat{m}_t \\
 \Delta\theta_t &= -\frac{\hat{m}_t}{\sqrt{\hat{n}_t} + \epsilon} \cdot \eta
 \end{aligned} \tag{1.42}$$

一般而言，在使用带动量的 RMSprop 或 Adam 问题，使用 Nadam 可以取得更好的效果。

1.3 拉格朗日乘子法

拉格朗日乘子法就是求函数 $f(x_1, x_2, \dots)$ 在约束条件 $g(x_1, x_2, \dots) = 0$ 下的极值的方法。步骤如下：

1. 设需要求极值的目标函数为 $f(x_1, x_2, \dots)$, 限制条件为 $g(x_1, x_2, \dots) = 0$;
2. 引入 n 个拉格朗日参数 $\lambda_i, i = 1, 2, \dots, n$;
3. 建立；拉格朗日函数

$$L(x_1, x_2, \dots, x_n, \lambda_1, \lambda_2, \dots, \lambda_n) = f(x_1, x_2, \dots, x_n) + \sum_{i=1}^n \lambda_i g_i(x_1, x_2, \dots, x_n) \tag{1.43}$$

4. 对每一个参数求偏导，并令其为 0，从而得到极值点：

$$\begin{aligned}\frac{\partial L}{\partial x_i} &= 0 \\ \frac{\partial L}{\partial \lambda_i} &= 0 \\ i &\in \{1, 2, \dots, n\}\end{aligned}\tag{1.44}$$

1.4 最小二乘法

最小二乘估计法，又称最小平方法，是一种数学优化技术。它通过最小化误差的平方和寻找数据的最佳函数匹配。利用最小二乘估计法可以简便地求得未知的数据，并使得这些求得的数据与实际数据之间误差的平方和为最小。常用于线性回归模型。

假设要回归的线性方程为 $y = \beta_0 + \beta_1 x$ ，它的损失函数也是十分简单，就是假设 y_i 是真实值， $\hat{y}_i = \beta_0 + \beta_1 x$ 是估计值。那么损失函数可以写为：

$$L = \sum_i^n (y_i - \hat{y}_i)^2 = \sum_i^n (y_i - \beta_0 - \beta_1 x_i)^2\tag{1.45}$$

要令 L 取最小值，分别对所有系数 β_0, β_1 进行求导：

$$\begin{aligned}\frac{\partial L}{\partial \beta_0} &= 2 \sum_i^n (y_i - \beta_0 - \beta_1 x_i)(-1) = 0 \\ \frac{\partial L}{\partial \beta_1} &= 2 \sum_i^n (y_i - \beta_0 - \beta_1 x_i)(-x_i) = 0\end{aligned}\tag{1.46}$$

经过整理我们就可以求出 β_0, β_1 ：

$$\begin{aligned}\beta_0 &= \frac{\sum_i^n x_i^2 \sum_i^n y_i - \sum_i^n x_i \sum_i^n x_i y_i}{n \sum_i^n x_i^2 - (\sum_i^n x_i)^2} \\ \beta_1 &= \frac{n \sum_i^n x_i y_i - \sum_i^n x_i \sum_i^n y_i}{n \sum_i^n x_i^2 - (\sum_i^n x_i)^2}\end{aligned}\tag{1.47}$$

有了 β_0, β_1 两个参数，就可以回归线性方程了。

Chapter 2

线性模型 (linear model)

2.1 基本形式

给定一个由 d 个属性描述的样本 $x = (x_1; x_2; \dots; x_i)$, 其中 x_i 表示 x 在第 i 个属性上的取值, 那么线性模型可以表示为:

$$f(x) = w_1x_1 + w_2x_2 + \dots + w_dx_d + b \quad (2.1)$$

用向量形式来表示:

$$f(x) = w^T x + b \quad (2.2)$$

其中 $w = (w_1; w_2; \dots; w_d)$, 一旦确定 w, b 模型就可以得到确定。

2.2 线性回归 (linear regression)

线性回归试图学习:

$$f(x_i) = w x_i + b, \text{使得 } f(x_i) \simeq y_i \quad (2.3)$$

为了得到最好的 w^*, b^* , 我们可以采用均方误差 (欧氏距离) 最小化的

方法：

$$\begin{aligned}
 (w^*, b^*) &= \arg \min_{(w,b)} \sum_{i=1}^m (f(x_i) - y_i)^2 \\
 &= \arg \min_{(w,b)} \sum_{i=1}^m (y_i - w^T x_i - b)^2 \\
 &= E_{(w,b)}
 \end{aligned} \tag{2.4}$$

求解 2.4 的过程，被称为线性回归模型的最小二乘“参数估计 (parameter estimation)”: 将其分别对 w, b 求偏导：

$$\begin{aligned}
 \frac{\partial E_{w,b}}{\partial w} &= \frac{\partial}{\partial w} \sum_{i=1}^m [(y_i - b - wx_i)]^2 \\
 &= \frac{\partial}{\partial w} \sum_{i=1}^m [(y_i - b)^2 - 2w(y_i - b)x_i + w^2x_i^2] \\
 &= \sum_{i=1}^m [2wx_i^2 - 2(y_i - b)x_i] \\
 &= 2 \left(w \sum_{i=1}^m x_i^2 - \sum_{i=1}^m (y_i - b)x_i \right) \\
 &= 0
 \end{aligned} \tag{2.5}$$

$$\begin{aligned}
 \frac{\partial E_{w,b}}{\partial b} &= \frac{\partial}{\partial b} \sum_{i=1}^m [(y_i - b - wx_i)]^2 \\
 &= \frac{\partial}{\partial b} \sum_{i=1}^m [(y_i - b)^2 - 2w(y_i - b)x_i + w^2x_i^2] \\
 &= \sum_{i=1}^m [-2(y_i - b) + 2wx_i] \\
 &= 2 \left(\sum_{i=1}^m b - \sum_{i=1}^m (y_i - wx_i) \right) \\
 &= 0
 \end{aligned} \tag{2.6}$$

令??中两式为 0, 即可求出 w, b 的最优解的闭式解:

$$\begin{aligned} w &= \frac{\sum_{i=1}^m y_i(x_i - \bar{x})}{\sum_{i=1}^m x_i^2 - \frac{1}{m}(\sum_{i=1}^m x_i)^2} \\ b &= \frac{1}{m} \sum_{i=1}^m (y_i - wx_i) \\ \bar{x} &= \frac{1}{m} \sum_{i=1}^m x_i, \quad \bar{x} \text{ 为 } x \text{ 的均值} \end{aligned} \tag{2.7}$$

2.3 多元线性回归 (multivariate linear regression)

上一节中的 x 仅由单个属性描述, 若其由 d 个属性进行了描述, 就可以拓展为多元线性回归。

将 w, b 写成向量形式 $\hat{w} = (w; b)$, 同时把数据集表示成一个 $m \times (d+1)$ 大小的矩阵 X (m 代表样本个数, d 代表样本对应的属性个数), X 中的元素 x_{ij} 代表第 i 个样本的第 j 个属性, 最后一列恒为 1:

$$X = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1d} & 1 \\ x_{21} & x_{22} & \cdots & x_{2d} & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{md} & 1 \end{pmatrix} = \begin{pmatrix} x_1^T & 1 \\ x_2^T & 1 \\ \vdots & \vdots \\ x_m^T & 1 \end{pmatrix} \tag{2.8}$$

与 2.4类似

$$E_{\hat{w}} = \hat{w}^* = \arg \min_{\hat{w}} (y - X\hat{w})^T (y - X\hat{w}) \tag{2.9}$$

$$\begin{aligned} \frac{\partial E_{\hat{w}}}{\partial \hat{w}} &= \frac{\partial}{\partial \hat{w}} (y^2 - 2\mathbf{X}^T y \hat{w} + \mathbf{X}^T \mathbf{X} \hat{w}^2) \\ &= -2\mathbf{X}^T y + 2\mathbf{X}^T \mathbf{X} \hat{w} \\ &= 2\mathbf{X}^T (\mathbf{X} \hat{w} - y) \\ &= 0 \end{aligned} \tag{2.10}$$

若 $\mathbf{X}^T \mathbf{X}$ 为满秩矩阵或者正定矩阵, 则:

$$\hat{w}^* = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T y \tag{2.11}$$

令 $\hat{x}_i = (x_i, 1)$, 则最终学得的多元回归模型:

$$f(\hat{x}_i) = \hat{x}_i^T (X^T X)^{-1} X^T y \quad (2.12)$$

2.4 对数线性回归 (log-linear regression)

一般来说我们得到的线性回归模型可以简写为:

$$y = w^T x + b \quad (2.13)$$

我们把 y 取对数, 那它的本质是试图让 $e^{w^T x + b}$ 逼近 y , 即:

$$\ln y = w^T x + b \quad (2.14)$$

一般地, 考虑单调可微的函数 $g(\cdot)$, 令:

$$y = g^{-1}(w^T x + b) \quad (2.15)$$

这个模型就被称为广义上的线性模型。

2.5 对数几率回归/逻辑回归 (logistic regression)

首先介绍 *Sigmoid* 函数:

$$y = \frac{1}{1 + e^{-z}} \quad (2.16)$$

它将 z 值转化为一个接近 0 或者 1 的 y 值, 并且在 $z = 0$ 附近变化很陡, 带入上节的对数几率函数:

$$y = \frac{1}{1 + e^{-(w^T x + b)}} \quad (2.17)$$

若将 y 视为样本 x 作为正例的可能性, 那么 $1 - y$ 为其反例的可能性,

两者比值取对数:

$$\begin{aligned}
 y &= \frac{1}{1 + e^{-(w^T x + b)}} \\
 \Rightarrow y + ye^{-(w^T x + b)} &= 1 \\
 \Rightarrow e^{-(w^T x + b)} &= \frac{1 - y}{y}
 \end{aligned} \tag{2.18}$$

取对数:

$$\begin{aligned}
 \Rightarrow -(w^T x + b) &= \ln \frac{1 - y}{y} \\
 \Rightarrow \ln \frac{y}{1 - y} &= w^T x + b
 \end{aligned}$$

这个比值称为对数几率 (log odds, 也叫 logit)。

若将 y 视为后验概率 $p(y = 1|x)$, 则:

$$\begin{aligned}
 p(y = 1|x) &= \frac{e^{w^T x + b}}{1 + e^{w^T x + b}} \\
 p(y = 0|x) &= \frac{1}{1 + e^{w^T x + b}}
 \end{aligned} \tag{2.19}$$

我们可以采用极大似然估计法 (maximum likelihood method) 来估计 w 和 b , 那么上述的对数似然可以写为:

$$\ell(w, b) = \sum_{i=1}^m \ln p(y_i|x_i; w, b) \tag{2.20}$$

令 $\beta = (w; b)$, $\hat{x} = (x; 1)$, 那么 $w^T x + b = \beta^T \hat{x}$ 。

令 $p_1(\hat{x}; \beta) = p(y = 1|\hat{x}; \beta)$, 那么 $p_0(\hat{x}; \beta) = p(y = 0|\hat{x}; \beta) = 1 - p_1(\hat{x}; \beta)$

那么 2.20 可重写为:

$$p(y_i|x_i; w, b) = y_i p_1(\hat{x}_i; \beta) + (1 - y_i) p_0(\hat{x}_i; \beta) \tag{2.21}$$

由上述几个公式, 重写 2.20:

$$\ell(\beta) = \sum_{i=1}^m (-y_i \beta^T \hat{x}_i + \ln(1 + e^{\beta^T \hat{x}_i})) \tag{2.22}$$

2.22是高阶可导的连续凸函数，根据凸优化理论，梯度下降法或者牛顿迭代法均可以求出最优解。

$$\beta^* = \arg \min_{\beta} \ell(\beta) \quad (2.23)$$

采用牛顿法，即：

$$\beta^{t+1} = \beta^t - \frac{\ell'(\beta)}{\ell''(\beta)} \quad (2.24)$$

2.6 线性判别分析 (linear discriminant analysis, LDA)

LDA 思想特别朴素：给定训练样例集，设法将样例投影到一条直线上，使得同类样例的投影点尽可能的近，异类样例的投影点尽可能远。即不同分类的样例在直线上是聚集在一起的，像一个个部落一样。

设 μ_0, μ_1 分别是两个分类的样本中心点，那么他们在直线上的投影分别是 $w^T \mu_0, w^T \mu_1$ 。若将所有样本点都投影到直线上，这两类样本的协方差分别为 $w^T \sum_0 w, w^T \sum_1 w$ 。

- 欲使同类样例投影点尽可能接近，可以让协方差尽可能小： $w^T \sum_0 w + w^T \sum_1 w$
- 欲使异类样例的投影点尽可能远离，可以让类中心之间的距离尽可能大： $\|w^T \mu_0 - w^T \mu_1\|_2^2$

综合上面两点，最大化目标 J 可写为：

$$\begin{aligned} J &= \frac{\|w^T \mu_0 - w^T \mu_1\|_2^2}{w^T \sum_0 w + w^T \sum_1 w} \\ &= \frac{w^T (\mu_0 - \mu_1)(\mu_0 - \mu_1)^T w}{w^T (\sum_0 + \sum_1) w} \end{aligned} \quad (2.25)$$

定义“类内散度矩阵”(within-class scatter matrix)：

$$\begin{aligned} S_w &= \sum_0 + \sum_1 \\ &= \sum_{x \in X_0} (x - \mu_0)(x - \mu_0)^T + \sum_{x \in X_1} (x - \mu_1)(x - \mu_1)^T \end{aligned} \quad (2.26)$$

定义“类间散度矩阵”(between-class scatter matrix):

$$S_b = (\mu_0 - \mu_1)(\mu_0 - \mu_1)^T \quad (2.27)$$

重写 2.25:

$$J = \frac{w^T S_b w}{w^T S_w w} \quad (2.28)$$

由于 2.28 的解与 w 的长度无关, 只与其方向有关, 不是一般性, 令 $w^T S_w w = 1$, 最大化分子 $w^T S_b w$, 一般采用拉格朗日乘子法。可得:

$$f(w) = S_b w - \lambda S_w w = 0 \quad (2.29)$$

其中 $S_b w$ 的方向恒为 $\mu_0 - \mu_1$, 那么我们可以令:

$$S_b w = \lambda(\mu_0 - \mu_1) \quad (2.30)$$

那么我们可得:

$$w = S_w^{-1}(\mu_0 - \mu_1) \quad (2.31)$$

为了求 S_w^{-1} , 通常的做法是对 S_w 进行奇异值分解, $S_w = U \sum V^T$, 得到 $S_w^{-1} = V \sum^{-1} U^T$, 从而进行求解。

结合贝叶斯理论, 当两类数据满足先验概率相同、服从高斯分布且协方差相等, LDA 可以达到最优分类

2.7 多元 LDA

假定存在 N 个类, 且第 i 类示例数为 m_i , 我们可以定义 S_w, S_b 和全局散度矩阵 S_t

$$\begin{aligned} S_w &= \sum_{i=1}^N S_{w_i} = \sum_{x \in X_i} (x - \mu_i)(x - \mu_i)^T \\ S_b &= \sum_{i=1}^N m_i(\mu_i - \mu)(\mu_i - \mu)^T \\ S_t &= S_w + S_b = \sum_{i=1}^m (x_i - \mu)(x_i - \mu)^T \end{aligned} \quad (2.32)$$

其中 μ 是所有示例的均值向量。通常知道上式三者中的两者即可，采用优化目标：

$$\max_W \frac{\text{tr}(W^T S_b W)}{\text{tr}(W^T S_w W)} \quad (2.33)$$

$$S_b W = \lambda S_w W \quad (2.34)$$

故 W 的闭式解是 $S_w^{-1} S_b$ 的 $N - 1$ 个最大广义特征值所对应的特征向量组成的矩阵。由于投影有降维的作用，故 LDA 也被视为一种经典的监督降维技术。

2.8 多分类学习

经典的拆分策略有三个，涉及到编码不详细说明，详情见周志华《机器学习》 $p.63 \sim p.66$:

- 一对一：One vs One, OvO
- 一对其余：One vs Rest, OvR
- 多对多：Many vs Many, MvM

2.9 类别不平衡问题 (class imbalance)

若训练集中正 (m^+) 反 (m^-) 例子数目不均等，那么：

$$\frac{y}{1-y} > \frac{m^+}{m^-}, \text{ 预测为正例} \quad (2.35)$$

故对样本进行再缩放 (rescaling)：

$$\frac{y'}{1+y'} = \frac{y}{1+y} \times \frac{m^-}{m^+} \quad (2.36)$$

总之，处理类别不平衡主要有三种方法：

- 欠采样 (undersampling): 去除一定样本，使得正负样本数目趋于平衡。
- 过采样 (oversampling): 增加一些样本。
- 再缩放策略。

2.10 小结

本章是机器学习理论的基础基础章节，介绍的是最基本的线性模型。

- 线性模型：
 - 形式简单、易于建模，许多功能更加强大的非线性模型大都是在线性模型的基础上通过引入层级结构或者高维映射而得。
 - 线性模型中的 w 具有很好的解释性，便于理解。
- 逻辑回归，又名对数几率回归：
 - 可以直接对分类可能性进行建模，无需事先假设数据的分布，就可以避免假设分布不准确所带来的问题。
 - 它不仅可以进行分类，还可以得到近似的概率预测。
 - 对率函数是任意阶可导的凸函数，具有很好的数学性质许多数值化方法都可以用于求取最优解。
- 线性判别分析 (LDA)：是一种采用投影的策略，被视为一种经典的监督降维技术。

Chapter 3

决策树 (decision tree)

3.1 基本流程

决策树其实是一个递归建树的流程，具体步骤如下：

1. 对于属性集 A ，采用信息增益或者基尼指数等方式确定当做根节点的 a_i
2. 根据 a_i 的取值将样本分成几份 $D = \{D_1^{a_i}, D_2^{a_i}, \dots, D_n^{a_i}\}$
3. 对于每一个样本子集 $D_j^{a_i}$ ，重复上述的 (1), (2) 两步。直到样本子集里的每个样本属于同一类别 C 。

3.2 划分选择

3.2.1 信息增益 (information gain)

首先需要了解信息熵 (information entropy)，信息熵代表了信息不确定的程度：信息熵越大，说明信息越不确定，那么纯度越低；反之，若信息熵很小，说明信息确定程度高，那么信息纯度越高。假设当前样本集合 D 中第 k 类样本所占的比例为 p_k , ($k = 1, 2, \dots, |\mathcal{Y}|$)，那么信息熵可以被定义为：

$$Ent(D) = - \sum_{k=1}^{|\mathcal{Y}|} p_k \log_2 p_k \quad (3.1)$$

由于信息熵取值和纯度大小呈反比，为了便于理解，我们引入“信息增益”，假设一个属性 a 的取值为 $\{a^1, a^2, \dots, a^V\}$ ，那么样本集合 D 可以按照 a 分成 V 份，假设按照 a^v 分的样本子集是 D^v ，那么信息增益 $Gain(D, a)$ 可以写为：

$$Gain(D, a) = Ent(D) - \sum_{v=1}^V \frac{|D_v|}{|D|} Ent(D^v) \quad (3.2)$$

为了得到最大的信息增益，我们可以求出所有属性 a^i 的信息增益 $Gain(D, a^i)$ ，从中选择信息增益最大的 a_* 作为当前的节点，这个思想就是**ID3 决策树学习算法。**

$$a_* = \arg \max_{a \in A} Gain(D, a) \quad (3.3)$$

3.2.2 信息增益率 (information gain ratio)

信息增益准则会对取值数目较多的属性有所偏好，为了减少这种不利影响，我们采用增益率 (gain ratio) 来划分最优属性，这个思想就是**C4.5 决策树算法。**

$$\begin{aligned} Gain_ratio(D, a) &= \frac{Gain(D, a)}{IV(a)} \\ IV(a) &= - \sum_{v=1}^V \frac{|D^v|}{|D|} \log_2 \frac{|D^v|}{|D|} \end{aligned} \quad (3.4)$$

3.3 基尼指数 (Gini index)

CART 决策树采用基尼指数来选择划分属性，数据集 D 的纯度可以用基尼值 $Gini(D)$ 来测量：

$$Gini(D) = \sum_{k=1}^{|\mathcal{Y}|} \sum_{k' \neq k} p_k p_{k'} = 1 - \sum_{k=1}^{|\mathcal{Y}|} p_k^2 \quad (3.5)$$

和信息熵类似，基尼值越小，数据集的纯度越高。基尼指数可以定义为：

$$Gini_index(D, a) = \sum_{v=1}^V \frac{|D^v|}{|D|} Gini(D^v) \quad (3.6)$$

与信息增益相反，我们选择基尼系数最小的属性当做划分属性：

$$a_* = \arg \min_{a \in A} Gini_index(D, a) \quad (3.7)$$

3.4 剪枝处理 (pruning)

剪枝是为了避免决策树算法是否进入‘过拟合’的手段。

3.4.1 预剪枝 (prepruning)

预剪枝是指在决策树生成过程中对当前节点进行估计，若当前节点的划分不能带来决策树泛化性能的提升，则停止划分并将当前节点标记为叶节点。

- 计算不分叶节点之前验证集的精度 p_{pre} 。
- 计算分开的叶节点之后的验证集精度 p_{post}
- 若 $p_{post} > p_{pre}$ 则扩展该节点，否则直接将其作为叶节点。
- 预剪枝可以减少很多不必要的分支，时间开销较小，但是会带来更大的欠拟合风险。

3.4.2 后剪枝 (postpruning)

- 和预剪枝类似，也是采用划分前后的验证集精度来决定是否进行剪枝。
- 不同的是，后剪枝是先分叶节点后再剪枝。
- 相比于预剪枝，后剪枝通常可以保留更加多的叶节点，所以后剪枝的欠拟合风险较小，泛化性能优于预剪枝，但是后剪枝需要在决策树生成后才能进行，训练的开销要大于预剪枝。

3.5 连续之和缺失值

3.5.1 连续值

对于连续值，一般采用连续属性离散化技术，最简单的策略是采用二分法 (bi-partition) 对连续属性进行处理，**C4.5 决策树算法就是采用这种方法。**

给定样本集 D 和连续属性 a ，假定 a 在 D 上出现了 n 个不同的取值，将其按从大到小的顺序进行排列，记为 $\{a_1, a_2, \dots, a_n\}$ ，基于划分点 t 可将 D 分为子集 D_t^- 和 D_t^+ ，其中 $a_i \leq a_t, a_i \in D_t^-$ 且 $a_j > a_t, a_j \in D_t^+$ ，对于相邻的属性 a_t, a_{t+1}, t 在区间 $[a_t, a_{t+1}]$ 上取任意值的划分相同。那么对于一个连续属性 a_t 我们就可以考察包含 $n - 1$ 个袁术的划分点集合，即：

$$T_n = \left\{ \frac{a_i + a_{i+1}}{2} \mid 1 \leq i \leq n - 1 \right\} \quad (3.8)$$

上述公式的意思就是把区间 $[a_t, a_{t+1}]$ 上的中位点 $\frac{a_i + a_{i+1}}{2}$ 作为候选的划分点，从而连续值就变成了离散值。

3.5.2 缺失值

对于缺失值，我们通常是采用给特定的信息增益赋予权值 ρ 的方式。

1. 假设某一个属性 a_i 的集合为 A ，缺失的属性集合为 A^* ，那么在 D 上有， $|D_A| = |A| + |A^*|$ ，其中 $|A|$ 代表集合 A 中的属性个数。
2. 我们把 A 当做一个无缺失值的属性，计算出相应的信息增益 $Gain(A, a_i)$ 。
3. 实际上属性的信息增益：

$$Gain(D_A, a_i) = \rho \times Gain(A, a_i) = \frac{|A|}{|A| + |A^*|} \times Gain(A, a_i) \quad (3.9)$$

4. 若用该属性作为父节点，那么缺失值将同时进入所有的子节点，在每个节点计算信息增益时，它的权重：

$$\rho_{child} = \frac{\text{该子节点不缺失的属性个数}}{\text{父节点不缺失属性的个数}} \quad (3.10)$$

3.6 多变量决策树 (multivariate decision tree)

上述所有的决策树算法的节点都是以单个属性为准，而我们实际的情况下，经常会用到多变量作为决策树的分界点，每个非叶结点都是形如 $\sum_{i=1}^d w_i a_i = t$ 的线性分类器，这个分类器可能会采取 *softmax* 之类的方式进行决策，不太便于解释。

3.7 随机森林 (Random Forest)

虽然拥有剪枝技术，但是决策树还是会存在过拟合的问题，随机森林可以很好地解决这个问题：

随机森林顾名思义，是用随机的方式建立一个森林，森林里面有很多的决策树组成，随机森林的每一棵决策树之间是没有关联的。在得到森林之后，当有一个新的输入样本进入的时候，就让森林中的每一棵决策树分别进行一下判断，看看这个样本应该属于哪一类（对于分类算法），然后看看哪一类被选择最多，就预测这个样本为那一类。

随机森林是一个最近比较火的算法，它有很多的优点：

- 在数据集上表现良好
- 在当前的很多数据集上，相对其他算法有着很大的优势
- 它能够处理很高维度 (feature 很多) 的数据，并且不用做特征选择
- 在训练完后，它能够给出哪些 feature 比较重要
- 在创建随机森林的时候，对误差 (generalization error) 使用的是无偏估计
- 训练速度快
- 在训练过程中，能够检测到 feature 间的互相影响
- 容易做成并行化方法，实现比较简单

每棵树的按照如下规则生成：

1. 如果训练集大小为 N , 对于每棵树而言, 随机且有放回地从训练集中的抽取 N 个训练样本 (这种采样方式称为 bootstrap sample 方法), 作为该树的训练集; 如果不进行随机抽样, 每棵树的训练集都一样, 那么最终训练出的树分类结果也是完全一样的, 这样的话完全没有 bagging 的必要; 我理解的是这样的: 如果不是有放回的抽样, 那么每棵树的训练样本都是不同的, 都是没有交集的, 这样每棵树都是“有偏的”, 都是绝对“片面的” (当然这样说可能不对), 也就是说每棵树训练出来都是有很大的差异的; 而随机森林最后分类取决于多棵树 (弱分类器) 的投票表决, 这种表决应该是“求同”, 因此使用完全不同的训练集来训练每棵树这样对最终分类结果是没有帮助的, 这样无异于是“盲人摸象”。
2. 如果每个样本的特征维度为 M , 指定一个常数 $m \ll M$, 随机地从 M 个特征中选取 m 个特征子集, 每次树进行分裂时, 从这 m 个特征中选择最优的;
3. 每棵树都尽最大程度的生长, 并且没有剪枝过程。

两个随机性的引入对随机森林的分类性能至关重要。由于它们的引入, 使得随机森林不容易陷入过拟合, 并且具有很好得抗噪能力 (比如: 对缺省值不敏感)。

随机森林分类效果 (错误率) 与两个因素有关:

- 森林中任意两棵树的相关性: 相关性越大, 错误率越大;
- 森林中每棵树的分类能力: 每棵树的分类能力越强, 整个森林的错误率越低。
- 减小特征选择个数 m , 树的相关性和分类能力也会相应的降低; 增大 m , 两者也会随之增大。所以关键问题是如何选择最优的 m (或者是范围), 这也是随机森林唯一的一个参数。

3.8 迭代决策树 (Gradient Boost Decision Tree, GBDT)

首先要了解 Boost 算法：原始的 Boost 算法是在算法开始的时候，为每一个样本赋上一个权重值，初始的时候，大家都是一样重要的。在每一步训练中得到的模型，会使得数据点的估计有对有错，我们就在每一步结束后，增加分错的点的权重，减少分对的点的权重，这样使得某些点如果老是被分错，那么就会被“严重关注”，也就被赋上一个很高的权重。然后等进行了 N 次迭代（由用户指定），将会得到 N 个简单的分类器（basic learner），然后我们将它们组合起来（比如说可以对它们进行加权、或者让它们进行投票等），得到一个最终的模型。

而 Gradient Boost 与传统的 Boost 的区别是，每一次的计算是为了减少上一次的残差 (residual)，而为了消除残差，我们可以在残差减少的梯度 (Gradient) 方向上建立一个新的模型。所以说，在 Gradient Boost 中，每个新的模型的简历是为了使得之前模型的残差往梯度方向减少，与传统 Boost 对正确、错误的样本进行加权有着很大的区别。

建立 GBDT 流程：

1. 给定一个初始化的值， $F_0(x)$
2. 迭代 M 次，建立 M 颗决策树
3. 对 $F_i(x)$ 进行逻辑回归。
4. 求得残差减少的梯度方向
5. 类似随机森林方式投票确定当前决策树，与原来的决策树合在一起作为一个新的模型。

3.9 小结

这章主要是介绍了决策树算法，随机森林和 GBDT 会在书上第八章再详细进行说明。

Chapter 4

神经网络 (neural networks)

4.1 神经元模型

设输入的样本属性 X 和对应的连接权重 W , 阈值 θ , 那么神经元可以被表示为:

$$y = f(WX - \theta) = f\left(\sum_{i=1}^n w_i x_i - \theta\right) \quad (4.1)$$

其中 y 表示输出, $f(x)$ 表示激活函数 (activation function)。

4.2 感知机 (Perceptron) 和多层网络

最简单的有两层 (输入层和输出层)。如图所示:

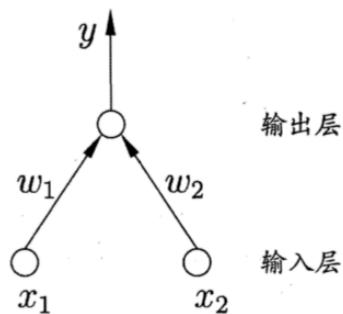


图 5.3 两个输入神经元的感知机网络结构示意图

一般的，给定训练的数据集，权重 $w_i (i = 1, 2, \dots, n)$ 和阈值 θ 可以通过学习得到。对于训练的样例 (x, y) ，若当前感知机的输出为 \hat{y} ，感知机的权重可以调整为：

$$\begin{aligned} w_i &\leftarrow w_i + \Delta w_i, \\ \Delta w_i &= \eta(y - \hat{y})x_i \end{aligned} \quad (4.2)$$

其中 $\eta \in (0, 1)$ 称为学习率 (learning rate)，若感知机预测正确， $\hat{y} = y$ ，否则根据错误程度进行权重调整。

单层感知机只能解决线性可分的问题，若不是线性可分的问题，那么我们得改成多层感知机才行，如输入层与输出层之间加入一层隐藏层。如下图就是典型的多层前馈神经网络 (multi-layer feedforward neural networks)：

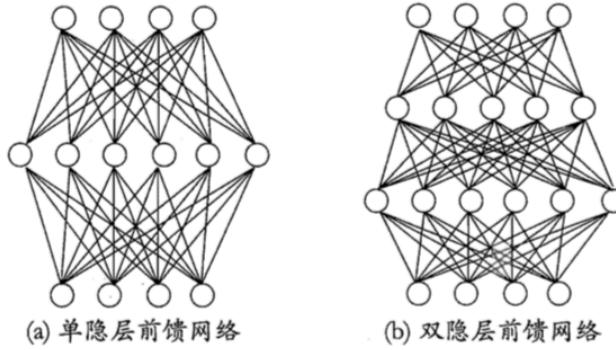


图 5.6 多层前馈神经网络结构示意图

4.3 误差逆传播算法 (BackPropagation, BP)

BP 算法可以说是 NN 算法中最杰出的代表，大部分时间都是用于多层前馈网络。

给定训练集 $D = \{(x_1, y_1), (x_2, y_2), \dots, (x_m, y_m)\}, x_i \in \mathbb{R}^d, y_i \in \mathbb{R}^l$ ，即输入示例由 d 个属性进行描述，输出 l 维向量，即 d 个输入神经元和 l 个输出神经元， q 个隐藏层神经元。其中输出层的第 j 个神经元的阈值为 θ_j ，隐层第 h 个神经元用 γ_h ，输入层第 i 个神经元和隐层的第 h 个神经元连接权

值为 v_{ih} , 隐层第 h 个神经元与输出层第 j 个神经元之间的连接权值为 w_{hj} 。记隐层第 h 个神经元接收到的输入为 $a_h = \sum_{i=1}^d v_{ih}x_i$, 输出层第 j 个神经元接收到的输入为 $\beta_j = \sum_{h=1}^q w_{hj}b_h$, 其中 b_h 为隐层第 h 个神经元的输出, 如下图所示:

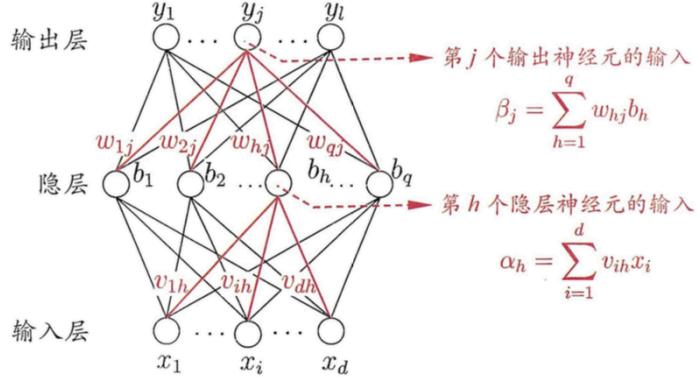


图 5.7 BP 网络及算法中的变量符号

对训练例子 (x_k, y_k) , 假定神经网络的输出为 $\hat{y}_k = (\hat{y}_1^k, \hat{y}_2^k, \dots, \hat{y}_l^k)$, 可以表示为:

$$\hat{y}_j^k = f(\beta_j - \theta_j) \quad (4.3)$$

那么在 (x_k, y_k) 的均方误差为:

$$E_k = \frac{1}{2} \sum_{j=1}^l (\hat{y}_j^k - y_j^k)^2 \quad (4.4)$$

BP 算法实际上是一个迭代学习的过程, 下面以隐藏层到输出层的连接权值 w_{hj} 为例进行推导。由于 BP 算法基于梯度下降 (gradient descent) 策略, 以目标的负梯度方向对参数进行调整, 对上面的均方误差 E_k 及给定的学习率 η , 有:

$$\Delta w_{hj} = -\eta \frac{\partial E_k}{\partial w_{hj}} \quad (4.5)$$

而从图 4.3可以看出 w_{hj} , 首先影响第 j 个输出层神经元的输入值 β_j , 再影响其输出值 \hat{y}_j^k , 最后影响到 E_k , 根据复合微分和马尔科夫过程, 有:

$$\frac{\partial E_k}{\partial w_{hj}} = \frac{\partial E_k}{\partial \hat{y}_j^k} \cdot \frac{\partial \hat{y}_j^k}{\partial \beta_j} \cdot \frac{\partial \beta_j}{\partial w_{hj}} \quad (4.6)$$

而根据 $\beta_j = \sum_{h=1}^q w_{hj} b_h$, 可得:

$$\frac{\partial \beta_j}{\partial w_{hj}} = b_h \quad (4.7)$$

我们选择 *Sigmoid* 函数作为激活函数, 是由于它有一个特别好的性质:

$$f' = f(x)(1 - f(x)) \quad (4.8)$$

根据公式 4.3 和 4.4 可得:

$$g_j = -\frac{\partial E_k}{\partial \hat{y}_j^k} \cdot \frac{\partial \hat{y}_j^k}{\partial \beta_j} = -(\hat{y}_j^k - y_j^k) f'(\beta_j - \theta_j) = \hat{y}_j^k (1 - \hat{y}_j^k) (y_j^k - \hat{y}_j^k) \quad (4.9)$$

这个式子表示 g_j 可以采用实际输出 \hat{y}_j^k 和样本真实输出 y_j^k 进行表示, 那么结合前面的式子 4.5-4.9 :

$$\Delta w_{hj} = \eta g_j b_h \quad (4.10)$$

同理, 我们可以求出其他的参数:

$$\Delta \theta_j = -\eta g_j$$

$$\Delta v_{ih} = \eta e_h x_i$$

$$\Delta \gamma_h = -\eta e_h$$

其中: (4.11)

$$\begin{aligned} e_h &= -\frac{\partial E_k}{\partial b_h} \cdot \frac{\partial b_h}{\partial \alpha_h} = -\sum_{j=1}^l \frac{\partial E_k}{\partial \beta_j} \cdot \frac{\partial \beta_j}{\partial b_h} f'(\alpha_h - \gamma_h) \\ &= b_h (1 - b_h) \sum_{j=1}^l w_{hj} g_j \end{aligned}$$

学习率 $\eta \in (0, 1)$ 控制算法每一轮迭代中的更新步长, 若太大则容易造成震荡, 太小会导致收敛速度太慢。

BP 算法的目标是要最小化训练集 D 上的积累误差:

$$E = \frac{1}{m} \sum_{k=1}^m E_k \quad (4.12)$$

一般来说，只要一个包含足够多神经元的隐层，多层前馈网络就能以任意精度逼近任意复杂度的连续函数，实践中通常采用试错法 (trial-by-error) 进行调整。

为了防止过拟合，通常采用两种策略：

- 早停 (early stopping): 将数据分成训练集和测试集，在更新权值时，若训练集误差降低而测试集误差升高，则停止训练。
- 正则化 (regularization): 在目标函数中增加一个用于描述网络复杂度的部分，如连接层权值与阈值的平方和，用 E_k 表示第 k 个训练样例上的误差， w_i 表示连接权值和阈值的平方和。那么 E 可以改写为：

$$E = \lambda \frac{1}{m} \sum_{k=1}^m E_k + (1 - \lambda) \sum_i w_i^2 \quad (4.13)$$

其中 $\lambda \in (0, 1)$ 表示经验误差与网络复杂度这两项的折中，通常采用交叉验证法进行估计。

4.4 全局最小与局部极小

一般来说，用的最多的参数寻优的方法是梯度搜索。梯度下降法是沿着函数值下降得最快的方向进行搜索，若误差函数在当前值为零了，说明已经到达局部极小，但是不一定到达全局最小。

为了跳出局部极小，通常采用如下策略：

- 以多组不同的参数值初始化神经网络，但是可能会陷入多个不同的局部极小。
- 模拟退火 (simulated annealing)，其策略是在每一步有一定的概率接收比当前解更差的结果，即选择次优解。
- 采用随机梯度算法，这样可以保证即使到局部极小时它的梯度仍不为零，从而跳出局部极小。
- 或者采用遗传算法 (genetic algorithms)。

4.5 其他常见神经网络

4.5.1 径向基函数网络，RBF

RBF 网络是一种单隐层神经网络，采用径向基函数作为隐层神经元的激活函数：

$$\begin{aligned}\varphi(x) &= \sum_{i=1}^q w_i \rho(x, c_i) \\ \rho(x, c_i) &= e^{-\beta_i \|x - c_i\|^2}\end{aligned}\tag{4.14}$$

已经被证明：拥有足够度的隐层神经元的 RBF 网络，能以任意精度逼近任意连续函数。

4.5.2 ART 网络

ART 网络是竞争 (competitive learning) 型网络，是神经网络中常用的无监督学习网络，其原理是所有网络的输出神经元都进行相互的竞争，每一时刻有且仅有一个神经元获胜而被激活，剩下的神经元都处于抑制状态。

ART 可以缓解竞争学习中的‘可塑性-稳定性窘境 (stability-plasticity dilemma)’，可塑性 (plasticity) 指的是神经网络要有学习新知识的能力，稳定性 (stability) 指的是神经网络在学习心得知识时要保持旧的知识，这样 ART 可以进行增量学习 (incremental learning) 或者在线学习 (online learning).

4.5.3 SOM 网络

自组织映射网络 (Self-Organizing Map)，是一种竞争学习型的无监督映射网络，它能够将高维数据映射到低维空间，同时保持输入数据在高维空间的拓扑结构。

4.5.4 级联相关网络

级联相关 (Cascade-Correlation) 网络能够在训练过程中改变自身的网络拓扑结构，与一般的前馈神经网络相比，级联相关网络无需设置网络层数、隐层神经元数目，训练速度较快，但是在数据量较小时容易陷入过拟合。

4.5.5 Elman 网络

递归神经网络 (recurrent neural networks,RNN) 允许网络中出现环形结构，从而让一些神经元的输出反馈来作为输入信号，使得网络在 t 时刻的输出状态不仅与 t 时刻的输入有关，还与 $t - 1$ 时刻的网络状态有关，从而能处理与实践相关的动态变化。Elman 网络就是常用的 RNN 一种。

4.5.6 Boltzmann 机

一种基于能量 (energy) 的模型，当能量最小化时取得网络的理想状态。

4.6 深度学习

4.7 小结

Chapter 5

支持向量机 (Support Vector Machine,SVM)

5.1 间隔与支持向量

在样本空间中，划分超平面可以通过如下线性方程组进行描述：

$$w^T x + b = 0 \quad (5.1)$$

样本空间中任一点 x ，到超平面 (w, b) 的距离可以写为：

$$\tau = \frac{|w^T x + b|}{\|w\|} \quad (5.2)$$

假设超平面 (w, b) 能使训练样本正确分类，即对于 $(x_i, y_i) \in D$ ，有：

$$\begin{cases} w^T x_i + b \geq +1, & y_i = +1; \\ w^T x_i + b \leq -1, & y_i = -1. \end{cases} \quad (5.3)$$

对于距离超平面最近的几个点使得上式的等号成立，那么它们被称为“支持向量 (support vector)”，如下图所示，两个异类支持向量到超平面的距离之和为：

$$\gamma = \frac{2}{\|w\|} \quad (5.4)$$

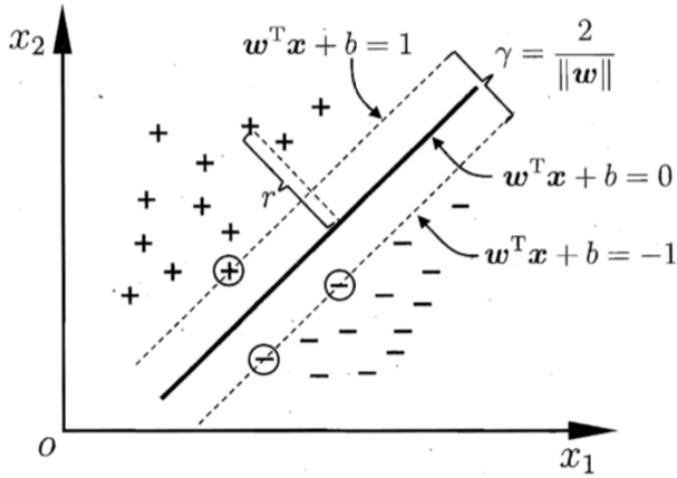


图 6.2 支持向量与间隔

欲找到最大间隔 (maximum margin) 来划分超平面, 也就是要找到能满足约束的 w, b , 使得 γ 最大, 即:

$$\max_{w,b} \frac{2}{\|w\|}, \quad s.t. \quad y_i(w^T x_i + b) \geq 1, i = 1, 2, \dots, m \quad (5.5)$$

即只需要最大化 $\frac{1}{\|w\|}$, 即最小化 $\|w\|^2$ 即可, 所以将上式改写一下:

$$\min_{w,b} \frac{1}{2} \|w\|^2, \quad s.t. \quad y_i(w^T x_i + b) \geq 1, i = 1, 2, \dots, m \quad (5.6)$$

这个就是支持向量机 (Support Vector Machine, SVM) 的基本型.

5.2 对偶问题

公式 5.6 是一个凸二次规划问题, 可以使用拉格朗日乘子法得到其的“对偶问题”(dual problem), 即对公式 5.6 每个约束添加拉格朗日乘子 $\alpha_i \geq 0$:

$$L(w, b, \alpha) = \frac{1}{2} \|w\|^2 + \sum_{i=1}^m \alpha_i (1 - y_i(w^T x_i + b)) \quad (5.7)$$

令 $L(w, b, \alpha)$ 对 w 和 b 的偏导为零可得：

$$\begin{aligned} w &= \sum_{i=1}^m a_i y_i x_i \\ 0 &= \sum_{i=1}^m a_i y_i \end{aligned} \tag{5.8}$$

将 5.8 带入 5.7，可以将 w 和 b 消去，就可以得到 5.6 的对偶问题：

$$\begin{aligned} \max_{\alpha} & \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m \alpha_i \alpha_j y_i y_j x_i^T x_j \\ \text{s.t. } & \sum_{i=1}^m \alpha_i y_i = 0, \alpha_i \geq 0, i = 1, 2, \dots, m \end{aligned} \tag{5.9}$$

求解出 α 后，再求出 w 和 b 即可求出模型：

$$f(x) = w^T x + b = \sum_{i=1}^m \alpha_i y_i x_i^T x + b \tag{5.10}$$

上述过程要满足 KKT(Karush-Kuhn-Tucker) 条件，即要求：

$$\begin{cases} \alpha_i \leftarrow 0; \\ y_i f(x_i) - 1 \geq 0; \\ \alpha_i (y_i f(x_i) - 1) = 0 \end{cases} \tag{5.11}$$

上述约束条件显示了 SVM 一个重要的性质：训练完成后，大部分的训练样本都不需要保留，最终模型仅仅与支持向量有关。

为了求解 5.9，我们常常采用 SMO 算法，其基本思路：先固定 α_i 之外的所有参数，然后求 α_i 上的极值。由于存在约束 $\sum_{i=1}^m \alpha_i y_i = 0$ ，所以 SMO 每次选择两个变量 α_i 和 α_j 并固定其它参数，不断重复以下的步骤知道收敛：

- 选取一对需要更新的变量 α_i 和 α_j ；
- 固定 α_i 和 α_j ，求解 4.9 获得更新后的 α_i 和 α_j ；

一般来说，只要 α_i 和 α_j 有一个不满足 KKT 条件，目标函数 5.9 就会在迭代中减小，违背 KKT 的程度越大，那么目标函数值减幅也会更大。基于这个性质 SMO 先选取违背 KKT 条件程度最大的变量，第二个变量选取一个使得目标函数值减小最快的变量，为了简便，所选取的两个变量对应的样本之间的间隔应该最大。

SMO 算法之所以高效，是应为它仅仅需要优化两个参数 α_i 和 α_j ，将 5.9 重写为：

$$\begin{aligned} \alpha_i y_i = \alpha_j y_j &= c, \alpha_i \geq 0, \alpha_j \geq 0, \\ c &= - \sum_{k \neq i, j} \alpha_k y_k \end{aligned} \quad (5.12)$$

用 5.12 消去 5.9 中的变量 α_j ，得到一个关于 α_i 的单变量二次规划的问题，仅有约束 $\alpha_i \geq 0$ 。

对于任意的支持向量 (x_s, y_s) 都有 $y_s f(x_s) = 1$ ，即：

$$y_s \left(\sum_{i \in S} \alpha_i y_i x_i^T x_s + b \right) = 1 \quad (5.13)$$

其中 $S = \{i | \alpha_i > 0, i = 1, 2, \dots, m\}$ 为了简便，直接求解所有支持向量的平均值得到偏置 b ：

$$b = \frac{1}{|S|} \sum_{s \in S} \left(y_s - \sum_{i \in S} \alpha_i y_i x_i^T x_s \right) \quad (5.14)$$

5.3 核函数

在实际应用中，往往遇到的是线性不可分的问题，遇到这种问题，一般的做法是将样本映射到一个更加高维的空间，使得样本能够在这个特征空间内线性可分。

令 $\phi(x)$ 表示将 x 映射后的特征向量，于是，在特征空间中划分超平面所对应的模型可表示为：

$$f(x) = w^T \phi(x) + b \quad (5.15)$$

类似上面的 5.6 和 5.9，有：

$$\min_{w, b} \frac{1}{2} \|w\|^2 \quad s.t. \quad y_i(w^T \phi(x_i) + b) \geq 1, \quad i = 1, 2, \dots, m \quad (5.16)$$

其对偶问题为：

$$\begin{aligned} & \max_{\alpha} \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m \alpha_i \alpha_j y_i y_j \phi(x_i)^T \phi(x_j) \\ & s.t. \sum_{i=1}^m \alpha_i y_i = 0, \quad \alpha_i \geq 0, \quad i = 1, 2, \dots, m \end{aligned} \quad (5.17)$$

假设一个函数 $\kappa(x_i, x_j)$ 表示 $\phi(x_i)^T \phi(x_j)$ 来计算内积，这个函数 κ 我们称为‘核函数’(kernel function)：

$$\kappa(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle = \phi(x_i)^T \phi(x_j) \quad (5.18)$$

重写 5.17：

$$\begin{aligned} & \max_{\alpha} \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m \alpha_i \alpha_j y_i y_j \kappa(x_i, x_j) \\ & s.t. \sum_{i=1}^m \alpha_i y_i = 0, \quad \alpha_i \geq 0, \quad i = 1, 2, \dots, m \end{aligned} \quad (5.19)$$

求解后：

$$f(x) = w^T \phi(x) + b = \sum_{i=1}^m \alpha_i y_i \kappa(x, x_i) + b \quad (5.20)$$

一般来说，只要一个对称函数所对应的核矩阵是半正定的，它就能作为核函数使用，下图是一些常用的核函数：

表 6.1 常用核函数

名称	表达式	参数
线性核	$\kappa(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$	
多项式核	$\kappa(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^T \mathbf{x}_j)^d$	$d \geq 1$ 为多项式的次数
高斯核	$\kappa(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\ \mathbf{x}_i - \mathbf{x}_j\ ^2}{2\sigma^2}\right)$	$\sigma > 0$ 为高斯核的带宽(width)
拉普拉斯核	$\kappa(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\ \mathbf{x}_i - \mathbf{x}_j\ }{\sigma}\right)$	$\sigma > 0$
Sigmoid 核	$\kappa(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\beta \mathbf{x}_i^T \mathbf{x}_j + \theta)$	\tanh 为双曲正切函数, $\beta > 0, \theta < 0$

核函数还会满足下面三个性质：

- 若 κ_1, κ_2 为核函数, 那么对于任意正数 γ_1, γ_2 , 其线性组合也为核函数

$$\gamma_1 \kappa_1 = \gamma_2 \kappa_2 \quad (5.21)$$

- 若 κ_1, κ_2 为核函数, 则核函数的直积也为核函数

$$\kappa_1 \otimes \kappa_2(x, z) = \kappa_1(x, z)\kappa_2(x, z) \quad (5.22)$$

- 若 κ_1 为核函数, 则对任意函数 $g(x)$,

$$\kappa(x, z) = g(x)\kappa_1(x, z)g(z) \quad (5.23)$$

也是核函数

5.4 软间隔与正则化

在现实任务中, 我们往往很难确定一个适合的核函数使得训练样本在特征空间中线性可分, 很难断定这个貌似线性可分的结果是不是由于过拟合造成的。缓解该问题的一个办法是允许 SVM 在一些样本上出错, 为此我们引入‘软间隔’(soft margin), 但是在最大化间隔的同时, 不满足约束的样本应该尽可能减少, 故优化目标可以写为:

$$\min_{w,b} \frac{1}{2} \|w\|^2 + C \sum_{i=1}^m \ell_{0/1}(y_i(w^T x_i + b) - 1) \quad (5.24)$$

其中: $C > 0$ 是一个常数, $\ell_{0/1}$ 是一个“0/1 损失函数”

$$\ell_{0/1}(z) = \begin{cases} 1, & \text{if } z < 0; \\ 0, & \text{otherwise.} \end{cases} \quad (5.25)$$

显由于 $\ell_{0/1}$ 性质不太好 (非凸, 非连续), 我们常用其它一些函数进行代替, 称为代替损失 (surrogate loss), 下面列举了常用的代替损失函数, 如下图所示:

$$\begin{aligned} \text{hinge 损失: } & \ell_{hinge}(z) = \max(0, 1 - z) \\ \text{指数损失 (exponential loss): } & \ell_{exp}(z) = \exp(-z) \\ \text{对率损失 (logistic loss): } & \ell_{log}(z) = \log(1 + \exp(-z)) \end{aligned} \quad (5.26)$$

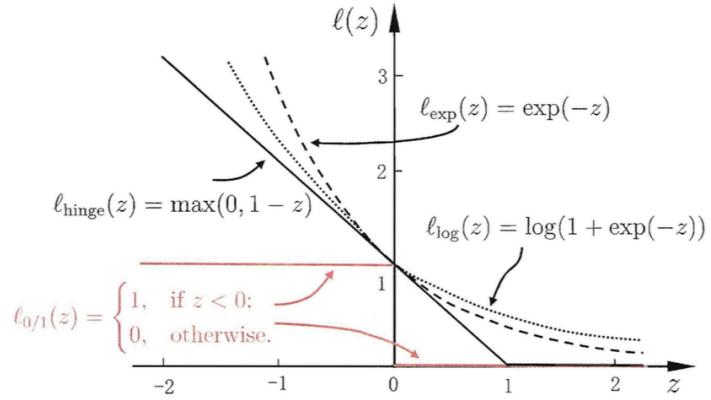


图 6.5 三种常见的替代损失函数: hinge 损失、指数损失、对率损失

若采用 hinge 损失, 那么改写 5.24:

$$\min_{w,b} \frac{1}{2} \|w\|^2 + C \sum_{i=1}^m \max(0, 1 - y_i(w^T x_i + b)) \quad (5.27)$$

引入松弛变量 (slack variables) $\xi_i \geq 0$, 重写上式:

$$\min_{w,b,\xi_i} \frac{1}{2} \|w\|^2 + C \sum_{i=1}^m \xi_i \quad (5.28)$$

$$s.t. y_i(w^T x_i + b) \geq 1 - \xi_i, \xi_i \geq 0, i = 1, 2, \dots, m$$

这称为常用的‘软间隔 SVM’, 通过拉格朗日乘子法可得:

$$\begin{aligned} L(w, b, \alpha, \xi, \mu) = & \\ \frac{1}{2} \|w\|^2 + C \sum_{i=1}^m \xi_i + \sum_{i=1}^m \alpha_i (1 - \xi_i - y_i(w^T x_i + b)) - \sum_{i=1}^m \mu_i \xi_i & \end{aligned} \quad (5.29)$$

其中 $\alpha_i \geq 0, \mu_i \geq 0$ 是拉格朗日乘子。令 $L(w, b, \alpha, \xi, \mu)$ 对 w, b, ξ_i 求偏导为零可得:

$$\begin{aligned} w &= \sum_{i=1}^m \alpha_i y_i x_i \\ 0 &= \sum_{i=1}^m \alpha_i y_i \\ C &= \alpha_i + \mu_i \end{aligned} \quad (5.30)$$

联立上面两个公式，可得 5.28 的对偶问题：

$$\begin{aligned} & \max_{\alpha} \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m \alpha_i \alpha_j y_i y_j x_i^T x_j \\ s.t. \quad & \sum_{i=1}^m \alpha_i y_i = 0, 0 \leq \alpha \leq C, i = 1, 2, \dots, m. \end{aligned} \quad (5.31)$$

软间隔与硬间隔的唯一差别是在于对偶变量的约束不同，前者是 $0 \leq \alpha_i \leq C$ ，后者是 $0 \leq \alpha_i$ ，那么软间隔 KKT 条件相应改为：

$$\begin{cases} \alpha_i \geq 0, \quad \mu_i \geq 0, \\ y_i f(x_i) - 1 + \xi_i \geq 0, \\ \alpha_i (y_i f(x_i) - 1 + \xi_i) = 0, \\ \xi_i \geq 0, \quad \mu_i \xi_i = 0 \end{cases} \quad (5.32)$$

通过上述 KKT 条件，可以看出软间隔 SVM 最终模型仅仅与支持向量有关，即通过采用 hinge 损失函数仍然保持了稀疏性。

我们可以把上面的优化目标写成更加一般的形式，其中第一项用来描述划分超平面的‘间隔’大小，另一项 $\sum_{i=1}^m \ell(f(x_i), y_i)$ 用来表述训练集上的误差：

$$\min_f \Omega(f) + C \sum_{i=1}^m \ell(f(x_i), y_i) \quad (5.33)$$

其中 $\Omega(f)$ 称为结构分析按 (structural risk)，用来描述模型 f 的某些性质， $\sum_{i=1}^m \ell(f(x_i), y_i)$ 称为经验风险 (empirical risk)，用于描述模型与训练数据的契合程度， C 对两者进行权重。 $\Omega(f)$ 也称为正则化项， C 称为正则化函数。

5.5 支持向量回归 (Support Vector Regression, SVR)

SVR 与传统的回归模型有些许不同，他假设我们能容忍模型输出 $f(x)$ 与真实输出 y 之间最多有 ϵ ，当且仅当 $|f(x) - y| > \epsilon$ 才计算损失，相当于以 $f(x)$ 为中间，构建一个宽度为 2ϵ 的间隔带，若样本落入这个间隔带，则认为分类时正确的。

SVR 问题可以形式化为：

$$\min_{w,b} \frac{1}{2} \|w\|^2 + C \sum_{i=1}^m \ell_\epsilon(f(x_i) - y_i) \quad (5.34)$$

其中 ℓ_ϵ 称为 ϵ -不敏感损失函数 (ϵ -insensitive loss):

$$\ell_\epsilon(z) = \begin{cases} 0, & \text{if } |z| \leq \epsilon \\ |z| - \epsilon, & \text{otherwise} \end{cases} \quad (5.35)$$

引入松弛变量 ξ_i 和 $\hat{\xi}_i$, 重写上上式:

$$\begin{aligned} & \min_{w,b,\xi_i,\hat{\xi}_i} \frac{1}{2} \|w\|^2 + C \sum_{i=1}^m (\xi_i + \hat{\xi}_i) \\ & \text{s.t. } f(x_i) - y_i \leq \epsilon + \xi_i, \quad y_i - f(x_i) \leq \epsilon + \hat{\xi}_i, \\ & \quad \xi_i, \hat{\xi}_i \geq 0, i = 1, 2, \dots, m \end{aligned} \quad (5.36)$$

引入拉格朗日乘子法: $\mu_i \geq 0, \hat{\mu}_i \geq 0, \alpha_i \geq 0, \hat{\alpha}_i \geq 0$:

$$\begin{aligned} L(w, b, \alpha, \hat{\alpha}, \xi, \hat{\xi}, \mu, \hat{\mu}) = & \\ & \frac{1}{2} \|w\|^2 + C \sum_{i=1}^m (\xi_i + \hat{\xi}_i - \sum_{i=1}^m \mu_i \xi_i - \sum_{i=1}^m \hat{\mu}_i \hat{\xi}_i + \\ & \sum_{i=1}^m \alpha_i (f(x_i) - y_i - \epsilon - \xi_i) + \sum_{i=1}^m \hat{\alpha}_i (y_i - f(x_i) - \epsilon - \hat{\xi}_i) \end{aligned} \quad (5.37)$$

对上式进行求导:

$$\begin{aligned} w &= \sum_{i=1}^m (\hat{\alpha}_i - \alpha_i) x_i \\ 0 &= \sum_{i=1}^m (\hat{\alpha}_i - \alpha_i) \\ C &= \alpha_i + \mu_i \\ C &= \hat{\alpha}_i + \hat{\mu}_i \end{aligned} \quad (5.38)$$

所以我们可以计算出 SVR 的对偶问题:

$$\begin{aligned} \max_{\alpha, \hat{\alpha}} \quad & \sum_{i=1}^m y_i (\hat{\alpha}_i - \alpha_i) - \epsilon (\hat{\alpha}_i + \alpha_i) - \frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m (\hat{\alpha}_i (\hat{\alpha}_j - \alpha_j)) x_i^T x_j \\ \text{s.t.} \quad & \sum_{i=1}^m (\hat{\alpha}_i + \alpha_i) = 0, \quad 0 \leq \alpha_i, \hat{\alpha}_i \leq C \end{aligned} \quad (5.39)$$

满足以下 KKT 条件:

$$\begin{cases} \alpha_i (f(x_i) - y_i - \epsilon - \xi_i) = 0, \\ \hat{\alpha}_i (y_i - f(x_i) - \epsilon - \hat{\xi}_i) = 0, \\ \alpha_i \hat{\alpha}_i = 0, \xi_i \hat{\xi}_i = 0, \\ (C - \alpha_i) \xi_i = 0, (C - \hat{\alpha}_i) \hat{\xi}_i = 0 \end{cases} \quad (5.40)$$

综上, SVR 的解形可以写为:

$$f(x) = \sum_{i=1}^m (\hat{\alpha}_i - \alpha_i) x_i^T x + b \quad (5.41)$$

能使得上式中的 $\hat{\alpha}_i - \alpha_i \neq 0$ 的样本即为 SVR 的支持向量, 他们必然落于 ϵ -间隔带外, 在得到 α_i 后, 若 $0 < \alpha_i < C$, 则必有 $\xi_i = 0$, 进而有:

$$b = y_i + \epsilon - \sum_{i=1}^m (\hat{\alpha}_i - \alpha_i) x_i^T x \quad (5.42)$$

一般在实践中, 我们采取更加鲁棒的做法: 选取多个满足条件 $0 < \alpha_i < C$ 的样本后求解 b 取平均值, 考虑到映射的形式, SVR 可表示为:

$$f(x) = \sum_{i=1}^m (\hat{\alpha}_i - \alpha_i) \kappa(x, x_i) + b \quad (5.43)$$

其中 $\kappa(x_i, x_j) = \phi(x_i)^T \phi(x_j)$ 为核函数。

5.6 核方法

表示定理: 满足 Ω 单挑递增, ℓ 非负损失函数, 对于优化问题:

$$\min_{h \in \mathbb{H}} F(h) = \Omega(\|h\|_{\mathbb{H}}) + \ell(h(x_1), h(x_2), \dots, h(x_m)) \quad (5.44)$$

的解总可写为：

$$h^*(x) = \sum_{i=1}^m \alpha_i \kappa(x, x_i) \quad (5.45)$$

一般来说，人们把这种方法称为核方法 (kernel methods)，最常见的就是通过引入核函数来核化将线性学习期拓展为非线性拓展，从而得到核线性判别分析 (Kernelized Linear Discriminant Analysis, KLDA).

我们可以假设通过某种映射 $\phi : \mathcal{X} \rightarrow \mathbb{F}$ 将样本空间到一个特征空间 \mathbb{F} ，然后在 \mathbb{F} 中执行线性判别分析，以求得：

$$h(x) = w^T \phi(x) \quad (5.46)$$

那么 KLDA 的学习目标：

$$\max_w J(w) = \frac{w^T S_b^\phi w}{w^T S_w^\phi w} \quad (5.47)$$

其中 $w^T S_b^\phi w$ 表示 \mathbb{F} 中的类间散度矩阵，而 $w^T S_w^\phi w$ 类内散度矩阵。令 X_i 表示第 $i \in 0, 1$ 类样本的集合，其样本数为 m_i ，总样本数为 $m = m_0 + m_1$ ，那么第 i 类样本在特征空间 \mathbb{F} 的均值为：

$$\mu_i^\phi = \frac{1}{m_i} \sum_{x \in X_i} \phi(x) \quad (5.48)$$

两个散度矩阵分别为：

$$\begin{aligned} S_b^\phi &= (\mu_1^\phi - \mu_0^\phi)(\mu_1^\phi - \mu_0^\phi)^T \\ S_w^\phi &= \sum_{i=0}^1 \sum_{x \in X_i} (\phi(x) - \mu_i^\phi)(\phi(x) - \mu_i^\phi)^T \end{aligned} \quad (5.49)$$

我们用核函数 $\kappa(x, x_i)$ 来表示，那么 $h(x)$ 可写为：

$$h(x) = \sum_{i=1}^m \alpha_i \kappa(x, x_i) \quad (5.50)$$

故：

$$w = \sum_{i=1}^m \alpha_i \phi(x_i) \quad (5.51)$$

令 $K \in \mathbb{R}^{m \times m}$ 为 κ 的核矩阵, 那么:

$$\begin{aligned}\hat{\mu}_0 &= \frac{1}{m_0} Kl_0, \\ \hat{\mu}_1 &= \frac{1}{m_1} Kl_1, \\ M &= (\hat{\mu}_0 - \hat{\mu}_1)(\hat{\mu}_0 - \hat{\mu}_1)^T, \\ N &= KK^T - \sum_{i=0}^l m_i \hat{\mu}_i \hat{\mu}_i^T,\end{aligned}\tag{5.52}$$

于是 5.47 等价为:

$$\max_{\alpha} J(\alpha) = \frac{\alpha^T M \alpha}{\alpha^T N \alpha} \tag{5.53}$$

用线性判别分析求解方法即可得到 α , 从而求出 $h(x)$ 。

5.7 小结

Chapter 6

贝叶斯分类器 (bayes classifier)

由于在深极做过一次 ppt 的演讲了，所以这一章不用写得太详细。

6.1 贝叶斯公式

贝叶斯公式的本质就是通过条件概率 (也叫似然) 之间的转化，建立先验概率 $P(x|c)$ 与后验概率 $P(c|x)$ 之间的联系。

$$P(c|x) = \frac{P(c, x)}{P(x)} = \frac{P(x|c)P(c)}{P(x)} \quad (6.1)$$

6.2 贝叶斯决策论 (bayesian decision theory)

假设有 N 种可能的类别标记。即 $\mathcal{Y} = \{c_1, c_2, \dots, c_N\}$, λ_{ij} 是将一个真实标记为 c_j 的样本误分类为 c_i 的损失，那么基于后验概率 $P(c_i|x)$ 就可以获得将样本 x 分类为 c_i 所产生的期望损失，即 x 上的条件风险 (conditional risk)。

$$R(c_i|x) = \sum_{j=1}^N \lambda_{ij} P(c_j|x) \quad (6.2)$$

我们的目标即找出 x 的分类，使得期望风险 $R(c|x)$ 最小：

$$h^*(x) = \arg \min_{c \in \mathcal{Y}} R(c|x) \quad (6.3)$$

6.3 极大似然估计 (Maximum Likelihood Estimation, MLE)

- 对 θ_c 进行极大似然估计。就是寻找能最大化似然 $P(D_c|\theta_c)$ 的参数值 $\hat{\theta}_c$ 。换句话说，就是遍历 θ_c 所有可能的取值，找出一个使数据出现的“可能性”最大的一个。
- 为了加快计算速度和减少溢出的可能性，可以用取对数相加代替连乘的操作：

$$LL(\theta_c) = \log P(D_c|\theta_c) = \sum_{x \in D_c} \log P(x|\theta_c) \quad (6.4)$$

- 此时参数 θ_c 的极大似然估计 $\hat{\theta}_c$ 可以写为：

$$\hat{\theta}_c = \arg \max_{\theta_c} LL(\theta_c) \quad (6.5)$$

- 总之，MLE 的思想可以总结为：已知某个参数能使这个样本出现的概率最大，我们当然不会再选择其他小概率的样本，所以干脆就把这个参数作为估计的真实值。

6.4 朴素贝叶斯分类器 (Naïve Bayes Classifier)

- 朴素 (Naïve)，指的是“属性条件独立性假设”，即对于已知类别，假设所有的属性相互独立。聊天监控系统里采用的贝叶斯算法就是基于 NBC 的。
- 在这个前提下，贝叶斯公式可以改写：

$$P(c|x) = \frac{P(c)P(x|c)}{P(x)} = \frac{P(c)}{P(x)} \prod_{i=1}^d P(x_i|c) \quad (6.6)$$

其中， d 为属性的数目， x_i 为 x 在第 i 个属性上的取值。

- 由上，我们可得朴素贝叶斯分类器的表达式。

$$h_{nb}(x) = \arg \max_{c \in \mathcal{Y}} P(c) \prod_{i=1}^d P(x_i|c) \quad (6.7)$$

- 对于贝叶斯分类器，若前提有大量的训练集，我们可以事先训练贝叶斯分类所有的概率估值，进行测试时，我们只需进行查表操作即可，借助哈希表、二叉树等数据结构进行存储，贝叶斯分类器的时间复杂度为 $O(n)$ 。

6.5 半朴素贝叶斯分类器 (semi-Naïve Bayes Classifier)

- NBC 是基于属性之间是相互独立的假设，但是在现实条件下这个假设是很难实现的，所以我们提出了半朴素贝叶斯分类器，它的基本想法是适当考虑一部分属性间的相互依赖信息，从而既不需要进行完全联合计算，又不至于彻底忽略了比较强的属性依赖关系。
- 采用得比较多得是“独依赖估计”(One-Dependent Estimator,ODE)，指的是每个属性最多仅依赖一个其他属性，即：

$$P(c|x) \propto P(c) \prod_{i=1}^d P(x_i|c, pa_i) \quad (6.8)$$

其中 pa_i 为属性 x_i 所依赖的属性，称为 x_i 的父属性。

6.6 贝叶斯网 (Bayesian network)

- 贝叶斯网也称为信念网 (belief network)。它借助有向无环图来刻画属性间的依赖关系，并使用条件概率表来描述属性的联合概率分布。
- 贝叶斯网 B 可以表示成如下公式：

$$B = \langle G, \Theta \rangle \quad (6.9)$$

其中 G 表示一个有向无环图， Θ 定量描述两个属性之间的直接依赖关系。假设属性 x_i 在 G 中的父结点集为 π_i ，则 Θ 包含了每个属性的条件概率表 $\theta_{x_i|\pi_i} = P_B(x_i|\pi_i)$ 。

- 贝叶斯网学习的首要任务是通过训练集构建一个最合理的贝叶斯网，一般采用评分搜索的办法。
- 首先定义一个评分函数(score function)，基于信息论准则，其目标是找到一个能以最小编码长度描述训练模型，即“最小描述长度”(Minimal Description Length,MDL)

6.6.1 贝叶斯网 (Bayesian network)-学习

求 MDL 的过程可以描述如下：

1. 给定训练集 $D = \{x_1, x_2, \dots, x_m\}$ ，那么贝叶斯网 $B = \langle G, \Theta \rangle$ 的评分函数可以写为：

$$s(B|D) = f(\theta)|B| - LL(B|D) \quad (6.10)$$

其中 $|B|$ 是贝叶斯网的参数个数； $f(\theta)$ 表示描述每个参数所需的字节数， $LL(B|D)$ 表示贝叶斯网 B 的对数似然。

$$LL(B|D) = \sum_{i=1}^m \log P_B(x_i) \quad (6.11)$$

2. 学习任务转化为一个优化任务，即寻找一个贝叶斯网 B 使评分函数 $s(B|D)$ 最小。
3. 若 $f(\theta) = 0$ ，即不计算对网络编码的长度，评分函数退化成负对数似然，那么学习任务退化成极大似然估计。

$$s(B|D) = -LL(B|D) = - \sum_{i=1}^m \log P_B(x_i) \quad (6.12)$$

4. 若 $B = \langle G, \Theta \rangle$ 的网络结构 G 固定，则 $s(B|D)$ 等价于对参数 Θ 的极大似然估计，那么 $\theta_{x_i|\pi_i}$ 可以直接在训练数据 D 上通过经验估计得到：

$$\theta_{x_i|\pi_i} = \hat{P}_D(x_i|\pi_i) \quad (6.13)$$

其中 $\hat{P}_D(\cdot)$ 是 D 上的经验分布。

5. 为了最小化评分函数 $s(B|D)$, 只需要对网络结构进行搜索, 而候选结构的最优参数可以直接在训练集上计算得到。
6. 搜索出贝叶斯网最优结构是一个 NP 难的问题, 难以快速求解, 一般常用两种方法保证在有限时间内求得近似解。
 - 采用贪心策略, 从某个网络结构出发, 每次调整一条边, 直到评分函数值不再降低为止。
 - 通过网络结构施加约束来削减搜索空间, 例如将网络结构限定为树形结构。

6.6.2 贝叶斯网 (Bayesian network)-推断

- 通过前面的训练和学习, 贝叶斯网就可以通过一些属性变量的观测值来推测其他属性变量的取值, 这个过程我们称为“推断”(inference), 已知变量观测值称为“证据”(evidence)。
- 理想情况下是直接根据贝叶斯网定义的联合概率分布计算后验概率, 但前面已经说明搜索最优结构是 NP 难的, 在现实应用中, 贝叶斯网的近似推断常使用吉布斯采样 (Gibbs sampling) 来完成。

6.7 EM(Expectation-Maximization) 算法

- 在实际应用的时候, 我们很难获得所有属性变量的值, 即训练样本是不完整的, 像这种无法获得属性变量的“未观测”变量, 我们称为“隐变量”(latent variable)。
- EM(Expectation-Maximization) 算法, 就是常用的估计参数隐变量的算法。它的基本思想很简单: 若参数 Θ 已知, 则可根据训练数据推断出最优隐变量 Z 的值 (E 步); 反之, 若 Z 的值已知, 则可方便地对 Θ 做极大似然估计 (M 步)。

若我们不是取 Z 的期望, 而是基于 Θ^t 计算隐变量 Z 的概率分布 $P(Z|X, \Theta^t)$, 那么 em 算法两步可以这样直接定义:

- E 步 (Expectation): 以当前参数 Θ^t 推断隐变量分布 $P(Z|X, \Theta^t)$, 并计算对数似然 $LL(\Theta|X, Z)$ 关于 Z 的期望。

$$Q(\Theta|\Theta^t) = \mathbb{E}_{Z|X, \Theta^t} LL(\Theta|X, Z) \quad (6.14)$$

- M 步 (Maximization): 寻找参数最大化期望似然:

$$\Theta^{t+1} = \arg \max_{\Theta} Q(\Theta|\Theta^t) \quad (6.15)$$

6.8 小结

贝叶斯模型是一个 precision 很高的模型, 而且属于线性模型, 十分便于工程实现。

最简单的朴素贝叶斯分类器, 在很多情况下都能够获得相当好的性能, 十分适合信息检索领域, 是常用的文本分类策略之一。

Chapter 7

集成学习 (ensemble learning)

7.1 个体与集成

集成学习指的是将几个弱学习器 (weaker learner, 一般就是指基学习器) 按照某种策略组合在一起进行学习。首先说明几个概念：

- 基学习器：表示构成的集成所用的算法都是一致的
- 组件学习器：表示构成的集成所用的算法都是不一致的

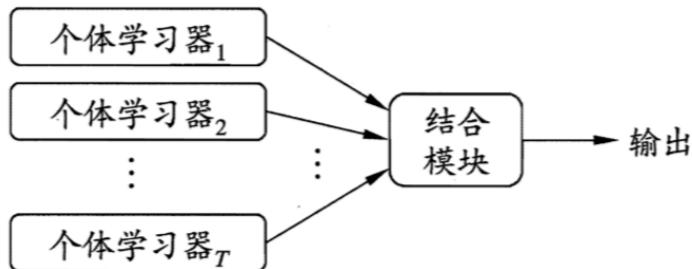


图 8.1 集成学习示意图

假定基分类器的错误率为 ϵ , 即对每个基分类器 h_i 有：

$$P(h_i(x) \neq f(x)) = \epsilon \quad (7.1)$$

若采用投票法，即超过半数基分类器认为是正确的，则集成分类就正确：

$$H(x) = \text{sign} \left(\sum_{i=1}^T h_i(x) \right) \quad (7.2)$$

若基分类器之间的错误率相互之间是独立的，那么根据 Hoeffding 不等式，集成错误率为：

$$P(h_i(x) \neq f(x)) = \sum_{k=0}^{[T/2]} \binom{T}{k} (1-\epsilon)^k \epsilon^{T-k} \leq \exp \left(-\frac{1}{2} T (1-2\epsilon)^2 \right) \quad (7.3)$$

说明随着个体分类器数目 T 的不断增大，集成的错误率将逐渐下降。当前二种集成学习的思想：

- 各个个体学习期之间存在强依赖的关系、必须串行生成序列化的方法，如 Boosting
- 各个个体学习期之间不存在强依赖关系、可以同时生成的并行化方法，如 Bagging 或者随机森林。

7.2 Boosting

Boosting 算法的工作机制很类似：

1. 从初始训练集中选择一个基学习器
2. 再根据基学习器的表现对训练样本分布进行调整，使之前分类错误的训练样本在后面受到更多关注。
3. 基于调整后的样本分布来训练下一个基学习器
4. 直到基学习器数目到达事先指定的值 T ，最终将这 T 个基学习器进行加权结合。

其中最出名的 Boosting 算法是 AdaBoost，一般用的是基学习器的线性组合：

$$H(x) = \sum_{t=1}^T \alpha_t h_t(x) \quad (7.4)$$

来最小化指数损失函数 (exponential loss function):

$$\ell_{\text{exp}}(H|\mathcal{D}) = \mathbb{E}_{x \sim \mathcal{D}}[e^{-f(x)H(x)}] \quad (7.5)$$

对应的算法流程图:[算法用 LATEX 打起来太麻烦了，先用截图，以后有时间再改成 LATEX](#)

输入: 训练集 $D = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_m, y_m)\}$;
 基学习算法 \mathfrak{L} ;
 训练轮数 T .

过程:

- 1: $\mathcal{D}_1(\mathbf{x}) = 1/m$.
- 2: **for** $t = 1, 2, \dots, T$ **do**
- 3: $h_t = \mathfrak{L}(D, \mathcal{D}_t)$;
- 4: $\epsilon_t = P_{\mathbf{x} \sim \mathcal{D}_t}(h_t(\mathbf{x}) \neq f(\mathbf{x}))$;
- 5: **if** $\epsilon_t > 0.5$ **then break**
- 6: $\alpha_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$;
- 7: $\mathcal{D}_{t+1}(\mathbf{x}) = \frac{\mathcal{D}_t(\mathbf{x})}{Z_t} \times \begin{cases} \exp(-\alpha_t), & \text{if } h_t(\mathbf{x}) = f(\mathbf{x}) \\ \exp(\alpha_t), & \text{if } h_t(\mathbf{x}) \neq f(\mathbf{x}) \end{cases}$
 $= \frac{\mathcal{D}_t(\mathbf{x}) \exp(-\alpha_t f(\mathbf{x}) h_t(\mathbf{x}))}{Z_t}$
- 8: **end for**

输出: $H(\mathbf{x}) = \text{sign} \left(\sum_{t=1}^T \alpha_t h_t(\mathbf{x}) \right)$

图 8.3 AdaBoost 算法

对 $H(x)$ 进行偏导:

$$\frac{\partial \ell_{\text{exp}}(H|\mathcal{D})}{\partial H(x)} = -e^{-H(x)}P(f(x) = 1|x) + e^{H(x)}P(f(x) = -1|x) \quad (7.6)$$

令上式为 0, 有:

$$H(x) = \frac{1}{2} \ln \frac{P(f(x) = 1|x)}{P(f(x) = -1|x)} \quad (7.7)$$

于是：

$$\begin{aligned}
\text{sign}(H(x)) &= \text{sign} \left(\frac{1}{2} \ln \frac{P(f(x) = 1|x)}{P(f(x) = -1|x)} \right) \\
&= \begin{cases} 1, & P(f(x) = 1|x) > P(f(x) = -1|x) \\ -1, & P(f(x) = 1|x) < P(f(x) = -1|x) \end{cases} \\
&= \arg \max_{y \in \{-1, 1\}} P(f(x) = y|x)
\end{aligned} \tag{7.8}$$

上式表明，若使得指数损失函数最小化了，那么分类错误率也将得到最小化；在 AdaBoost 算法中，第一个基分类器 h_1 是通过直接将基学习算法用于初始数据分布而得到的，此后迭代地生成 h_t 和 α_t ，当基分类器 h_t 的基于分布 \mathcal{D} 产生后，该基分类器的权重 α_t 应使得 $\alpha_t h_t$ 最小化指数函数：

$$\ell_{\text{exp}}(\alpha_t h_t | \mathcal{D}_t) = e^{-\alpha_t} (1 - \epsilon_t) + e^{\alpha_t} \epsilon_t \tag{7.9}$$

其中 $\epsilon_t = P_{x \sim \mathcal{D}_t}(h_t(x) \neq f(x))$ ，对 α_t 进行求导：

$$\frac{\partial \ell_{\text{exp}}(\alpha_t h_t | \mathcal{D}_t)}{\partial \alpha_t} = -e^{-\alpha_t} (1 - \epsilon_t) \tag{7.10}$$

令上式为 0，可得：

$$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right) \tag{7.11}$$

AdaBoost 算法能在获得 H_{t-1} 之后的样本分布进行相应的调整，使得下一轮的基学习器 h_t 能够纠正 H_{t-1} 的一些错误，理想的情况下， h_t 能纠正 H_{t-1} 的所有错误，达到理想化：

$$\begin{aligned}
h_t(x) &= \arg \min_h \ell_{\text{exp}}(H_{t-1} + h | \mathcal{D}) \\
&= \arg \max_h \mathbb{E}_{x \sim \mathcal{D}} = \left[\frac{e^{-f(x)H_{t-1}(x)}}{\mathbb{E}_{x \sim \mathcal{D}}[e^{-f(x)H_{t-1}(x)}]} f(x) h(x) \right]
\end{aligned} \tag{7.12}$$

其中 $\mathbb{E}_{x \sim \mathcal{D}}[e^{-f(x)H_{t-1}(x)}]$ 是一个常数，令 \mathcal{D}_t 表示一个分布：

$$\mathcal{D}_t(x) = \frac{\mathcal{D}(x) e^{-f(x)H_{t-1}(x)}}{\mathbb{E}_{x \sim \mathcal{D}}[e^{-f(x)H_{t-1}(x)}]} \tag{7.13}$$

根据数学的期望，这等价于：

$$h_t(x) = \arg \max_h \mathbb{E}_{x \sim \mathcal{D}_t} [f(x)h(x)] \quad (7.14)$$

由于 $f(x), h(x) \in \{-1, +1\}$ ，有：

$$h_t(x) = \arg \min_h \mathbb{E}_{x \sim \mathcal{D}_t} [\mathbb{I}(f(x) \neq h(x))] \quad (7.15)$$

综上：理想的 h_t 将在分布 \mathcal{D}_t 上最小化分类误差，类似于残差逼近的思想。考虑到 \mathcal{D}_t 和 \mathcal{D}_{t+1} 的关系，有：

$$\mathcal{D}_{t+1}(x) = \mathcal{D}_t(x) \cdot e^{-f(x)\alpha_i h_t(x)} \frac{\mathbb{E}_{x \sim \mathcal{D}} [e^{-f(x)H_{t-1}(x)}]}{\mathbb{E}_{x \sim \mathcal{D}} [e^{-f(x)H_t(x)}]} \quad (7.16)$$

对应算法流程图中的第七行样本分布的更新公式。

Boosting 算法要求基学习器能对特定的数据分布进行学习，可以通过‘重赋权法’实施，即在迭代的每一轮中，根据样本分布为每个训练样本重新赋予一个权重，对于无法接受代权样本的基学习方法，可以通过重采样法进行处理，即在每一轮学习中，根据样本分布对训练集进行重新采样，再进行训练。需要注意，Boosting 算法在训练每一轮都要检查当前生成的基学习器是否满足基本条件，一旦不满足，直接抛弃。

从偏差-方差分解的角度来说，Boosting 主要关注如何降低偏差，因此 Boosting 能基于泛化性能相当弱的学习器构建出很强的集成。

7.3 Bagging 与随机森林

在上一节中，我们可以知道，想得到泛化性能强的集成，集成中的个体学习器应当尽可能的相互独立，在实际应用中，应当使个体学习器尽可能有较大的差异。

7.3.1 Bagging

Bagging 可以说是并行式集成学习的著名代表，它是基于自主采样法（bootstrap sampling），给定包含 m 个样本的数据集，我们先随机选择一个样本放入采样集中，然后将其放回初始数据集。照这样，我们可以采样出 T

个含有 m 个训练样本的采样集，然后基于每个采样集训练出一个基学习器，再将这些基学习器进行组合，这就是 bagging 的基本思想。如下图所示：

输入: 训练集 $D = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_m, y_m)\}$;
基学习算法 \mathfrak{L} ;
训练轮数 T .

过程:

- 1: **for** $t = 1, 2, \dots, T$ **do**
- 2: $h_t = \mathfrak{L}(D, \mathcal{D}_{bs})$
- 3: **end for**

输出: $H(\mathbf{x}) = \arg \max_{y \in \mathcal{Y}} \sum_{t=1}^T \mathbb{I}(h_t(\mathbf{x}) = y)$

图 8.5 Bagging 算法

假定基学习器的时间复杂度为 $O(m)$ ，那么 Bagging 的复杂度大概为 $T(O(m) + O(s))$ ，说明 Bagging 是一个近似的线性时间复杂度算法，能够不经修改地应用于多分类、回归等任务。

从第二章我们知道，每个基学习器只使用了大约 63.2% 的样本，意味着剩下的 36.8% 可以用于验证集对泛化性能的包外估计 (out-of-bag estimate)。包外样本有许多用途：

- 当基学习器是决策树时，可以采用包外样本进行减值减少过拟合
- 当基学习器是神经网络时，可以用包外样本来辅助早期的停止来减少过拟合。

7.3.2 随机森林 (Random Forest, RF)

随机森林是 Bagging 的一个扩展变体，RF 在以决策树为基学习器的基础上，进一步在决策树的训练过程中引入随机属性的选择，在 RF 中，对于基决策树的每个节点，先从该节点的属性集合中随机选择一个包含 k 个属性的子集，然后在这个子集中选择一个最优属性用于划分。这里的参数 k 表示

了随机性的引入程度，若 $k = d$ 表明与传统决策树相同，若 $k = 1$ 表明随机只选择一个属性用于划分，随机性最大，一般情况下，推荐选择 $k = \log_2 d$.

7.4 结合策略

组合策略值得是将几个基学习器组合在一起，来避免以下情况：

- 单学习器泛化性能不佳。
- 避免陷入局部极小。
- 可以扩大相应的假设空间。

7.4.1 平均法 (averaging)

对于数值型的输出， $h_i(x) \in \mathbb{R}$, 最常用的就是平均法了：

- 简单平均 (simple averaging):

$$H(x) = \frac{1}{T} \sum_{i=1}^T h_i(x) \quad (7.17)$$

- 加权平均 (weighted averaging):

$$H(x) = \sum_{i=1}^T w_i h_i(x) \quad (7.18)$$

加权平均法的权重一般都是从训练数据中获得，但是由于数据不充分或者存在噪声，使得权重并不是完全可靠。若规模较大，容易陷入过拟合。一般而言，个体学习器的性能相差较大时采用加权平均法，相近时采用简单平均法。

7.4.2 投票法 (voting)

- 绝对多数投票法 (majority voting), 若某个 label 投票超过半数, 那么就预测为该 label。

$$H(x) = \begin{cases} c_j, & \text{if } \sum_{i=1}^T h_j^i(x) > 0.5 \sum_{k=1}^N \sum_{i=1}^T h_i^k(x); \\ \text{reject}, & \text{otherwise.} \end{cases} \quad (7.19)$$

- 相对多数投票法 (plurality voting), 选择得票数最高的 label, 若最高有多个, 则随机选择一个。

$$H(x) = c_{\arg \max_j \sum_{i=1}^T h_i^j(x)} \quad (7.20)$$

- 加权投票法 (weighted voting):

$$H(x) = c_{\arg \max_j \sum_{i=1}^T w_i h_i^j(x)} \quad (7.21)$$

7.4.3 学习法

学习法的原理是把初级学习器的输出结果当做次级训练器的样本进行训练。如 Stacking 算法:

输入: 训练集 $D = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_m, y_m)\}$;
初级学习算法 $\mathfrak{L}_1, \mathfrak{L}_2, \dots, \mathfrak{L}_T$;
次级学习算法 \mathfrak{L} .

过程:

```
1: for  $t = 1, 2, \dots, T$  do
2:    $h_t = \mathfrak{L}_t(D)$ ;
3: end for
4:  $D' = \emptyset$ ;
5: for  $i = 1, 2, \dots, m$  do
6:   for  $t = 1, 2, \dots, T$  do
7:      $z_{it} = h_t(\mathbf{x}_i)$ ;
8:   end for
9:    $D' = D' \cup ((z_{i1}, z_{i2}, \dots, z_{iT}), y_i)$ ;
10: end for
11:  $h' = \mathfrak{L}(D')$ ;
```

输出: $H(\mathbf{x}) = h'(h_1(\mathbf{x}), h_2(\mathbf{x}), \dots, h_T(\mathbf{x}))$

图 8.9 Stacking 算法

7.5 多样性

7.5.1 误差-分歧分解

定义学习器 h_i 的分歧为:

$$A(h_i|x) = (h_i(x) - H(x))^2 \quad (7.22)$$

则集成的分歧:

$$\bar{A}(h|x) = \sum_{i=1}^T w_i A(h_i|x) = \sum_{i=1}^T w_i (h_i(x) - H(x))^2 \quad (7.23)$$

分歧项表明了个体学习器在样本 x 上的不一致性，即在一定程度上反映了个体学习器的多样性，那么个体学习器 h_i 和集成 H 的平方误差分别为：

$$\begin{aligned} E(h_i|x) &= (f(x) - h_i(x))^2 \\ E(H|x) &= (f(x) - H(x))^2 \end{aligned} \quad (7.24)$$

令 $\bar{E}(h|x) = \sum_{i=1}^T w_i \cdot E(h_i|x)$ 表示个体学习器误差的加权均值，有：

$$\bar{A}(h|x) = \bar{E}(h|x) - E(H|x) \quad (7.25)$$

令 $p(x)$ 表示样本的概率密度，则在全样本上有：

$$\begin{aligned} \sum_{i=1}^T w_i \int A(h_i|x)p(x)dx &= \\ \sum_{i=1}^T \int E(h_i|x)p(x)dx - \int E(H|x)p(x)dx \end{aligned} \quad (7.26)$$

其中 $E_i = \int E(h_i|x)p(x)dx$ 为全样本的泛化误差， $A_i = \int A(h_i|x)p(x)dx$ 位分歧项，而 $E = \int E(H|x)p(x)dx$ 为集成的泛化误差。

综上：

$$E = \bar{E} - \bar{A} \quad (7.27)$$

上式表明，个体学习器准确性越高，多样性越大，则集成越好。这就是集成-分期分解，但是这个理论中的 \bar{A} 不是一个可以直接操作的多样性度量，只适合进行回归学习而不是分类学习。

7.5.2 多样性的度量 (diversity measure)

多样性度量用于估算个体学习器的多样化程度，典型的做法是考虑个体分类器的两两相似/不相似性。

给定数据集 $D = \{(x_1, y_1), (x_2, y_2), \dots, (x_m, y_m)\}$ ，对于二分类的任务， $y_i \in \{-1, +1\}$ ，则分类器 h_i 和 h_j 预测结果如下表：

		$h_i = +1$	$h_i = -1$
$h_j = +1$	$h_j = +1$	a	c
	$h_j = -1$	b	d

其中: $a + b + c + d = m$, 那么:

- 不合度量 (disagreement measure)

$$dis_{ij} = \frac{b + c}{m} \quad (7.28)$$

其中 $dis_{ij} \in [0, 1]$, 值越大说明多样性越大。

- 相关系数 (correlation coefficient)

$$\rho_{ij} = \frac{ad - bc}{\sqrt{(a+b)(a+c)(c+d)(b+d)}} \quad (7.29)$$

$\rho_{ij} \in [-1, 1]$, 若 h_i 与 h_j 无关, 那么 $\rho_{ij} = 0$, 正相关 $\rho_{ij} > 0$, 负相关则 $\rho_{ij} < 0$

- Q -统计量 (Q -stastistic)

$$Q_{ij} = \frac{ad - bc}{ad + bc} \quad (7.30)$$

Q_{ij} 与相关系数 ρ_{ij} 相同, 且 $|Q_{ij}| \leq |\rho_{ij}|$

- κ -统计量 (κ -statistic)

$$\kappa = \frac{p_1 - p_2}{1 - p_2} \quad (7.31)$$

其中, p_1 是两个分类器取得一致的概率, p_2 是两个分类器偶然达成一致的概率。由下面公式进行估算

$$\begin{aligned} p_1 &= \frac{a + d}{m} \\ p_2 &= \frac{(a + b)(a + c) + (c + d)(b + d)}{m^2} \end{aligned} \quad (7.32)$$

7.5.3 多样性增强

原理是在学习过程中引入随机性, 常见的做法是对数据样本、输入属性、输出熟悉、算法参数进行扰动。

- 数据样本扰动

给定初始数据集, 进行随机采样, 对于不稳定学习器 (那些训练样本稍微有些改变就会导致学习器有显著变动, 如决策树、神经网络等)

- 输入属性扰动
一般采用随机子空间算法 (random subspace):

输入: 训练集 $D = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_m, y_m)\}$;

基学习算法 \mathfrak{L} ;

基学习器数 T ;

子空间属性数 d' .

过程:

```

1: for  $t = 1, 2, \dots, T$  do
2:    $\mathcal{F}_t = \text{RS}(D, d')$ 
3:    $D_t = \text{Map}_{\mathcal{F}_t}(D)$ 
4:    $h_t = \mathfrak{L}(D_t)$ 
5: end for

```

输出: $H(\mathbf{x}) = \arg \max_{y \in \mathcal{Y}} \sum_{t=1}^T \mathbb{I}(h_t(\text{Map}_{\mathcal{F}_t}(\mathbf{x})) = y)$

图 8.11 随机子空间算法

- 输出表示扰动

对输出进行扰动，比如随机改变一些样本的标记、引入纠错码等。

- 算法参数扰动

就是基学习器的调参

7.6 小结

Chapter 8

聚类 (clustering)

8.1 聚类任务

在无监督学习 (unsupervised learning) 中，训练样本的标记信息是未知的，目标是通过无标记的训练样本的学习来揭示数据的内在性质及规律，此类学习任务一般就是研究聚类任务了。

聚类是指将数据集中的样本划分成若干个通常不相交的子集，每个子集被称为一个簇 (cluster).

8.2 性能度量

一般来说我们希望进行聚类后‘簇内相似度 (intra-cluster similarity)’高同时‘簇间相似度 ()’

性能度量一般分为两类：

- 外部指标 (external index): 即将聚类结果与某个外部模型进行比较。
- 内部指标 (internal index): 只考察聚类结果而不利用任何模型。

假设数据集给出 $D = \{x_1, x_2, \dots, x_m\}$ 通过聚类给出的簇划分结果为 $C = \{C_1, C_2, \dots, C_n\}$, 参考模型给出的划分 $C^* = \{C_1^*, C_2^*, \dots, C_n^*\}, \lambda, \lambda^*$

分别表示对应的簇标记向量，将其两两配对：

$$\begin{aligned}
 a &= |SS|, SS = \{(x_i, x_j) | \lambda_i = \lambda_j, \lambda_i^* = \lambda_j^*, i < j\}, \\
 b &= |SD|, SD = \{(x_i, x_j) | \lambda_i = \lambda_j, \lambda_i^* \neq \lambda_j^*, i < j\}, \\
 c &= |DS|, DS = \{(x_i, x_j) | \lambda_i \neq \lambda_j, \lambda_i^* = \lambda_j^*, i < j\}, \\
 d &= |DD|, DD = \{(x_i, x_j) | \lambda_i \neq \lambda_j, \lambda_i^* \neq \lambda_j^*, i < j\}, \\
 a + b + c + d &= \frac{m(m-1)}{2}
 \end{aligned} \tag{8.1}$$

- 聚类的性能度量外部指标：

- Jaccard 系数：

$$JC = \frac{a}{a + b + c} \tag{8.2}$$

- FM 指数：

$$FMI = \sqrt{\frac{a}{a+b} \cdot \frac{a}{a+c}} \tag{8.3}$$

- Rand 指数：

$$RI = \frac{2(a+d)}{m(m-1)} \tag{8.4}$$

上述三个指标都在 $[0, 1]$ 区间，值越大越好。

- 聚类的性能度量内部指标：首先定义几个变量：

- $dist(\cdot, \cdot)$ 表示两个样本之间的距离；
- $avg(C)$ 表示簇 C 之间的样本的平均距离。

$$avg(C) = \frac{2}{|C|(|C|-1)} \sum_{1 \leq i < j \leq |C|} dist(x_i, x_j) \tag{8.5}$$

- $diam(C)$ 表示簇 C 内样本的最远距离：

$$diam(C) = \max_{1 \leq i < j \leq |C|} dist(x_i, x_j) \tag{8.6}$$

- $d_{\min}(C_i, C_j)$, 对应簇 C_i 到 C_j 最近样本间距：

$$d_{\min}(C_i, C_j) = \min_{x_i \in C_i, x_j \in C_j} dist(x_i, x_j) \tag{8.7}$$

- $d_{\text{cen}}(C_i, C_j)$, 表示簇 C_i 到 C_j 中心点的距离。

$$d_{\text{cen}}(C_i, C_j) = \text{dist}(\mu_i, \mu_j) \quad (8.8)$$

其中 $\mu = \frac{1}{|C|} \sum_{1 \leq i \leq |C|}$, 表示簇 C 的中心点

那么我们可以推出聚类的性能度量内部指标:

- DB 指数:

$$DBI = \frac{1}{k} \sum_{i=1}^k \max_{j \neq i} \left(\frac{\text{avg}(C_i) + \text{avg}(C_j)}{d_{\text{cen}}(\mu_i, \mu_j)} \right) \quad (8.9)$$

- Dunn 指数:

$$DI = \min_{1 \leq i \leq k} \left\{ \min_{j \neq i} \left(\frac{d_{\min}(C_i, C_j)}{\max_{1 \leq l \leq k} \text{diam}(C_l)} \right) \right\} \quad (8.10)$$

显然 DBI 越小越好而 DI 越大越好。

8.3 距离计算

函数 $\text{dist}(\cdot, \cdot)$, 记录了两个样本上的距离, 需满足以下的性质:

- 非负性: $\text{dist}(x_i, x_j) \geq 0$;
- 同一性: $\text{dist}(x_i, x_j) = 0$ 当且仅当 $x_i = x_j$;
- 对称性: $\text{dist}(x_i, x_j) = \text{dist}(x_j, x_i)$;
- 直递性: $\text{dist}(x_i, x_j) \leq \text{dist}(x_i, x_k) + \text{dist}(x_k, x_j)$

对于给定样本 $x_i = (x_{i1}; x_{i2}; \dots; x_{in})$ 和 $x_j = (x_{j1}; x_{j2}; \dots; x_{jn})$, 计算两点最常用的距离是‘闵可夫斯基距离 (Minkowski distance,MK)’

$$\text{dist}_{mk}(x_i, x_j) = \left(\sum_{u=1}^n |x_{iu} - x_{ju}|^p \right)^{\frac{1}{p}} \quad (8.11)$$

当 $p = 2$, 即二维向量时, mk 距离即为欧氏距离 (Euclidean distance):

$$dist_{ed}(x_i, x_j) = \|x_i - x_j\|_{p=2} = \sqrt{\sum_{u=1}^n |x_{iu} - x_{ju}|^2} \quad (8.12)$$

当 $p = 1$, 即曼哈顿距离 (Manhattan distance):

$$dist_{man}(x_i, x_j) = \|x_i - x_j\|_{p=1} = \sum_{u=1}^n |x_{iu} - x_{ju}| \quad (8.13)$$

mk 距离仅适合如 $\{1, 2, 3\}$ 之类的有序属性, 对于无序属性采用 VDM(Value Difference Metric) 计算距离;

令 $m_{u,a}$ 表示在属性 u 上取值为 a 的样本数, $m_{u,a,i}$ 表示在第 i 个样本簇中在属性 u 上取值为 a 的样本数, k 为样本簇数, 那么属性 u 上的两个离散值 a, b 之间的 VDM 距离为:

$$VDM_p(a, b) = \sum_{i=1}^k \left| \frac{m_{u,a,i}}{m_{u,a}} - \frac{m_{u,b,i}}{m_{u,b}} \right|^p \quad (8.14)$$

结合两种计算距离的算法, 假设有 n_c 个有序属性, $n - n_c$ 个无序属性, 令有序属性排列在无序属性之前, 则:

$$\begin{aligned} MinkovDM_p(x_i, x_j) = \\ \left(\sum_{u=1}^{n_c} |x_{iu} - x_{ju}|^p + \sum_{u=n_c+1}^n VDM_p(x_{iu}, x_{ju}) \right)^{\frac{1}{p}} \end{aligned} \quad (8.15)$$

当样本空间中不同属性的重要性不同, 可使用‘加权距离’。

8.4 原型聚类

聚类的常用算法, 先对原型进行初始化, 然后对原型进行更新迭代来进行求解。

8.4.1 k 均值算法 (k-means)

给定样本集 $D = \{x_1, x_2, \dots, x_m\}$, k-means 针对聚类所得簇划分 $C = \{C_1, C_2, \dots, C_k\}$ 的最小化平方误差为:

$$E = \sum_{i=1}^k \sum_{x \in C_i} \|x - \mu_i\|_2^2 \quad (8.16)$$

其中 $\mu_i = \frac{1}{|C_i|} \sum_{x \in C_i} x$, E 的值越小说明簇内样本相似度越高:

Algorithm 1 k-means 算法

In: 样本集 $D = \{x_1, x_2, \dots, x_m\}$; 聚类簇数 k ;

Out: 簇划分 $C = \{C_1, C_2, \dots, C_k\}$;

```
1: function K-MEANS( $D, k$ )
2:   从  $D$  中随机选择  $k$  个样本作为初始的均值向量  $\{\mu_1, \mu_2, \dots, \mu_k\}$ ;
3:   repeat
4:     令  $C_i = \emptyset (1 \leq i \leq k)$ ;
5:     for  $j = 1, 2, \dots, m$  do
6:       计算样本  $x_j$  与各均值向量  $\mu_i (1 \leq i \leq k)$  的距离:  $d_{ji} = \|x_j - \mu_i\|_2$ .
7:       根据距离最近的均值向量确定  $x_j$  的簇标记:  $\lambda_j = \arg \min_{i \in \{1, 2, \dots, k\}} d_{ji}$ ;
8:       将样本  $x_j$  划入相应的簇:  $C_{\lambda_j} = C_{\lambda_j};$ 
9:     end for
10:    for  $i = 1, 2, \dots, k$  do
11:      计算新的均值向量:  $\mu'_i = \frac{1}{|C_i|} \sum_{x \in C_i} x$ ;
12:      if  $\mu'_i \neq \mu_i$  then
13:        将当前均值向量  $\mu_i$  更新为  $\mu'_i$ ;
14:      else
15:        保持当前均值向量不变;
16:      end if
17:    end for
18:    until 当前的均值向量均未更新
19: end function
```

8.4.2 学习向量量化

跟 k-means 算法类似，学习向量量化 (Learning Vector Quantization, LVQ) 试图找到一组原型向量来刻画聚类结构，但是与一般的聚类算法不同的是，LVQ 假设样本带有类别标记，学习的过程利用样本的监督信息来辅助聚类。算法如下图：

输入: 样本集 $D = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_m, y_m)\}$;
 原型向量个数 q , 各原型向量预设的类别标记 $\{t_1, t_2, \dots, t_q\}$;
 学习率 $\eta \in (0, 1)$.

过程:

- 1: 初始化一组原型向量 $\{\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_q\}$
- 2: **repeat**
- 3: 从样本集 D 随机选取样本 (\mathbf{x}_j, y_j) ;
- 4: 计算样本 \mathbf{x}_j 与 \mathbf{p}_i ($1 \leq i \leq q$) 的距离: $d_{ji} = \|\mathbf{x}_j - \mathbf{p}_i\|_2$;
- 5: 找出与 \mathbf{x}_j 距离最近的原型向量 p_{i^*} , $i^* = \arg \min_{i \in \{1, 2, \dots, q\}} d_{ji}$;
- 6: **if** $y_j = t_{i^*}$ **then**
- 7: $\mathbf{p}' = \mathbf{p}_{i^*} + \eta \cdot (\mathbf{x}_j - \mathbf{p}_{i^*})$
- 8: **else**
- 9: $\mathbf{p}' = \mathbf{p}_{i^*} - \eta \cdot (\mathbf{x}_j - \mathbf{p}_{i^*})$
- 10: **end if**
- 11: 将原型向量 \mathbf{p}_{i^*} 更新为 \mathbf{p}'
- 12: **until** 满足停止条件

输出: 原型向量 $\{\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_q\}$

图 9.4 学习向量量化算法

8.4.3 高斯混合聚类

高斯混合聚类采用的是概率模型来表达聚类模型, 而 k-means 与 LVQ 是采用原型向量来刻画的。

对 n 维样本空间 \mathcal{X} 的随机向量为 x , 若 x 服从高斯分布, 其密度函数:

$$p(x) = \frac{1}{(2\pi)^{\frac{n}{2}} |\Sigma|^{\frac{1}{2}}} e^{-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)} \quad (8.17)$$

其中 μ 是 n 维均值向量, Σ 是 $n \times n$ 的协方差矩阵, 将概率密度函数记为 $p(x|\mu, \Sigma)$. 我们可以定义高斯混合分布为:

$$p_{\mathcal{M}}(x) = \sum_{i=1}^k \alpha_i \cdot p(x|\mu_i, \Sigma_i), \quad (8.18)$$

该分布由 k 个混合分布组成, 每个混合分布对应相应的一个高斯分布, 其中 μ_i 与 Σ_i 是第 i 个高斯混合成分的参数。而 $\alpha_i > 0$ 为对应的混合系数 (mixture coefficient)。

那么样本的生成过程可以由高斯分布给出：

1. 根据 $\alpha_1, \alpha_2, \dots, \alpha_k$ 定义的先验分布选择高斯混合成分，
2. 根据被选择的混合成分的概率密度函数进行采样并且生成对应的样本。

若训练集 $D = \{x_1, x_2, \dots, x_m\}$ 由上述过程进行生成，令随机变量 $z_j \in \{1, 2, \dots, k\}$ 表示样本 x_j 的高斯混合成分，那么 z_j 的后验分布为：

$$p_{\mathcal{M}}(z_j = i|x_j) = \frac{\alpha_i \cdot p(x_j|\mu_i, \sum_i)}{\sum_{l=1}^k \alpha_l \cdot p(x_j|\mu_l, \sum_i)} \quad (8.19)$$

将 $p_{\mathcal{M}}(z_j = i|x_j)$ 简记为 γ_{ji} ($i = 1, 2, \dots, k$)，当高斯混合分布已知时，高斯混合聚类将样本集 D 划分为 k 个簇 $C = \{C_1, C_2, \dots, C_k\}$ 每个样本 x_j 的簇标记记为 λ_j 如下确定：

$$\lambda_j = \arg \max_{i \in \{1, 2, \dots, k\}} \gamma_{ji} \quad (8.20)$$

对于模型参数 $\{(\alpha_i, \mu_i, \sum_i) | 1 \leq i \leq k\}$ 可以采用极大似然估计，最大化对数似然：

$$LL(D) = \ln \left(\prod_{j=1}^m p_{\mathcal{M}}(x_j) \right) = \sum_{j=1}^m \ln \left(\sum_{i=1}^k \alpha_i \cdot p(x_j|\mu_i, \sum_i) \right) \quad (8.21)$$

令 $\frac{\partial LL(D)}{\partial \mu_i} = 0$ ，有：

$$\sum_{j=1}^m \frac{\alpha_i \cdot p(x_j|\mu_i, \sum_i)}{\sum_{l=1}^k \alpha_l \cdot p(x_j|\mu_l, \sum_l)} (x_j - \mu_i) = 0 \quad (8.22)$$

由 8.19 及 $\gamma_{ji} = p_{\mathcal{M}}(z_j = i|x_j)$ ，有：

$$\mu_i = \frac{\sum_{j=1}^m \gamma_{ji} x_j}{\sum_{j=1}^m \gamma_{ji}} \quad (8.23)$$

令： $\frac{\partial LL(D)}{\partial \sum_i} = 0$ 得：

$$\sum_i = \frac{\sum_{j=1}^m \gamma_{ji} (x_j - \mu_i)(x_j - \mu_i)^T}{\sum_{j=1}^m \gamma_{ji}} \quad (8.24)$$

对于混合系数 α_i , 考虑 $LL(D)$ 的拉格朗日形式:

$$LL(D) + \lambda \left(\sum_{i=1}^k \alpha_i - 1 \right) \quad (8.25)$$

其中 λ 为拉格朗日乘子, 取上式对 α_i 的偏导为 0, 有:

$$\sum_{j=1}^m \frac{p(x_j | \mu_i, \Sigma_i)}{\sum_{l=1}^k \alpha_l \cdot p(x_j | \mu_l, \Sigma_l)} + \lambda = 0 \quad (8.26)$$

两边同时乘 α_i 并对所有样本进行求和可知 $\lambda = -m$, 有:

$$\alpha_i = \frac{1}{m} \sum_{j=1}^m \gamma_{ji} \quad (8.27)$$

综上可以推导出高斯混合模型的 EM 算法, 算法流程如下图所示:

- E 步: 根据当前参数来计算每个样本属于每个高斯成分的后验概率 γ_{ji}
- M 步: 更新模型参数 $\{(\alpha_i, \mu_i, \Sigma_i) | 1 \leq i \leq k\}$

输入: 样本集 $D = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_m\}$;
高斯混合成分个数 k .

过程:

- 1: 初始化高斯混合分布的模型参数 $\{(\alpha_i, \mu_i, \Sigma_i) \mid 1 \leq i \leq k\}$
- 2: **repeat**
- 3: **for** $j = 1, 2, \dots, m$ **do**
- 4: 根据式(9.30)计算 \mathbf{x}_j 由各混合成分生成的后验概率, 即
 $\gamma_{ji} = p_M(z_j = i \mid \mathbf{x}_j)$ ($1 \leq i \leq k$)
- 5: **end for**
- 6: **for** $i = 1, 2, \dots, k$ **do**
- 7: 计算新均值向量: $\mu'_i = \frac{\sum_{j=1}^m \gamma_{ji} \mathbf{x}_j}{\sum_{j=1}^m \gamma_{ji}}$;
- 8: 计算新协方差矩阵: $\Sigma'_i = \frac{\sum_{j=1}^m \gamma_{ji} (\mathbf{x}_j - \mu'_i)(\mathbf{x}_j - \mu'_i)^T}{\sum_{j=1}^m \gamma_{ji}}$;
- 9: 计算新混合系数: $\alpha'_i = \frac{\sum_{j=1}^m \gamma_{ji}}{m}$;
- 10: **end for**
- 11: 将模型参数 $\{(\alpha_i, \mu_i, \Sigma_i) \mid 1 \leq i \leq k\}$ 更新为 $\{(\alpha'_i, \mu'_i, \Sigma'_i) \mid 1 \leq i \leq k\}$
- 12: **until** 满足停止条件
- 13: $C_i = \emptyset$ ($1 \leq i \leq k$)
- 14: **for** $j = 1, 2, \dots, m$ **do**
- 15: 根据式(9.31)确定 \mathbf{x}_j 的簇标记 λ_j ;
- 16: 将 \mathbf{x}_j 划入相应的簇: $C_{\lambda_j} = C_{\lambda_j} \cup \{\mathbf{x}_j\}$
- 17: **end for**

输出: 簇划分 $\mathcal{C} = \{C_1, C_2, \dots, C_k\}$

图 9.6 高斯混合聚类算法

8.5 密度聚类 (density-based clustering)

密度聚类假定聚类结构能够通过样本分布的紧密程度进行确定。

DBSCAN 是一种著名的密度聚类算法, 它基于一组领域参数 ϵ 来刻画样本分布的紧密程度, 首先定义几个概念, 给定数据集 $D = \{x_1, x_2, \dots, x_m\}$:

- ϵ -邻域: 对 $x_j \in D$, 其 ϵ -邻域包含样本集 D 中与 X_j 距离不大于 ϵ 的样本, 即 $N_\epsilon(x_j) = \{x_i \in D \mid dist(x_i, x_j) \leq \epsilon\}$;
- 核心对象 (core object): 若 x_j 的 ϵ -邻域至少包含 $MinPts$ 个样本。即 $|N_\epsilon(x_j)| \geq MinPts$, 则 x_j 是一个核心对象。

- 密度直达 (directly density-reachable): 若 x_j 位于 x_i 的 ϵ -邻域中, 且 x_i 是核心对象, 则称 x_j 由 x_i 密度直达;
- 密度可达 (density-reachable): 对 x_i 与 x_j , 若存在样本序列 p_1, p_2, \dots, p_n , 其中 $p_1 = x_i, p_n = x_j$ 且 p_{i+1} 由 p_i 密度直达, 则称 x_j 由 x_i 密度可达;
- 密度相连 (density-connected): 对于 x_i 与 x_j , 若存在 x_k 使得 x_i 与 x_j 均有 x_k 密度可达, 那么称为 x_i 与 x_j 密度相连。

如下图所示:

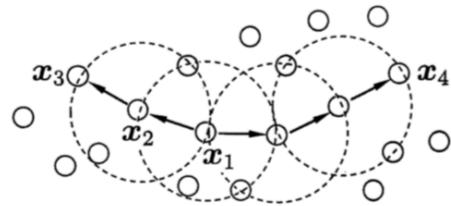


图 9.8 DBSCAN 定义的基本概念($MinPts = 3$): 虚线显示出 ϵ -邻域, x_1 是核心对象, x_2 由 x_1 密度直达, x_3 由 x_1 密度可达, x_3 与 x_4 密度相连.

基于如上的概念, DBSCAN 将簇定义为: 由密度可达关系导出的最大密度相连的样本集合。形式化来说, 给定邻域参数 $(\epsilon, MinPts)$, 簇 $C \subseteq D$ 满足以下两个性质的非空子集:

1. 连续性 (connextivity): $x_i \in C, x_j \in C \Rightarrow x_i$ 与 x_j 密度相连
2. 最大性 (maxmality): $x_i \in C, x_j$ 由 x_i 密度可达 $\Rightarrow x_j \in C$

算法流程如下图进行:

输入: 样本集 $D = \{x_1, x_2, \dots, x_m\}$;
邻域参数 $(\epsilon, MinPts)$.

过程:

- 1: 初始化核心对象集合: $\Omega = \emptyset$
- 2: **for** $j = 1, 2, \dots, m$ **do**
- 3: 确定样本 x_j 的 ϵ -邻域 $N_\epsilon(x_j)$;
- 4: **if** $|N_\epsilon(x_j)| \geq MinPts$ **then**
- 5: 将样本 x_j 加入核心对象集合: $\Omega = \Omega \cup \{x_j\}$
- 6: **end if**
- 7: **end for**
- 8: 初始化聚类簇数: $k = 0$
- 9: 初始化未访问样本集合: $\Gamma = D$
- 10: **while** $\Omega \neq \emptyset$ **do**
- 11: 记录当前未访问样本集合: $\Gamma_{old} = \Gamma$;
- 12: 随机选取一个核心对象 $o \in \Omega$, 初始化队列 $Q = < o >$;
- 13: $\Gamma = \Gamma \setminus \{o\}$;
- 14: **while** $Q \neq \emptyset$ **do**
- 15: 取出队列 Q 中的首个样本 q ;
- 16: **if** $|N_\epsilon(q)| \geq MinPts$ **then**
- 17: 令 $\Delta = N_\epsilon(q) \cap \Gamma$;
- 18: 将 Δ 中的样本加入队列 Q ;
- 19: $\Gamma = \Gamma \setminus \Delta$;
- 20: **end if**
- 21: **end while**
- 22: $k = k + 1$, 生成聚类簇 $C_k = \Gamma_{old} \setminus \Gamma$;
- 23: $\Omega = \Omega \setminus C_k$
- 24: **end while**

输出: 簇划分 $\mathcal{C} = \{C_1, C_2, \dots, C_k\}$

图 9.9 DBSCAN 算法

8.6 层次聚类 (hierarchical clustering)

层次聚类试图在不同层次上对数据集进行划分, 从而形成树形的聚类结构。

AGNES 是一种采用自底向上聚合策略的层次聚类算法。他先将数据集中的每一个样本看做一个初始聚类簇，然后在算法运行的每一步中找出最近的两个聚类簇进行合并，直到聚类簇个数减小到阈值。定义下面三种聚类簇之间的距离：

- 最小距离: $d_{\min}(C_i, C_j) = \min_{x \in C_i, z \in C_j} dist(x, z)$
- 最大距离: $d_{\max}(C_i, C_j) = \max_{x \in C_i, z \in C_j} dist(x, z)$
- 平均距离: $d_{avg}(C_i, C_j) = \frac{1}{|C_i||C_j|} \sum_{x \in C_i} \sum_{z \in C_j} dist(x, z)$

综上 AGNES 算法可以定义出如下流程图：

输入: 样本集 $D = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_m\}$;
聚类簇距离度量函数 d ;
聚类簇数 k .

过程:

```
1: for  $j = 1, 2, \dots, m$  do
2:    $C_j = \{\mathbf{x}_j\}$ 
3: end for
4: for  $i = 1, 2, \dots, m$  do
5:   for  $j = 1, 2, \dots, m$  do
6:      $M(i, j) = d(C_i, C_j)$ ;
7:      $M(j, i) = M(i, j)$ 
8:   end for
9: end for
10: 设置当前聚类簇个数:  $q = m$ 
11: while  $q > k$  do
12:   找出距离最近的两个聚类簇  $C_{i^*}$  和  $C_{j^*}$ ;
13:   合并  $C_{i^*}$  和  $C_{j^*}$ :  $C_{i^*} = C_{i^*} \cup C_{j^*}$ ;
14:   for  $j = j^* + 1, j^* + 2, \dots, q$  do
15:     将聚类簇  $C_j$  重编号为  $C_{j-1}$ 
16:   end for
17:   删除距离矩阵  $M$  的第  $j^*$  行与第  $j^*$  列;
18:   for  $j = 1, 2, \dots, q - 1$  do
19:      $M(i^*, j) = d(C_{i^*}, C_j)$ ;
20:      $M(j, i^*) = M(i^*, j)$ 
21:   end for
22:    $q = q - 1$ 
23: end while
```

输出: 簇划分 $\mathcal{C} = \{C_1, C_2, \dots, C_k\}$

图 9.11 AGNES 算法

8.7 小结

Chapter 9

降维与度量学习

9.1 k 近邻学习

k 近邻学习 (k -Nearest Neighbor, k NN)，是一种常用的监督学习方法，其原理很简单，就是看测试集中离输入样本 x 中最近的 k 个样本中分类最多的一个，将其结果作为 x 的分类结果进行返回。

k 近邻算法的是懒惰学习 (lazy learning) 的代表，此类算法的最大特点就是在训练阶段仅仅将样本保存起来，训练时间的开销为 0，在收到测试样本之后再进行相应的处理，那么那些在训练阶段就对样本进行学习的处理方法称为“急切学习”。

9.2 低维嵌入

高维空间会给计算距离造成大量的麻烦，我们将其称为维数灾难。

缓解维数灾难的一个重要途径就是进行降维 (dimension reduction)，若要求原始空间中样本的距离在低维空间中继续得以维持，即“多维缩放”，(Multiple Dimensional Scaling,MDS):

假定 m 个样本在原始的 d 维空间的距离矩阵为 $D \in \mathbb{R}^{m \times m}, dist_{ij}$ 指样本 x_i 和 x_j 的距离。我们的目标是要获得样本在 d' 维空间的表示 $Z \in \mathbb{R}^{d' \times m}, d' \leq d$ ，且任意两个样本 x_i 和 x_j 在 d 维空间和 d' 维空间之间的距

离相同。

令 $B = Z^T Z \in \mathbb{R}^{m \times m}$ 表示降维后的样本内积矩阵, $b_{ij} = z_i^T z_j$ 有,

$$dist_{ij}^2 = \|z_i^2\| + \|z_j^2\| - 2z_i^T z_j = b_{ii} + b_{jj} - 2b_{ij} \quad (9.1)$$

令

$$\begin{aligned} dist_{i\cdot}^2 &= \frac{1}{m} \sum_{j=1}^m dist_{ij}^2 \\ dist_{\cdot j}^2 &= \frac{1}{m} \sum_{i=1}^m dist_{ij}^2 \\ dist_{\cdot \cdot}^2 &= \frac{1}{m^2} \sum_{i=1}^m \sum_{j=1}^m dist_{ij}^2 \end{aligned} \quad (9.2)$$

综上可得:

$$b_{ij} = -\frac{1}{2}(dist_{ij}^2 - dist_{i\cdot}^2 - dist_{\cdot j}^2 + dist_{\cdot \cdot}^2) \quad (9.3)$$

上述公式说明可以通过降维前后保持不变的距离矩阵 D 求内积矩阵 B . 对矩阵 B 进行特征值分解, 有 $b = V \Lambda V^T$, 其中 $\Lambda = diag(\lambda_1, \lambda_2, \dots, \lambda_d)$ 特征值构成的对角矩阵。且满足

$$\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_d \quad (9.4)$$

V 为特征向量矩阵, 假定其中有 d^* 个非零特征值, 它们构成对角矩阵 $\Lambda - diag(\lambda_1, \lambda_2, \dots, \lambda_{d^*})$, 令 V_* 表示对应的特征向量矩阵, 那么 Z 可以表达为:

$$Z = \Lambda_*^{\frac{1}{2}} V_*^T \in \mathbb{R}^{d^* \times m} \quad (9.5)$$

有时候为了实现有效的降维, 降维后的距离只需尽可能地接近原始空间的距离。此时去 $d' \ll d$ 个最大特征值构成的对角矩阵 $\tilde{\Lambda} = diag(\lambda_1, \lambda_2, \dots, \lambda_{d'})$, 令 \tilde{V} 表示相应的特征向量矩阵, 那么将 Z 表示为:

$$Z = \tilde{\Lambda}^{\frac{1}{2}} \tilde{V}^T \in \mathbb{R}^{d' \times m} \quad (9.6)$$

Algorithm 2 MDS

Input: $D, dist_{ij}, d'$;

Output: 矩阵 $\tilde{V} \tilde{\Lambda}^{\frac{1}{2}} \in \mathbb{R}^{m \times d'}$

- 1: **function** MDS($D, dist_{ij}, d'$)
 - 2: 根据公式 9.2 计算 $dist_{\cdot i}^2, dist_{\cdot j}^2, dist^2_{\cdot \cdot}$;
 - 3: 根据公式 9.3 计算矩阵 B ;
 - 4: 对矩阵 B 做特征值分解;
 - 5: 取 $\tilde{\Lambda}$ 为 d' 个最大特征值构成的对角矩阵, \tilde{V} 为对应的特征向量矩阵;
 - 6: $Z = \tilde{V} \tilde{\Lambda}^{\frac{1}{2}} \in \mathbb{R}^{m \times d'}$;
 - 7: **end function**
-

一般来说, 想要获得低维的子空间, 最简单的方式就是对原始的高维空间进行线性变换。给定 d 维空间中的样本 $X = (x_1, x_2, \dots, x_m) \in \mathbb{R}^{d \times m}$, 那么变化后的样本为:

$$Z = W^T X \quad (9.7)$$

其中 $W \in \mathbb{R}^{d \times d'}$ 是变换矩阵, $Z \in \mathbb{R}^{d' \times m}$ 是样本在新空间中的表达。

给予线性变化来进行降维的方法称为线性降维方法。

9.3 主成分分析 (Principal Component Analysis,PCA)

PCA 是一种常用的降维方法, 我们设想存在一个超平面, 对于所有的样本:

- 最大重构性: 样本点到这个超平面的距离都足够近
- 最大可分性: 样本点在这个超平面上的投影能尽可能的分开

我们先从最大重构性进行推导。

假定数据样本已经进行了中心化, 即 $\sum_i x_i = 0$; 再假定投影变化后的新坐标系为 $\{w_1, w_2, \dots, w_d\}$, 其中 $\|w_i\|_2 = 1, w_i^T w_j = 0 (i \neq j)$, 若丢弃新坐标系中的部分坐标, 即将维度降到 $d' < d$, 那么 x_i 在低维坐标系中的投

影是 $z_i = (z_{i1}; z_{i2}; \dots; z_{id'})$, 其中 $z_{ij} = w_j^T x_i$ 指的是 x_i 在低维坐标系下第 j 维的坐标, 若基于 z_i 来重构 x_i , 则有 $\hat{x}_i = \sum_{j=1}^{d'} z_{ij} w_j$. 那么, 原样本点 x_i 与基于投影重构的样本点 \hat{x}_i 的距离为:

$$\begin{aligned} & \sum_{i=1}^m \left\| \sum_{j=1}^{d'} z_{ij} w_j - x_i \right\|_2^2 = \\ & \sum_{i=1}^m z_i^T z_i - 2 \sum_{i=1}^m z_i^T W^T x_i + \text{const} \propto -\text{tr} \left(W^T \left(\sum_{i=1}^m x_i x_i^T \right) W \right) \end{aligned} \quad (9.8)$$

根据最近重构性, 上式应该要被最小化, 考虑到 w_j 是标准正交基, 而 $\sum_i x_i x_i^T$ 是协方差矩阵, 有:

$$\min_W -\text{tr}(W^T X X^T W) \quad s.t. \quad W^T W = I \quad (9.9)$$

上面这个式子就是主成分分析的主要优化目标。为了使得样本点的投影能够尽可能的分开, 则应该使得投影后的样本点的方差能够最大化。假设样本投影后的方差是 $\sum_i W^T x_i x_i^T W$, 于是优化目标可以写为

$$\max_W \text{tr}(W^T X X^T W) \quad s.t. \quad W^T W = I \quad (9.10)$$

对上式采用拉格朗日乘子法可得:

$$X X^T W = \lambda W \quad (9.11)$$

只需对协方差矩阵 $X X^T$ 进行特征值分解, 再将求得的特征值进行排序: $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_d$, 再取前 d' 个特征值对应的特征向量构成 $W = (w_1, w_2, \dots, w_{d'})$, 即是主成分分析的解了。算法流程图可以描述如下:

Algorithm 3 PCA

Input: D, d' ;

Output: 投影矩阵 $W = (w_1, w_2, \dots, w_{d'})$;

- 1: **function** PCA(D, d')
 - 2: 对所有样本进行中心化: $x_i \leftarrow x_i - \frac{1}{m} \sum_{i=1}^m x_i$;
 - 3: 计算样本的协方差矩阵 $X X^T$;
 - 4: 对协方差矩阵 $X X^T$ 做特征值分解;
 - 5: 取最大的 d' 个特征值所对应的特征向量 $(w_1, w_2, \dots, w_{d'})$;
 - 6: **end function**
-

9.4 核化线性降维

对应的是将高维空间经过非线性的的映射到低维空间。

借鉴前面 SVM 的核函数方法，我们可以将转化矩阵进行核化，假设我们将在高维特征空间中把数据投影到由 W 确定的超平面上，即欲求解：

$$\left(\sum_{i=1}^m z_i z_i^T \right) = \lambda W \quad (9.12)$$

求出：

$$W = \frac{1}{\lambda} \left(\sum_{i=1}^m z_i z_i^T \right) = \sum_{i=1}^m z_i \frac{z_i^T W}{\lambda} = \sum_{i=1}^m z_i \alpha_i \quad (9.13)$$

其中 z_i 是样本点 x_i 在高维特征空间中的像，是由 x_i 通过映射 ϕ 映射产生的，那么可以改写 9.12 为核函数公式：

$$\left(\sum_{i=1}^m \phi(x_i) \phi(x_i)^T \right) W = \lambda W \quad (9.14)$$

再将 9.13 改写成：

$$W = \sum_{i=1}^m \phi(x_i) \alpha_i \quad (9.15)$$

核函数可以写为：

$$\kappa(x_i, x_j) = \phi(x_i)^T \phi(x_j) \quad (9.16)$$

综上：

$$\mathcal{K}A = \lambda A \quad (9.17)$$

其中 \mathcal{K} 是 κ 对应的核矩阵，显然上式是一个特征值分解问题，只需取 \mathcal{K} 的最大的 d' 个特征值对应的特征向量即可。

那么对于一个新的样本 x ，其投影之后的第 $j (j = 1, 2, \dots, d')$ 维坐标为：

$$z_j = \sum_{i=1}^m \alpha_i^j \kappa(x_i, x) \quad (9.18)$$

上述的公式说明，为了获得投影后的坐标，KPCA 需要对所有的样本进行求和，具有很大的开销。

9.5 流形学习 (manifold learning)

流形学习是一类借鉴了拓扑流形结构的降维方法，它在局部具有欧氏空间的性质，能够用欧式距离进行计算。若能够将低维流形嵌入高维空间中，就可以让高维空间的局部具有欧氏空间的性质，容易建立局部的降维映射，然后再将局部推广到全局中。

9.5.1 等度量映射 (Isometric Mapping, Isomap)

其思想是对于每个点基于欧式距离找出其邻近点，建立一个近邻连接图，那么计算问题就将转化为计算近邻连接图上两点的最短路径问题了。算法流程如下：

Algorithm 4 Isomap

Input: D, d' , 近邻参数 k ;

Output: D 在低维空间的投影 $Z = \{z_1, z_2, \dots, z_m\}$;

```
1: function ISOMAP( $D, d', k$ )
2:   for  $i = 1, 2, \dots, m$  do
3:     确定  $x_i$  的近邻;
4:      $x_i$  与  $k$  近邻点之间的距离设置为欧式距离，与其他点的距离设置
      成无穷大;
5:   end for
6:   采用 Dijkstra 或者 Floyd 等算法计算任意两个样本之间的距离
       $dist(x_i, x_j)$ ;
7:   将  $dist(x_i, x_j)$  作为 MDS 算法的输入;
8:   return MDS 算法的输出  $Z$ ;
9: end function
```

我们可以将训练样本的高维空间坐标作为输入，低维的空间坐标作为输出，训练一个回归学习器来对新样本进行一个预测，这是目前比较常用的方法。

对于如何构建近邻图，通常有两种办法：

- 指定近邻点的个数 k , 以最近的 k 个点作为近邻点, 这样的图是 k 近邻图。
- 另一种是指定阈值 ϵ , 所有小于 ϵ 的点都会被认为是近邻点, 这样的图被称为 ϵ 近邻图。

两种办法都存在一些问题, 如果范围确定得太大, 那么会把距离较远得点也算进来, 造成”短路”, 如果范围确定得太小, 则会造成”断路”, 从而损失掉一定的信息。

9.5.2 局部线性嵌入 (Locally Linear Embedding,LLE)

LLE 试图保存邻域内样本之间的线性关系, 假设样本点 x_i 得坐标能够通过它的邻域样本 x_j, x_k 的坐标线性组合而重构得来的, 即:

$$x_i = w_{ij}x_j + w_{ik}x_k \quad (9.19)$$

LLE 希望在低维的空间中仍然能够保持这个趋势。首先对于每个样本 x_i 先找到它的邻近下标集合 Q_i , 然后计算出基于 Q_i 中的样本点对于 x_i 进行线性重构的系数 w_i ;

$$\min_{w_1, w_2, \dots, w_m} \sum_{i=1}^m \|x_i - \sum_{j \in Q_i} w_{ij} x_j\|_2^2 \quad s.t. \quad \sum_{j \in Q_i} w_{ij} = 1 \quad (9.20)$$

令 $C_{jk} = (x_i - x_j)^T(x_i - x_k)$, 那么可以解出:

$$w_{ij} = \frac{\sum_{k \in Q_i} C_{jk}^{-1}}{\sum_{l, s \in Q_i} C_{ls}^{-1}} \quad (9.21)$$

那么 x_i 对应的低维空间的坐标 z_i 可以通过下面的式子进行求解:

$$\min_{z_1, z_2, \dots, z_m} \sum_{i=1}^m \|z_i - \sum_{j \in Q_i} w_{ij} z_j\|_2^2 \quad (9.22)$$

再令 $Z = (z_1, z_2, \dots, z_m) \in \mathbb{R}^{d' \times m}$, $(W)_{ij} = w_{ij}$, 有:

$$M = (I - W)^T(I - W) \quad (9.23)$$

重写 9.22:

$$\min_Z \operatorname{tr}(ZMZ^T) \quad s.t. \quad ZZ^T = I \quad (9.24)$$

其中 M 最小的 d' 个特征值对应的特征向量组成了矩阵 Z^T . 其算法如下图所示:

Algorithm 5 LLE

Input: $D, d',$ 近邻参数 $k;$

Output: D 在低维空间的投影 $Z = \{z_1, z_2, \dots, z_m\};$

```

1: function LLE( $D, d', k$ )
2:   for  $i = 1, 2, \dots, m$  do
3:     确定  $x_i$  的近邻;
4:     根据公式 9.20 和 9.21 求得  $w_{ij}, j \in Q_i;$ 
5:     若  $j \notin Q_i$ , 则  $w_{ij} = 0;$ 
6:   end for
7:   从公式 9.24 求出  $M;$ 
8:   对  $M$  进行特征值分解;
9:   return  $M$  的最小  $d'$  个特征值所对应的特征向量。
10: end function

```

9.6 度量学习

前面的算法的目的是找到一个适合的低维空间来尽可能的还原高维空间, 度量学习是学习如何制造一个距离度量来直接衡量降维的样本:

对于两个 d 维的样本 x_i 和 x_j , 它们之间的平方欧式距离可以写为:

$$\begin{aligned}
\operatorname{dist}_{wed}^2(x_i, x_j) &= \|x_i - x_j\|_2^2 \\
&= \sum_{k=1}^d w_k \operatorname{dist}_{ij,k}^2 \\
&= (x_i - x_j)^T W (x_i - x_j)
\end{aligned} \quad (9.25)$$

其中 $\operatorname{dist}_{ij,k}$ 指的是 x_i 和 x_j 在第 k 维的距离, w_k 指的是距离的权重。 W 是权重矩阵, $(W)_{ii} = w_i.$

W 所有的非对角元素都是 0, 说明 W 对应的属性都是正交不相关的, 这在实际的应用中是不合理的, 对此, 我们将 W 替换成一个普通的半正定对称矩阵 M , 有正交基 P 使得 $M = PP^T$:

$$dist_{mah}^2(x_i, x_j) = (x_i - x_j)^T M (x_i - x_j) = \|x_i - x_j\|_M^2 \quad (9.26)$$

上述式子就是马氏距离 (Mahalanobis distance)。度量学习就是对 M 进行一系列的学习。常用的方法称为近邻成分分析 (Neighbourhood Component Analysis,NCA), NCA 在进行判别时通常采用多数投票法, 若替换成概率投票法, 对于任意样本, 有 x_j , 它对 x_i 分类影响的概率为:

$$p_{ij} = \frac{\exp(-\|x_i - x_j\|_M^2)}{\sum_l \exp(-\|x_i - x_l\|_M^2)} \quad (9.27)$$

计算 x_i 的留一法正确率, 即它被自身之外的所有样本正确分类的概率为:

$$p_i = \sum_{j \in \Omega_i} p_{ij} \quad (9.28)$$

其中 Ω_i 表示与 x_i 属于相同类别的样本的下标集合, 故整个样本采用留一法的正确率可以表示为:

$$\sum_{i=1}^m p_i = \sum_{i=1}^m \sum_{j \in \Omega_i} p_{ij} \quad (9.29)$$

综上, 我们可以得出 NCA 的优化目标了:

$$\min_P \quad 1 - \sum_{i=1}^m \sum_{j \in \Omega_i} \frac{\exp(-\|P^T x_i - P^T x_j\|_2^2)}{\sum_l \exp(-\|P^T x_i - P^T x_l\|_2^2)} \quad (9.30)$$

9.7 小结

Chapter 10

计算学习理论 (Computational Learning Theory)

10.1 基础知识

计算学习理论是通过计算来进行学习的理论，即有关于机器学习的理论基础。

给定样例集 $D = \{(x_1, y_1), (x_2, y_2), \dots, (x_m, y_m)\}, x_i \in \mathcal{X}$ 。假设所有的样本都是独立同分布的样本，并且服从一个隐含未知的分布 \mathcal{D} 令 h 为 \mathcal{X} 到 \mathcal{Y} 的一个映射，

其泛化误差为：

$$E(h; \mathcal{D}) = P_{x \sim \mathcal{D}}(h(x) \neq y) \quad (10.1)$$

其经验误差为：

$$\hat{E}(h; D) = \frac{1}{m} \sum_{i=1}^m \mathbb{I}(h(x_i) \neq y_i) \quad (10.2)$$

由于 D 是 \mathcal{D} 的独立同分布采样，那么 h 的经验误差的期望等于其泛化的误差，令 ϵ 为 $E(h)$ 的上限，即 $E(h) \leq \epsilon$ ，我们将 ϵ 称为误差参数。

对于两个映射 $h_1, h_2 \in \mathcal{X} \rightarrow \mathcal{Y}$ 来度量他们之间的差别：

$$d(h_1, h_2) = P_{x \sim \mathcal{D}}(h_1(x) \neq h_2(x)) \quad (10.3)$$

下面将会介绍一些常用的不等式：

- Jensen 不等式：对于任意的凸函数 $f(x)$ 有：

$$f(\mathbb{E}(x)) \leq \mathbb{E}(f(x)) \quad (10.4)$$

- Hoeffding 不等式：若 x_1, x_2, \dots, x_m 为 m 个独立随机变量，且满足 $0 \leq x_i \leq 1$ ，那么对于任意 $\epsilon > 0$ 有：

$$\begin{aligned} P\left(\frac{1}{m} \sum_{i=1}^m x_i - \frac{1}{m} \sum_{i=1}^m \mathbb{E}(x_i) \geq \epsilon\right) &\leq \exp(-2m\epsilon^2), \\ P\left(\left|\frac{1}{m} \sum_{i=1}^m x_i - \frac{1}{m} \sum_{i=1}^m \mathbb{E}(x_i)\right| \geq \epsilon\right) &\leq 2 \exp(-2m\epsilon^2) \end{aligned} \quad (10.5)$$

- McDiarmid 不等式：若 x_1, x_2, \dots, x_m 为 m 个独立随机变量，且对任意 $1 \leq i \leq m$ ，函数 f 满足：

$$\sup_{x_1, \dots, x_m, x'_i} |f(x_1, \dots, x_m) - f(x_1, \dots, x_{i-1}, x'_i, x_{i+1}, \dots, x_m)| \leq c_i, \quad (10.6)$$

对于任意 $\epsilon > 0$ ，有：

$$\begin{aligned} P(f(x_1, \dots, x_m) - \mathbb{E}(f(x_1, \dots, x_m)) \geq \epsilon) &\leq \exp\left(\frac{-2\epsilon^2}{\sum_i c_i^2}\right) \\ P(|f(x_1, \dots, x_m) - \mathbb{E}(f(x_1, \dots, x_m))| \geq \epsilon) &\leq 2 \exp\left(\frac{-2\epsilon^2}{\sum_i c_i^2}\right) \end{aligned} \quad (10.7)$$

10.2 PAC 学习

计算学习理论中最基本的就是概率近似正确 (Probably Approximately Correct, PAC)，令 c 表示概念 (concept)，这样就完成了样本空间 \mathcal{X} 到标记空间 \mathcal{Y} 的映射，它决定了样本示例 x 的真实标记 y ，若对于任何样例，均有 $c(x) = y$ 成立，则称 c 为目标概念；我们用 \mathcal{C} 表示概念类。

给定了学习算法 \mathcal{L} ，它所考虑的所有可能概念的集合就被称为假设空间 (hypothesis space)，我们可以用符号 \mathcal{H} 进行表示，学习算法一般会把自认

为可能的目标概念集中起来构成 \mathcal{H} , 对于 $h \in \mathcal{H}$, 由于无法确定他是否是真的是目标概念, 因此可以将其称为假设。

给定训练集 D , 我们希望学习算法 \mathcal{L} 所得到的模型所对应的假设 h 应该尽可能接近目标概念 c , 我们希望以比较大的把握学得比较好的模型, 也就是说, 以较大的概率学得误差满足预设上限的模型; 形象地说, 令 δ 表示置信度, 定义:

PAC 辨识 (PAC Identify): 对 $0 < \epsilon, \delta < 1$, 所有的 $c \in \mathcal{C}$ 和分布 \mathcal{D} , 若存在学习算法 \mathcal{L} , 其输出假设 $h \in \mathcal{H}$ 满足:

$$P(E(h) \leq \epsilon) \geq 1 - \delta \quad (10.8)$$

那么称学习算法 \mathcal{L} 能从假设空间 \mathcal{H} 中辨识概念类 \mathcal{C} 。

这样的学习算法 \mathcal{L} 能以较大的概率 (至少 $1 - \delta$) 学得目标概念 c 的近似 (误差至多 ϵ)。为此我们定义:

PAC 可学习 (PAC Learnable): 令 m 表示从分布 \mathcal{D} 中独立同分布采样得到的样例数目, $0 < \epsilon, \delta < 1$, 对于所有分布 \mathcal{D} , 若存在学习算法 \mathcal{L} 和多项式函数 $poly(\cdot, \cdot, \cdot, \cdot)$, 使得对于任何 $m \geq poly(\frac{1}{\epsilon}, \frac{1}{\delta}, size(x), size(c))$, 若 \mathcal{L} 能够从假设空间 \mathcal{H} 中 PAC 辨识概念类 \mathcal{C} , 则称为概念类 \mathcal{C} 是可 PAC 可学习的。

PAC 学习算法 (PAC Learning Algorithm): 若学习算法 \mathcal{L} 使得概念类 \mathcal{C} 为 PAC 可学习的, 且 \mathcal{L} 的运行时间也是多项式函数 $poly(\frac{1}{\epsilon}, \frac{1}{\delta}, size(x), size(c))$, 则称概念类 \mathcal{C} 是高效 PAC 可学习的。

样本复杂度 (Sample Complexity): 满足 PAC 学习算法 \mathcal{L} 所需的 $m \geq poly(\frac{1}{\epsilon}, \frac{1}{\delta}, size(x), size(c))$ 中的最小 m , 那我们将其称为学习算法 \mathcal{L} 的样本复杂度。

在 PAC 学习中一个关键因素是假设空间 \mathcal{H} 的复杂度, 一般而言, \mathcal{H} 越大, 其包含任意目标概念的可能性将会越大, 但是从其中找到某个具体目标概念的难度也相应增大了。 $|\mathcal{H}|$ 有限时, 我们称 \mathcal{H} 为“有限假设空间”, 否则称为“无限假设空间”。

10.3 有限假设空间

10.3.1 可分情形

可分情形意味着目标概念 c 属于假设空间 \mathcal{H} , 即 $c \in \mathcal{H}$; 容易想到一种简单的学习策略, 对于任何训练集 D 上出现标记错误的假设肯定不是目标概念 c , 于是我们只需保留与 D 一致的假设, 剔除不一致的假设, 直到 \mathcal{H} 中仅剩下一个假设为止。但是 \mathcal{H} 中可能存在不止一个与 D 一致的等效假设, 这样的的话上面的算法就失效了。

对于上述情况, PAC 学习只要训练集 D 的规模能使学习算法 \mathcal{L} 以概率 $1 - \delta$ 找到目标假设 ϵ 即可。

假定 h 的泛化误差大于 ϵ , 对于分布 \mathcal{D} 上随机采样得到的任何样例 (x, y) 都有:

$$P(h(x) = y) = 1 - P(h(x) \neq y) = 1 - E(h) < 1 - \epsilon \quad (10.9)$$

那么 h 于 D 表现一致的概率为:

$$P((h(x_1) = y_1) \wedge \cdots \wedge (h(x_m) = y_m)) = (1 - P(h(x) \neq y))^m < (1 - \epsilon)^m \quad (10.10)$$

由于我们事先不知道学习算法 \mathcal{L} 会输出 \mathcal{H} 的那个假设, 所以我们只需保证泛化误差大于 ϵ , 且训练集上表现完美的所有假设出现概率之和不大于 δ 即可:

$$P(h \in \mathcal{H} : E(h) > \epsilon \wedge \hat{E}(h) = 0) < |\mathcal{H}|(1 - \epsilon)^m < |\mathcal{H}|e^{-m\epsilon} \quad (10.11)$$

我们令 10.11, 不大于 δ : 即

$$|\mathcal{H}|e^{-m\epsilon} \leq \delta \quad (10.12)$$

可得:

$$m \geq \frac{1}{\epsilon}(\ln |\mathcal{H}| + \ln \frac{1}{\delta}) \quad (10.13)$$

上面的公式说明, 输出假设 h 的泛化误差随着样本数目的增多而收敛到 0, 其收敛速度为 $O(\frac{1}{m})$.

10.3.2 不可分情形

对于不可分的清醒，目标概率 c 往往不存在于假设空间 \mathcal{H} 中，假定 \mathcal{H} 中的任意一个假设都会在训练集上或多或少的错误，由 Hoeffding 不等式可知：

- 若训练集 D 包含 m 个从分布 \mathcal{D} 上独立同分布采样而得的样例， $0 < \epsilon < 1$ ，则对任意 $h \in \mathcal{H}$ ，有

$$\begin{aligned} P(\hat{E}(h) - E(h) \geq \epsilon) &\leq \exp(-2m\epsilon^2) \\ P(E(h) - \hat{E}(h) \geq \epsilon) &\leq \exp(-2m\epsilon^2) \\ P(|E(h) - \hat{E}(h)| \geq \epsilon) &\leq 2 \exp(-2m\epsilon^2) \end{aligned} \quad (10.14)$$

- 若训练集 D 包含 m 个从分布 \mathcal{D} 上独立同分布采样而得的样例， $0 < \epsilon < 1$ ，则对任意 $h \in \mathcal{H}$ ，下面的公式至少以 $1 - \delta$ 的概率成立。

$$\hat{E}(h) - \sqrt{\frac{\ln(2/\delta)}{2m}} \leq E(h) \leq \hat{E}(h) + \sqrt{\frac{\ln(2/\delta)}{2m}} \quad (10.15)$$

- 对于有限假设空间，且 $0 < \epsilon < 1$ ，则对任意 $h \in \mathcal{H}$ ，有：

$$P\left(|E(h) - \hat{E}(h)| \leq \sqrt{\frac{\ln |\mathcal{H}| + \ln(2/\delta)}{2m}}\right) \geq 1 - \delta \quad (10.16)$$

显然，当 $c \notin \mathcal{H}$ ，学习算法 \mathcal{L} 是无法学得目标概念 c 的 ϵ 近似的；但是，若假设空间 \mathcal{H} 给定时，其中必然存在一个泛化误差最小的假设，我们可以选择这个假设即可。这种方法被称为“不可知学习 (agnostic learning)”

不可知 PAC 可学习 (agnostic PAC learnable): 令 m 表示从分布 \mathcal{D} 中独立同分布采样得到的样例数目， $0 < \epsilon, \delta < 1$ ，对所有的分布 (D) ，若存在学习算法 \mathcal{L} 和多项式函数 $\text{poly}(\cdot, \cdot, \cdot, \cdot)$ ，使得对于任意 $m \geq \text{poly}(\frac{1}{\epsilon}, \frac{1}{\delta}, \text{size}(x), \text{size}(c))$ ，满足如下公式，则称假设空间 \mathcal{H} 是不可知 PAC 可学习的。

10.4 VC 维

现实的学习任务通常都是面临无限的假设空间，这时我们想度量假设空间的复杂度，最常见方法就是考虑假设空间的 VC 维 (VapnikChervonenkis dimension)，给定假设空间 \mathcal{H} 和示例集 $D = \{x_1, x_2, \dots, x_m\}$ ，则 \mathcal{H} 中每个假设 h 都能对 D 中示例赋予标记，标记结果可以表示为：

$$h|_D = \{(h(x_1), h(x_2), \dots, h(x_m))\} \quad (10.17)$$

我们引入几个概念：

- 增长函数 (growth function): 对于所有的 $m \in N$ ，假设空间 \mathcal{H} 的增长函数 $\Pi_{\mathcal{H}}(m)$ 为：

$$\Pi_{\mathcal{H}}(m) = \max_{\{x_1, \dots, x_m\} \subseteq \mathcal{X}} |\{(h(x_1), \dots, h(x_m)) | h \in \mathcal{H}\}| \quad (10.18)$$

增长函数描述了假设空间 \mathcal{H} 的表示能力，由此反映出假设空间的复杂度

- 对分 (dichotomy) 和打散 (shattering): 增长函数可以用来估计经验误差与泛化误差之间的关系：对假设空间 \mathcal{H} ， $m \in N, 0 < \epsilon < 1$ 和任意 $h \in \mathcal{H}$ 有：

$$P(|E(h) - \hat{E}(h)| > \epsilon) \leq 4\Pi_{\mathcal{H}}(2m) \exp\left(-\frac{m\epsilon^2}{8}\right) \quad (10.19)$$

对于 m 个示例，最多有 2^m 个可能的结果。对于二分类问题来说， \mathcal{H} 中的假设对 D 中的示例赋予标记的每种可能结果称为对 D 的一种“对分”，若假设空间 \mathcal{H} 能实现示例集 D 上的所有对分，即 $\Pi_{\mathcal{H}}(m) = 2^m$ ，那么称示例集 D 能被假设空间 \mathcal{H} 打散。

综上，我们可以定义 VC 维了：假设空间 \mathcal{H} 的 VC 维是能被 \mathcal{H} 打散的最大示例集的大小，即：

$$VC(\mathcal{H}) = \max\{m : \Pi_{\mathcal{H}}(m) = 2^m\} \quad (10.20)$$

其中 $VC(\mathcal{H} = d$ 表明存在大小为 d 的示例集能被假设空间 \mathcal{H} 打散，与数据分布是否可知无关。

VC 维跟增长函数有密切联系，假设空间 \mathcal{H} 的 VC 维为 d ，那么对任意的 $m \in \mathbb{N}$ 有：

$$\Pi_{\mathcal{H}}(m) \leq \sum_{i=0}^d \binom{m}{i} \quad (10.21)$$

若假设空间 \mathcal{H} 的 VC 维为 d ，那么对任意的整数 $m \geq d$ ：

$$\Pi_{\mathcal{H}}(m) \leq \left(\frac{e \cdot m}{d}\right)^d \quad (10.22)$$

若假设空间 \mathcal{H} 的 VC 维为 d ，那么对任意的 $m > d, 0 < \delta < 1, h \in \mathcal{H}$ 有：

$$P \left(E(h) - \hat{E}(h) \leq \sqrt{\frac{8d \ln \frac{2em}{d} + 8 \ln \frac{4}{\delta}}{m}} \right) \geq 1 - \delta \quad (10.23)$$

上面上个公式表明：VC 维的泛化误差界只与样例数目有关，收敛速度为 $O(\frac{1}{\sqrt{m}})$ ，与数据分布 \mathcal{D} 和样例集 D 无关。因此，基于 VC 维的泛化误差界是分布无关、数据独立的。

令 h 表示学习算法 \mathcal{L} 输出的假设，若 h 满足：

$$\hat{E}(h) = \min_{h' \in \mathcal{H}} \hat{E}(h') \quad (10.24)$$

则称 \mathcal{L} 满足经验风险最小化 (Empirical Risk Minimization, ERM)，那么基于这个算法，我们可知任何 VC 维有限的假设空间 \mathcal{H} 都是 (不可知)PAC 可学习的。

10.5 Rademacher 复杂度

上节提到的 VC 维的泛化误差界是分布无关、数据独立的，也就是说 VC 维具有一定的普适性，而造成的代价就是对于一些学习问题的典型情况效果不太好。

Rademacher 复杂度是另一种刻画假设空间复杂度的途径，与 VC 维不同的是，它在一定程度上考虑到了数据分布。给定数据集 $D = \{(x_1, y_1), \dots, (x_m, y_m)\}$ ，假设 h 的经验误差为：

$$\hat{E}(h) = \frac{1}{m} \sum_{i=1}^m \mathbb{I}(h(x_i) \neq y_i) = \frac{1}{2} - \frac{1}{2m} \sum_{i=1}^m y_i h(x_i) \quad (10.25)$$

那么我们可以得出经验误差的假设:

$$\arg \max_{h \in \mathcal{H}} \frac{1}{m} \sum_{i=1}^m y_i h(x_i) \quad (10.26)$$

在实际应用之中, 我们还要考虑噪声的影响, 引入随机变量 δ_i , 它以 0.5 的概率取 -1, 0.5 的概率取 +1, 称为 Rademacher 随机变量, 重写上式:

$$\sup_{h \in \mathcal{H}} \frac{1}{m} \sum_{i=1}^m \delta_i h(x_i) \quad (10.27)$$

取期望可表示为:

$$\mathbb{E}_\delta \left[\sup_{h \in \mathcal{H}} \frac{1}{m} \sum_{i=1}^m \delta_i h(x_i) \right] \quad (10.28)$$

考虑到实值函数空间 $\mathcal{F}: \mathcal{Z} \rightarrow \mathbb{R}$, 令 $Z = \{z_1, z_2, \dots, z_m\}$, 其中 $z_i \in \mathcal{Z}$ 将上式的 \mathcal{X}, \mathcal{H} 替换成 \mathcal{Z}, \mathcal{F} 可得。

函数空间 \mathcal{F} 关于 Z 的经验 Rademacher 复杂度:

$$\hat{R}_Z(\mathcal{F}) = \mathbb{E}_\delta \left[\sup_{f \in \mathcal{F}} \frac{1}{m} \sum_{i=1}^m \delta_i f(z_i) \right] \quad (10.29)$$

函数空间 \mathcal{F} 关于 Z 上分布 \mathcal{D} 的 Rademacher 复杂度:

$$R_m(\mathcal{F}) = \mathbb{E}_{Z \subseteq \mathcal{Z}: |Z|=m} \left[\hat{R}_Z(\mathcal{F}) \right] \quad (10.30)$$

对实值函数空间 $\mathcal{F}: \mathcal{Z} \rightarrow [0, 1]$, 根据分布 \mathcal{D} 从 \mathcal{Z} 中独立同分布采样得到的示例集 $Z = \{z_1, z_2, \dots, z_m\}$ $z_i \in \mathcal{Z}, 0 < \delta < 1$, 那么对于任意的 $f \in \mathcal{F}$, 至少以 $1 - \delta$ 的概率满足:

$$\begin{aligned} \mathbb{E}[f(z)] &\leq \frac{1}{m} \sum_{i=1}^m f(z_i) + 2R_m(\mathcal{F}) + \sqrt{\frac{\ln(1/\delta)}{2m}} \\ \mathbb{E}[f(z)] &\leq \frac{1}{m} \sum_{i=1}^m f(z_i) + 2\hat{R}_Z(\mathcal{F}) + 3\sqrt{\frac{\ln(2/\delta)}{2m}} \end{aligned} \quad (10.31)$$

对于上式, 由于函数空间 \mathcal{F} 是区间 $[0, 1]$ 上的实值函数, 因此上式仅仅适用于回归问题, 对于二分类问题, 我们可以有如下的定理:

对于假设空间 $\mathcal{H} : \mathcal{X} \rightarrow \{-1, +1\}$, 根据分布 \mathcal{D} 从 \mathcal{X} 中独立同分布采样得到的示例集 $D = \{x_1, x_2, \dots, x_m\}, x_i \in \mathcal{X}, 0 < \delta < 1$, 对于任意 $h \in \mathcal{H}$, 至少有 $1 - \delta$ 的概率:

$$\begin{aligned} E(h) &\leq \hat{E}(h) + R_m(\mathcal{H}) + \sqrt{\frac{\ln(1/\delta)}{2m}} \\ E(h) &\leq \hat{E}(h) + \hat{R}_D(\mathcal{H}) + 3\sqrt{\frac{\ln(2/\delta)}{2m}} \end{aligned} \quad (10.32)$$

对于 Rademacher 复杂度与增长函数, 满足如下定理:

假设空间 \mathcal{H} 的 Rademacher 复杂度 $R_m H$ 与增长函数 $\Pi_{\mathcal{H}}(m)$ 满足:

$$R_m H \leq \sqrt{\frac{2 \ln \Pi_{\mathcal{H}}(m)}{m}} \quad (10.33)$$

结合上式:

$$E(h) \leq \hat{E}(h) + \sqrt{\frac{2d \ln \frac{em}{d}}{m}} + \sqrt{\frac{\ln \frac{1}{\delta}}{2m}} \quad (10.34)$$

10.6 稳定性

无论是 VC 维还是 Rademacher 复杂度都是与具体算法无关的, 为了得到与算法有关的分析结果, 我们引入稳定性 (stability) 来分析这方面。

算法的稳定性指的是算法的输入发生变化时, 输出是否会随之发生较大的变化。假设 $D^{/i}$ 表示移除 D 中的第 i 个样例得到的集合, 而 D^i 表示替换第 i 个样例得到的集合。我们假设 \mathcal{L}_D 如下的损失:

- 泛化损失:

$$\ell(\mathcal{L}, \mathcal{D}) = \mathbb{E}_{x \in \mathcal{X}, z=(x,y)} [\ell(\mathcal{L}_D, z)] \quad (10.35)$$

- 经验损失:

$$\hat{\ell}(\mathcal{L}, D) = \frac{1}{m} \sum_{i=1}^m \ell(\mathcal{L}_D, z_i) \quad (10.36)$$

- 留一损失:

$$\ell_{loo}(\mathcal{L}, D) = \frac{1}{m} \sum_{i=1}^m \ell(\mathcal{L}_{D^{/i}}, z_i) \quad (10.37)$$

对于任意 $x \in \mathcal{X}, z = (x, y)$, 若学习算法 \mathcal{L} 满足:

$$|\ell(\mathcal{L}_D, z) - \ell(\mathcal{L}_{D^{(i)}}, z)| \leq \beta, i = 1, 2, \dots, m \quad (10.38)$$

则称 \mathcal{L} 关于损失函数 ℓ 满足 β -均匀稳定性。若损失函数 ℓ 的上界为 M , $0 < \delta < 1$, 则对任意 $m \geq 1$, 以至少 $1 - \delta$ 的概率有:

$$\begin{aligned} \ell(\mathcal{L}, \mathcal{D}) &\leq \hat{\ell}(\mathcal{L}, D) + 2\beta + (4m\beta + M)\sqrt{\frac{\ln \frac{1}{\delta}}{2m}} \\ \ell(\mathcal{L}, \mathcal{D}) &\leq \ell_{loo}(\mathcal{L}, D) + \beta + (4m\beta + M)\sqrt{\frac{\ln \frac{1}{\delta}}{2m}} \end{aligned} \quad (10.39)$$

上式说明经验损失和泛化损失之间差别的收敛率为 $\beta\sqrt{m}$, 若取 $\beta = O(\frac{1}{m})$, 则可以保证收敛率为 $O(\frac{1}{\sqrt{m}})$ 。我们假设 $\beta = \frac{1}{m}$, 则有:

$$\ell(\mathcal{L}, \mathcal{D}) \leq \hat{\ell}(\mathcal{L}, D) + \frac{2}{m} + (4 + M)\sqrt{\frac{\ln \frac{1}{\delta}}{2m}} \quad (10.40)$$

上式说明若算法 \mathcal{L} 所输出的假设满足经验损失最小化, 则称算法 \mathcal{L} 满足经验风险最小化 (Empirical Risk Minimization), 简称算法是 ERM 的, 若该算法 \mathcal{L} 还是稳定的, 那么假设空间 \mathcal{H} 是可学习的。

10.7 小结

Chapter 11

特征选择与稀疏学习

11.1 子集搜索与评价

我们一般将属性称为特征 (feature)，跟当前任务有关的属性称为相关特征 (relevant feature) 而没什么用的属性被我们称为无关特征 (irrelevant feature)，而从给定的特征集合中选择相关特征子集的过程，就称为特征选择 (feature selection)。

特征选择可以算是一个重要的数据预处理过程，特征过程中必须确保不丢失不丢失重要的特征、并尽可能的去剔除无关特征。

欲从特征集合中选择一个包含来所有重要特征的特征子集，最好的方法就是产生一个”候选的子集”，评价它的好坏，基于评价结果产生下一个候选子集，再对其进行评价。那我们要解决的问题就变成了如何评价一个候选子集的好坏了。

第一个环节是进行子集搜索，在给定的特征集合 $\{a_1, a_2, \dots, a_d\}$ 我们可将每个特征看作一个候选的子集.

1. 假定我们得出 $\{a_2\}$ 最优，我们将其加入选定集中，此时选定集为 $\{a_2\}$.
2. 假定我们在剩下的 $d - 1$ 个子集中得到了 $\{a_4\}$ 最优，那么我们把 $\{a_2, a_4\}$ 与当前的选定集 $\{a_2\}$ 进行比较，如果前者优于后者，那么我们用前者作为新的选定集，否则选定集不作出任何更改。

3. 直到第 $k + 1$ 轮时, 最优的候选 $(k + 1)$ 特征子集不如上一轮的选定集, 那么算法停止。

上述这种能够逐渐增加相关特征的策略称为”前向 (forward) 搜索”, 类似的, 我们的选定集从全集开始逐渐减少的方式我们可以称为”后向 (backward)”, 也可以两者同时进行, 这种策略称为”双向 (bidirection) 搜索”.

但是无论是哪种策略, 都是贪心的, 仅仅可能得到的局部最优解。

第二个环节是一个子集评价 (subset evaluation) 问题, 在决策树那章提出的信息增益:

$$Gain(A) = Ent(D) - \sum_{v=1}^V \frac{|D^v|}{|D|} Ent(D^v) \quad (11.1)$$

信息熵将被定义为:

$$Ent(D) = - \sum_{i=1}^{|\mathcal{Y}|} p_k \log_2 p_k \quad (11.2)$$

信息增益越大, 说明有助于分类的信息将会越多。那么常用的特征选择方式我们也以分为三种: 过滤式 (filter)、包裹式 (wrapper)、嵌入式 (embedding)。

11.2 过滤式选择

filter 算法首先要对初始特征进行过滤, 再用过滤后的特征来训练模型。Relief 算法就是其中的一个典型的算法, 该方法设计来一个相关统计量来度量, 该统计量是个向量, 其每个分量对应于一个初始特征, 而特征子集的重要性指的是由子集中每个特征所对应的相关统计量的分量之和来决定的。所以只需指定一个阈值 τ , 然后选择比 τ 大的相关统计量分量所对应的特征即可;

显然, Relief 的关键是如何确定相关统计量, 给定训练集 $\{(x_1, y_1), \dots, (x_m, y_m)\}$, 对于每个示例 x_i , Relief 先在 x_i 的同类样本中寻找最近邻 $x_{i,nh}$, 我们将其称为”猜中近邻 (near-hit)”, 再从 x_i 的异类样本中寻找其最近邻 $x_{i,nm}$, 我们称为”猜错近邻 (near-miss)”, 然后统计量对应于属性 j 的分量可以表示为:

$$\sigma^j = \sum_j -diff(x_i^j, x_{i,nh}^j)^2 + diff(x_i^j, x_{i,nm}^j)^2 \quad (11.3)$$

从上式可以看出，若 x_i 与其猜中近邻 $x_{i,nh}$ 在属性 j 上的距离小于 x_i 与其猜错近邻 $x_{i,nm}$ 的距离，说明属性 j 对区分同类和异类样本是有益的，于是应该增大属性 j 所对应的统计量分量。否则将减小 j 所对应的统计量分量，最后，若分量值越大，则对应属性的分类能力就越强。

在实际使用的过程中，Relief 只需在数据集的采样上而不必在整个数据集上估计相关统计量。是一个运行效率很高的过滤式特征选择算法。

我们可以将其拓展到多分类的问题，假定数据集 D 中的样本来自 $|\mathcal{Y}|$ 个类别。对于示例 x_i ，若它属于第 k 类 ($k \in \{1, 2, \dots, |\mathcal{Y}|\}$)，Relief-F 先在第 k 类样本中寻找 x_i 最近邻示例 $x_{i,nh}$ ，并将其作为猜中近邻，然后在第 k 类样本外的每个类都找一个 x_i 的最近邻示例作为猜错近邻，极为 $x_{i,l,nm}$ ($l = 1, 2, \dots, |\mathcal{Y}|; l \neq k$)，于是相关的统计量对应于属性 j 的分量为：

$$\sigma^j = \sum_i -\text{diff}(x_i^J, x_{i,nh}^j)^2 + \sum_{l \neq k} (p_l \times \text{diff}(x_i^J, x_{i,l,nm}^j)^2) \quad (11.4)$$

其中 p_l 作为第 l 类样本在数据集 D 中所占的比例。

11.3 包裹式选择

与过滤式特征选择不考虑后续学习器不同，包裹式特征选择直接把最终将要使用的学习器的性能作为特征子集的评价准则，换言之，包裹式特征选择的目的就是为给定学习器选择最有利于其性能的特征子集。

一般来说，包裹式选择器是直接针对最后的学习器，那么性能将会优于过滤式特征选择。但是由于在迭代过程中需要多次训练学习器，因此包裹式特征选择的开销要大得多。

常用的包裹式特征选择方法是 LVW(LAs Vegas Wrapper)，它是在拉斯维加斯方法 (Las Vegas method) 框架下使用随即策略来进行自己搜索。并且以最终分类器的误差为特征子集的评价标注，算法流程图如下：

输入: 数据集 D ;
特征集 A ;
学习算法 \mathfrak{L} ;
停止条件控制参数 T .

过程:

```
1:  $E = \infty$ ;  
2:  $d = |A|$ ;  
3:  $A^* = A$ ;  
4:  $t = 0$ ;  
5: while  $t < T$  do  
6:   随机产生特征子集  $A'$ ;  
7:    $d' = |A'|$ ;  
8:    $E' = \text{CrossValidation}(\mathfrak{L}(D^{A'}))$ ;  
9:   if  $(E' < E) \vee ((E' = E) \wedge (d' < d))$  then  
10:     $t = 0$ ;  
11:     $E = E'$ ;  
12:     $d = d'$ ;  
13:     $A^* = A'$   
14:   else  
15:     $t = t + 1$   
16:   end if  
17: end while
```

输出: 特征子集 A^*

图 11.1 LVW 算法描述

需要特别留意，由于 LVW 算法中特征子集搜索采用类随即策略，每次特征子集评价都需要训练学习器，导致计算的开销会很大，因此为了加快算法的收敛速度，设置停止条件控制参数 T ，但是由于随机的选择太多了，可能无法在有限时间内给出相应的解。

11.4 嵌入式选择与 L_1 正则化

与包裹式和过滤式特征选择方法不同，嵌入式特征选择是将特征选择过程与学习器的训练过程融为一体，两者在同一个优化过程中完成。在此之前，我们要先明白什么是范数：

- L_0 范数：指的是向量中非 0 元素的个数，如果我们用 L_0 范数来规则化一个参数矩阵 W ，那么就是希望 W 的大部分元素都是 0，即让 W 是稀疏的。
- L_1 范数：指的是向量中各个元素的绝对值之和，也称为“稀疏规则算子”(Lasso regularization)，存在下面一个定理：任何的规则化算子，如果在 W_i 的地方不可微，并且可以分解出一个求和的形式，那么这个规则化算子就可以实现稀疏。至于为何一般采用 L_1 而不是 L_0 来稀疏，原因有两个：
 - L_1 范数是 L_0 范数的最优凸近似，并且 L_1 范数比 L_0 更加容易优化求解。
 - L_0 范数求解是一个 NP 难的问题。

一般来说我们稀疏参数是为了进行更好的特征选择和增强特征的可解释性。

- L_2 范数是指向量各元素的平方和然后求平方根，采用 L_2 范数会导致 W 的每个元素都很小，但是会让元素们只是接近 0 而不是等于 0。越小的参数说明模型越简单，越简单的模型则越不容易产生过拟合现象。这说明 L_2 范数不但可以防止过拟合，还可以让我们的优化求解变得更加稳定和快速。

给定数据集 D ，我们考虑最简单的线性回归模型，以平方误差作为损失函数，则优化目标为：

$$\min_w \sum_{i=1}^m (y_i - w^T x_i)^2 \quad (11.5)$$

当样本的特征很多，而样本数量很少时，上式容易陷入过拟合，为了缓解过拟合问题，我们通常的做法是引入正则化项，若使用 L_2 范数进行正则化，则有：

$$\min_w \sum_{i=1}^m (y_i - w^T x_i)^2 + \lambda \|w\|_2^2 \quad (11.6)$$

其中正则化参数 $\lambda > 0$ ，将上述公式称为“岭回归 (ridge regression)”。

若采用 L_1 范数代替 L_2 的范数，则有：

$$\min_w \sum_{i=1}^m (y_i - w^T x_i)^2 + \lambda \|w\|_1 \quad (11.7)$$

我们将上述公式称为 LASSO，相比于 L_2 范数， L_1 范数更加容易获得一个稀疏 (sparse) 的解。

L_1 正则化问题的求解可使用近端梯度下降 (PGD)，一般来说令 ∇ 表示微分算子，对优化目标 $\min_x f(x) + \lambda \|x\|_1$ ，若 $f(x)$ 可导，并且 ∇f 满足 L-Lipschitz 条件，即存在常数 $L > 0$ 使得：

$$\|\nabla f(x') - \nabla f(x)\|_2^2 \leq L\|x' - x\|_2^2, (\forall x, x') \quad (11.8)$$

则在 x_k 附近可讲 $f(x)$ 通过二阶泰勒展开式近似为：

$$\begin{aligned} \hat{f}(x) &= f(x_k) + \langle \nabla f(x_k), x - x_k \rangle + \frac{L}{2} \|x - x_k\|^2 \\ &= \frac{L}{2} \|x - \left(x_k - \frac{1}{L} \nabla f(x_k) \right)\|_2^2 + \text{const} \end{aligned} \quad (11.9)$$

其中 const 是一个与 x 无关的常数，显然上式的最小值在 x_{k+1} 获得：

$$x_{k+1} = x_k - \frac{1}{L} \nabla f(x_k) \quad (11.10)$$

若是通过梯度下降法对 $f(x)$ 进行最小化，则每一步梯度下降迭代实际上等价于最小化二次函数 $\hat{f}(x)$ ，所以每步迭代可以写为：

$$x_{k+1} = \arg \min_x \frac{L}{2} \|x - \left(x_k - \frac{1}{L} \nabla f(x_k) \right)\|_2^2 + \lambda \|x\|_1 \quad (11.11)$$

在迭代下降的同时也要同时考虑 L_1 范数最小化，令 $z = x_k - \frac{1}{L} \nabla f(x_k)$ ：

$$x_{k+1} = \arg \min_x \frac{L}{2} \|x - z\|_2^2 + \lambda \|x\|_1 \quad (11.12)$$

其中 x 的各个分量互不影响, 于是上式将会出现闭式解:

$$x_{k+1}^i = \begin{cases} z^i - \lambda/L, & \lambda/L < z^i; \\ 0, & |z^i| \leq \lambda/L \\ z^i + \lambda/L, & z^i < -\lambda/L \end{cases} \quad (11.13)$$

综上, 说明 PGD 能使 LASSO 和其他基于 L_1 范数最小化的方法得以快速求解。

11.5 稀疏表示与字典学习

我们可以把数据集 D 考虑成一个矩阵, 每行对应一个样本, 每列对应一个特征, 若特征的选择所考虑的问题是特征具有稀疏性, 那么通过特征选择可以去除这些列。考虑另一种稀疏性, D 中对应的矩阵中存在许多零元素, 但是这些零元素并不是以整行或者整列出现的, 当样本具有这样的稀疏表达形式时, 对于学习任务来说会有不少好处, 因为类似的稀疏 one-hot 向量矩阵表达出了高度的稀疏性, 这就意味着多数问题将会变成线性可分的。而且稀疏矩阵的存储也是很简单的, 但是我们要注意稀疏矩阵要时恰当稀疏而不是过度稀疏。

在一般的学习任务中, 我们可能要创建一个类似的字典, 将样本转化成合适的稀疏表示形式, 从而使得学习任务得到简化, 模型的复杂度得到降低。这个过程我们一般称为字典学习 (dictionary learning) 或者稀疏编码 sparse coding。

给定数据集 D , 字典学习的最简单形式可以表示为:

$$\min_{B, \alpha_i} \sum_{i=1}^m \|x_i - B\alpha_i\|_2^2 + \lambda \sum_{i=1}^m \|\alpha_i\|_1 \quad (11.14)$$

其中 $B \in \mathbb{R}^{d \times k}$ 为字典矩阵, k 称为字典的词汇量, 通常是由用户指定; $\alpha_i \in \mathbb{R}^k$ 则表示样本 $x_i \in \mathbb{R}^d$ 的稀疏表示。上式表明 α_i 能更好地重构 x_i , 并且 α_i 自身尽可能地稀疏。我们一般采取变量交替优化的策略来求解上式。

首先第一步, 我们固定住字典 B , 将 11.14 按照分量展开, 从而为每个

样本 x_i 找到相应的 α 。

$$\min_{\alpha_i} \|x_i - B\alpha_i\|_2^2 + \lambda \|\alpha_i\|_1 \quad (11.15)$$

第二步，我们固定住 α 来更新字典 B ，此时可将 11.14 写为：

$$\min_B \|X - BA\|_F^2 \quad (11.16)$$

其中 $\|\cdot\|_F$ 是矩阵的 Frobenius 范数，而 11.16 常用的求解方法是基于逐列更新策略的 KSVD，令 b_i 表示字典矩阵 B 的第 i 列， α^i 表示稀疏矩阵 A 的第 i 行，重写 11.16：

$$\begin{aligned} \min_B \|X - BA\|_F^2 &= \min_{b_i} \|X - \sum_{j=1}^k b_j \alpha^j\|_F^2 \\ &= \min_{b_i} \left\| \left(X - \sum_{j \neq i} b_j \alpha^j \right) - b_i \alpha^i \right\|_F^2 \\ &= \min_{b_i} \|E_i - b_i \alpha^i\|_F^2 \end{aligned} \quad (11.17)$$

原则上只需对 E_i 进行奇异值分解取其最大奇异值所对应的正交向量，但是直接对 E_i 进行奇异值分解会同时修改 b_i 和 α^i ，从而破坏 A 的稀疏性，为了避免这种情况，KSVD 对 E_i 和 α^i 进行专门处理； α^i 只保留非零元素， E_i 则仅保留 b_i 与 α 的非零元素的乘积项，然后再进行奇异值分解，从而保证了稀疏性。

11.6 压缩感知 (compressed sensing)

类比于通信原理中的采样算法，只要采样的频率足够高（原始模拟信号的两倍）就可以重构出信号，放在机器学习之中就是压缩感知。

假定有长度为 m 的离散信号 x ，采样后得到长度为 n 的信号 y ， $n \ll m$ 即：

$$y = \Phi x \quad (11.18)$$

其中 $\Phi \in \mathbb{R}^{n \times m}$ 是是对信号 x 的测量矩阵。

一般的，进行采样后我们要想办法将信号进行还原，假设存在某个线性变换 $\Psi \in \mathbb{R}^{m \times m}$ ，使得 x 可表示为 Ψs 那么 y 可以表示为：

$$y = \Phi \Psi s = As \quad (11.19)$$

有趣的是，若 s 具有稀疏性，上式就能够很好的恢复出原始信号。

基于此压缩感知一般分为两个阶段：感知测量和重构恢复，我们一般所说的压缩感知一般指代的是后者。

压缩感知的理论一般而言还是比较复杂的，下面要简要地介绍一下”限定等距性 (RIP)”: 对于大小为 $n \times m$ 的矩阵 A ，若存在常数 $\delta \in (0, 1)$ 对于任意的向量 s 和子矩阵 A_k ，有：

$$(1 - \delta_k) \|s\|_2^2 \leq \|A_k s\|_2^2 \leq (1 + \delta_k) \|s\|_2^2 \quad (11.20)$$

则称 A 满足 k 限定等距性，此时可以通过下面的优化问题近乎完美地从 y 中恢复出稀疏信号 s ，从而恢复出 x ：

$$\min_s \|s\|_0 \quad s.t. \quad y = As \quad (11.21)$$

这样的话，压缩感知问题就可以通过 L_1 范数最小化问题进行求解，比如转化成 LASSO 的等价形式再通过近端梯度下降法进行求解，即使用”基寻踪去噪”。

在实际的运用中，可能会遇到信息缺失的情形，这时我们就要用到矩阵补全技术来解决这个问题，其形式为：

$$\min_X \text{rank}(X) \quad s.t. (X)_{ij} = (A)_{ij}, (i, j \in \Omega) \quad (11.22)$$

其中 Ω 是 A 中的不缺少元素的集合。注意到 $\text{rank}(X)$ 在集合 $\{X \in \mathbb{R}^{m \times n} : \|X\|_F^2 \leq 1\}$ 的凸包是 X 的”核范数”：

$$\|X\|_* = \sum_{j=1}^{\min\{m,n\}} \sigma_j(X) \quad (11.23)$$

其中 $\sigma_j(X)$ 表示 X 的奇异值，即核范数的奇异值之和，于是可通过最小化矩阵核范数来求解近似解：

$$\min_X \|X\|_* \quad s.t. \quad (X)_{ij} = (A)_{ij}, (i, j) \in \Omega \quad (11.24)$$

理论上已经证明，满足一定条件时，若 A 的秩为 r , $n \ll m$, 则只需观察到 $O(mr \log^2 m)$ 个元素就能够完美恢复出 A 。

11.7 小结

Chapter 12

半监督学习 (semi-supervised learning)

12.1 未标记样本

假设 D_l 指的是已经有标记的数据集, D_u 的标记是未知的。若 $l \ll u$, 则训练样本不足, 学得的模型泛化性能往往不佳。为了利用 D_u 我们可以事先用 D_l 训练一个模型, 用这个模型分类 D_u , 并把重新获得的有标记的样本加入到 D_l 中重新训练一个模型, 在进行第二次挑选..., 这样只需要少量 D_u 样本就可以构建出比较强的模型了, 这样可以大幅降低标记成本, 这样的学习方式称为“主动学习”(active learning), 其目的是采用较少的查询 (query) 来获得尽量好的性能。

为了获得从 D_u 中采样的样本标记, 由于采样是独立同分布的, 所以我们可以根据样本的分布信息来较大把握地确定样本标记。让学习器能够不依赖外界交互、自动地利用未标记样本来提升学习性能, 就是半监督学习了。

要利用未标记的样本, 必须要做的是将一些未标记样本所揭示的数据分布信息与类别标记相联系的假设, 而最常见的假设是聚类假设 (cluster assumption), 即假设数据存在簇结构, 同一簇结构属于同一类别。另一种假设是“流形假设 (manifold assumption)”, 即假设数据分布在一个流行结构上, 邻近的样本拥有相似的输出值, 可以看作是聚类假设的推广。两种假设

都是基于“相似的样本拥有相似的输出”这一个基本假设。

半监督学习可进一步划分为纯 (pure) 半监督学习和直推学习 (transductive learning)。前者假定训练数据中的未标记样本并非待预测的数据，而后者假定学习过程中所考虑的未标记样本恰是待预测数据。

12.2 生成式方法 (generative methods)

生成式方法是直接基于生成式模型的方法，这种方法假设所有的数据都是由同一个潜在的模型生成的。

给定样本 x ，其真实类别标记为 $y \in \mathcal{Y}, \mathcal{Y} = \{1, 2, \dots, N\}$ 为所有可能的类别，假设样本由高斯混合模型生成，且每个类别对应一个高斯混合成分，换言之，数据样本是基于如下概率密度生成：

$$p(x) = \sum_{i=1}^N \alpha_i \cdot p(x|\mu_i, \sum_i) \quad (12.1)$$

其中，混合系数 $\alpha_i \geq 0, \sum_{i=1}^N \alpha_i = 1$;。令 $f(x) \in \mathcal{Y}$ 表示模型 f 对 x 的预测标记， $\Theta \in \{1, 2, \dots, N\}$ ，由最大化后验概率可知：

$$f(x) = \arg \max_{j \in \mathcal{Y}} \sum_{i=1}^N p(y=j|\Theta=i, x) \cdot p(\Theta=i|x) \quad (12.2)$$

其中：

$$p(\Theta=i|x) = \frac{\alpha_i \cdot p(x|\mu_i, \sum_i)}{\sum_{i=1}^N \alpha_i \cdot p(x|\mu_i, \sum_i)} \quad (12.3)$$

不难发现，上式中估计 $p(y=j|\Theta=i, x)$ 需要知道样本的标记，因此仅能使用有标记数据，而 $(\Theta=i|x)$ 不涉及样本标记，两者均可使用，随着未标记数据的大量地引入，这一项估计有望更加准确。假设样本 D_l 和 D_u 是独立同分布，且都是由同一个高斯混合模型生成的。用极大似然估计法来估计高斯混合模型的参数 $\{(\alpha_i, \mu_i, \sum_i) | 1 \leq i \leq N\}$ ，那么 $D_l \cup D_u$ 的对数似

然是：

$$\begin{aligned} LL(D_l \bigcup D_u) &= \sum_{(x_j, y_j) \in D_l} \ln \left(\sum_{i=1}^N \alpha_i \cdot p(x_j | \mu_i, \sum_i) \cdot p(y_j | \Theta = i | x_j) \right) \\ &\quad + \sum_{x_j \in D_u} \ln \left(\sum_{i=1}^N \alpha_i \cdot p(x_j | \mu_i, \sum_i) \right) \end{aligned} \quad (12.4)$$

对于上式，我们可以用 EM 算法进行求解：

- E 步：根据当前模型参数计算未标记样本 x_j 属于各高斯混合成分的概率：

$$\gamma_{ji} = \frac{\alpha_i \cdot p(x_j | \mu_i, \sum_i)}{\sum_{i=1}^N \alpha_i \cdot p(x_j | \mu_i, \sum_i)} \quad (12.5)$$

- M 步：基于 γ_{ji} 更新模型参数，其中 l_i 表示第 i 类的有标记样本数目：

$$\begin{aligned} \mu_i &= \frac{1}{\sum_{x_j \in D_u} \gamma_{ji} + l_i} \left(\sum_{x_j \in D_u} \gamma_{ji} x_j + \sum_{(x_j, y_j) \in D_l \wedge y_j = i} x_j \right) \\ \sum_i &= \frac{1}{\sum_{x_j \in D_u} \gamma_{ji} + l_i} \left(\sum_{x_j \in D_u} \gamma_{ji} (x_j - \mu_i) (x_j - \mu_i)^T + \sum_{(x_j, y_j) \in D_l \wedge y_j = i} (x_j - \mu_i) (x_j - \mu_i)^T \right) \\ \alpha_i &= \frac{1}{m} \left(\sum_{x_j \in D_u} \gamma_{ji} + l_i \right) \end{aligned} \quad (12.6)$$

一直迭代上述过程直到收敛，即可获得模型参数，然后就能够进行相应的分类了，此方法简单，易于实现，但是关键是模型假设必须要准确，遗憾的是，在现实任务中往往很难事先做出准确的模型假设。

12.3 半监督 SVM

半监督支持向量机 (S3VM) 是 SVM 在半监督学习上的推广，S3VM 试图找到能将两类有标记样本分开，且穿过数据低密度区域的划分超平面。其中的典型代表为 TSVM，TSVM 也是针对二分类问题的策略，其试图尝试

将每个未标记样本分别作为正例或者反例，然后在所有这些结果中，寻求一个在所有样本上间隔最大化的划分超平面。一旦划分超平面得到确定，那么样本标记也就是当前的指派结果。

显然，上述策略是一个穷举的算法，为了加快效率，TSVM 采用局部搜索迭代来寻找届世界，如下如图所示流程：

输入: 有标记样本集 $D_l = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_l, y_l)\}$;
 未标记样本集 $D_u = \{\mathbf{x}_{l+1}, \mathbf{x}_{l+2}, \dots, \mathbf{x}_{l+u}\}$;
 折中参数 C_l, C_u .

过程:

- 1: 用 D_l 训练一个 SVM_l ;
- 2: 用 SVM_l 对 D_u 中样本进行预测，得到 $\hat{\mathbf{y}} = (\hat{y}_{l+1}, \hat{y}_{l+2}, \dots, \hat{y}_{l+u})$;
- 3: 初始化 $C_u \ll C_l$;
- 4: **while** $C_u < C_l$ **do**
- 5: 基于 $D_l, D_u, \hat{\mathbf{y}}, C_l, C_u$ 求解式(13.9)，得到 $(\mathbf{w}, b), \xi$;
- 6: **while** $\exists\{i, j \mid (\hat{y}_i \hat{y}_j < 0) \wedge (\xi_i > 0) \wedge (\xi_j > 0) \wedge (\xi_i + \xi_j > 2)\}$ **do**
- 7: $\hat{y}_i = -\hat{y}_i$;
- 8: $\hat{y}_j = -\hat{y}_j$;
- 9: 基于 $D_l, D_u, \hat{\mathbf{y}}, C_l, C_u$ 重新求解式(13.9)，得到 $(\mathbf{w}, b), \xi$
- 10: **end while**
- 11: $C_u = \min\{2C_u, C_l\}$
- 12: **end while**

输出: 未标记样本的预测结果: $\hat{\mathbf{y}} = (\hat{y}_{l+1}, \hat{y}_{l+2}, \dots, \hat{y}_{l+u})$

图 13.4 TSVM 算法

12.4 图半监督学习

给定一个数据集，我们可以将其映射成一个图，每个样本对应图中一个节点，若两个样本之间的相似度很高，对应节点之间存在一条边，边的强度正比于样本之间的相似度。我们称有标记样本对应的结点是染过色的，而未标记样本所对应未染色的，那么半监督学习就相当于颜色在图上的传播了。

我们先基于 D_l 和 D_u 构建一个图 $G = \langle V, E \rangle$ ，其中结点集 $V =$

$\{x_1, \dots, x_{l+1}, \dots, x_{l+u}\}$, 边集 E 可以表示一个亲和矩阵, 其高斯定义为:

$$(W)_{ij} = \begin{cases} \exp\left(\frac{-||x_i - x_j||_2^2}{2\sigma^2}\right), & \text{if } i \neq j; \\ 0, & \text{otherwise.} \end{cases} \quad (12.7)$$

假定从图 $G = \langle V, E \rangle$ 将学得一个实值函数 $f : V \rightarrow \mathbb{R}$, 其对应的分类规则为: $y_i = \text{sign}(f(x_i))$, $y_i \in \{-1, +1\}$, 定义关于 f 的能量函数:

$$\begin{aligned} E(f) &= \frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m (W)_{ij} (f(x_i) - f(x_j))^2 \\ &= \frac{1}{2} \left(\sum_{i=1}^m d_i f^2(x_i) + \sum_{j=1}^m d_j f^2(x_j) - 2 \sum_{i=1}^m \sum_{j=1}^m (W)_{ij} f(x_i) f(x_j) \right) \\ &= \sum_{i=1}^m d_i f^2(x_i) - \sum_{i=1}^m \sum_{j=1}^m (W)_{ij} f(x_i) f(x_j) \\ &= f^T (D - W) f \end{aligned} \quad (12.8)$$

其中 $f = (f_l^T f_u^T)^T$, f_l 和 f_u 分别指 f 在有标记样本和无标记样本的预测结果。 $D = \text{diag}(d_1, d_2, \dots, d_{l+u})$ 是一个对角矩阵, 其对角元素 $d_i = \sum_{j=1}^{l+u} (W)_{ij}$ 为矩阵 W 的第 i 行元素之和。

具有最小能量的函数 f 在有标记样本上满足 $f(x_i) = y_i$ ($i = 1, 2, \dots, l$), 在未标记样本中满足 $\Delta f = 0$, 其中 $\Delta = D - W$ 为拉普拉斯矩阵, 采用分块矩阵重写上式:

$$\begin{aligned} W &= \begin{bmatrix} W_{ll} & W_{lu} \\ W_{ul} & W_{uu} \end{bmatrix} \\ D &= \begin{bmatrix} D_{ll} & 0_{lu} \\ 0_{ul} & D_{uu} \end{bmatrix} \\ E(f) &= (f_l^T f_u^T) \left(\begin{bmatrix} D_{ll} & 0_{lu} \\ 0_{ul} & D_{uu} \end{bmatrix} - \begin{bmatrix} D_{ll} & 0_{lu} \\ 0_{ul} & D_{uu} \end{bmatrix} \right) \begin{bmatrix} f_l \\ f_u \end{bmatrix} \\ &= f_l^T (D_u - W_u) f_l - 2 f_u^T W_{ul} f_l + f_u^T (D_{uu} - W_{uu}) f_u \end{aligned} \quad (12.9)$$

由 $\frac{\partial E(f)}{\partial f_u} = 0$ 可得：

$$f_u = (D_{uu} - W_{uu})^{-1}W_{ul}f_l \quad (12.10)$$

令：

$$P = D^{-1}W = \begin{bmatrix} D_{ll}^{-1}W_{ll} & D_{ll}^{-1}W_{lu} \\ D_{uu}^{-1}W_{ul} & D_{uu}^{-1}W_{uu} \end{bmatrix} \quad (12.11)$$

重写 12.10，可得：

$$f_u = (I - P_{uu})^{-1}P_{ul}f_l \quad (12.12)$$

上式说明，在 D_l 上的标记信息作为 $f_l = (y_1; y_2; \dots; y_l)$ 代入上式，即可求出 f_u 对于未标记样本的预测。

对该方法进行拓展，使其能够适用于多分类的标记方法。定义一个 $(l+u) \times |\mathcal{Y}|$ 的非负标记矩阵 $F = (F_1^T, \dots, F_{l+u}^T)^T$ ，其第 i 行元素 $F_i = ((F)_{i1}, \dots, (F)_{i|\mathcal{Y}|})$ 为示例的标记向量，其对应的分类规则为： $y_i = \arg \max_{1 \leq j \leq |\mathcal{Y}|} (F)_{ij}$ 。将 F 初始化为：

$$F(0) = (Y)_{ij} = \begin{cases} 1, & \text{if } (1 \leq i \leq l) \wedge (y_i = j); \\ 0, & \text{otherwise} \end{cases} \quad (12.13)$$

基于 W 构造的一个标记传播矩阵 $S = D^{-\frac{1}{2}}WD^{-\frac{1}{2}}$ ，其中 $D^{-\frac{1}{2}} = \text{diag}(\frac{1}{\sqrt{d_1}}, \dots, \frac{1}{\sqrt{d_{l+u}}})$ ，有如下迭代计算式：

$$F(t+1) = \alpha SF(t) + (1 - \alpha)Y \quad (12.14)$$

其中 $\alpha \in (0, 1)$ ，为用户指定参数，收敛上式可得：

$$F^* = \lim_{t \rightarrow \infty} F(t) = (1 - \alpha)(I - \alpha S)^{-1}Y \quad (12.15)$$

由 F^* 即可获得 D_u 中的样本标记了，算法描述如下：

输入: 有标记样本集 $D_l = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_l, y_l)\}$;
未标记样本集 $D_u = \{\mathbf{x}_{l+1}, \mathbf{x}_{l+2}, \dots, \mathbf{x}_{l+u}\}$;
构图参数 σ ;
折中参数 α .

过程:

- 1: 基于式(13.11)和参数 σ 得到 \mathbf{W} ;
- 2: 基于 \mathbf{W} 构造标记传播矩阵 $\mathbf{S} = \mathbf{D}^{-\frac{1}{2}} \mathbf{W} \mathbf{D}^{-\frac{1}{2}}$;
- 3: 根据式(13.18)初始化 $\mathbf{F}(0)$;
- 4: $t = 0$;
- 5: **repeat**
- 6: $\mathbf{F}(t + 1) = \alpha \mathbf{S} \mathbf{F}(t) + (1 - \alpha) \mathbf{Y}$;
- 7: $t = t + 1$
- 8: **until** 迭代收敛至 \mathbf{F}^*
- 9: **for** $i = l + 1, l + 2, \dots, l + u$ **do**
- 10: $y_i = \arg \max_{1 \leq j \leq |\mathcal{Y}|} (\mathbf{F}^*)_{ij}$
- 11: **end for**

输出: 未标记样本的预测结果: $\hat{\mathbf{y}} = (\hat{y}_{l+1}, \hat{y}_{l+2}, \dots, \hat{y}_{l+u})$

图 13.5 迭代式标记传播算法

上述算法对应于正则化框架:

$$\min_F \frac{1}{2} \left(\sum_{i,j=1}^{l+u} (W)_{ij} \left\| \frac{1}{\sqrt{d_i}} F_i - \frac{1}{\sqrt{d_j}} F_j \right\|^2 \right) + \mu \sum_{i=1}^l \|F_i - Y_i\|^2, \quad (12.16)$$

其中 μ 为正则化参数, 当 $\mu = \frac{1-\alpha}{\alpha}$ 时, 最优解为 F^* .

图半监督学习方法在概念上相当清晰, 易于通过矩阵运算探索算法的性质, 但是涉及大量矩阵运算也会造成算法的开销较大, 并且很难判断新样本在图中的位置, 或者需要引入额外的预测机制。

12.5 基于分歧的方法 (disagreement-based methods)

基于分歧的方法采用多个学习器, 一般采用的算法时协同训练;

一个数据对象往往拥有多个属性集，每个属性集将会构成一个视图，假设不同视图具有相容性 (compatibility)，即其包含的关于输出空间 \mathcal{Y} 的信息是一致的，这样就可以让不同视图之间输出的信息可能相同或者互补，对学习器的构建带来便利。协同训练就是在不同的视图中输出相应的信息，再提供给别的视图进行学习训练，类似“相互学习，共同进步”，一直迭代进行，知道所有的分类器都不在发生变化。

协同过程虽然很简单，但是理论证明：若两个视图充分且条件独立，则可利用未标记样本通过协同训练将弱分类器的泛化性能提高到任意高。研究表明，即使没有那么条件独立，协同训练仍可以有效地提升弱分类器的性能。算法如下：

输入: 有标记样本集 $D_l = \{(\langle \mathbf{x}_1^1, \mathbf{x}_1^2 \rangle, y_1), \dots, (\langle \mathbf{x}_l^1, \mathbf{x}_l^2 \rangle, y_l)\}$;
 未标记样本集 $D_u = \{\langle \mathbf{x}_{l+1}^1, \mathbf{x}_{l+1}^2 \rangle, \dots, \langle \mathbf{x}_{l+u}^1, \mathbf{x}_{l+u}^2 \rangle\}$;
 缓冲池大小 s ;
 每轮挑选的正例数 p ;
 每轮挑选的反例数 n ;
 基学习算法 \mathcal{L} ;
 学习轮数 T .

过程:

- 1: 从 D_u 中随机抽取 s 个样本构成缓冲池 D_s ;
- 2: $D_u = D_u \setminus D_s$;
- 3: **for** $j = 1, 2$ **do**
- 4: $D_l^j = \{(\mathbf{x}_i^j, y_i) \mid (\langle \mathbf{x}_i^j, \mathbf{x}_i^{3-j} \rangle, y_i) \in D_l\}$;
- 5: **end for**
- 6: **for** $t = 1, 2, \dots, T$ **do**
- 7: **for** $j = 1, 2$ **do**
- 8: $h_j \leftarrow \mathcal{L}(D_l^j)$;
- 9: 考察 h_j 在 $D_s^j = \{\mathbf{x}_i^j \mid \langle \mathbf{x}_i^j, \mathbf{x}_i^{3-j} \rangle \in D_s\}$ 上的分类置信度, 挑选 p 个正例置信度最高的样本 $D_p \subset D_s$ 、 n 个反例置信度最高的样本 $D_n \subset D_s$;
- 10: 由 D_p^j 生成伪标记正例 $\tilde{D}_p^{3-j} = \{(\mathbf{x}_i^{3-j}, +1) \mid \mathbf{x}_i^j \in D_p^j\}$;
- 11: 由 D_n^j 生成伪标记反例 $\tilde{D}_n^{3-j} = \{(\mathbf{x}_i^{3-j}, -1) \mid \mathbf{x}_i^j \in D_n^j\}$;
- 12: $D_s = D_s \setminus (D_p \cup D_n)$;
- 13: **end for**
- 14: **if** h_1, h_2 均未发生改变 **then**
- 15: **break**
- 16: **else**
- 17: **for** $j = 1, 2$ **do**
- 18: $D_l^j = D_l^j \cup (\tilde{D}_p^j \cup \tilde{D}_n^j)$;
- 19: **end for**
- 20: 从 D_u 中随机抽取 $2p + 2n$ 个样本加入 D_s
- 21: **end if**
- 22: **end for**

输出: 分类器 h_1, h_2

图 13.6 协同训练算法

上面说明, 基于分歧的算法只需要采用合适的基学习器, 就能够较少地收到模型假设、损失函数非凸性和数据规模的影响, 学习方法简单有效, 但是如何取相对独立的视图, 需要巧妙的设计。

12.6 半监督聚类 (semi-supervised clustering)

聚类任务中获得的监督信息大致有两种类型, 第一种类型是“必连 (must-link)”; 第二种为“勿连 (cannot-link)”。如下可引出采用第一种监督信息的约束 k 均值 (Constrained k -maens) 算法:

输入: 样本集 $D = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_m\}$;
 必连约束集合 \mathcal{M} ;
 勿连约束集合 \mathcal{C} ;
 聚类簇数 k .

过程:

- 1: 从 D 中随机选取 k 个样本作为初始均值向量 $\{\mu_1, \mu_2, \dots, \mu_k\}$;
- 2: **repeat**
- 3: $C_j = \emptyset$ ($1 \leq j \leq k$);
- 4: **for** $i = 1, 2, \dots, m$ **do**
- 5: 计算样本 \mathbf{x}_i 与各均值向量 μ_j ($1 \leq j \leq k$) 的距离: $d_{ij} = \|\mathbf{x}_i - \mu_j\|_2$;
- 6: $\mathcal{K} = \{1, 2, \dots, k\}$;
- 7: is_merged=false;
- 8: **while** \neg is_merged **do**
- 9: 基于 \mathcal{K} 找出与样本 \mathbf{x}_i 距离最近的簇: $r = \arg \min_{j \in \mathcal{K}} d_{ij}$;
- 10: 检测将 \mathbf{x}_i 划入聚类簇 C_r 是否会违背 \mathcal{M} 与 \mathcal{C} 中的约束;
- 11: **if** \neg is_violated **then**
- 12: $C_r = C_r \cup \{\mathbf{x}_i\}$;
- 13: is_merged=true
- 14: **else**
- 15: $\mathcal{K} = \mathcal{K} \setminus \{r\}$;
- 16: **if** $\mathcal{K} = \emptyset$ **then**
- 17: **break** 并返回错误提示
- 18: **end if**
- 19: **end if**
- 20: **end while**
- 21: **end for**
- 22: **for** $j = 1, 2, \dots, k$ **do**
- 23: $\mu_j = \frac{1}{|C_j|} \sum_{\mathbf{x} \in C_j} \mathbf{x}$;
- 24: **end for**
- 25: **until** 均值向量均未更新

输出: 簇划分 $\{C_1, C_2, \dots, C_k\}$

图 13.7 约束 k 均值算法

而第二种监督信息可引出约束种子 k 均值 (Constrained Seed k -maens) 算法。

输入: 样本集 $D = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_m\}$;
 少量有标记样本 $S = \bigcup_{j=1}^k S_j$;
 聚类簇数 k .

过程:

```

1: for  $j = 1, 2, \dots, k$  do
2:    $\mu_j = \frac{1}{|S_j|} \sum_{\mathbf{x} \in S_j} \mathbf{x}$ 
3: end for
4: repeat
5:    $C_j = \emptyset$  ( $1 \leq j \leq k$ );
6:   for  $j = 1, 2, \dots, k$  do
7:     for all  $\mathbf{x} \in S_j$  do
8:        $C_j = C_j \cup \{\mathbf{x}\}$ 
9:     end for
10:   end for
11:   for all  $\mathbf{x}_i \in D \setminus S$  do
12:     计算样本  $\mathbf{x}_i$  与各均值向量  $\mu_j$  ( $1 \leq j \leq k$ ) 的距离:  $d_{ij} = \|\mathbf{x}_i - \mu_j\|_2$  ;
13:     找出与样本  $\mathbf{x}_i$  距离最近的簇:  $r = \arg \min_{j \in \{1, 2, \dots, k\}} d_{ij}$  ;
14:     将样本  $\mathbf{x}_i$  划入相应的簇:  $C_r = C_r \cup \{\mathbf{x}_i\}$ 
15:   end for
16:   for  $j = 1, 2, \dots, k$  do
17:      $\mu_j = \frac{1}{|C_j|} \sum_{\mathbf{x} \in C_j} \mathbf{x}$ ;
18:   end for
19: until 均值向量均未更新
输出: 簇划分  $\{C_1, C_2, \dots, C_k\}$ 
```

图 13.9 约束种子 k 均值算法

12.7 小结

Chapter 13

概率图模型 (probabilistic graphical model)

13.1 隐马尔可夫模型

机器学习最重要的任务，是根据一些以观察到的证据（如训练样本）来对感兴趣的未知变量（如类别标记）进行估计和推测。

概率图模型是一类用图来表示变量相关关系的概率模型，大致可以分为两类：一是使用有向无环图表示变量间的依赖关系，称为有向图模型或贝叶斯网；第二类是使用无向图表示变量间的相关关系，称为无向图模型或者马尔可夫网。

隐马尔可夫模型 (Hidden Markov Model,HMM) 是结构最简单的动态贝叶斯网，是一种著名的有向图模型。如下图所示，HMM 变量可分为两组，第一组是状态变量 $\{y_1, y_2, \dots, y_n\}$ ，其中 $y_i \in \mathcal{Y}$ 表示第 i 时刻的系统状态。通常假定状态变量是隐藏的、不可被观测的，故状态变量亦称为隐变量 (hidden variable)。第二组是观测变量 $\{x_1, x_2, \dots, x_n\}$ ，其中 $x_i \in \mathcal{X}$ 表示第 i 时刻的观测值。

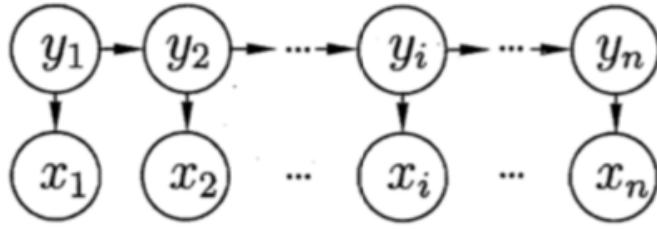


图 14.1 隐马尔可夫模型的图结构

从图上可知, t 时刻的状态 y_t 仅依赖于 $t-1$ 时刻的状态 y_{t-1} , 而与其余的 $n-2$ 个状态无关, 这就是所谓的马尔可夫链: 系统下一时刻的状态仅由当前状态决定, 不依赖以往的任何状态, 基于这种依赖关系, 所有变量的联合概率分布可以写为:

$$P(x_1, y_1, \dots, x_n, y_n) = P(y_1)P(x_1|y_1) \prod_{i=2}^n P(y_i|y_{i-1})P(x_i|y_i) \quad (13.1)$$

除了拥有结构信息, 欲确定一个隐马尔可夫模型还需要三组参数:

- 状态转移概率: 模型在各个状态间转化的概率, 通常记为矩阵 $A = [a_{ij}]_{N \times N}$; 其中

$$a_{ij} = P(y_{t+1} = s_j | y_t = s_i), \quad 1 \leq i, j \leq N \quad (13.2)$$

该公式表示在任意时刻 t , 若状态为 s_i , 则下一时刻状态为 s_j 的概率。

- 输出观测概率: 模型根据当前状态获得哥哥观测值的概率, 通常记为矩阵 $B = [b_{ij}]_{N \times M}$, 其中:

$$b_{ij} = P(x_t = o_j | y_t = s_i), \quad 1 \leq i \leq N, 1 \leq j \leq M \quad (13.3)$$

表示在任意时刻 t , 若状态为 s_i , 则观测值 o_j 被获取的概率。

- 初始状态概率: 模型在初始时刻各状态出现的概率, 通常记为: $\pi = (\pi_1, \pi_2, \dots, \pi_N)$

$$\pi_i = P(y_1 = s_i), \quad 1 \leq i \leq N \quad (13.4)$$

对于上述三种参数，通常用其参数 $\lambda = [A, B, \pi]$ 来指代，给定一个输入 λ ，他按照如下过程产生观测序列 $\{x_1, x_2, \dots, x_n\}$:

1. 设置 $t = 1$ ，并根据初始状态概率 π 选择初始状态 y_1 ；
2. 根据状态 y_t 和输出观测概率 B 选择观测变量取值 x_t ；
3. 根据状态 y_t 和状态转移矩阵 A 转移模型状态，确定 y_{t+1} ；
4. 若 $t < n$ ，设置 $t = t + 1$ ，并转到第二步，否则停止。

13.2 马尔可夫随机场 (Markov Random Field)

MRF 是典型的马尔可夫网，一种著名的无向图模型。如下图所示，图中的每一个节点表示一个或一组变量，节点之间的边表示两个变量之间的依赖关系。还有一组势函数 (potential function)，亦称为因子 (factor)，主要用于定义概率分布函数。

对于图中节点的一个子集，若其中任意两个节点间都有边进行连接，则称该节点子集为一个团 (clique)，若加入一个节点后不再形成团，那么该团就称为极大团。一个节点至少出现在一个极大团之中。

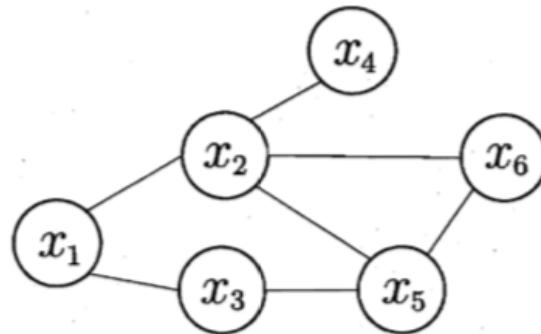


图 14.2 一个简单的马尔可夫随机场

在 MRF 中，多个变量之间的联合概率分布能基于团分解为多个因子的乘积，每个因子仅与一个团相关。假设所有团构成的集合为 \mathcal{C} ，与团 $Q \in \mathcal{C}$

的变量集合为 x_Q , 则联合概率 $P(X)$ 定义为:

$$P(x) = \frac{1}{Z} \prod_{Q \in \mathcal{C}} \psi_Q(x_Q) \quad (13.5)$$

其中 ψ_Q 为团 Q 对应的势函数, 用于对团 Q 中的变量关系进行建模, $X = \sum_x \prod_{Q \in \mathcal{C}} \psi_Q(x_Q)$ 为规范化因子, 而实际中, 精确计算 Z 通常是一件很困难的问题, 许多任务往往并不需要精确地获得 Z 的精确值。

MRF 借助分离的概念从而得到所谓的条件独立性, 如下图所示, 若从结点集 A 中的结点到结点集 B 中的结点都必须经过结点集 C, 则称结点集 A 和 B 被结点集 C 分离, 那么称 C 为分离集, 对马尔可夫随机场, 有: 给定两个变量子集的分离集已知, 则两个变量子集条件独立。这个结论称为全局马尔可夫性;

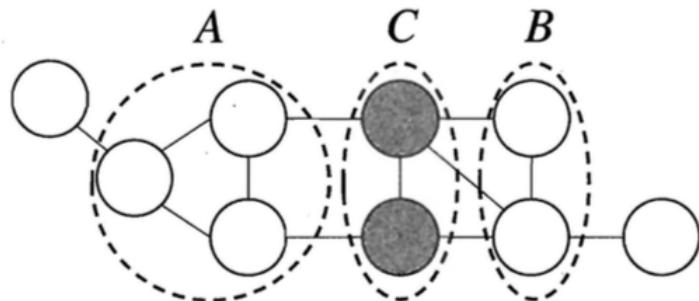


图 14.3 结点集 A 和 B 被结点集 C 分离

由全局马尔可夫性可以得到两个很有用的推论:

- 局部马尔可夫性 (local Markov property): 给定某变量的邻接变量, 则该条件变量独立于其他变量。
- 成对马尔可夫性 (pairwise Markov property): 给定所有其他变量, 两个非邻接变量条件独立。

为了满足非负性, 指数函数通常被用于定义势函数, 即:

$$\psi_Q(x_Q) = e^{-H_Q(x_Q)} \quad (13.6)$$

其中 $H_Q(x_Q)$ 是一个定义在变量 x_Q 上的实值函数，其常见形式为：

$$H_Q(x_Q) = \sum_{u,v \in Q, u \neq v} \alpha_{uv} x_u x_v + \sum_{v \in Q} \beta_v x_v \quad (13.7)$$

13.3 条件随机场 (Conditional Random Field,CRF)

CRF 是一种判别式无向图模型，生成式模型是直接对联合分布进行建模，而判别式模型则是对条件分布进行建模。CRF 视图对多个变量在给定观测值后的条件概率进行建模，具体来说，令 $x = \{x_1, x_2, \dots, x_n\}$ 为观测序列， $y = \{y_1, y_2, \dots, y_n\}$ 为与之对应的标记序列，则条件随机场的目标是构建条件概率模型 $P(y|x)$ 。

令 $G = \langle V, E \rangle$ 表示结点与标记变量 y 中元素一一对应的无向图， y_v 表示与结点 v 对应的标记变量，而 $n(v)$ 表示结点 v 的邻接结点，若 G 中的每个变量 y_v 都满足马尔可夫性，即：

$$P(y_v|x, y_{V/\{v\}}) = P(y_v|x, y_{n(v)}) \quad (13.8)$$

条件随机场使用势函数和图结构上的团来定义条件概率 $P(y|x)$ 。通过选用指数势函数并引入特征函数 (feature function)，条件概率被定义为：

$$P(y|x) = \frac{1}{Z} \exp \left(\sum_j \sum_{i=1}^{n-1} \lambda_i t_j(y_{i+1}, y_i, x, i) + \sum_k \sum_{i=1}^n \mu_k s_k(y_i, x, i) \right) \quad (13.9)$$

其中 $t_j(y_{i+1}, y_i, x, i)$ 是定义在观测序列的两个相邻标记位置上的转移特征函数 (transition feature function)，用于刻画相邻标记变量之间的相关关系以及观测序列对他们的影响， $s_k(y_i, x, i)$ 是定义在观测序列的标记位置 i 上的特征函数。

CRF 和 MRF 都是使用团上的势函数定义概率，两者并没有显著区别，但是 CRF 处理的是条件概率，而 MRF 处理的是联合概率。

13.4 学习与推断

基于概率图模型定义的联合概率分布，使得我们能够对目标变量的边际分布 (marginal distribution) 或以某些可观测变量为条件的条件分布进行

推断，即变成对联合分布中其他无关变量进行积分的过程，被称为”边际化 (marginalization)”。

具体来说，假设图模型所对应的变量集 $x = \{x_1, x_2, \dots, x_N\}$ 能分为 x_E 和 X_F 两个不相交的变量集，推断问题的目标就是计算边际概率 $P(X_F)$ 或条件概率 $P(x_F|x_E)$ ，由条件概率定义可知：

$$P(x_F|x_E) = \frac{P(x_F, x_E)}{P(x_E)} = \frac{P(x_F, x_E)}{\sum x_F P(x_F, x_E)} \quad (13.10)$$

其中联合概率 $P(x_F, x_E)$ 可以基于概率图模型进行获得，因此推断问题的关键就是如何高效的计算边际分布。即：

$$P(x_E) = \sum_{x_F} P(x_E, x_F) \quad (13.11)$$

概率图模型的推断方法大致可分为两类，第一类是精确推断方法，希望能够计算出目标变量的边际分布或条件分布的精确值；遗憾的是，一般情形下，此类算法复杂度会随着极大团规模的增长而呈指数增长。第二类是近似推断方法，希望在较低的时间复杂度下获得原问题的近似解。在现实任务中更加常用。

13.4.1 变量消去

精确推断的实质是一类动态规划的算法，它利用图模型所描述的条件独立性来消减计算目标概率值所需要的计算量，变量消去法是最直观的精确推断算法，也是其他精确算法的基础。这类算法的核心是通过乘法对加法的分配律，变量消去法把多个变量的积的求和问题，转化为对部分变量交替进行求积求和的问题，这种转化将每次求积和求和运算限制在局部，仅与部分变量有关，从而简化了计算。

变量消去法有一个明显的缺点：若需要计算多个边际分布，会造成大量的冗余计算，导致计算效率并不是很高。

13.4.2 信念传播 (Belief Propagation)

信念传播算法将变量消去法中的求和操作看作是一个信息传递过程，从而能够较好的解决求解多个边际分布时的重复计算问题，具体来说，变量消

去法通过求和操作消去变量 x_i :

$$m_{ij}(x_j) = \sum_{x_i} \psi(x_i, x_j) \prod_{k \in n(i)/j} m_{ki}(x_i) \quad (13.12)$$

其中 $n(i)$ 表示结点的邻接结点，在信念传播算法中，这个操作，将会被看成从 x_i 向 x_j 传递了一个消息 $m_{ij}(x_j)$ ，不难发现，每次消息的传递操作仅仅与变量 x_i 及其邻接节点有关，说明消息传递的相关计算是被限制在图的局部进行的。

在信念传播算法中，一个结点仅仅接收来自其他所有结点的消息后才能够向另一个结点发送消息，且结点的边际分布正比于它所接收的消息的乘积，即：

$$P(x_i) \propto \prod m_{ki}(x_i) \quad (13.13)$$

若图结构中没有环，则信念传播算法经过两个步骤即可完成所有的消息传递，进而能计算所有变量上的边际分布：

- 指定一个根结点，从所有叶结点开始向根结点传递消息，直到根结点收到所有邻接结点的消息。
- 从根结点开始向叶结点传递消息，直到所有叶结点均收到消息。

13.5 近似判断

精确推断方法通常需要很大的开销，因此在现实应用中采用近似推断的方法，近似推断的方法大致可以分为两大类：第一类是采样 (sampling)，通过随机化方法完成近似；第二类是使用确定性近似完成近似判断，典型代表就是变分推断 (variational inference)。

13.5.1 MCMC 采样

在很多的任务重，我们关心某些概率分布并非因为对这些概率分布本身感兴趣，而是要基于他们计算某些期望。若直接计算或者逼近这个期望比推断概率分布更容易，那么可以更加快速地进行求解。

采样法正是基于这个思路，具体来说，假定我们的目标是计算函数 $f(x)$ 在概率密度函数 $p(x)$ 下的期望：

$$\mathbb{E}_p[f] = \int f(x)p(x)dx \quad (13.14)$$

则可根据 $p(x)$ 抽取一组样本 $\{x_1, x_2, \dots, x_N\}$ ，然后计算 $f(x)$ 在这些样本上的均值。

$$\hat{f} = \frac{1}{N} \sum_{i=1}^N f(x_i) \quad (13.15)$$

并且以此来近似目标期望 $\mathbb{E}[f]$ ，若样本独立，基于大数定律，这种通过大量采样的方法就能够获得较高的近似精度。

概率图模型中最常用的采样技术是马尔可夫链蒙特卡洛 (MCMC)，给定连续变量 $x \in X$ 的概率密度函数 $p(x)$, x 在区间 A 中的概率可计算为：

$$P(A) = \int_A p(x)dx \quad (13.16)$$

若有函数 $f : X \rightarrow \mathbb{R}$ ，则可计算 $f(x)$ 的期望为：

$$p(f) = \mathbb{E}_p[f(X)] = \int_x f(x)p(x)dx \quad (13.17)$$

如果 x 不是单变量而是一个高维多元变量 x ，且服从一个非常复杂的分布，则对上式直接求积分是很困难的，这时候 MCMC 先构造出一个服从 p 分布的独立同分布的随机变量 $\{x_1, x_2, \dots, x_N\}$ ，然后再得到上式的无偏估计：

$$\bar{p}(f) = \frac{1}{N} \sum_{i=1}^N f(x_i) \quad (13.18)$$

若概率密度函数很复杂，则构造服从 p 分布的独立同分布样本也很困难，而 MCMC 方法的关键就在于如何构建“平稳分布为 p 的马尔可夫链”来产生样本；若马尔可夫链收敛到平稳状态，那么此时产出的样本 x 近似服从于 p ，一般来说。若某个时刻马尔可夫链满足平稳条件：

$$p(x^t)T(x^{t-1}|x^t) = p(x^{t-1})T(x^T|x^{t-1}) \quad (13.19)$$

则 $p(x)$ 就是该马尔可夫链的平稳分布了，其中 $T(x'|x)$ 是马尔可夫链从状态 x 转移到 x' 的状态转移概率。

上面公式说明，MCMC 方法先设法构造一条马尔可夫链，使其收敛至平稳状态，再进行采样计算。

Metropolis-Hastings(MH) 算法是 MCMC 的重要代表，它的思想是采用拒绝采样来逼近平稳分布。算法每次根据上一轮采样的结果 x^{t-1} 来采样获得候选样本 x^* ，但这个候选样本有一定的概率会被拒绝掉，假定从状态 x^{t-1} 到状态 x^* 的状态转移概率为 $Q(x^*|x^{t-1})A(x^*|x^{t-1})$ ，其中 $Q(x^*|x^{t-1})$ 使用用户给定的先验概率， $A(x^*|x^{t-1})$ 是 x^* 被接受的概率，具体算法如下图：

输入：先验概率 $Q(\mathbf{x}^* | \mathbf{x}^{t-1})$.
过程：
1: 初始化 \mathbf{x}^0 ;
2: **for** $t = 1, 2, \dots$ **do**
3: 根据 $Q(\mathbf{x}^* | \mathbf{x}^{t-1})$ 采样出候选样本 \mathbf{x}^* ;
4: 根据均匀分布从 $(0, 1)$ 范围内采样出阈值 u ;
5: **if** $u \leq A(\mathbf{x}^* | \mathbf{x}^{t-1})$ **then**
6: $\mathbf{x}^t = \mathbf{x}^*$
7: **else**
8: $\mathbf{x}^t = \mathbf{x}^{t-1}$
9: **end if**
10: **end for**
11: **return** $\mathbf{x}^1, \mathbf{x}^2, \dots$
输出：采样出的一个样本序列 $\mathbf{x}^1, \mathbf{x}^2, \dots$

图 14.9 Metropolis-Hastings 算法

除此之外，还可以采用吉布斯采样 (Gibbs sampling):

1. 随机或者以某个次序选取某个变量 x_i
2. 根据 x 中除 x_i 外的变量的现有取值，计算条件概率 $p(x_i|X_i)$
3. 根据 $p(x_i|X_i)$ 对变量 x_i 采样，用采样值代替原值。

13.5.2 变分推断

变分推断是通过使用已知简单分布来逼近需要推断的复杂分布，并通过限制近似分布的类型，从而得到局部最优、但具有确定解的近似后验分布。

采用盘式记法，那么所有能观察到的变量 x 的联合分布的概率密度函数是：

$$p(x|\Theta) = \prod_{i=1}^N \sum_z p(x_i, z|\theta) \quad (13.20)$$

对应的对数似然函数为：

$$\ln p(x|\Theta) = \sum_{i=1}^N \ln \left\{ \sum_z p(x_i, z|\Theta) \right\} \quad (13.21)$$

对于概率模型的参数估计通常以最大化对数似然函数为手段，如 EM 算法：

- E 步：根据 t 时刻的参数 Θ^t 对 $p(z|x, \Theta^t)$ 进行推断，并计算联合似然函数 $p(x, z|\Theta)$ ；
- M 步：在基于 E 步的结果进行最大化寻优，即对关于变量 Θ 的函数 $Q(\Theta; \Theta^T)$ 进行最大化从而求取：

$$\begin{aligned} \Theta^{t+1} &= \arg \max_{\Theta} Q(\Theta; \Theta^t) \\ &= \arg \max_{\Theta} \sum_z p(z|x, \Theta^t) \ln p(x, z|\Theta) \end{aligned} \quad (13.22)$$

通过上面的 EM 算法我们得到了一个 z 服从的近似分布 $q(z)$ ；在现实任务中，E 步对 $p(z|x, \Theta^T)$ 的推断很可能由于 z 模型复杂而难以进行，此时可以借助变分推断，通常假设 z 服从分布：

$$q(z) = \prod_{i=1}^M q_i(z_i) \quad (13.23)$$

然而在实践中使用变分法是，最重要的考虑如何对隐变量进行拆解，以及假设各变量自己服从何种分布。若选择不好，会导致变分法效率低、效果差。

13.6 话题模型 (topic model)

话题模型是一族生成式有向图模型，主要用于处理离散型的数据，其中最典型的代表就是 LDA 模型。

现实任务中可以通过统计文档中出现的次来获得词频向量 $w_i (i = 1, 2, \dots, T)$ ，LDA 认为每篇文档包含多个话题，用向量 $\Theta_t \in \mathbb{R}^K$ 表示文档 t 中包含的每个话题的比例， $\Theta_{t,k}$ 即表示文档 t 中包含话题 k 的比例，进而通过下面的步骤由话题“生成”文档 t ：

1. 根据参数为 α 的狄利克雷分布随机采样一个话题的分布 Θ_t
2. 按照下面的步骤生成文档中的 N 个词。
 - (a) 根据 Θ 进行话题指派，得到文档 t 中词 n 的话题 $z_{t,n}$ ；
 - (b) 根据指派的话题所对应的词频分布 β_k 随机采样生成词。

如图所示，文档的词频 $w_{t,n}$ 是唯一的已观测变量，它依赖于对这个词进行的话题指派 $z_{t,n}$ ，以及话题所对应的词频 β_k ，同时话题指派 $z_{t,n}$ 依赖于话题分布 Θ_t ，而 Θ 依赖于狄利克雷分布参数 α ，而话题词频依赖于参数 η

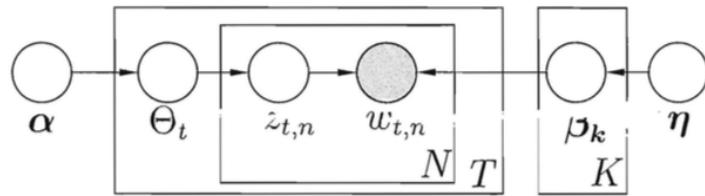


图 14.12 LDA 的盘式记法图

故 LDA 模型所对应的概率分布：

$$p(W, z, \beta, \Theta | \alpha, \eta) = \prod_{t=1}^T p(\Theta_t | \alpha) \prod_{k=1}^K p(\beta_k | \eta) \left(\prod_{n=1}^N P(w_{t,n} | z_{t,n}, \beta_k) P(z_{t,n} | \Theta_t) \right) \quad (13.24)$$

其中 $p(\Theta_t | \alpha)$ 和 $p(\beta_k | \eta)$ 分别设置为以 α 和 η 为参数的 K 维和 N 维狄利克雷分布。而 α, η .

为 α 和 η 最大化对数似然

$$LL(\alpha, \eta) = \sum_{t=1}^T \ln p(w_t | \alpha, \eta) \quad (13.25)$$

为了方便求解，我们通常是根据词频 $w_{t,n}$ 来推断文档集所对应的话题结构，通过求解：

$$p(z, \beta, \Theta | W, \alpha, \eta) = \frac{p(W, z, \beta, \Theta | \alpha, \eta)}{p(W | \alpha, \eta)} \quad (13.26)$$

而在实际应用中，常常采用吉布斯采样或者变分法来近似推断上式的分母 $p(W | \alpha, \eta)$

13.7 小结

Chapter 14

规则学习

规则学习这一章没什么难度，简略说明

14.1 基本概念

规则 (relu) 通常是指语意明确、能描述数据分布所隐含的客观规律或领域概念。

14.2 序贯覆盖

规则学习的目标是产生一个能覆盖尽可能多的样例的规则集，最直接的做法是：序贯覆盖。即进行逐条归纳：在训练集上每学到一条规则，就将该规则覆盖的训练样例去除，将剩下的训练样例组成训练集重复上述的过程。也称为分支策略。

14.3 剪枝优化

规则生成本质上是一个贪心搜索过程，需要有一定机制来缓解过拟合风险，最常见的做法是进行剪枝，具体做法参见决策树。

14.4 一阶规则学习

14.5 归纳逻辑程序设计

14.5.1 最小一般泛化

14.6 小结

Chapter 15

强化学习 (Reinforcement Learning)

15.1 任务与奖赏

强化学习通常可以用马尔可夫决策过程 (Markov Decision Process,MDP) 来描述：机器处于环境 E 中，状态空间为 X ，机器算法决策的动作集 A ，状态转移函数 P ，回馈奖赏机制 (reward) R ，那么强化学习任务就对应一个四元组： $E = \langle X, A, P, R \rangle$ 。

强化学习算法要做的就是通过在 E 中不断地进行尝试从而学到一个策略 (policy) π ，根据俄这个策略，在状态 x 下就能知道下一步执行的动作 $a = \pi(x)$ 。策略一般有两种表现方法，第一是将策略表现为函数 $\pi : X \rightarrow A$ ，常用于确定性表示，另一种是概率表示法 $\pi : X \times A \rightarrow \mathbb{R}$ ，用 $\pi(x, a)$ 表示状态 x 下选择动作 a 的概率，必须有 $\sum_a \pi(x, a) = 1$ 。

15.2 K-摇臂赌博机

15.2.1 探索与利用

由于强化学习的最终奖赏一般需要在很多步后才能观察到，采用贪心的策略，我们最大化单步的奖赏 (reward)。完成这个任务，需要知道每个动作

带来的奖赏和执行奖赏的最大化动作，若一个动作的奖赏值来自于一个概率分布，那么仅通过一次尝试并不能确切地获得平均奖赏值。

单步强化学习对应了一个理论模型，即“K-摇臂赌博机”，通过探索每个摇臂来确定奖赏期望，通过利用来获得最大期望。

探索和期望两者是矛盾的，因为尝试次数是有限的，会造成此长彼消的窘境（探索-利用窘境 (Exploration-Exploitation dilemma)），欲得到最大化奖赏，那么必须在探索和利用中达成较好的折中。

15.2.2 ϵ -贪心

ϵ -贪心法是基于一个概率来进行探索和利用：每次尝试时，以 ϵ 的概率进行探索，并且以 $1 - \epsilon$ 来进行利用。

令 $Q(k)$ 记录摇臂 k 的平均奖赏，为了减少尝试次数，采用增量式计算均值，即记录已尝试次数 ($n-1$) 和当前的平均奖赏 $Q_{n-1}(k)$ ，令 v_n 表示第 n 次的奖赏，则：

$$Q_n(k) = \frac{(n-1) \times Q_{n-1}(k) + v_n}{n} \quad (15.1)$$

算法流程为：

Algorithm 6 ϵ -贪心算法

In: 摆臂编号 K ; 奖赏函数 R ; 尝试次数 T ; 探索概率 ϵ

Out: 累计奖赏 r ;

```
1: function  $\epsilon$ -GREEDY( $K, R, T, \epsilon$ )
2:    $r = 0$ ;
3:    $\forall i = 1 \sim K : Q(i) = 0, count(i) = 0$ 
4:   for  $t = 1, 2, \dots, T$  do
5:     if  $\text{rand}() < \epsilon$  then
6:        $k$  从  $1, 2, \dots, K$  随机选取一个摆臂编号, 用作探索
7:     else
8:        $k = \arg \max_i Q(i)$ , 用作利用
9:     end if
10:     $v = R(k), r = r + v$ ;
11:    采用增量式计算  $Q(k)$ 
12:     $count(k) = count(k) + 1$ ;
13:   end for
14: end function
```

一般的, 通常令 ϵ 取一个较小的常数, 如 0.1 或者 0.01; 即进行少量探索大量利用。一段时间后, 若摆臂奖赏都能很好地近似出来, 那么可以让 ϵ 随着尝试次数逐渐减小, 如 $\epsilon = \frac{1}{\sqrt{t}}$.

15.2.3 softmax

softmax 算法是基于当前已知的摆臂平均奖励来进行探索和折中:

softmax 算法中的摆臂选择概率为:

$$P(k) = \frac{e^{\frac{Q(k)}{\tau}}}{\sum_{i=1}^K e^{\frac{Q(i)}{\tau}}} \quad (15.2)$$

其中 $Q(i)$ 记录的是当前摆臂的平均奖赏, τ 称为温度, 其值越小则说明平均奖赏高的摆臂被选择的概率会越高。 τ 趋于 0 时, 将趋于仅利用, 而 τ 趋于无穷大时, 将趋于仅探索。

Algorithm 7 softmax 算法

In: 摆臂编号 K ; 奖赏函数 R ; 尝试次数 T ; 温度参数 τ

Out: 累计奖赏 r ;

```
1: function SOFTMAX( $K, R, T, \epsilon$ )
2:    $r = 0$ ;
3:    $\forall i = 1 \sim K : Q(i) = 0, count(i) = 0$ 
4:   for  $t = 1, 2, \dots, T$  do
5:      $k$  从  $1, 2, \dots, K$  根据上式随机选取
6:      $v = R(k), r = r + v$ ;
7:     采用增量式计算  $Q(k)$ 
8:      $count(k) = count(k) + 1$ ;
9:   end for
10: end function
```

k -揆臂赌博机的做法有很多局限，他没有考虑强化学习任务中的马尔可夫决策过程。

15.3 有模型学习

考虑多步强化学习的过程，暂且先假定任务对应的马尔可夫决策过程的四元组 $E = \langle X, A, P, R \rangle$ 均为已知。在情况下状态转移概率 $P_{x \rightarrow x'}^a$ 是已知的，该转移带来的奖励 $R_{x \rightarrow x'}^a$ 也是已知的。

15.3.1 策略评估

令 $V^\pi(x)$ 表示从 x 出发，使用策略 π 所带来的积累奖赏，函数 $Q^\pi(x, a)$ 表示从 x 出发，执行动作 a 后再使用策略 π 所带来的积累奖赏。有下面的状态值函数和状态-动作值函数：

$$\begin{cases} V_T^\pi(x) = \mathbb{E}_\pi \left[\frac{1}{T} \sum_{t=1}^T \tau_t | x_0 = x \right], T \text{步积累奖励} \\ V_\gamma^\pi(x) = \mathbb{E}_\pi \left[\sum_{t=0}^{+\infty} \gamma^t \tau_{t+1} | x_0 = x \right], \gamma \text{折扣积累奖赏} \end{cases} \quad (15.3)$$

$$\begin{cases} Q_T^\pi(x, a) = \mathbb{E}_\pi \left[\frac{1}{T} \sum_{t=1}^T \tau_t | x_0 = x, a_0 = a \right], \\ Q_\gamma^\pi(x, a) = \mathbb{E}_\pi \left[\sum_{t=0}^{+\infty} \gamma^t \tau_{t+1} | x_0 = x, a_0 = a \right]; \end{cases} \quad (15.4)$$

由于 MDP 的马尔可夫性质，对于 T 步的积累奖赏和 γ 折扣奖赏有：

$$V_T^\pi(x) = \sum_{a \in A} \pi(x, a) \sum_{x' \in X} P_{x \rightarrow x'}^a \left(\frac{1}{T} R_{x \rightarrow x'}^a + \frac{T-1}{T} V_{T-1}^\pi(x') \right) \quad (15.5)$$

$$V_\gamma^\pi(x) = \sum_{a \in A} \pi(x, a) \sum_{x' \in X} P_{x \rightarrow x'}^a (R_{x \rightarrow x'}^a + \gamma V_\gamma^\pi(x')) \quad (15.6)$$

假设值函数的初始值 V_0^π 开始，通过一次迭代能计算出每个状态的单步奖励 V_1^π ，从而通过迭代来求出 V_n^π ：

Algorithm 8 基于 T 步积累奖励的策略评估算法

In: MDP 四元组 $E = \langle X, A, P, R \rangle$; 被评估的策略 π ; 积累奖励参数 T ;

Out: 状态值函数 V ;

```

1: function T-STEPREWARDS( $E, \pi, T$ )
2:    $\forall x \in X : V(x) = 0$ 
3:   for  $t = 1, 2, \dots, n$  do
4:      $\forall x \in X : V'(x) = \sum_{a \in A} \pi(x, a) \sum_{x' \in X} P_{x \rightarrow x'}^a (\frac{1}{t} R_{x \rightarrow x'}^a +$ 
       $\frac{t-1}{t} V(x'))$ ;
5:     if  $t = T + 1$  then
6:       break
7:     else
8:        $V = V'$ 
9:     end if
10:   end for
11: end function
```

对于 V_γ^π ，若 $\lim_{t \rightarrow \infty} (\gamma)^t = 0$ ，也能采用上面的算法。为了防止迭代次数过多，可以设置一个阈值 θ ，若 $|V - V'| < \theta$ ，则停止算法。

若我们通过迭代计算出了 V , 那么可以直接计算出状态-动作值函数。

$$\begin{cases} Q_T^\pi(x, a) = \sum_{x' \in X} P_{x \rightarrow x'}^a (\frac{1}{T} R_{x \rightarrow x'}^a + \frac{T-1}{T} V_{T-1}^\pi(x')) \\ Q_\gamma^\pi(x, a) = \sum_{x' \in X} P_{x \rightarrow x'}^a + \gamma V_\gamma^\pi(x') \end{cases} \quad (15.7)$$

15.3.2 策略改进

对于一个策略的积累奖赏进行评估之后, 若发现它并非最优策略, 则希望能对其进行改进。理想的策略是希望能够最大化积累奖励:

$$\pi^* = \arg \max_{\pi} \sum_{x \in X} V^\pi(x) \quad (15.8)$$

一般来说, 一个强化学习任务可能具有许多个最优策略, 而最优策略所对应的值函数 V^* 则称为最优值函数:

$$\forall x \in X : V^*(x) = V^{\pi^*}(x) \quad (15.9)$$

对 15.5 和 15.6 进行改动, 求和改成求最优:

$$\begin{cases} V_T^*(x) = \max_{a \in A} \sum_{x' \in X} P_{x \rightarrow x'}^a (\frac{1}{T} R_{x \rightarrow x'}^a + \frac{T-1}{T} V_{T-1}^*(x')); \\ V_\gamma^*(x) = \max_{a \in A} \sum_{x' \in X} P_{x \rightarrow x'}^a (R_{x \rightarrow x'}^a + \gamma V_\gamma^*(x')) \end{cases} \quad (15.10)$$

同理可得最优状态-动作值函数:

$$\begin{cases} Q_T^*(x, a) = \sum_{x' \in X} P_{x \rightarrow x'}^a (\frac{1}{T} R_{x \rightarrow x'}^a + \frac{T-1}{T} \max_{a' \in A} Q_{T-1}^*(x', a')); \\ Q_\gamma^*(x, a) = \sum_{x' \in X} P_{x \rightarrow x'}^a (R_{x \rightarrow x'}^a + \gamma \max_{a' \in A} Q_\gamma^*(x', a')) \end{cases} \quad (15.11)$$

上述式子说明, 将策略选择的动作改变成当前最优的动作, 这样能够使得策略更加好。即:

$$\pi'(x) = \arg \max_{a \in A} Q^\pi(x, a) \quad (15.12)$$

当 $\pi' = \pi$ 时, 就满足了 Bellman 等式, 找到了最优策略。

15.3.3 策略迭代与值迭代

从一个初始策略（通常是随机策略）出发，先进行策略评估，然后改进策略，评估改进的策略，再进一步地改进策略，不断迭代进行策略评估和改进，直到策略收敛，不再发生改变为止。这种做法叫做“策略迭代（policy iteration）”，下面算法就是 t 步积累奖励算法的策略迭代算法。

Algorithm 9 基于 T 步积累奖励的策略迭代算法

In: MDP 四元组 $E = \langle X, A, P, R \rangle$; 积累奖励参数 T ;

Out: 最优策略 π

```
1: function T-STEPPOLICY( $E, T$ )
2:   loop
3:     for  $t = 1, 2, \dots, n$  do
4:        $\forall x \in X : V'(x) = \sum_{a \in A} \pi(x, a) \sum_{x' \in x} P_{x \rightarrow x'}^a (\frac{1}{t} R_{x \rightarrow x'}^a +$ 
         $\frac{t-1}{t} V(x'));$ 
5:       if  $t = T + 1$  then
6:         break
7:       else
8:          $V = V'$ 
9:       end if
10:      end for
11:       $\forall x \in X : \pi'(x) = \arg \max_{a \in A} Q(x, a)$ 
12:      if  $\forall x : \pi'(x) = \pi(x)$  then
13:        break
14:      else
15:         $\pi = \pi'$ 
16:      end if
17:    end loop
18: end function
```

同理可以得到值迭代（value iteration）算法：

Algorithm 10 基于 T 步积累奖励的值迭代算法

In: MDP 四元组 $E = \langle X, A, P, R \rangle$; 积累奖励参数 T ; 收敛阈值 θ

Out: 策略 $\pi(x) = \arg \max_{a \in A} Q(x, a)$;

```
1: function T-STEPVALUEPOLICY( $E, T, \theta$ )
2:    $\forall x \in X : V(x) = 0$ 
3:   for  $t = 1, 2, \dots, n$  do
4:      $\forall x \in X : V'(x) = \max_{a \in A} \sum_{x' \in X} P_{x \rightarrow x'}^a (\frac{1}{t} R_{x \rightarrow x'}^a + \frac{t-1}{t} V(x'))$ ;
5:     if  $\max_{x \in X} |V(x) - V'(x)| < \theta$  then
6:       break
7:     else
8:        $V = V'$ 
9:     end if
10:   end for
11: end function
```

若要采用 γ 折扣奖励, 只需将上面的第三行代码替换成:

$$\forall x \in X : V'(x) = \max_{a \in A} \sum_{x' \in X} P_{x \rightarrow x'}^a (R_{x \rightarrow x'}^a + \gamma V(x')) \quad (15.13)$$

这说明, 在模型已知是强化学习任务能归结为基于动态规划的寻优问题。

15.4 免模型学习

在现实任务中, 很难找出现成的模型, 若学习算法不依赖于环境的建模, 我们称为”免模型的学习 (model-free learning)”:

15.4.1 蒙特卡洛强化学习

受到 k -摇臂赌博机的启发, 一种直接策略评估替代的方法是多次进行”采样”, 然后求取平均积累奖赏来作为期望积累奖赏的近似, 这就是蒙特卡洛学习方法, 由于采样必须为有限的次数, 故该方法更加适合 T 步积累奖赏的强化学习任务。

在模型未知的情况下，机器只能从一个启示状态开始探索环境，所以只能在探索的过程中逐渐发现各个状态并估计各个状态-动作对的值函数。

综上，在模型未知的情形下，我们从启示状态出发，使用某种策略进行采样，那么执行该策略 T 步并获得轨迹：

$$\langle x_0, a_0, \tau_1, x_1, a_1, \tau_2, \dots, x_{T-1}, a_{T-1}, \tau_T, x_T \rangle \quad (15.14)$$

上面公式的格式是记录每一对状态-动作 (x, a) ，及之后的奖赏之和 τ 。记录多条轨迹后，进行求和平均，即可得到状态-动作值函数的估计。同理，为了避免老是获得相同的轨迹，我们采用 ϵ -贪心法。以 ϵ 的概率从所有的动作中随机选取一个，以 $1 - \epsilon$ 的概率选取当前最优动作。这样的话每个动作都有可能会被选，说明多次采样将会产生不同的采样轨迹。同理可用上一张的方式进行相应的策略改进。

在上述的算法描述之中，被评估与被改进的是用一个策略，因此称为“同策略 (on-policy)”蒙特卡洛强化学习算法，算法中奖赏均值采用增量式计算，每采样出一条轨迹，就根据这条轨迹进行值更新计算：

Algorithm 11 同策略蒙特卡洛强化学习算法

In: 环境 E ; 动作空间 A ; 起始状态 x_0 ; 策略执行步数 T

Out: 策略 π ;

```
1: function ON-POLICY-MONTE-CARLO-METHOD( $E, A, x_0, T$ )
2:    $Q(x, a) = 0, count(x, a) = 0, \pi(x, a) = \frac{1}{|A(x)|}$ 
3:   for  $s = 1, 2, \dots, n$  do
4:     在  $E$  中执行策略  $\pi$  产生轨迹;
5:     for  $t = 0, 1, \dots, T - 1$  do
6:        $R = \frac{1}{T-t} \sum_{i=t+1}^T \tau_i;$ 
7:        $Q(x_t, a_t)$  增量式更新;
8:        $count(x_t, a_t) = count(x_t, a_t) + 1;$ 
9:     end for
10:    对所有已见状态  $x$ :
           
$$\pi(x, a) \begin{cases} \arg \max_{a'} Q(x, a'), & p = 1 - \epsilon \\ random(A), & p = \epsilon \end{cases}$$

11:   end for
12: end function
```

上述算法最终产生的是 ϵ -贪心策略，然而引入 ϵ -策略仅仅是为了评估策略，而不需要在使用策略时使用，这就需要异策蒙特卡洛来实现了。

假设两个不同的策略 π 和 π' 产生采样轨迹，两者的区别在于每个“状态-动作对”被采样的概率不同。使用策略 π 的采样轨迹来评估策略 π ，实际上就是累积奖励的估计期望：

$$Q(x, a) = \frac{1}{m} \sum_{i=1}^m \tau_i \quad (15.15)$$

若改用策略 π' 的采样轨迹来评估策略 π ，则仅需对积累奖励进行加权，即：

$$Q(x, a) = \frac{1}{m} \sum_{i=1}^m \frac{P_i^\pi}{P_i^{\pi'}} \tau_i \quad (15.16)$$

其中 P_i^π 和 $P_i^{\pi'}$ 分别表示两个策略产生第 i 条轨迹的概率，给定一条轨迹，策略 π 产生该轨迹的概率为：

$$P^\pi = \prod_{i=0}^{T-1} \pi(x_i, a_i) P_{x_i \rightarrow x_{i+1}^{a_i}} \quad (15.17)$$

故：

$$\frac{P^\pi}{P^{\pi'}} = \prod_{i=0}^{T-1} \frac{\pi(x_i, a_i)}{\pi'(x_i, a_i)} \quad (15.18)$$

若 π 为确定性的概率而 π' 是关于 π 的 ϵ -贪心策略，则 $\pi(x, a)$ 始终为 1， $\pi'(x_i, a_i)$ 为 $\frac{\epsilon}{|A|}$ 或 $1 - \epsilon + \frac{\epsilon}{|A|}$ ，这就是异策略的蒙特卡洛强化学习：

Algorithm 12 异策略蒙特卡洛强化学习算法

In: 环境 E ; 动作空间 A ; 起始状态 x_0 ; 策略执行步数 T

Out: 策略 π ;

```

1: function OFF-POLICY-MONTE-CARLO-METHOD( $E, A, x_0, T$ )
2:    $Q(x, a) = 0, count(x, a) = 0, \pi(x, a) = \frac{1}{|A(x)|}$ 
3:   for  $s = 1, 2, \dots, n$  do
4:     在  $E$  中执行策略  $\pi$  产生轨迹;
5:
6:      $p_i = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|A|}, & a_i = \pi(x); \\ \frac{\epsilon}{|A|}, & a_i \neq \pi(x) \end{cases}$ 
7:     for  $t = 0, 1, \dots, T - 1$  do
8:        $R = \frac{1}{T-t} \sum_{i=t+1}^T (\tau_i \times \prod_{j=i}^{T-1} \frac{1}{p_j})$ ;
9:        $Q(x_t, a_t)$  增量式更新;
10:       $count(x_t, a_t) = count(x_t, a_t) + 1$ ;
11:      end for
12:       $\pi(x) = \arg \max_{a'} Q(x, a')$ ;
13:    end for
end function

```

15.4.2 时序差分学习

蒙特卡洛算法必须得到一条完整的采样轨迹后才能开始更新策略值函数，这样就无法利用动态规划，导致计算效率会低得多，时序差分差分算法 *Temporal Difference, TD* 就是结合和动态规划和蒙特卡洛方法。

蒙特卡洛方法的本质是多次尝试后求平均来作为期望累积奖赏的近似，但是我们可以采用增量式进行更新，假设给予 t 个采样已经估计出的值函数 $Q_t^\pi(x, a) = \frac{1}{t} \sum_{i=1}^t \tau_i$ ，那么得到第 $t+1$ 个采样 τ_{t+1} 有：

$$Q_{t+1}^\pi = Q_t^\pi(x, a) + \frac{1}{t+1}(\tau_{t+1} - Q_t^\pi(x, a)) \quad (15.19)$$

一般的，将 $\frac{1}{t+1}$ 替换成系数 α_{t+1} 。通常令 α 为一个较小的正数值 α ，那么更新步长 α 越大，说明越靠后的积累奖励越重要。

以 γ 折扣积累奖励为例，利用动态规划方法可知：

$$Q^\pi(x, a) = \sum_{x' \in X} P_{x \rightarrow x'}^a (R_{x \rightarrow x'}^a + \gamma \sum_{a' \in A} \pi(x', a') Q^\pi(x', a')) \quad (15.20)$$

通过增量求和：

$$Q_{t+1}^\pi(x, a) = Q_t^\pi(x, a) + \alpha(R_{x \rightarrow x'}^a + \gamma Q_t^\pi(x', a') - Q_t^\pi(x, a)) \quad (15.21)$$

根据上面的式子，得到的就是 Sarsa 算法：

Algorithm 13 Sarsa 强化学习算法

In: 环境 E ; 动作空间 A ; 起始状态 x_0 ; 奖励折扣 γ ; 更新步长 α ;

Out: 策略 π ;

```
1: function SARSA( $E, A, x_0, T$ )
2:    $Q(x, a) = 0, \pi(x, a) = \frac{1}{|A(x)|};$ 
3:    $x = x_0, a = \pi(x);$ 
4:   for  $s = 1, 2, \dots, n$  do
5:      $r, x' =$  在  $E$  中执行动作  $a$  产生的奖赏与转移的状态;
6:      $a' = \pi^\epsilon(x');$ 
7:      $Q(x, a) = Q(x, a) + \alpha(r + \gamma Q(x', a') - Q(x, a));$ 
8:      $\pi(x) = \arg \max_{a''} Q(x, a'');$ 
9:      $x = x', a = a';$ 
10:    end for
11: end function
```

若将 Sarsa 修改为异策略算法，则得到 Q -学习 (Q-learning) 算法：

Algorithm 14 Q 强化学习算法

In: 环境 E ; 动作空间 A ; 起始状态 x_0 ; 奖励折扣 γ ; 更新步长 α ;

Out: 策略 π ;

```
1: function Q-LEARNING( $E, A, x_0, T$ )
2:    $Q(x, a) = 0, \pi(x, a) = \frac{1}{|A(x)|};$ 
3:    $x = x_0;$ 
4:   for  $s = 1, 2, \dots, n$  do
5:      $r, x' =$  在  $E$  中执行动作  $\pi^\epsilon(x)$  产生的奖赏与转移的状态;
6:      $a' = \pi^\epsilon(x');$ 
7:      $Q(x, a) = Q(x, a) + \alpha(r + \gamma Q(x', a') - Q(x, a));$ 
8:      $\pi(x) = \arg \max_{a''} Q(x, a'');$ 
9:      $x = x', a = a';$ 
10:    end for
11: end function
```

15.5 值函数近似

在现实任务中的状态空间往往是连续的，无穷个状态。我们不妨直接对连续状态空间的值函数进行学习，假定状态空间为 n 维实数空间 $X = \mathbb{R}^n$ ，考虑简单情形，即值函数能表达为状态的线性函数：

$$V_\theta(x) = \theta^T x \quad (15.22)$$

其中 x 为状态向量， θ 为参数向量。这样的值函数的求解被称为值函数近似 (value function approximation)，与真实值函数 V^π 的近似程度通常采用最小二乘误差来度量：

$$E_\theta = \mathbb{E}_{x \sim \pi} [(V^\pi(x) - V_\theta(x))^2] \quad (15.23)$$

为了使误差最小化，采用梯度下降法，对误差求负导数：

$$-\frac{\partial E_\theta}{\partial \theta} = \mathbb{E}_{x \sim \pi} [2(V^\pi(x) - V_\theta(x))x] \quad (15.24)$$

得到单个样本的更新规则：

$$\theta = \theta + \alpha(V^\pi(x) - V_\theta(x))x \quad (15.25)$$

由于我们并不知道策略的真实值函数 V^π ，但可借助时序差分学习，基于 $V^\pi(x) = r + \gamma V^\pi(x')$ ：

$$\theta = \theta + \alpha(r + \gamma \theta^T x' - \theta^T x)x \quad (15.26)$$

基于线性值函数近似代替 Sarsa 算法中的值函数，既可以得到线性值函数近似 Darsa 算法：

Algorithm 15 线性值函数近似 Sarsa 强化学习算法

In: 环境 E ; 动作空间 A ; 起始状态 x_0 ; 奖励折扣 γ ; 更新步长 α ;

Out: 策略 π ;

```
1: function LINEAR-SEMI-SARSA( $E, A, x_0, T$ )
2:    $\theta = 0$ ;
3:    $x = x_0, a = \pi(x) = \arg \max_{a''} \theta^T(x; a'')$ ;
4:   for  $s = 1, 2, \dots, n$  do
5:      $r, x' =$  在  $E$  中执行动作  $a$  产生的奖赏与转移的状态;
6:      $a' = \pi^\epsilon(x')$ ;
7:      $\theta = \theta + \alpha(r + \gamma \theta(x'; a') - \theta^T(x; a))(x; a)$ ;
8:      $\pi(x) = \arg \max_{a''} Q(x, a'')$ ;
9:      $x = x', a = a'$ ;
10:    end for
11: end function
```

15.6 模仿学习

强化学习模仿人类专家系统进行的学习。

15.6.1 直接模仿学习

强化学习的状态空间过大，如果能模仿人类专家的”状态-动作对”，可以显著缓解这一困难，我们称之为”直接模仿学习”。

基于人类专家的策略可以构建一个初始策略，进而再加以更新改进。

15.6.2 逆强化学习

在很多任务中，设计奖励函数往往相当困难，从人类专家提供的范例数据中反推出奖励函数有助于解决该问题，这就是逆强化学习。

在逆强化学习中，已知 X, A 和一个决策轨迹数据集 $\{\tau_1, \tau_2, \dots, \tau_m\}$ ，其基本思路是：要使得机器做出与范例一致的行为，等价于在某个激励函数

的环境中求解最优策略。换句话说，我们要寻找某种奖励函数使得范例数据是最优的。

假设奖励函数能表达为状态特征的线性函数，即 $R(x) = w^T w$ ，于是策略 π 的累计奖励可以写为：

$$\rho^\pi = \mathbb{E} \left[\sum_{t=0}^{+\infty} \gamma^t R(x_t) | \pi \right] = w^T \mathbb{E} \left[\sum_{t=0}^{+\infty} \gamma^t x_t | \pi \right] \quad (15.27)$$

将状态向量的期望 $\mathbb{E}[\sum_{t=0}^{+\infty} \gamma^t x_t | \pi]$ 简写为 \bar{x}^π ，采用蒙特卡洛采样法，将每条范例轨迹上的状态加权求和再平均，记为 \bar{x}^* ，对于最优奖励函数 $R(x) = w^{*T} x$ 和其他任意策略产生的 \bar{x}^π ，有：

$$w^{*T} \bar{x}^* - w^{*T} \bar{x}^\pi \geq 0 \quad (15.28)$$

若能计算出 $(\bar{x}^* - \bar{x}^\pi)$ ，则可以解出：

$$w^* = \arg \max_w \min_\pi w^T (\bar{x}^* - \bar{x}^\pi), \quad s.t. \quad \|w\| \leq 1 \quad (15.29)$$

算法如下图所示：

Algorithm 16 迭代式逆强化学习算法

In: 环境 E ; 状态空间 X ; 动作空间 A ; 范例轨迹数据集 D ;

Out: 奖励函数 $R(x) = w^{*T} x$ 和策略 π ;

- 1: **function** INVERSE-RL(E, X, A, D)
 - 2: x^* = 从范例轨迹中算出状态加权和的均值向量。
 - 3: π 为随机策略；
 - 4: **for** $t = 1, 2, \dots, n$ **do**
 - 5: \bar{x}_t^π = 从 π 的采样轨迹中算出状态加权和的均值向量。
 - 6: 求解 $w^* = \arg \max_w \min_{i=1}^T w^T (\bar{x}^* - \bar{x}^\pi)$, s.t. $\|w\| \leq 1$
 - 7: π = 在环境 $\langle X, A, R(x) = w^{*T} x \rangle$ 中求解最优策略；
 - 8: **end for**
 - 9: **end function**
-

15.7 小结