机器学习导论札记

杨跞仁

2024 年 7 月

目录

1 引言

在当今的科技时代，机器学习已成为推动创新和变革的重要力量。它不仅是计算机科学的一部分，更是各行各业实现智能化转型的关键技术。从自动驾驶汽车到智能语音助手，机器学习的应用无处不在。作为人工智能的基础学科，了解和掌握机器学习技术是每个计算机学子的必修课。

这本《机器学习导论札记》是笔者在大二上学期于华中科技大学武汉光电国家研究中心，在全廷伟教授的指导下完成的自学《机器学习》课程的学习笔记。同时，它也是笔者作为数学与统计学院智慧开源项目负责人期间撰写的一本开源笔记。在这里，也是非常感谢数学与统计学院胡颖教授对我的栽培。札记包含了西瓜书前 9 章的主要内容，笔者分为线性模型、聚类、决策树、集成学习、支持向量机、神经网络、贝叶斯分类器七章进行讲解。关于实战，大家可以关注 Datawhale 获取更多资讯。

希望这本《机器学习导论札记》能帮助大家更好地学习机器学习。在学习过程中，笔者参考了南京大学周志华教授撰写的《机器学习》一书（俗称“西瓜书”）。西瓜书内容科学严谨，作者识见高远，但对入门者来说可能稍显艰深。入门学习时，可以暂时不必过于纠结于数学公式的推导，更应注重理解基本思想，力求能用简单通俗的语言将相关方法脱口而出。本札记的创作初心是整理自己零散的学习笔记，供自己在遗忘时进行翻阅。笔者仅仅是一名普通本科的在读学生，见识难免短浅，笔调难免稚嫩，内容难免粗糙。因此，还望读者诸君海涵。如发现错误，笔者将非常荣幸得到您的不吝赐教。大家可以关注《师苑数模》公众号，与我们联系。

2 线性模型

2.1 线性模型原理的讲述

下面的流程图呈现了我对于线性模型的大致梳理过程，下面的讲述也将按照这个顺序进行。

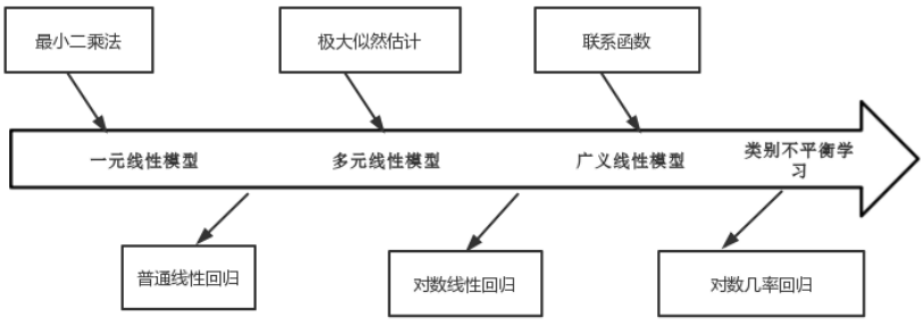

图 1. 线性模型梳理流程图

2.1.1 线性模型的目的及效果

在学习任何模型之前，首先应当清楚其需要达到的目的（一针见血地道出是在做什么事）。而**线性模型** (linear model) 就是在试图学得一个通过属性的线性组合来进行预测的函数：

$$f(x) = w_1x_1 + w_2x_2 + \cdots + w_dx_d + b \tag{1}$$

上式写成向量形式为：

$$f(x) = w^Tx + b \tag{2}$$

所以，我们需要做的便是学得参数 w, b，来将模型确定。由此可见，线性模型的效果是：**简单、基本、可解释性好**。

2.1.2 线性回归

线性回归的本质是希望学得：

$$f(x_i) = wx_i + b, s.t. f(x_i) \approx y_i \tag{3}$$

若涉及离散的数据，采取的措施如下：若有“序”(order)，则连续化；否则，转化为 k 维向量。下面就该想想如何确定参数 w, b，关键在于如何找到预测函数 $f(x)$ 与真实值 (ground truth) 之间的差异，这一差异可以利用均方误差进行度量，令**均方误差**最小化，有：

$$\begin{aligned}(w^*, b^*) &= \arg\min_{(w,b)} \sum_{i=1}^{m} (f(x_i) - y_i)^2 \\ &= \arg\min_{(w,b)} \sum_{i=1}^{m} (y_i - wx_i - b)^2\end{aligned} \tag{4}$$

对 $E_{(w,b)} = \sum_{i=1}^{m}(y_i - wx_i - b)^2$ 进行最小二乘估计，分别对 w 和 b 求偏导，并令偏导数为 0, 得到闭式 (closed-form) 解：

$$w = \frac{\sum_{i=1}^{m} y_i(x_i - \overline{x})}{\sum_{i=1}^{m} x_i^2 - \frac{1}{m}(\sum_{m}^{i=1} x_i)^2}, b = \frac{1}{m}\sum_{m}^{i=1}(y_i - wx_i) \tag{5}$$

2.1.3 多元线性回归

把 w 和 b 吸收入向量形式 $\hat{w} = (w, b)$ 的数据集表示为：

$$X = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1d} & 1 \\ x_{21} & x_{22} & \cdots & x_{2d} & 1 \\ \vdots & \vdots & \cdots & \vdots & 1 \\ x_{m1} & x_{m2} & \cdots & x_{md} & 1 \end{bmatrix} = \begin{bmatrix} x_1^T & 1 \\ x_2^T & 1 \\ \vdots & \vdots \\ x_m^T & 1 \end{bmatrix}, y = (y_1, y_2, \cdots, y_n) \tag{6}$$

同样采用最小二乘法求解，有：

$$\hat{w}^* = \arg\min_{\hat{w}}(y - X\hat{w})^T(y - X\hat{w}) \tag{7}$$

令 $E_{\hat{w}}$ 为式（7）中的主体目标函数，对 $\hat{w}$ 求偏导：

$$\frac{\partial E_{\hat{w}}}{\partial \hat{w}} = 2X^T(X\hat{w} - y) \tag{8}$$

令此偏导数为 0，即可得到 $\hat{w}$ 的最优解的闭式解。然而不幸的是，这样处理将会遇到一个大麻烦——**涉及矩阵求逆：**

（1）若 X^TX 满秩或正定，则 $\hat{w}^* = (X^TX)^{-1}X^Ty$

（2）若 X^TX 不满秩，则可解出多个 $\hat{w}$，此时往往需求助于归纳偏好，或引入 **正则化** (regularization)。

线性模型看似简单，但是依然灵活多变。在线性回归中，本质是将预测值逼近真实值 y。事实上，我们还可以令预测值逼近 y 的衍生物，这样就可以得到更多的多元线性回归模型。例如令 $\ln y = w^Tx + b$，就得到了**对数线性回归** (log-linear regression)，如图 2 所示，这种做法其实是在用 e^{w^Tx+b} 逼近 y。

2.1.4 广义线性回归与对率回归

引子：在上例对数线性回归模型中，对数函数起到的作用便是将线性模型的预测值与真实标记联系起来的作用。那么推广到更一般的情况，我们考虑单调可微函数 $g(\cdot)$，得到广义线性回归的一般形式：

$$y = g^{-1}(w^Tx + b) \tag{9}$$

式（9）中的 g^{-1} 便称作单调可微的**联系函数** (link function)，它的作用是把线性回归产生的结果与实际需要的结果联系起来。那么正是因为有联系函数作为纽带，所以回归模

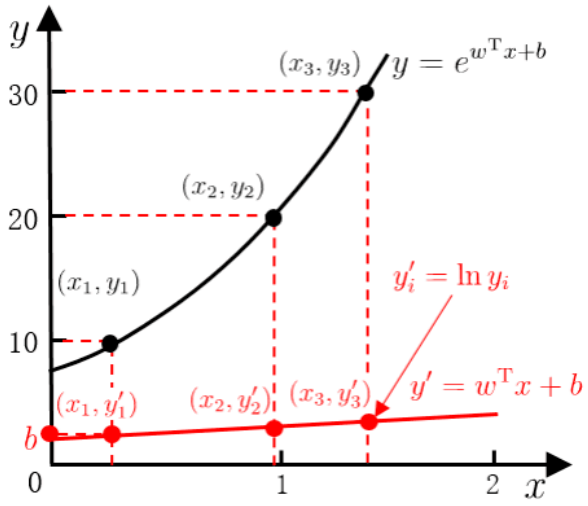

图 2. 对数线性回归示意图

型能处理分类任务。而下面要讲述的**对数几率回归** (logistic odds regression) 也是一种**分类学习算法**。

以二分类任务为例，线性回归模型产生的实值输出为 $z = w^T x + b$，但我们的期望输出为 $y \in [0, 1]$。因此，我们需要找到一个替代函数，将实值转化到 [0,1] 区间内。我们可以找到一个理想的**“单位阶跃函数”** (unit-step function)：

$$y = \begin{cases} 0, & z < 0 \\ 1/2, & z = 0 \\ 1, & z > 0 \end{cases} \tag{10}$$

式（10）中，若预测值 z 大于 0 则判定为正例，小于 0 则判定为负例，等于临界值 0 则可以任意判别。但是由于此函数不连续，性质不好，所以我们需要找到一个**连续**的常用**可微**且**任意阶可导**的替代函数，而 Sigmoid(S 型函数) 正是具备以上优美性质的函数，如图 3，其解析式见式（11）：

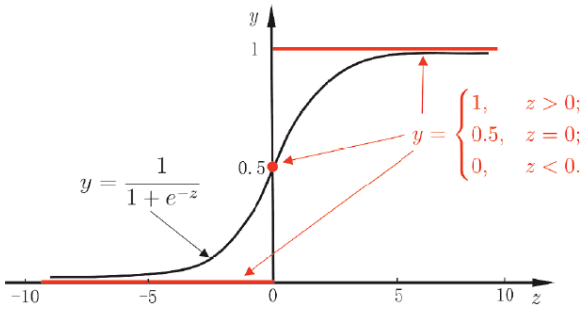

图 3. S 形曲线示意图

$$y = \frac{1}{1+e^{-z}} = \frac{1}{1+e^{-(w^T x+b)}} \tag{11}$$

这个替代函数便称作**对数几率函数** (logistic function)。简称“**对率函数**”。以对率函数为联系函数，式（11）可以化简为：

$$\ln \frac{y}{1-y} = w^T x + b \tag{12}$$

式（12）中的 $\frac{y}{1-y}$ 则反映了 x 作为正例的可能性，即几率 (odds) 的得名由来。下面给出对数几率回归的求解思路：若将 y 看作类后验概率估计 $p(y=1 \mid x)$，则式（12）可以改写为：

$$\ln \frac{p(y=1 \mid x)}{p(y=0 \mid x)} = w^T x + b \tag{13}$$

于是，可使用“极大似然法”(maximum likelihood method)，给定数据集 $\{(x_i, y_i)\}_{i=1}^m$，其最大化对数似然函数 (log-likelihood function) 可以写成：

$$l(w,b) = \sum_{i=1}^{m} \ln p(y_i \mid x_i; w, b) \tag{14}$$

式（14）的本质是令每个样本属于其真实标记的概率越大越好，即要求：

$$\max\{p(\text{真实为正例})p(\text{预测为正例}) + p(\text{真实为负例})p(\text{预测为负例})\} \tag{15}$$

令 $\beta = (w; b), \quad \hat{x} = (x; 1)$, 则 $w^T x + b$ 可以简记为 $\beta^T \hat{x}$。再令：

$$\begin{aligned} p_1(\hat{x}_i, \beta) &= p(y=1 \mid \hat{x}_i; \beta) = \frac{e^{w^T x + b}}{1 + e^{w^T x + b}}, \\ p_0(\hat{x}_i, \beta) &= p(y=0 \mid \hat{x}_i; \beta) = 1 - p(y=1 \mid \hat{x}_i; \beta) = \frac{1}{1 + e^{w^T x + b}}, \end{aligned} \tag{16}$$

结合上述各式可将似然项重写为：

$$p(y_i \mid \hat{x}_i; w, b) = y_i p_1(\hat{x}_i, \beta) + (1 - y_i) p_0(\hat{x}_i, \beta) \tag{17}$$

于是，最大化对数似然函数（15）等价于最小化下式：

$$l(\beta) = \sum_{i=1}^{m} (-y_i \beta^T \hat{x}^i + \ln(1 + e^{\beta^T \hat{x}^i})) \tag{18}$$

显然，式（17）是关于的一个高阶可导连续凸函数，可用经典的数值优化方法：如梯度下降法 (gradient descent method) 或者牛顿法 (Newton method) 求得最优解。

2.1.5 类别不平衡学习

简而言之，类别不平衡 (class-imbalance)，便是指不同类别的样本比例相差很大的情况，此时“小类”往往更重要。以正例较少反例较多来举例，其基本思路是若 $\frac{y}{1-y} > \frac{m^+}{m^-}$，（$m^+$ 表示正例数目，m^- 表示负例数目）则预测为正例。又因为分类器是在基于进行决策的，因此我们采用“再缩放”(rescaling) 的策略进行决策：分类器是在基于若

$\frac{y}{1-y} > 1$ 则预测为正例进行决策的，因此我们采用“再缩放”(rescaling) 的策略进行决策：

$$\frac{y'}{1-y'} = \frac{y}{1-y} \times \frac{m^+}{m^-} \tag{19}$$

然而，这是基于“训练集是真实样本总体的无偏采样”这个假设下才成立的，实际任务中往往不会如此理想，精确估计观测几率 $\frac{m^+}{m^-}$ 往往很困难，因此无法使用训练集观测几率来推断出真实几率。我们在现实中常见的类别不平衡学习方法有：过采样 (oversampling)，欠采样 (undersampling) 以及阈值移动 (threshold-moving)。

2.2 基于线性模型的一些问题探讨

2.2.1 在极大似然法中，为什么通常取自然对数？

因为概率通常很小，计算机很难处理非常小的小数，取自然对数可以有效防止浮点数下溢的情况。

2.2.2 对数几率回归有哪些优点？

（1）无需事先假设数据分布；

（2）可得到“类别”的近似概率预测；

（3）可直接应用现有数值优化算法求取最优解。

2.2.3 在对数几率回归中，为什么不能用最小二乘法（基于均方误差的性能度量）去处理？

先回顾一下最小二乘法的步骤：

对于函数：$f(x) = \frac{1}{1+e^{-(w^T x+b)}}$，找 $[f(x)-y]^2_{\min} \Rightarrow$ 找$[\frac{1}{1+e^{-(w^T x+b)}} - y]_{\min}$ 此后令偏导数为 0，得到最优闭式解。

不难看出，最小二乘法的最终解一定是梯度为 0 处的解。事实上，梯度为 0 处的解不一定是最优解，最小二乘法这样处理的原理是基于优化**凸函数**的前提下，梯度为 0 处的解才对应最优解。而在对数几率回归中，可以证明需要优化的函数（20）是非凸的，所以最小二乘法不适用。

$$g(x,y) = \frac{1}{1+e^{-(w^T x+b)}} - y \tag{20}$$

2.2.4 最小二乘法与极大似然法有什么联系？

笔者将二者的关系概括如下：极大似然估计是就是确定（估计）一个模型参数是当前事件最有可能得到，当数据测量误差项服从标准正态分布时，其结果和最小二乘法一致。

2.2.5 极大似然估计与贝叶斯估计有什么差异？

在已知数据集 D 服从某种概率分布 P，P 中参数 θ 未知。极大似然估计认为未知 θ 是一个定值，目标是找到这个参数 θ 使得数据集 D 发生的概率最大，记为 $\max p(D \mid \theta)$。而贝叶斯估计则是认为未知参数 θ 本身服从一定的概率分布，目标是找到当数据集 D 发生的前提下，哪一个参数 θ 发生的概率最大，记为：$\max p(\theta \mid D)$。

数据集	参数估计方法	数学刻画
D	极大似然估计	$\hat{\theta} = \arg\max p(D \mid \theta)$
D	贝叶斯估计	$\hat{p}(\theta \mid D) = \frac{p(\theta)p(D\|\theta)}{P(D)}$

表 1. 两种估计的数学刻画对比表

由此，贝叶斯估计引入了未知参数 θ 所服从的先验概率，去求得当数据集 D 发生前提下的后验概率。假设先验概率分布是均匀分布，然后取后验概率最大，就能从贝叶斯估计中得到极大似然估计。

3 基础聚类

3.1 常见聚类方法

聚类（clustering）是在“**无监督学习**”任务中研究最多、应用最广的一类方法。其目标是将数据样本划分为若干个通常不相交的“簇”(cluster)，既可以作为一个单独过程（用于找寻数据内在的分布结构），也可作为分类等其他学习任务的前驱过程。

各位宝子们可以欣赏故事一则：

老师拿来苹果和梨，让小朋友分成两份。

小明把大苹果大梨放一起，小个头的放一起，老师点头，嗯，体量感。

小芳把红苹果挑出来，剩下的放一起，老师点头，颜色感。

小武的结果？不明白。小武掏出眼镜：最新款，能看到水果里有几个籽，左边这堆单数，右边双数。

老师很高兴：新的聚类算法诞生了。

这则故事说明了什么呢？聚类的标准往往无需墨守陈规，而是基于我们当前的任务灵活选择的。因此，聚类也许是机器学习中“新算法”出现最多、最快的领域，总能找到一个新的“标准”，使以往算法对它无能为力。

常见的聚类方法可以分类如下：

原型聚类

- 亦称“基于原型的聚类”(prototype-based clustering)
- 假设：聚类结构能通过一组原型刻画
- 过程：先对原型初始化，然后对原型进行迭代更新求解
- 代表：k 均值聚类，学习向量量化 (LVQ)，高斯混合聚类

密度聚类

- 亦称“基于密度的聚类”(density-based clustering)
- 假设：聚类结构能通过样本分布的紧密程度确定
- 过程：从样本密度的角度来考察样本之间的可连接性，并基于可连接样本不断扩展聚类簇
- 代表：DBSCAN, OPTICS, DENCLUE

层次聚类 (hierarchical clustering)

- 假设：能够产生不同粒度的聚类结果
- 过程：在不同层次对数据集进行划分，从而形成树形的聚类结构
- 代表：AGNES (自底向上)，DIANA (自顶向下)

接下来，笔者将为大家介绍两种用的最多的聚类方式。

3.2 K-Means 聚类

此法是原型聚类的一个古老且经典的算法，其原理很简单：首先随机地选取出若干个样本作为初始的均值向量（簇）；然后分别计算每一个样本点与初始均值向量的距离，与哪个均值向量最近就属于哪个簇；最后将每个簇重新计算中心点，重复第二步直到收敛。

在搞清操作步骤之后，我们需要了解在上述操作中都蕴含了什么知识呢？笔者概括如下：

（1）K-means 算法的核心原理是什么？

此算法最终肯定希望得到“簇内相似度”高且“簇间相似度”低的聚类效果，而对于 K-均值的理解，便是要找到一个最优解来刻画每一个簇内的样本围绕这个簇的均值（初始均值向量）的紧密程度高。用公式表达如下：对于簇划分 $C = \{c_1, c_2, ..., c_n\}$，最小化平方误差：

$$E = \sum_{i=1}^{k} \sum_{x \in c_i} \|x - \mu_i\|_2^2, \text{其中均值向量} \mu_i = \frac{1}{|c_i|} \sum_{x \in c_i} x \tag{1}$$

（2）K-means 算法为什么要做重复第二步的操作直到收敛？

因为要寻找最小化的 E 值是一个 NP 难问题，所以只能通过贪心的思想，通过一次又一次的迭代优化来近似求解。

（3）如何计算距离？

给定样本集 $x_i = (x_{i1}, x_{i2}, ..., x_{in})$ 与样本集 $x_j = (x_{j1}, x_{j2}, ..., x_{jn})$，通常使用**闵可夫斯基距离**（Minkowski distance）来计算：

$$dist_{mk}(x_i, x_j) = (\sum_{\mu=1}^{n} |x_{iu} - x_{ju}|^p)^{1/p} \tag{2}$$

式（2）中，当 $p \to \infty$ 时，则得到**切比雪夫距离**；

当 p=2 时，则得到**欧式距离**:

$$dist_{ed}(x_i, x_j) = \sqrt{\sum_{\mu=1}^{n} |x_{iu} - x_{ju}|^2} \tag{3}$$

当 p=1 时，则得到**曼哈顿距离**：

$$dist_{man}(x_i, x_j) = \sum_{\mu=1}^{n} |x_{iu} - x_{ju}| \tag{4}$$

在处理连续属性时通常使用上述的闵可夫斯基距离来计算，但当处理离散型属性时，闵可夫斯基距离就不再适用，此时将采用 VDM 距离进行计算：

$$VDM_{p(a,b)} = \sum_{i=1}^{k} \left| \frac{m_{u,a,i}}{m_{u,a}} - \frac{m_{u,b,i}}{m_{u,b}} \right|^p \tag{5}$$

式（5）中，令 $m_{u,a}$ 表示属性 u 上取值为 a 的样本数，$m_{u,a,i}$ 表示在第 i 个样本簇中属性 u 上取值为 a 的样本数，k 为样本簇数，则表达两个离散数据 a 与 b 之间的 VDM 距离。

若数据样本既有连续值又有离散值该怎么办呢？很简单，将闵可夫斯基距离与 VDM 距离混合起来，就可以处理混合数据：

$$Mincov DM_p = (\sum_{u=1}^{n_c} |x_{iu} - x_{ju}|^p + \sum_{u=n_c+1}^{n} VDM_p(x_{iu} - x_{ju}))^{1/p} \tag{6}$$

当样本空间中不同属性的重要性不同时，可以使用加权闵可夫斯基距离：

$$dist_{wmk}(x_i, x_j) = (w_1 |x_{i1} - x_{j1}|^p + ... + w_n |x_{in} - x_{jn}|^p)^{1/p} \tag{7}$$

在现实任务中，有必要基于数据样本来确定合适的距离计算计算式。

3.3 高斯混合聚类

此法的基本原理是：高斯混合分布并不是高斯模型，而是若干个模型的混合物形态，哪个数据样本更加属于高斯模型，就被分到哪一类中。与 k-Means 不同的是，高斯混合聚类是通过概率模型来表达聚类原型的方法。

基本操作步骤概括如下：（1）初始化高斯混合成分的个数 k，假设高斯混合分布模型参数　为（高斯混合系数），u 为均值，Σ 为协方差矩阵。

（2）分别计算每个样本点的后验概率（该样本点属于每一个高斯模型的概率）。

（3）迭代，重复第二步操作直至收敛。

通过分析不难发现，高斯混合聚类的难点就在于后验概率的计算（该样本点属于每一个高斯模型的概率）。具体做法参见方羽新《机器学习基础实验 GMM 和 PCA》，在此就不多做赘述了。

4 决策树

4.1 决策树的认识

决策树是基于树的结构进行决策的算法，每个“内部结点”对应于某个属性上的“测试”，每个分支对应于该测试的一种可能结果（即该属性的某个取值），每个“叶结点”对应于一个“预测结果”。

我们可以用两种视角来理解决策树：

其一，可以将决策树看作是一条由 **if-then** 规则构成的链条，即从根结点到每一个叶子结点的每一条路径都构建一个规则，那么这条路径中的内部结点特征表示**规则的条件**，其对应的叶子结点则表示**规则的结论**。

其二，可以将决策树模型理解为一种概率模型。若 X 为特征的随机变量，Y 为类的随机变量，相应的条件概率分布可以表示为：$P(Y \mid X)$。当叶子结点上的条件概率分布偏向于某一类时，那么属于该类的概率更大。

4.2 决策树的基本流程

处理决策树问题的核心策略是“**分而治之**”（divide-and-conquer），这种处理思路类似于数据结构讲排序算法中的快速排序。这样做是因为决策树的本质其实是自根至叶的递归过程，我们需要在每个中间结点寻找一个“划分”（split or test）属性。其基本流程可以概述如下：

学习过程：通过对训练样本的分析来确定“划分属性”（即内部结点所对应的属性）。

预测过程：将测试示例从根结点开始，沿着划分属性所构成的“判定测试序列”下行，直到叶结点。

下面引用周志华老师在西瓜书中所呈现的伪代码流程（如表 2）：

上述的基本流程中，有以下几点值得关注：

（1）决策树基本流程中的三种**停止条件**有：

情形一：当前结点包含的样本全属于同一类别，无需划分（过程 2 4）；

情形二：当前属性集为空，或是所有样本在所有属性上取值相同，无法划分（过程 5 7）；

情形三：当前结点包含的样本集合为空，不能划分（过程 11 13）。

（2）过程中蓝色矩形所标注的做法的实质是**利用当前结点的后验分布**；橙色矩形所标注的做法的实质是**将父结点的样本分布作为当前结点的先验分布**。

（3）过程中红色矩形所标注的做法是**决策树算法的核心！**从训练数据中众多的特征中选择一个最优特征作为当前结点的分裂标准的做法称为**特征选择**。

表 2. 决策树算法流程

输入: 训练集 $D = \{(x_1, y_1), (x_2, y_2), ..., (x_m, y_m)\}$;
属性集 $A = \{\alpha_1, \alpha_2, ..., \alpha_d\}$;

过程: 函数 TreeGenerate(D, A)
1. 生成结点 node;
2. **if**: D 中样本全属于同一类别 C，**then**
3. 将 node 标记为 C 类叶结点; **return**
4. **end if**
5. **if**: $A = \emptyset$ or D 中样本在 A 上取值相同 **then**
6. 将 node 标记为叶结点，其类别标记为 D 中样本数最多的类; **return**
7. **end if**
8. 从 A 中选择最优划分属性 a_*;
9. **for** a_* 的每一个值 a'_* do
10. 为 node 生成一个分支; 令 D'_* 表示 D 中在 a_* 上取值为 a'_* 的样本子集;
11. **if**: D'_* 为空 **then**
12. 将分支结点标记为叶结点，其类别标记为 D 中样本最多的类; **return**
13. **else**
14. 以 TreeGenerate($D'_*, A \setminus \{a'_*\}$) 为分支结点
15. **end if**
16. **end if**
17. **end for**
18. **输出**: 以 node 为根结点的一棵决策树

4.3 决策树的特征选择

4.3.1 信息增益（Information Gain）

信息增益是 ID3（Iterative Dichotomiser，迭代二分器 Three）算法中应用的特征选择指标。

信息熵（entropy）是度量样本集合“纯度”最常用的一种指标。假定当前样本集合 D 中第 k 类样本所占的比例为 p_k，则 D 的信息熵定义为:

$$Ent(D) = -\sum_{k=1}^{|y|} p_k \log_2 p_k \tag{1}$$

在计算信息熵时，我们约定：若 p=0，则 $p \log_2 p = 0$。由上式可以看出 Ent(D) 的最小值为 0，最大值为 $\log_2 |y|$。Ent(D) 的值越小，D 的纯度越高。信息增益直接以信息熵为基础，计算当前划分对信息熵所造成的变化。若离散属性 a 的取值是：$\{a^1, a^2, ..., a^V\}$，

第 V 个分支 D^{ν} 表示 D 中在 a 上取值 D^{ν} 等于的样本集合，那么以属性 a 对数据集 D 进行划分所获得的信息增益为：

$$Gain(D,a) = Ent(D) - \sum_{\nu=1}^{V} \frac{|D^{\nu}|}{|D|} Ent(D^{\nu}) \tag{2}$$

其中，Ent(D) 指的是划分前的信息熵，$\frac{|D^{\nu}|}{|D|}$ 指的是第 ν 个分支的权重，$Ent(D^{\nu})$ 指的是划分后的信息熵。

4.3.2 增益率（Gain Ratio）

增益率是 C4.5 决策树算法中应用的特征选择指标。上一点中提到的信息增益有一个明显的弱点，那就是对可取值数目较多的属性有所偏好。比如以**索引编号**作为划分属性，就会导致每个分支结点都有且仅有一个样本，纯度达到最大，很明显毫无泛化能力。为了消除这一不利影响，著名的 C4.5 算法不直接使用信息增益，而是选择了增益率作为特征选择标准。

增益率的定义是：

$$Gainratio(D,a) = \frac{Gain(D,a)}{IV(a)} \tag{3}$$

其中 a 的**固有值**（intrinsic value）定义为：

$$IV(a) = -\sum_{\nu=1}^{V} \frac{|D^{\nu}|}{|D|} \log_2 \frac{|D^{\nu}|}{|D|} \tag{4}$$

属性 a 的可能取值数目越多（即 V 越大），则 IV(a) 的值通常就越大。

因为增益率准则对可取值数目较少的属性有所偏好，所以 C4.5 算法没有直接选择增益率最大的候选属性，而是采用一个**启发式**：先从候选划分属性中找出信息增益高于平均水平的，再从中选取增益率最高的。

4.3.3 基尼指数（Gini Index）

基尼指数是 CART（Classification and Regression Tree）算法中应用的特征选择指标。

CART 是在给定输入随机变量条件下输出随机变量的条件概率分布的学习方法。CART 算法通过选择最优特征和特征值进行划分，将输入空间也就是特征空间划分为有限个单元，并在这些单元上确定预测的概率分布，也就是在输入给定的条件下输出条件概率分布。其主要包括回归树和分类树两种。回归树用于目标变量为连续型的建模任务，其特征选择准则用的是**平方误差最小准则**。分类树用于目标变量为离散型的的建模任务，其特征选择准则用的是基尼指数，这也有别于此前 ID3 的信息增益准则和 C4.5 的增益率准则。无论是回归树还是分类树，其算法核心都在于递归地选择最优特征构建决策树。

数据集 D 的纯度可以由**基尼值**来度量，基尼值定义为：

$$Gini(D) = \sum_{k=1}^{|y|} \sum_{k' \neq k} p_k p_{k'} = 1 - \sum_{k=1}^{|y|} p_k^2 \tag{5}$$

式 (5) 中，Gini(D) 反映了从 D 中随机抽取两个样例，其类别标记不一致的概率。因此，Gini(D) 越小，数据集 D 的纯度越高。属性 a 的基尼指数定义为：

$$GiniIndex(D, a) = \sum_{\nu=1}^{V} \frac{|D^\nu|}{|D|} Gini(D^\nu) \tag{6}$$

在候选属性集合中，选取那个使划分后基尼指数最小的属性即可。

4.4 决策树的剪枝处理

研究表明，划分选择的各种准则虽然对决策树的尺寸有较大影响，但对泛化性能的影响很有限，例如信息增益与基尼指数产生的结果，仅在约 2% 的情况下不同。而剪枝方法和程度对决策树泛化性能的影响更为显著，在数据带噪声时甚至可能将泛化性能提升 25%。为什么会产生这样的结果？这是因为剪枝（pruning）是决策树对付“过拟合”的主要手段！剪枝处理是为了尽可能正确分类训练样本，通过主动去掉一些分支来降低过拟合的风险的操作。

剪枝处理的基本策略有两种：

情形一：**预剪枝**（pre-pruning），即**提前终止某些分支的生长**。因为其在训练开始前就提前砍掉了很多分支，所以在降低过拟合风险的同时也节省了时间及成本的开销。但是，预剪枝基于“贪心”本质禁止这些分支展开，这样也会给预剪枝决策树带来了欠拟含的风险。

情形二：**后剪枝**（post-pruning），即**先生成一个完整树，再回头剪枝**。一般情形下，后剪枝决策树的欠拟合风险很小，泛化性能往往预剪枝决策树更好。但是由于要自底向上对树中的所有非叶结点进行逐一考察，其训练时间和成本开销会大得多。

剪枝过程中需评估剪枝前后决策树的优劣，即要使用到机器学习中的**评估方法**。如留出法、交叉验证法和自助法。

4.5 缺失值处理策略

如果遇到某些样本数据丢失的情况，我们如何解决以下两个问题？Q1：如何进行划分属性选择？Q2：给定划分属性，若样本在该属性上的值缺失，如何进行划分？

基本处理思路可以概括为**“样本赋权，权重划分”**，这个思路不仅适用于决策树模型，也可适用于集成学习中的提升树模型。

针对 Q1，用无缺样本代替总体来进行计算，从而进行划分。注意，这里“无缺样本”指的并不是“所有属性都完整的样本”，而是对于某一属性，集合内该属性没有缺

失的样本（有可能一个样本可以参与属性 a 的计算但是不可以参与属性 b 的计算）给出新的信息增益计算公式：

$$Gain(D,a) = \rho \times Gain(D,a) = \rho \times [Ent(D) - \sum_{\nu=1}^{V} \tilde{r}_\nu Ent(\tilde{D}^\nu)] \tag{7}$$

其中，$Ent(\tilde{D}) = -\sum_{k=1}^{|y|} \widetilde{p}_k \log_2 \widetilde{p}_k$。对于属性 a，$\rho$ 是无缺样本所占比例，$\tilde{p}_k$ 是无缺样本中第 k 类样本所占比例，$\tilde{r}_\nu$ 是无缺样本中在属性 a 上取值为 a^ν 的所占比例。

针对 Q2，以 $\tilde{r}_\nu$ 为新权重，划分到每个子结点中（每个子结点的权重不同，和为 1）。权重划分进入分支的过程可以理解为**把有值的样本进入各个属性划分结果（后验概率）当作了无值样本进入各个属性划分结果（先验概率）**。

决策树的原理相对来说很容易理解，它的提出是受到了香农提出的信息论的影响。所以，无论是 ID3、CART 还是 C4.5，它们的划分准则指标都与信息相关。然而令人耳目一新的是，我们也可以从条件概率分布的角度理解决策树，在第二节中呈现出来了，这非常美妙！

最后是代码，可以去尝试使用经典的数据集“今天你是否要打高尔夫”，利用 sklearn 库中的 tree 模型构建决策树。笔者构建的 ID3 大致效果如图 4，囿于篇幅，代码就不展示了。

图 4. 一棵 ID3 决策树

5 集成学习

5.1 集成学习综述

在学习一个新的领域之前，必须清楚其基本思想。古语有云“三个臭皮匠，顶个诸葛亮”，关于集成学习的研究开端，是源于 1989 年莱斯利·维利昂特和迈克尔·肯斯提出的一个公开问题：“弱可学习性是否等价于强可学习性？”这个问题也就是说如果一个机器学习任务存在着比“随机猜测”好一些的弱学习办法，就必然存在着准确率任意高的强学习办法。所谓**集成学习**（ensemble learning）便是先产生一组**个体学习器**（individual learner），再通过某种策略将其结合起来，从而得到更好的效果解决问题。

5.1.1 如何得到好的集成

要想得到好的集成效果，个体学习器应该**“好而不同”**：个体学习器间应当有一定的准确率（比随机猜测的准确率高），且要具有**多样性**（diversity）（学习器间应有差异）。

可以从**误差-分歧分解**（error-ambiguity decomposition）的角度来说明多样性是得到好的集成的关键所在：

$$E = \overline{E} - \overline{A} \tag{1}$$

式（1）中，E 指的是集成学习的误差（ensemble error），$\overline{E}$ 指的是每个个体的错误取平均值（Ave.error of individuals），$\overline{A}$ 指的是每个个体间的差异度（Ave.ambiguity of individuals）。这个式子的含义是个体学习器准确性越高、多样性越大，集成效果越好。但是，如何取舍 error 与 ambiguity 的问题成为了设计算法的挑战。

那么有没有一种数学方法能够度量集成学习中个体分类器的多样性呢？事实上，现在已经提出了很多种**多样性度量**（diversity measure）方法，如不合度量、相关系数、Q-统计量、K-统计量..... 但是遗憾的是，迄今为止都没有找到一种最合适的数学刻画得到认可。因此，如何理解和度量多样性也成为了集成学习领域的**圣杯**（holy grail）问题。

5.1.2 常用集成学习方法

1.Boosting 算法家族：

这是一类**序列化**集成学习算法，即个体学习器间存在强依赖关系，必须串行生成。使用同质分类器，顺序训练，后续学习器着重解决前者预测不准的样本，将多个弱学习器加权求和组合成一个强学习器，强学习器的预测结果作为最终预测结果。其流程图如图 5.

2.Bagging 算法：

这是一类**并行化**集成学习算法，即个体学习器间不存在强依赖关系、可同时生成。使用同质分类器，并行训练，最终对多个弱学习器的结果进行投票或求均值的方式作为最终预测结果。著名的**随机森林**（random forest）便是此类方法应用最多的变体。Bagging 的流程图如图 6.

3.Stacking 算法：

与前两类不同的是，这是一类使用异质分类器，多个弱分类器并行训练，然后最后一层加一个神经网络对弱分类器的结果进行总结，输出一个最终预测结果的算法。但是由于深度学习的发展，此类算法已经被弃如敝履，因此不多做赘述。

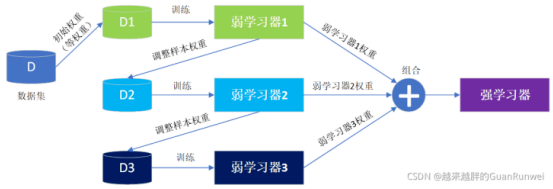

图 5. Boosting

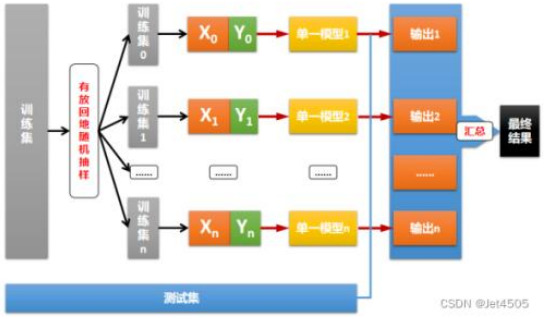

图 6. Bagging

5.2 AdaBoost

自适应增强算法（Adaptive Boosting）是 Boosting 家族的最著名的代表，原理颇有点“前人栽树，后人乘凉”的味道，其采用一种**顺序、级联**的结构，后一个模型的训练永远是在前一个模型的基础上完成的。AdaBoost 提高了对前一轮弱分类器分类错误样本的关注度（提高其权重），降低正确样本的权重；是对多个弱分类器进行线性组合的**加性模型**，提高分类效果好的弱学习器的权重，降低分类效果差的弱学习器的权重。其训练流程叙述如下：

给定训练样本集 $D = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$ 进行二分类任务：

Step1：权重初始化

初始化所有样本的困难度，即为每个训练样本分配一个初始权重为：

$$D_1 = \{w_{11}, w_{12}, ..., w_{1n}\}, w_{1i} = \frac{1}{N} \tag{2}$$

Step2：训练弱分类器

对包含权重分布的样本集的训练集 D_t 进行训练，得到弱分类器（weak learner）：$\lambda_t(x)$.

Step3：计算当前分类误差

弱学习器在当前训练集上的分类误差率为：

$$\varepsilon_t = p(\lambda_t(x_i) \neq y_i) = \sum_{i=1}^{n} w_{ti} I(\lambda_t(x_i) \neq y_i) \tag{3}$$

Step4：计算弱分类器权重

根据分类误差率计算当前弱分类器的权重系数为：

$$\alpha_t = \frac{1}{2} \ln \frac{1 - \varepsilon_t}{\varepsilon_t} \tag{4}$$

Step5：更新训练样本权重

迭代训练，调整训练集的权重分布：

$$w_{t+1,i} = \begin{cases} \frac{1}{2(1-\varepsilon)}, & \text{if 当前样本被正确分类} \\ \frac{1}{2\varepsilon}, & \text{if 当前样本被错误分类} \end{cases} \tag{5}$$

其中，ε 表示通过当前弱学习器产生的错误率。

Step6：得到最终的强分类学习器

$$f(x) = \sum_{i=1}^{T} \alpha_t \lambda_t(x) \Lambda(x) = sign(f(x)) \tag{6}$$

关于 AdaBoost，有以下性质要进行探讨：

1.AdaBoost 算法有一个奇异的现象，就是一般情况下永远不会发生过拟合，这也是它最大的优点，关于这一点的原因解释参见周志华《Boosting 学习理论的探索 —— 一个跨越 30 年的故事》一文。

2.Adaboost 算法不需要弱分类器的先验知识，最后得到的强分类器的分类精度依赖于所有弱分类器。无论是应用于人造数据还是真实数据，Adaboost 都能显著的提高学习精度。

3.Adaboost 算法不需要预先知道弱分类器的错误率上限，且最后得到的强分类器的分类精度依赖于所有弱分类器的分类精度，可以深挖分类器的能力。Adaboost 可以根据弱分类器的反馈，自适应地调整假定的错误率，执行的效率高。

4. 关于 AdaBoost 算法的缺点，在训练过程中，会使得难于分类样本的权值呈指数增长，训练将会过于偏向这类困难的样本，导致其易受噪声干扰。此外，其依赖于弱分类器，而弱分类器训练的时间成本很高。

5.3 GBDT

梯度提升决策树（Gradient Boosting Decision Tree，简称 GBDT），其基学习器顾名思义就是 CART 决策树。针对分类问题就是二叉分类树，针对回归问题就是二叉回归树。所以，我们要想了解 GBDT，就先从提升树说起。

5.3.1 提升树（Boosting tree）

提升树模型是统计学习中性能最好的方法之一，其基本形式写成：

$$f_M(x) = \sum_{m=1}^{M} T(x; \Theta_m) \tag{7}$$

上式中，M 指树的个数，T 指的是每一棵决策树，而 Θ_m 指的是决策树的参数。在分类任务中，可以使用指数损失作为二分类任务的损失函数，而对于多分类任务，则通常使

用 softmax 作为损失函数。我们可以将二分类任务提升树理解为 **AdaBoost 模型的一种特殊形式**。

为什么这么说呢？其一，此时的基学习器 $\lambda(x)$ 与 AdaBoost 中的弱学习器对应，限制为二叉分类树；其二，每个基分类器的权重均设置为 1。此时经集成后的最终模型可以写为：

$$f(x) = T(x, \theta_1) + T(x, \theta_2) + ... + T(x, \theta_m) \tag{8}$$

由此，只要我们使用的是指数损失函数，就可以用指数损失函数来调整样本数据的权值，从而让每个基学习器学到不同的内容。

接下来，我们需要探讨如何处理回归任务呢？和分类任务不同的是，这里需要用到回归树（regression tree），其形式如下：

$$T(x; \Theta) = \sum_{j=1}^{J} C_j I(x \in R_j), \text{即} f_M(x) = \sum_{m=1}^{M} T(x; \Theta_m) \tag{9}$$

我们通常采用的是**前向分步算法**（forward stagewise algorithm），该算法是采用贪心的策略，逐棵逐棵树进行优化。那么此时就有一个良好的性质：首先确定初始提升树 $f_0(x) = 0$，然后当我们优化到第 m 棵树时，前 m-1 棵树就已经成为了**已知的值**，所以我们到第 m 轮次优化时只需考虑的是第 m 棵树中的变量。这一性质非常重要且优美，后文简称优美性质。

在回归树中使用的损失函数是**平方误差损失**：

$$\begin{aligned} L(y, f(x)) &= (y - f(x))^2 \\ &= [y - f_{m-1}(x) - T(x, \theta_m)]^2 \\ &= [r - T(x, \theta_m)]^2 \end{aligned} \tag{10}$$

其中，$r = y - f_{m-1}(x)$ 称为在第 m-1 步时的**残差**（即第 m-1 步时的真实值减去预测值）。残差也正是回归树的拟合对象，结合前向分布算法有：

$$\begin{aligned} \hat{\theta}_m &= \arg\min_{\theta_m} \sum_{i=1}^{N} L\left(y^{(i)}, f_{m-1}(x^{(i)}) + T(x^{(i)}, \theta_m)\right) \\ &= \arg\min_{\theta_m} \sum_{i=1}^{N} \left(r_m^{(i)} - T(x^{(i)}, \theta_m)\right)^2 \end{aligned} \tag{11}$$

由此，我们可以概括一下回归树模型的思路：

（1）使用每一个残差为每一个基学习器的训练样本，得到个体学习器；

（2）采用加性模型的方式进行组合集成。模型的目标是使得总体损失在逐步下降，因此如果回归树越多，模型结果就越准。当达到实际问题所需精度时，便可停止训练。

5.3.2 GBDT 模型的推导

针对一般决策问题，我们可以使用 GBDT，其基本思路依然是应用前向分布算法的加性模型集成学习。我们要解决的问题本质是：损失函数不同，对应的凸优化问题就不

一样；我们希望找到一个通用的方式，求解一般性凸优化问题，这就是 GBDT 要做的事。

所用基学习器：回归树

一般损失函数，具体未知，根据实际灵活定义，但其基本形式可以写为：$L(y, f(x))$

我们的核心目标是使得：

$$L(y^{(i)}, f_m(x^{(i)})) < L(y^{(i)}, f_{m-1}(x^{(i)})) \tag{12}$$

移项，得：

$$L(y^{(i)}, f_{m-1}(x^{(i)})) - L(y^{(i)}, f_m(x^{(i)})) > 0 \tag{13}$$

在一般损失函数 $L(y, f_m(x))$ 中，只有 $f_m(x)$ 是未知量，其满足式（13）：

$$f_m(x) = f_{m-1}(x) + T(x, \theta_m), f_{m-1}(x)\text{是已知量.} \tag{14}$$

解决这个凸优化问题，采用的方式是**泰勒一阶展开**，因此，有：

$$\begin{aligned} L(y, f_m(x)) &\approx L(y, f_{m-1}(x)) + \left.\frac{\partial L(y, f_m(x))}{\partial f_m(x)}\right|_{f_m(x)=f_{m-1}(x)} \cdot (f_m(x) - f_{m-1}(x)) \\ &\approx L(y, f_{m-1}(x)) + \left.\frac{\partial L(y, f_m(x))}{\partial f_m(x)}\right|_{f_m(x)=f_{m-1}(x)} \end{aligned} \tag{15}$$

移项，得：

$$L(y, f_{m-1}(x)) - L(y, f_m(x)) \approx -\left.\frac{\partial L(y, f_m(x))}{\partial f_m(x)}\right|_{f_m(x)=f_{m-1}(x)} \cdot T(x, \theta_m) \tag{16}$$

我们希望 $L(y, f_{m-1}(x))$–$L(y, f_m(x))$ 能一直大于零，即让总体损失一直降低，模型精度便一直提升。

当 $T(x, \theta_m) \approx -\frac{\partial L(y, f_m(x))}{\partial f_m(x)}$ 时，才满足我们的希望。此时有残差：

$$r(x, y) = -\left[\frac{\partial L(y, f_m(x))}{\partial f_m(x)}\right]_{f_m(x)=f_{m-1}(x)} \tag{17}$$

接下来，将样本点 (x_i, y_i) 代入 $r(x, y)$，可以得到所有 r_m 的值，进而得到第 m 轮训练的数据集。

我们将 GBDT 的相关步骤总结如下：

Step1：初始化提升树模型，计算 $f_0(x)$；

Step2：对每个样本计算负梯度拟合的残差；

Step3：将上一步得到的残差作为样本新的真实值，继续训练下一棵回归树；

Step4：计算最优拟合值；Step5：更新提升树模型；

Step6：得到最终的梯度提升树。

5.4 XGBoost

极端梯度提升树（eXtreme Gradient Boosting，简称 XGBoost），是由著名学者陈天奇博士提出，基于 GBDT 的变形和优化的模型。相较于 GBDT，XGBoost 在精度、速度以及泛化能力三部分均有显著的提升。因此，在诸多数据竞赛和各种顶级解决方案中都不乏 XGBoost 模型的身影。

在精度上，XGBoost 通过将损失函数展开到二阶导数，更能逼近真实损失；在速度上，XGBoost 采用了**加权分位数**和**稀疏感知**这两个技巧，通过缓存优化和模型分块并行提高算法速度；在泛化性能上，XGBoost 对损失函数加入正则化项、加性模型中设置学习率和近似采样等方式，防止过拟合。

在 XGBoost 中，使用到的基学习器是回归树（regression tree）。我们依旧使用前向分布算法，依然可以使用那个优美性质：

$$\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t(x_i) \tag{18}$$

5.4.1 目标损失函数的确定

接下来我们来写一写目标损失函数（以下简称目标函数），当每一个样本输入到我们的模型之后，都可以得到一个预测值。和真实值（ground truth）相比，我们就可以计算出它们的损失。假设有 N 个样本，我们将这 N 个样本总体的损失值累加起来，便是我们要优化的目标函数：

$$obj^{(t)} = \sum_{i=1}^{N} L(y_i, \hat{y}_i^{(t)}) + \sum_{i=1}^{t} \Omega(f_j) \tag{19}$$

式（19）中，$\sum_{i=1}^{t} \Omega(f_j)$ 指的是将 t 棵树的**复杂度**累加起来，这称为**正则项**（regularizer）. 那么我们的优化目标当然是使这个目标函数取最小值，即总体损失达到最小：

$$(w_1^*, w_2^*, ..., w_j^*) = \arg\min obj^{(t)} \tag{20}$$

正则项的定义是：

$$\Omega(f_t) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 \tag{21}$$

式（21）中的 T 指的是当前回归树中叶子结点的个数，χ, γ 为两个可以控制惩罚力度的超参数。值得注意的有两点，其一：如果叶子结点的数量越多，就说明当前回归树越枝繁叶茂，从而更加容易产生过拟合；其二：如果结点值较大，就代表当前这一棵回归树在所有回归树中的占比较大，也容易发生过拟合。因此，正则项的作用便是缓解过拟合，通过调节控制惩罚力度的超参数来实现。根据优美的性质，正则项可以写成：

$$\sum_{j=1}^{t} \Omega(f_j) = \sum_{j=1}^{t-1} \Omega(f_j) + \Omega(f_t) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 + const \tag{22}$$

由此观之，当前待优化的正则项只和当前待优化的这棵树的叶子结点的值和其叶子结点的个数相关。

因为树模型往往是**阶跃而非连续的**，通常使用梯度下降法来优化不太合适，这是因为不连续的函数无法求导，那么它的梯度也是没有意义的。我们假设误差损失为平方误差损失，然后得到一个关于 w 的式子：

$$L(y_i, \hat{y}_i) = (y_i, \hat{y}_i)^2 \tag{23}$$

按样本的顺序进行遍历，将所有的损失累加起来。然后应用**目标拆解**的思想，把总体目标拆解成了以叶子结点为单位的一个个小目标逐一击破，且每个小目标中都只有一个变量。结合之前的推导，可将目标函数作一个初步的改写：

$$obj^{(t)} = \gamma T + \sum_{j=1}^{T} \left[\sum_{i \in I_j} L(y_i, \hat{y}_i^{(t-1)} + w_j) \right] + \frac{1}{2} \lambda w_j^2 \tag{24}$$

此时，我们面临的问题是，如何将式（24）中的 $L(y_i, \hat{y}_i^{(t-1)} + w_j)$ 化简成一个只含有 w 的式子呢？这时，就需要用到**泰勒二阶展开**来化简式子。

对于 $L(y_i, \hat{y}_i^{(t-1)} + w_j)$ 而言，若将 $\hat{y}_i^{(t-1)} + w_j$ 看作 x，$\hat{y}_i^{(t-1)}$ 看作 x_0，则有 w_j 可以看作是 $x - x_0$. 那么我们可将其化简为：

$$L(y_i, \hat{y}_i^{(t-1)} + w_j) \approx L(y_i, \hat{y}_i^{(t-1)}) + L'(y_i, \hat{y}_i^{(t-1)}) w_j + \frac{1}{2} L''(y_i, \hat{y}_i^{(t-1)}) w_j^2 \tag{25}$$

式（25）中，$L(y_i, \hat{y}_i^{(t-1)}), L'(y_i, \hat{y}_i^{(t-1)}), L''(y_i, \hat{y}_i^{(t-1)})$ 三个部分都为常量. 在做优化时，第一项对我们没有什么帮助，可以直接剔除。这里举个很简单的例子：$y = x^2, y = x^2 + 1$ 的极值点是一样的。为了方便书写，我们定义 g_i, h_i 了两种表示方法，分别表示目标泰勒展式的一阶导常量和二阶导常量，由此，可将上式变形：

$$L(y_i, \hat{y}_i^{(t-1)} + w_j) \approx g_i w_j + \frac{1}{2} h_i w_j^2 \tag{26}$$

代入目标函数的初步改写式，有：

$$obj^{(t)} = \gamma T + \sum_{j=1}^{T} [w_j \sum_{i \in I_j} g_i + \frac{1}{2} w_j^2 (\lambda + \sum_{i \in I_j} h_i)] \tag{27}$$

为了方便书写，我们定义 $\sum_{i \in I_j} g_i$为$G_i, \sum_{i \in I_j} h_i$为$H_i$. 上式可以改写为最终版本的目标函数：

$$obj^{(t)} = \gamma T + \sum_{j=1}^{T} [w_j G_i + \frac{1}{2} w_j^2 (\lambda + H_i)]. \tag{28}$$

5.4.2 确定树的结构（预排序算法）

当每层回归树的划分条件不一样时，我们得到的叶子结点所包含的样本就会不一样。因此，在不同的划分条件下得到的回归树经过 5.4.1 节的计算方式都能找到一个

obj^*，而我们的目的就是在众多的 obj^* 中找到那个最小值，这个最小值所对应的回归树就是我们需要确定的树的结构，并将其用作基学习器。

XGBoost 模型采用的是**精确贪心算法**（Precise greedy algorithm），其核心思想可以理解为**“分步 + 贪心”**的模式。如果我们将一棵整体的树看作是由一个个结点分裂而来的，也就是说，每一步都对应着一个结点进行分裂。基于此，我们每一次都只用关注一个结点是怎样进行分裂的，一步一步来，这便是“分步”。这样一来就可以大大降低计算量，因为我们**无需考虑组合起来的情况，每一次都只需关注当前正在考虑的这个结点的所有可能情况**。

假设决策树模型在某个结点进行了特征分裂，分裂前的损失函数写为：

$$obj^*_{before} = -\frac{1}{2}[\frac{(G_{left}+G_{right})^2}{H_{left}+H_{right}+\lambda}]+\gamma \tag{29}$$

分裂后的损失函数写为：

$$obj^*_{after} = -\frac{1}{2}[\frac{G^2_{left}}{H_{left}+\lambda}+\frac{G^2_{right}}{H_{right}+\lambda}]+2\gamma \tag{30}$$

所以，可以计算出分裂后的信息增益为：

$$Gain = obj^*_{before} - obj^*_{after} = \frac{1}{2}[\frac{G^2_{left}}{H_{left}+\lambda}+\frac{G^2_{right}}{H_{right}+\lambda}-\frac{(G_{left}+G_{right})^2}{H_{left}+H_{right}+\lambda}]-\gamma \tag{31}$$

对每一棵树都采用上式进行信息增益的计算，直到找出满足条件的回归树，这便是“贪心”的含义。总结一下，用分而治之的方法处理决策树。对于每棵树的生长，按照结点分步划分，后按照“贪心”的过程治之。那么什么时候结点不再进行分裂呢？满足下列两种情形之一的，结点不再进行分裂：

1. $\max gain \leq 0$
2. 叶子结点包含的样本个数 $i \leq 1$

这两点的解释是，当增益太小时，说明再分裂给我们带来的提升几乎为零，于是选择停止分裂。当层级越深，这棵树往往会学得更精细，容易学到不该学的东西，增加了过拟合的风险。所以我们可以**限制层级数、叶子结点的个数**等措施来防止过拟合。

关于 XGBoost，还有很多值得思考的地方，在此就不多做赘述了。感兴趣的读者可以去看看陈天奇博士的论文，或者参见杨跞仁《XGboost 原理理解》。最后，我们用一幅图总结下 XGBoost 如图 7：

5.5 LightGBM

5.5.1 从 XGBoost 说开去

XGBoost 也有很多值得优化的地方，其最显著的劣势便是占用空间内存太大，不适合处理数据量和特征量都十分多的情况。在寻找最优分裂点的算法复杂度可以估计为：

$$\Omega = \text{特征数} \times \text{特征分裂点数量} \times \text{样本量}$$

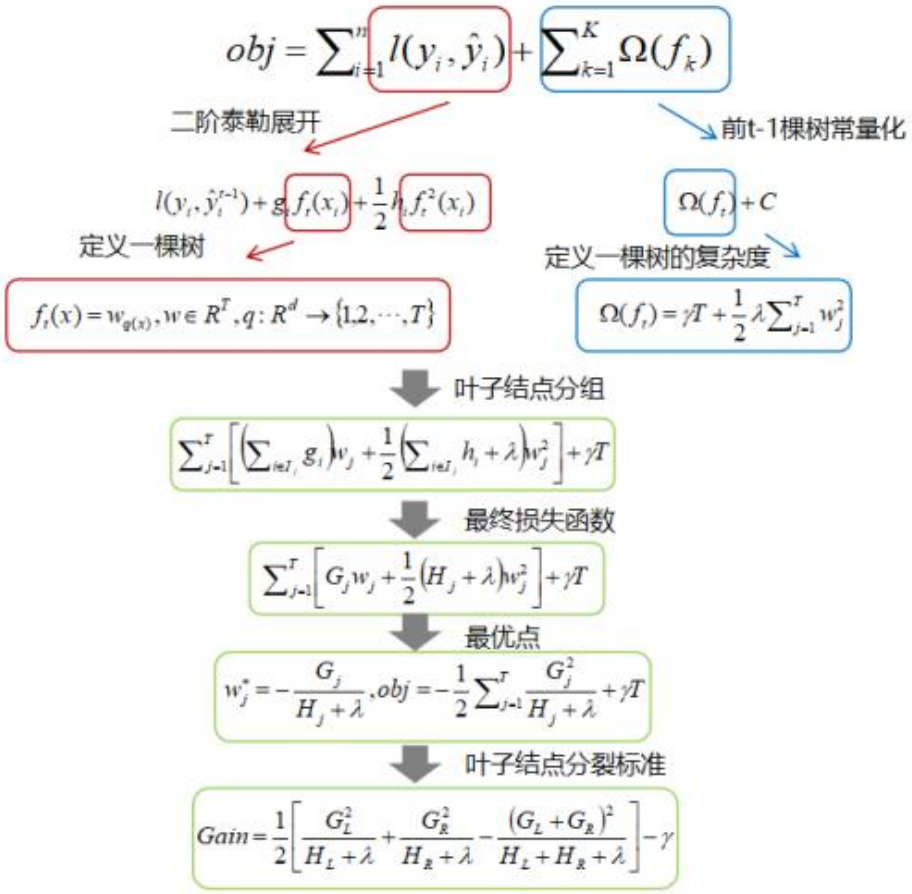

图 7. XGBoost 算法示意图

由此观之，要想对 XGBoost 进行优化，那么肯定是要从上式中的这三方面入手。而 LightGBM（**轻量梯度提升机**，Light Gradient Boosting machine）就可以看成是 XGBoost 的升级加强版本，2017 年经微软推出后，便成为各种数据竞赛中刷分拿奖的神兵利器。和 XGBoost 相比，其在大规模数据集上跑起来更加轻盈。对比以下几点：

（1）模型精度：XGBoost 和 LightGBM 相当；

（2）训练速度：LightGBM 远快于 XGBoost；

（3）内存消耗：LightGBM 远小于 XGBoost；

（4）缺失值特征：XGBoost 和 LightGBM 都可以自动处理特征缺失值；

（5）分类特征：XGBoost 不支持类别特征，需要 OneHot 编码预处理，LightGBM 直接支持类别特征。

于是，我们可以用以下公式来浅显的揭示二者之间的关系：

$$LightGBM = XGBoost + Histogram + GOSS + EFB \tag{32}$$

5.5.2 直方图算法（Histogarm）

为了减少特征分裂点的数量，更加高效的寻找最优特征分裂点，LightGBM 采用直方图算法寻找最优特征分裂点。直方图算法的基本思路解释是：通过将连续的浮点特征离散成个整数，并构造宽度为的直方图，然后遍历训练数据，统计每个离散值在直方图中的累计统计量。在进行特征选择时，只需要根据直方图的离散值，遍历寻找最优的分割点，全过程如图 8。

其优点有：

（1）占用内存更低，数据分隔的复杂度更低；

（2）从 8 位整型存储，内存消耗为原来的 $\frac{1}{8}$；

（3）时间开销由原来的 $O(data * feature)$ 显著降低为 $O(k * feature)$。

需注意的有：

（1）使用 bin 替代原始数据相当于增加了正则化；

（2）使用 bin 意味着更多细节特征被丢弃，相似的数据可能被划分到了相同的桶里；

（3）bin 数量选择决定了正则化的程度，bin 越少惩罚越严重，欠拟合风险越高；

（4）直方图除了保存划分阈值和当前 bin 内样本数以外，还保存了当前 bin 内所有样本的一阶梯度和（一阶梯度和的平方的均值等价于均方损失）；

（5）阈值的选取是按照直方图从小到大遍历，使用了上面的一阶梯度和，目的是得到划分后 loss 最大的特征及阈值。

此外，直方图算法的另一个好处在于**差加速**，一个叶子节点的直方图可由其父节点的直方图与其兄弟节点的直方图做差得到，这也可以加速特征节点分裂。通常构造直方图，需要遍历该叶子上的所有数据，但直方图做差仅需遍历直方图的 k 个桶。作差示意图见图 9.

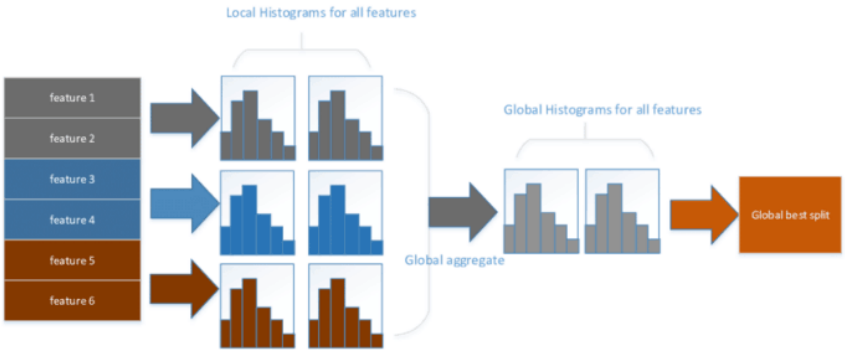

图 8. 直方图算法示意图

图 9. 作差图

5.5.3 单边梯度抽样算法（GOSS）

从减少样本的角度，GOSS 算法也算是 LightGBM 模型的核心算法。其本质很简单，就是将训练过程中大部分权重较小的样本剔除，仅对剩余样本数据计算信息增益，以达到最大效率的保留对计算信息增益有帮助的样本，提高模型训练速度的目的。

GOSS 的基本做法是先将需要进行分裂的特征按绝对值大小降序排序，取绝对值最大的 a% 前个数据，假设样本大小为 n，在剩下的 (1-a)% 个数据中随机选择 b% 个数据，将这个数据乘以一个常数 $\frac{1-a}{b}$（这个常数大于 1，可以理解为将 b% 中的数据放大），也就是使得算法**更加关注训练不够充分的样本**。最后使用这 (a+b)% 的数据来计算信息增益。

一言以蔽之，此法就是丢弃梯度较小的样本并保证在不损失太多精度的情况下，提升模型训练速度。

5.5.4 互斥特征捆绑算法（EFB）

从优化特征数量出发，EFB 算法通过将两个互斥 (两个特征不会同时为非零值) 的特征捆绑在一起，合为一个特征，在不丢失特征信息的前提下，减少特征数量，从而加速模型训练。在许多应用场景下，数据集中会有大量的稀疏特征，这些稀疏特征大部分样本都取值为 0，只有少数样本取值非 0。通常可以认为这些稀疏特征是互斥的，即它们几乎不会同时取非零值。利用这种特性，可以通过对某些特征的取值重新编码，将多个这样互斥的特征捆绑成为一个新的特征。有趣的是，对于类别特征，如果转换成 onehot 编码，则这些 onehot 编码后的多个特征相互之间是互斥的，从而可以被捆绑成为一个特征。因此，对于指定为类别特征的特征，LightGBM 可以直接将每个类别取值和一个 bin 关联，从而自动地处理它们。而无需预处理成 onehot 编码多此一举。

简言之，该算法的步骤可以概括为：

Step1：将特征按照非零值的个数进行排序；

Step2：计算不同特征之间的冲突比率；

Step3：遍历每个特征并尝试合并特征，使冲突比率最小化。

此法的直方图时间复杂度从 $O(data * feature)$ 降到 $O(data * bundle)$，由于 bundle < feature，我们能够极大地加速 GBDT 的训练过程而且未损失精度。此外，此法能够将许多互斥的特征变为低维稠密的特征，就能够有效的避免不必要 0 值特征的计算。

5.5.5 生长策略

1.XGBoost 采用 level-wise 的生长策略，好处是可以多线程优化，也方便控制模型复杂度，且不易过拟合。但缺点是不加区分的对待同一层所有叶子节点，大部分的节点分裂和增益计算不是必须的，带来了多余的计算开销。

2.LightGBM 则采用 lead-wise 的生长策略，每次从当前所有叶子中找到分裂增益最大（一般也是数据量最大）的一个叶子，然后分裂、如此循环.

优点：同 level-wise 相比，在分裂次数相同的情况下，leaf-wise 可以降低更多的误差，得到更好的精度。

缺点：可能会长出比较深的决策树，产生过拟合。可采用措施：在 leaf-wise 上增加了一个最大深度的限制，在保证高效率的同时防止过拟合。

LightGBM 经常活跃在各大机器学习、数据挖掘以及大数据挑战赛中，有人评价说“LightGBM 算法除了比 XGBoost 更准确和更省时外，还优于现有的其他 Boosting 算法”。这种观点虽然绝对，不符合没有免费的午餐（NFL 定理），却也不失为实战的一般经验之谈。总之，一个好的参赛方案 99% 会用到集成学习的思想，这是毋庸置疑的。

5.6 CatBoost

笔者个人认为 CatBoost 的原理要比前面讲述的 Boosting 算法难理解很多，且自己也没有学到位。故本节内容可能非常粗糙，仅供大家参考。

Categorical + Boosting（简称 CatBoost），由俄罗斯搜索引擎 Yandex 开发，因其能够高效处理数据中的类别特征而得名。与其他 Boosting 方法相比，CatBoost 主要创新了两点：类别特征处理和排序提升（Ordered Boosting）。

5.6.1 处理类别特征的方法

对于类别特征，如果类别数目不多，可以使用 one-hot 编码。否则，很容易造成维度爆炸。CatBoost 不提倡使用 one-hot 编码，它设计了一种基于预测目标统计值的方法可以将类别特征转化为数值特征，但又不是简单的统一使用标签均值代替分类特征，而是做了改进：添加**先验分布项**，用以减少噪声和低频类别型数据对于数据分布的影响。

假设 $\sigma = (\sigma_1, \sigma_2, ..., \sigma_n)$ 是随机排列序列，有：

$$X_{\sigma_p,k} = \frac{\sum_{j=1}^{p=1}[X_{\sigma_j,k} = X_{\sigma_p,k}] \cdot Y_{\sigma_j} + \alpha P}{\sum_{j=1}^{p=1}[X_{\sigma_j,k} = X_{\sigma_p,k}] + \alpha} \tag{33}$$

解释下式（33）：[ ] 代表指示函数，P 就代表先验项。对应回归任务，计算标签的平均值作为先验值；对于二分任务，将正类的出现概率作为先验值。α 就代表优先级的权重系数，这个是为防止低频次的特征带来的影响所用的平滑操作，如果不使用这个操作的话，当对于某一个特征只有一个样本的时候，其特征编码就为 1，会有过拟合的风险。

这种方法称为 Ordered Target Statistics 数值编码方法，可以有效解决预测漂移的问题，见 5.6.3 节。

5.6.2 特征组合

CatBoost 在构建新的分裂节点时，会采用贪心的策略考虑特征之间的组合。CatBoost 将当前树的所有组合、类别型特征与数据集中的所有类别型特征相结合，并将新的类别组合型特征动态地转换为数值型特征。

使用参数“max_ctr_complexity”控制特征组合的最大个数。

5.6.3 预测偏移

预测偏移（Prediction shift）现象，简而言之就是训练样本 X_k 的分布 $F(X_k)|X_k$ 与测试样本 X 的分布 $F(X)|X$ 之间所产生的偏差。这种现象存在于所有的梯度提升（Gradient Boosting）算法中，由**目标泄露**（target leakage）和**梯度偏差**引起。目标泄露是数据泄露的一种，指在训练数据中包含目标信息，但在预测时没有可用的类似数据。这会导致模型看起来很精确，但用模型做出来的决策却很不准确。

假设前一轮训练得到的强学习器为 $F^{t-1}(x)$，当前的损失函数为 $L(y, F^{t-1}(x))$，则本轮迭代要拟合的弱学习器为 h^t：

$$h^t = \arg\min_{h \in H} L(y, F^{t-1}(x) + h(x)) \tag{34}$$

进一步梯度表达为：

$$h^t = \arg\min_{h \in H} E(-g^t(x, y) - h(x))^2 \tag{35}$$

但因上式的数学期望未知，因此往往使用同样数据集来近似：

$$h^t = \arg\min_{h \in H} \frac{1}{\mathrm{n}} \sum_{n}^{k=1} (-g^t(X_k, y_k) - h(X_h))^2 \tag{36}$$

由这个近似表达式就能看出预测偏移产生的原因有以下三点：

1. 梯度 $g^t(X_k, y_k)|X_k$ 的条件分布与测试样本 $g^t(X, y)|X$ 的分布不同，产生偏移。即梯度的条件分布和测试数据的分布存在偏移；

2. 式中定义的 h^t 基础学习器与进一步梯度表达式存在偏移。即 h^t 的数据近似估计与梯度表达式之间存在偏移；

3. 预测偏移会影响到训练模型的泛化性能。

为了解决预测偏移的问题，可以假设数据集无穷大，在每一步提升中，独立采样一个新的数据集，将现有模型应用至新训练数据上会得到无偏残差，遗憾的是，数据集一定是有限的。因此，采用基于 Ordered TS 的**排序提升算法**（Ordered Boosting）。此法笔者学的不太通透，所以不能在此花费笔墨了。大家可以参见石晓文《数学推导 + 纯 Python 实现机器学习算法 19：CatBoost》。

5.7 Random Forest

随机森林（Random Forest，简称 RF 算法），是并行化集成学习算法 Bagging 的著名变体，其基本思想是**以弱搏强**。Bagging 的核心概念在于**自主采样**（bootstrap samping）。随机森林顾名思义，随机指的是随机的从数据集中采集一部分进行模型训练，即看问题的角度不一样，以保证每颗决策树的输出相似但不一样；森林则指的是有很多个决策树作为基学习器组成了整个模型。

随机森林的一般步骤是：

Step1：预设模型的超参数，几个树？分几层？

Step2：随机采样，训练每个决策树：

$$DATASET[N \times D] \Rightarrow datasubset[n \times d] \tag{37}$$

其中 N,n 表示样本数量，满足 $n << N$；D,d 表示特征数量，满足 $d << D$。

Step3：输入待测样本到每个树中，再将每个树的结果整合，从而组成森林：回归问题取均值，分类问题取众数（投票法）。

6　支持向量机

早些年神经网络还不被看好的时候，支持向量机一度被认为是最科学最完美的机器学习模型，这主要是因为其具有超高的数学可解释性，且效果瞩目。我们现在就来进入支持向量机的世界吧！

6.1　支持向量机的基本型

从分类学习任务说起，存在样本集 $D = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}, y_i \in \{-1, 1\}$，要在样本空间中找到一个划分超平面，将不同类型的样本分开。

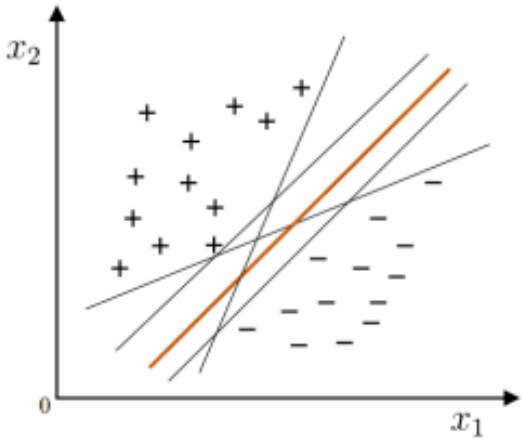

图 10. 一些划分超平面

如图 10 所示，存在很多种划分超平面，但是究竟哪一种更好呢？显然，选择正中间的更加合适，其结果更加鲁棒，泛化能力更强。

如果往更高的维度推而广之，那么这类问题便可以概括为需要在一个样本数 * 维度数 =m*n 的样本数据集（1）中，

$$X = \begin{bmatrix} x_{11} & x_{12} & x_{13} & x_{14} & ... & x_{1m} \\ x_{21} & x_{22} & x_{23} & x_{24} & ... & x_{2m} \\ x_{31} & x_{32} & x_{33} & x_{34} & ... & x_{3m} \\ \vdots & \vdots & \vdots & \vdots & ... & \vdots \\ x_{n1} & x_{n2} & x_{n3} & x_{n4} & ... & x_{nm} \end{bmatrix} \tag{1}$$

寻找一个划分超平面，其方程可以写成如下形式：

$$w_1 x_1 + w_2 x_2 + \cdots + w_m x_m + b = 0 \tag{2}$$

式（2）中，w 可以简单理解为 x 所对应的权重，其是该划分超平面的**法向量**，可以决定该超平面的方向；而 b 则是**偏置位移项**，可以决定该超平面与原点的距离。因此这个划分超平面将由 w 和 b 这两个参数决定。

如图 11 所示，正中间红色的超平面即为**决策边界** (decision boundary) 超平面，也就是式（2）。如果将决策边界分别向上、下移动个单位，就会形成分类决策的上下边界：

$$\begin{cases} w_1x_1 + w_2x_2 + \cdots + w_mx_m + b = c \\ w_1x_1 + w_2x_2 + \cdots + w_mx_m + b = -c \end{cases} \tag{3}$$

将式（3）中的每一项除以 c，则可将上下边界超平面方程写为：

$$\begin{cases} w_1'x_1 + w_2'x_2 + \cdots + w_m'x_m + b' = 1 \\ w_1'x_1 + w_2'x_2 + \cdots + w_m'x_m + b' = -1 \end{cases} \text{,其中} w_d' = \frac{w_d}{c}, b' = \frac{b}{c} \tag{4}$$

由于上下边界超平面一定会经过一些样本点，而这些样本点距离决策边界超平面最近，因此它们决定了间隔距离，于是我们将这些样本点称作**支持向量** (support vector)，两个异类的支持向量之间的距离称作**间隔** (margin)，其可以描述为：

$$\gamma = \frac{2}{\|w\|} \tag{5}$$

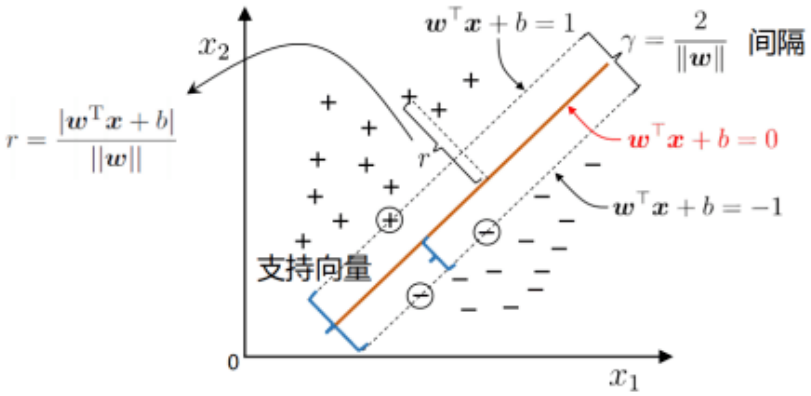

图 11. 支持向量机的基本结构

这个间隔我们可以理解为缓冲区，如果初始数据发生了如下变化：当存在异常值进入到图 11 所示的间隔中时，我们是否应该为异常值牺牲间隔距离呢？我们可以将间隔理解为收入，损失理解为成本，那么问题可以转化为：同时约束收入和成本使得利润最大化，此时在最优解下得到的间隔便称作**软间隔** (soft margin). 它有一定的容错率，其目的是在间隔距离与错误间找到一个平衡。关于软间隔会在后文进行描述。与之对应的有，如果要求所有样本都必须划分正确，这时得到的间隔便是**硬间隔** (hard margin).

那么我们如何找到具有最大间隔 (maximum margin) 的超平面呢？即寻找参数 w 与 b，使得 γ 最大：

$$\arg\max_{w,b} \quad \frac{2}{||w||} \text{s.t.} \begin{cases} y_i(w^T x_i + b) \geq 1, \\ i = 1, 2, \ldots, m \end{cases} \tag{6}$$

最大化 $\|w\|^{-1}$，等价于最小化 $\|w\|^2$，所以式（6）等价于：

$$\begin{aligned} &\arg\min_{w,b} \frac{1}{2}\|w\|^2 \\ &s.t. y_i(w^T x_i + b) \geq 1, i = 1, 2, \ldots, m \end{aligned} \tag{7}$$

这便是**支持向量机** (support vector machine，简记为 SVM) 的基本型，解决上述问题便成为了该模型的关键。综上所述，支持向量机的模型算法的本质是什么呢？一言以蔽之，是在 m 维度上找到一个（m-1）维超平面将两类样本区分开来，**SVM 是建立于结构风险最小原理的基础之上，以间隔最大化为学习策略，其算法本质是求解凸二次型规划的最优化算法**。

6.2 凸二次型优化理论概述

结合在 bilibili 上浙江大学胡浩基教授关于支持向量机的讲解以及南京大学周志华教授编写的西瓜书，下面我将本部分叙述如下：

我们首先定义一个关于优化理论的普适定义，这个很重要，因为在后续讲解支持向量机优化问题时需要一一对应：

原问题（prime problem）：

$$\min f(w) \tag{8}$$

限制条件：

$$\begin{cases} g_i(w) \leq 0, & i = 1, 2, \ldots, k \\ h_i(w) = 0, & i = 1, 2, \ldots, \upsilon \end{cases} \tag{9}$$

对式（7）求解的核心思路是**拉格朗日乘子法**。这里直接给出拉格朗日函数表达式：

$$L(w, \alpha, \beta) = f(w) + \sum_{i=1}^{k} \alpha_i g_i(w) + \sum_{i=1}^{\upsilon} \beta_i h_i(w) \text{.....(分量形式)} \tag{10}$$

$$= f(w) + \alpha^T g(w) + \beta^T h(w) \text{.....(向量形式)} \tag{11}$$

对偶问题（dual problem）可定义如下：

$$\begin{aligned} &\max \theta(\alpha^*, \beta^*) = \inf_{\text{所有} w} \{L(w, \alpha^*, \beta^*)\} \\ &s.t. \alpha_i^* \geq 0, i = 1, 2, ..., k \end{aligned} \tag{12}$$

式（12）有以下解释：inf 指的是求下确界（infimum），在本式中可以理解为限制 α, β，遍历所有的 w 求出 $L_{\min}$.

接着来讨论下**原问题与对偶问题的关系**：

定理：如果 w^* 是原问题的解，α^*, β^* 是对偶问题的解，则有：

$$f(w^*) \geq \theta(\alpha^*, \beta^*) \tag{13}$$

式（13）证明如下：

$$\theta(\alpha^*,\beta^*)=\inf_{所有w}\{L(w,\alpha^*,\beta^*)\}\leq L(w,\alpha^*,\beta^*) \tag{14}$$

式（14）很好理解，某一个特解 w^* 一定不小于函数 L 的下确界（也就是的最小值）。将式（10）代入式（14）可得：

$$续(14)=f(w^*)+\sum_{i=1}^{\kappa}\alpha_i^*g_i(w^*)+\sum_{i=1}^{\upsilon}\beta_i^*h_i(w^*)\quad\leq f(w^*) \tag{15}$$

式(14)的解释是:根据限制条件式(9),不难知道 $\alpha_i^*\geq 0,\quad g_i(w^*)\leq 0$,因此 $\alpha_i^*g_i(w^*)\leq 0$; 同理可以分析出 $\beta_i^*h_i(w^*)=0$. 因此，定理证毕。

定义：

$$G=f(w^*)-\theta(\alpha^*,\beta^*)\geq 0 \tag{16}$$

其中，G 称为原问题与对偶问题的**间距**（dualuty Gap），对于某些特定的优化问题，可以证明出 G=0，而强对偶优化问题正是这种情况：

强对偶定理：若 $f(w)$ 为凸函数，线性函数 $g(w)=Aw+b,h(w)=Cw+d$，则此优化问题的 G=0. 即若 w^* 是原问题的解，α^*,β^* 是对偶问题的解，则：

$$f(w^*)=\theta(\alpha^*,\beta^*) \tag{17}$$

这意味着两点，其一是：

$$\theta(\alpha^*,\beta^*)=\inf_{所有w}\{L(w,\alpha^*,\beta^*)\}=L(w,\alpha^*,\beta^*) \tag{18}$$

其二是：因为 $\alpha_i^*\geq 0,\quad g_i(w^*)\leq 0$，但是又要满足 $\alpha_i^*g_i(w^*)=0$，所以可以得到：

$$\begin{aligned}&\forall i=1,2,...,k,有:\\&\alpha_i^*=0 or g_i^*(w^*)=0\end{aligned} \tag{19}$$

式（19）中，在数学中 or 的含义是两者中至少有一个成立（包含同时成立的情况），该式就是著名的**库恩-塔克条件**（KKT 条件）。

回到支持向量机的基本型，我们将求解流程概述一遍：

Step1：引入拉格朗日乘子，得到拉格朗日表达式（8）；

Step2：对参数 w 与 b 求偏导，并令偏导为 0：

$$\begin{cases}w=\sum_{i=1}^{m}\alpha_iy_ix_i\\\\0=\sum_{i=1}^{m}\alpha_iy_i\end{cases} \tag{20}$$

Step3：回代，得到对偶问题：

$$\max_{\alpha}\sum_{i=1}^{m}\alpha_i-\frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_jy_iy_jx_i^Tx_j,s.t.\sum_{i=1}^{m}\alpha_iy_i=0,\alpha_i\geq 0,i=1,2,...,m. \tag{21}$$

Step4：解出 α 后可以确定出最终模型是：

$$f(x) = w^T x + b = \sum_{i=1}^{m} \alpha_i y_i x_i^T x + b \tag{22}$$

上述过程需要满足 KKT 条件 (Karush-Kuhn-Tucker)：

$$\begin{cases} \alpha_i \geq 0 \\ 1 - y_i f(x_i) \leq 0 \\ \alpha_i (1 - y_i f(x_i)) = 0 \end{cases} \Rightarrow \text{必有} \alpha_i = 0 \ y_i f(x_i) = 1. \tag{23}$$

这反映了**解的稀疏性**：即最终模型仅与支持向量有关，支持向量机因此而得名。

对于式（21）这样的二次规划问题，我们该如何求解呢？利用 SMO 算法求解，参见勿在浮沙筑高台《SMO 算法剖析》。

6.3 核支持向量机

6.3.1 核函数的原理

如果在该（较低）维度不存在一个能正确划分两类样本的超平面（即**低维不可分**问题），此时的处理方式是将样本从原始空间映射到一个更高维的特征空间，使样本在这个特征空间内线性可分。如果原始空间是有限维（属性数有限），那么一定存在一个高维特征空间使样本线性可分。（这一点已经得到严格证明）

在特征空间中，若样本 x 映射到**甚高维空间**（可理解为甚至可以高到无限维度的空间）后的向量为 $\phi(x)$，划分超平面为 $f(x) = w^T \phi(x) + b$，则原始问题是：

$$\min_{w,b} \frac{1}{2} \|w\|^2, s.t. y_i (w^T \phi(x) + b) \geq 1, i = 1, 2, ..., m \tag{24}$$

对偶问题是：

$$\begin{aligned} &\max_{\alpha} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j \phi(x_i)^T \phi(x_j) \\ &s.t. \sum_{i=1}^{m} \alpha_i y_i = 0, \alpha_i \geq 0, i = 1, 2, ..., m \end{aligned} \tag{25}$$

其预测函数为：

$$f(x) = w^T \phi(x) + b = \sum_{i=1}^{m} \alpha_i y_i \phi(x_i)^T \phi(x_j) + b \tag{26}$$

结合式（25）和（26）可以发现，$\phi(x_i)^T \phi(x_j)$ 部分只以内积形式出现，而计算甚高维空间的内积的计算量大到无法想象。此时，有没有一种方法可以绕过显式考虑特征映射和计算内积呢？这类问题的基本思路是设计一个**核函数** (kernal function)：

$$\kappa(x_i, x_j) = \phi(x_i)^T \phi(x_j) \tag{27}$$

根据 **Mercer 定理**：若一个对称函数所对应的核矩阵半正定，则它就能作为核函数来使用。这里的核矩阵可以理解为经过映射后在甚高维空间中的距离矩阵，那么问题就转化为了在该低维空间中使用核函数来求解。直白的说，就是将先把数据映射到高维空间中，再在高维空间中作内积的问题，转化为了先在原始低维空间中作内积，再用核函数将结果映射到高维空间中。任何一个核函数，都隐式地定义了一个 RKHS (Reproducing Kernel Hilbert Space, 再生核希尔伯特空间)。

下面给出几种常用的核函数如图 12，截取自西瓜书。

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

图 12. 一些常用的核函数

下面可将核函数的特点总结如下：

1. 无需知道非线性变换函数映射 ϕ 的形式和参数；
2. 避免了维数灾难，大大减小了计算量；
3. 核函数的选择成为了决定核支持向量机模型的**最大变数**。

6.3.2 软间隔支持向量机

现实中很难确定合适的核函数，使训练样本在特征空间中线性可分。即便貌似线性可分，也很难断定是否是因过拟合造成的，于是引入软间隔 (soft margin) 如图 13，允许在一些样本上不满足约束。

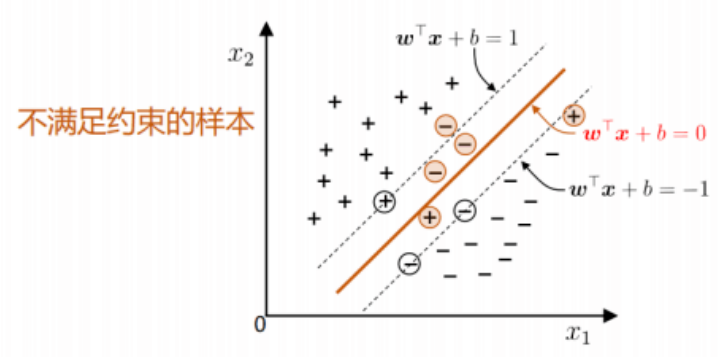

图 13. 软间隔 SVM 示意图

此时优化的基本思路是：最大化间隔的同时, 让不满足约束的样本尽可能少：

$$\min_{w,b} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m} \ell_{0/1}(y_i(w^T x_i + b) - 1) \tag{28}$$

式（28）中，L 是取值仅有 0 和 1 的损失函数 (loss function)：

$$\ell_{0/1}(z) = \begin{cases} 1, z < 0 \\ 0, else \end{cases} \tag{29}$$

这里有一个障碍，损失函数非凸、非连续且不易优化，所以采用**替代损失函数**来处理问题。根据替代损失函数得到的解仍是原问题的解，具有替代损失的**一致性** (consistency)，如图 14 所示：

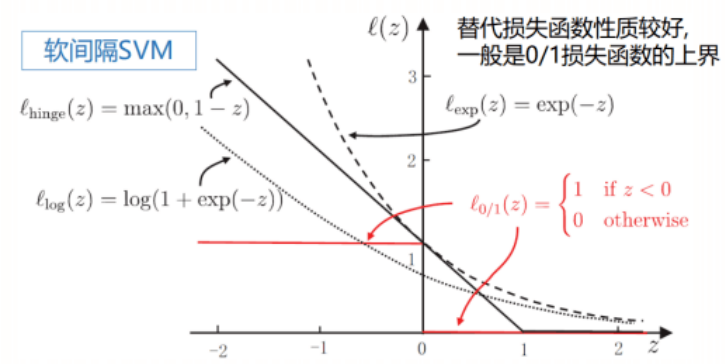

图 14. 三种替代损失函数（hinge，指数，对数）

此模型也被称作**软间隔支持向量机**，该模型处理的原始问题是：

$$\min_{w,b} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m} \max(0, 1 - y_i(w^T x_i + b)) \tag{30}$$

引入松弛变量 (可以解释为函数间隔，ξ_i 对应数据点 x_i 允许偏离的量)(slack variables)：

$$\begin{aligned} &\min_{w,b,\xi_i} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m} \xi_i \\ &s.t. y_i(w^T x_i + b) \geq 1 - \xi_i, \xi_i \geq 0, i = 1, 2, ..., m \end{aligned} \tag{31}$$

再求解其对偶问题如下：

$$\max_{\alpha} \sum_{i=1}^{m} \alpha_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j x_i^T x_j s.t. \sum_{i=1}^{m} \alpha_i y_i = 0, 0 \leq \alpha_i \leq C, i = 1, 2, ..., m \tag{32}$$

根据 KKT 条件可知，最终模型仅与支持向量有关，也即采用 hinge 损失函数后仍保持了解的稀疏性。

6.3.3 核支持向量机的推导

本小节内容非常精妙，可以打通核支持向量机的任督二脉！

针对线性不可分的一般形式的支持向量机，需要引入松弛变量，并映射到甚高维空间中，其方程如下：

原问题：

$$\min \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N}\xi_i \tag{33}$$

限制条件：

$$\begin{cases} y_i[w^T\varphi(x_i)+b] \geq 1-\xi_i \\ \xi_i \geq 0 \end{cases} \tag{34}$$

式（33）是一个凸函数，这里有必要将凸函数的定义回顾一下：

凸函数的几何定义：在一个凸函数上任意取两点，那么这两点的连线永远在函数曲线（面）的上方。换句话说，任取两点的中点处的函数值小于这两点连线段的中点纵坐标是恒成立的。将上述定义代数化，则可以得到：

凸函数的代数定义：

$$\begin{aligned} &\forall w_1, w_2, \lambda \in [0,1], \text{有}: \\ &f(\lambda w_1 + (1-\lambda)w_2) \leq \lambda f(w_1) + (1-\lambda)f(w_2) \text{恒成立} \end{aligned} \tag{35}$$

即使空间扩充到甚高维，该代数定义亦成立。

接下来，为使方程（33）、（34）与凸优化理论小节（参见 6.2 节式（8）、（9））的形式一一对应起来，我们改写如下：

原问题

$$\min \frac{1}{2}\|w\|^2 - C\sum_{i=1}^{N}\xi_i \tag{36}$$

限制条件：

$$\begin{cases} 1+\xi_i - y_i w^T\varphi(x_i) - y_i b \leq 0 \\ \xi_i \leq 0 \end{cases} \tag{37}$$

对偶问题：

$$\max_{\theta(\alpha,\beta)} \theta(\alpha,\beta) = \inf_{w,\xi_i,b} \frac{1}{2}\|w\|^2 - C\sum_{i=1}^{N}\xi_i + \sum_{i=1}^{N}\beta_i\xi_i + \sum_{i=1}^{N}\alpha_i[1+\xi_i - y_i w^T\varphi(x_i) - y_i b] \tag{38}$$

$$\text{s.t.} \begin{cases} \alpha_i \geq 0 \\ \beta_i \geq 0 \\ i = 1, 2, ..., N \end{cases}$$

式（38）与式（10）、（12）的对应关系解释如下：

其一，式（12）中遍历所有的 w 对应式（38）中遍历所有的 w, ξ_i, b；

其二，式（12）中限制 α 对应式（38）中限制 α, β；

其三，式（10）中的 $\sum_{i=1}^{k} \alpha_i g_i(w)$ 拆分成了式（38）中的 $\sum_{i=1}^{N} \beta_i \xi_i$ 与 $\sum_{i=1}^{N} \alpha_i [1 + \xi_i - y_i w^T \varphi(x_i) - y_i b]$ 两项。

那么，式（38）的最优解如何求呢？很明显最优解就在拉格朗日函数对三个变量分别求偏导，并令所求偏导数为零处。即：

$$\begin{cases} \frac{\partial L}{\partial w} = 0 \\ \frac{\partial L}{\partial \xi_i} = 0 \\ \frac{\partial L}{\partial b} = 0 \end{cases} \tag{39}$$

这时有一个麻烦，涉及矩阵求导，于是需要引入以下两个定理：

定理 1：如果 $f(w) = \frac{1}{2}\|w\|^2$，则 $\frac{\partial f}{\partial w} = w$.

定理 2：如果 $f(w) = w^T X$，则 $\frac{\partial f}{\partial w} = X$.

由以上两个定理，可求出式（39）的表达式是：

$$\begin{cases} w - \sum_{i=1}^{N} \alpha_i y_i \varphi(x_i) = 0 .....(40-1) \\ -C + \beta_i + \alpha_i = 0 \Rightarrow \alpha_i + \beta_i = C .....(40-2) \\ -\sum_{i=1}^{N} \alpha_i y_i = 0 \Longrightarrow \sum_{i=1}^{N} \alpha_i y_i = 0 .....(40-3) \end{cases} \tag{40}$$

接下来就是根据式（40）来处理式（38）的过程：

Step1：由式（40-2）代入式（38），观察可知形如 $-C\sum \xi_i + \sum \beta_i \xi_i + \sum \alpha_i \xi_i$ 的结构可消去；

Step2：

$$\begin{aligned} \frac{1}{2}\|w\|^2 &= \frac{1}{2} w^T w \\ &= \frac{1}{2} (\sum_{i=1}^{N} \alpha_i y_i \varphi(x_i))^T (\sum_{j=1}^{N} \alpha_j y_j \varphi(x_j)) \\ &= \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \varphi(x_i)^T \varphi(x_j) \\ &= \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \kappa(x_i, x_j) \end{aligned} \tag{41}$$

式（41）的化简结果非常精妙，其与核函数对应了起来，将所有甚高维空间中的映射向量内积的计算全部消去，转化为了使用核函数求解。

Step3：式（41）中最后剩余的部分化简如下：

$$
\begin{aligned}
&-\sum_{i=1}^{N}\alpha_i y_i w^T\varphi(x_i) = -\sum_{i=1}^{N}\alpha_i y_i(\sum_{j=1}^{N}\alpha_j y_j\varphi(x_j))^T\varphi(x_i)\\
&=-\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j\varphi(x_j)^T\varphi(x_i)\\
&=-\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j\kappa(x_i,x_j)
\end{aligned}
\tag{42}
$$

联立式（40）、（41）、（42）可得式（38）的最终化简形式为：

$$
\theta(\alpha,\beta)=\sum_{i=1}^{N}\alpha_i-\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j\kappa(x_i,x_j) \tag{43}
$$

至此，我们可以将核支持向量机的**对偶问题**形式完整的呈现如下：

$$
\max\theta(\alpha,\beta)=\sum_{i=1}^{N}\alpha_i-\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j\kappa(x_i,x_j) \tag{44}
$$

限制条件 1：由 $\alpha_i\geq 0,\beta_i\geq 0,\alpha_i+\beta_i=C$ 可得：$0\leq\alpha_i\leq C$.

限制条件 2：$\sum_{i=1}^{N}\alpha_i y_i=0$.

综上所述，此对偶问题是一个已知 y 和核函数，未知所有的拉格朗日乘子的一个凸优化问题，通常的解决策略是 SMO 算法。

这里我们需要思考一个问题，如何将求未知的 α 这个问题转变为求解 w 与 b？支持向量机模型的本质不正是要求得法向量 w 与偏置位移项 b 吗？我们从支持向量机模型的测试流程入手，其测试样本记作 X，测试原理如下：

$$
\begin{cases}
if w^T\varphi(x)+b\geq 0\ y=+1\\
if w^T\varphi(x)+b<0\ y=-1
\end{cases}
\tag{45}
$$

我们会发现一个非常巧妙的点,即我们根本无需知道 w 的具体值,只需知道 $w^T\varphi(x)+b$ 的结果，问题依然可以解决。这是因为支持向量机模型的本质是在做分类任务，我们只需关注分类结果为正例还是负例。

$$
\begin{aligned}
w^T\varphi(x)&=\sum_{i=1}^{N}[\alpha_i y_i\varphi(x_i)]^T\varphi(x)\\
&=\sum_{i=1}^{N}\alpha_i y_i\varphi(x_i)^T\varphi(x)\\
&=\sum_{i=1}^{N}\alpha_i y_i\kappa(x_i,x)
\end{aligned}
\tag{46}
$$

接下来，如何求解 b 呢？这就需要用到大名鼎鼎的 KKT 条件了：对 $\forall i = 1, 2, ..., N$ 有：

$$\begin{cases} \beta_i = 0 or \xi_i = 0 \\ \alpha_i = 0 or 1 + \xi_i - y_i w^T \varphi(x_i) - y_i b = 0 \end{cases} \tag{47}$$

取 $0 < \alpha_i < C$，则 $\beta_i = C - \alpha_i > 0$，这意味着在式（47）中对 $\forall i = 1, 2, ..., N$ 有：

$$\begin{cases} \beta_i eq 0 \Rightarrow \xi_i = 0 \\ \alpha_i eq 0 \Rightarrow 1 + \xi_i - y_i w^T \varphi(x_i) - y_i b = 0 \end{cases} \tag{48}$$

解之，得：

$$b^* = \frac{1 - y_i w^T \varphi(x_i)}{y_i} \quad = \frac{1 - y_i \sum_{i=1}^{N} \alpha_i \alpha_j \kappa(x_i, x_j)}{y_i} \tag{49}$$

关于 b 的求解，通常有两种做法。其一是随机选取，其二是在区间内均匀选取并分别求出多个 b，最后遍历所有的 b 取均值作为最终的特解 b^*，通常第二种做法更加科学，精确度更高，鲁棒性更高。

7　神经网络

神经网络作为机器学习领域发展最快的分支，自 21 世纪 10 年代以来，得益于计算机处理能力的飞速提升和海量数据的涌现，其复杂程度不断加深，层数日益增多，形态千变万化。神经网络已演化为深度学习，其也被誉为通向人工智能领域的金钥匙。

本章仅是管中窥豹，从最单纯的网络讲起，慢慢过度到深度学习，对 CV 和 NLP 领域的著名网络予以简单介绍。

7.1　神经元与感知机

现在流行的对于神经网络的公认定义是由 T. Kohonen, 1988,Neural Networks 给出的：“神经网络是由具有适应性的简单单元组成的广泛并行互连的网络，它的组织能够模拟生物神经系统对真实世界物体所作出的交互反应”。而神经网络是一个很大的学科领域，我们仅讨论神经网络与机器学习的交集，即“神经网络学习”亦称“**连接主义**”（connectionism）学习。

上述具有适应性的简单单元也称作神经元（neuron），神经元的模型如下：

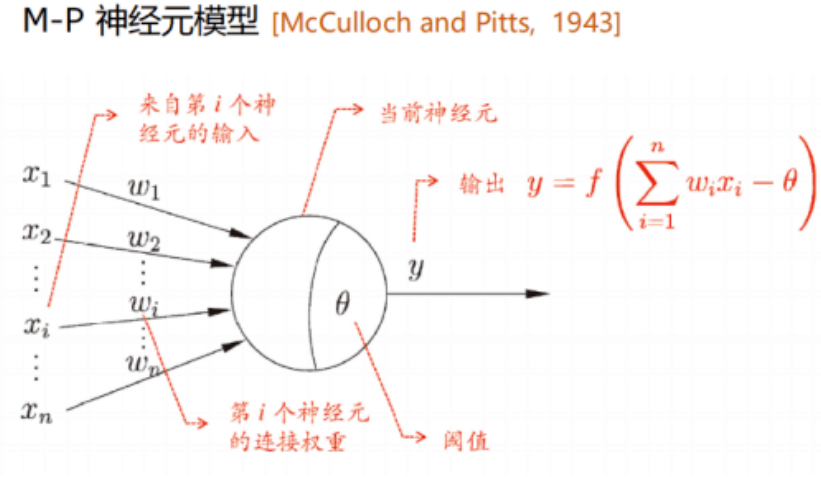

图 15. 神经元模型示意图

如上图，每一个神经元都接受来自其他神经元的输入信号，输入信号通过带有权重的连接传递，得到的总输入值和阈值进行比较，通过**激活函数**（activation function）处理后进行输出。需要注意的是，**神经网络学得的知识蕴含在连接权与阈值中**。

理想激活函数是阶跃函数，0 表示抑制神经元，1 表示激活神经元。但因为阶跃函数具有不连续、不光滑等不好的性质，如图 16，常用的是 Sigmoid 函数（后文会提到）。

1957 年，Frank Rosenblatt 指出能够从一些输入输出对 (X, y) 中通过学习算法获得权重 W 和 b。

感知机算法（Perceptron Algorithm）是由两层神经元组成的神经网络，我们定义增

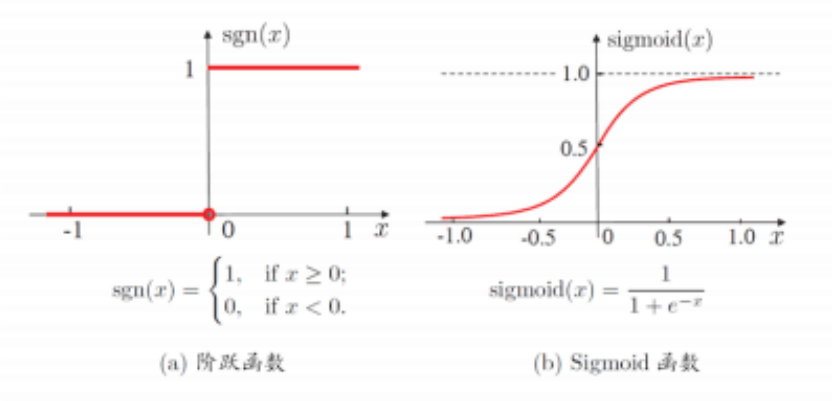

图 16. 激活函数

广向量 $\vec{X} = \begin{cases} \begin{bmatrix} X \\ 1 \end{bmatrix}, y = +1 \\ \begin{bmatrix} -X \\ -1 \end{bmatrix}, y = -1 \end{cases}$，$W = \begin{bmatrix} w \\ b \end{bmatrix}$，那么感知机算法可以描述如下：

输入：$\bar{X}_i$；

Step1：随机取 W；

Step2：若 $W^T\vec{X}_i < 0$，则 $W = W + \vec{X}_i$；

Step3：一直重复 Step2，直到对所有的 $\bar{X}_i$ 都不满足增广向量 $\vec{X}$ 时退出。

感知机收敛定理：输入 $\{\vec{X}_i\}, i = 1 \sim N$，若线性可分，即满足：

$$\exists W_{opt}, s.t W_{opt}^T \vec{X}_i > 0, i = 1 \sim N \tag{1}$$

则利用上述感知机算法，经过有限步骤后，必可以得到一个 W，$s.t. W^T\vec{X}_i > 0, i = 1 \sim N$.

注意：若存在一个超平面能够将数据分开，则必有无数个超平面能够将数据分开，因此 W_{opt} 与 W 仅仅有**十分微小的可能是**同一个超平面。

证明：不失一般性，不妨设 $\|W_{opt}\| = 1$（因为 W_{opt}与aW_{opt} 是同一个平面）

假设第 k 步的 W 是 W_k，且有一个 $\vec{X}_i$，使：$W_k^T\bar{X}_i < 0$

根据感知机算法：

$$\begin{aligned} \|W_{k+1} - aW_{opt}\|^2 &= \left\|(W_k - aW_{opt}) + \vec{X}_i\right\|^2 \\ &= \|(W_k - aW_{opt})\|^2 + \left\|\vec{X}_i\right\|^2 + 2W_k^T\vec{X}_i - 2aW_{opt}^T\vec{X}_i \end{aligned} \tag{2}$$

式（2）中，$2W_k^T\vec{X}_i < 0, 2aW_{opt}^T\vec{X}_i > 0$，因此一定可以取一个很大的 α，使：

$$\|W_{k+1} - aW_{opt}\|^2 < \|(W_k - aW_{opt})\|^2 \tag{3}$$

这可以反映出每一次迭代，都能使新的 W 到平面的距离减小，完成了定性证明。那么，究竟会不会一直减小直至为零呢？答案是会的，下见定量证明：

定义 $\beta = \max\left\{\left\|\vec{X}_i\right\|\right\}, \gamma = \min_{i=1\sim N}(W_{opt}^T X_i)$，取 $a = \frac{\beta^2+1}{2\gamma}$，则：

$$\|W_{k+1} - aW_{opt}\|^2 < \|(W_k - aW_{opt})\|^2 - 1 \tag{4}$$

取 $D = \|(W_0 - aW_{opt})\|$，则至多经过 D^2 步骤后，W 将收敛至超平面 aW_{opt}.

7.2 多层前馈神经网络

7.2.1 工作原理与万有逼近性

多层网络，是指包含隐层的网络；前馈网络，是指神经元之间不存在同层连接也不存在跨层连接（而并不是说信号只能向前传递）。其中，隐层和输出层神经元亦称“功能单元”（Functional Unit）。多层前馈神经网络具有强大的表示能力，即**万有逼近性：仅需一个包含足够多神经元的隐层，多层前馈神经网络就能以任意精度逼近任意复杂度的连续函数**。这个将在下文展开具体的叙述。需要注意的是，如傅里叶变换，泰勒展开等都具有此性质，这并不是神经网络所独有的优良性能。神经网络训练其实是一个“糊涂”的过程：即解一定存在，只是需要自己不断的去寻找。如何设置隐层神经元数量是一个悬而未决的问题，实际常用“试错法”。

我们以一个简单的两层神经网络为例，看看其是如何工作的：

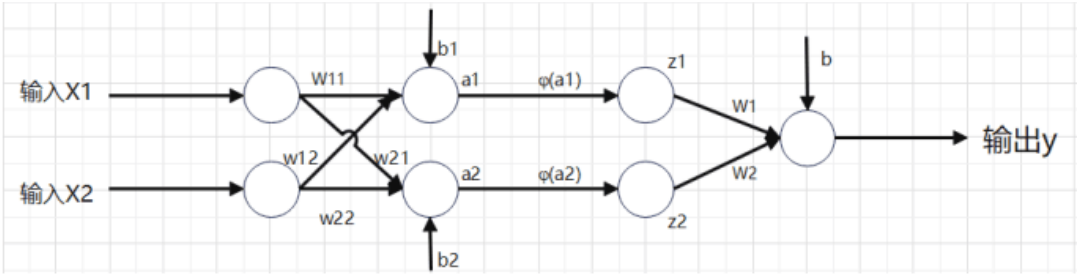

图 17. 一个非常简单的两层神经网络模型示意图

如图 17 所示，$\varphi(.)$ 是一个非线性函数，第一层神经元中有：

$$\begin{cases} a_1 = w_{11}X_1 + w_{12}X_2 + b_1 \\ a_2 = w_{21}X_1 + w_{22}X_2 + b_2 \end{cases} \tag{5}$$

第二层神经元有：

$$\begin{cases} z_1 = \varphi(a_1) \\ z_2 = \varphi(a_2) \end{cases} \tag{6}$$

因此输出是：

$$y = w_1 z_1 + w_2 z_2 + b \tag{7}$$

注意：必须要引进非线性函数，整个神经网络模型才会变成非线性模型，才能够去处理非线性问题！

万有逼近性定理：三层神经网络可以模拟所有决策面。

例 1：在空间 x_1Ox_2 中，有一个三角形的决策面，请问如何使用神经网络模型进行模拟？

此三角形决策面是由三条直线所围成的封闭图形，三条直线方程可以设置为：

$$\begin{cases} w_{11}X_1 + w_{12}X_2 + b_1 = 0 \\ w_{21}X_1 + w_{22}X_2 + b_2 = 0 \\ w_{31}X_1 + w_{32}X_2 + b_3 = 0 \end{cases} \tag{8}$$

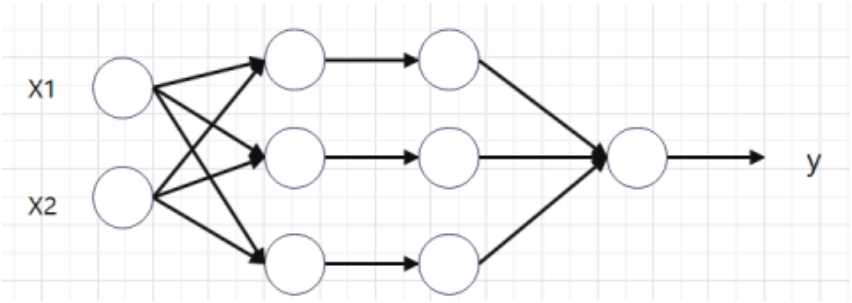

图 18. 两层神经网络模拟示意图

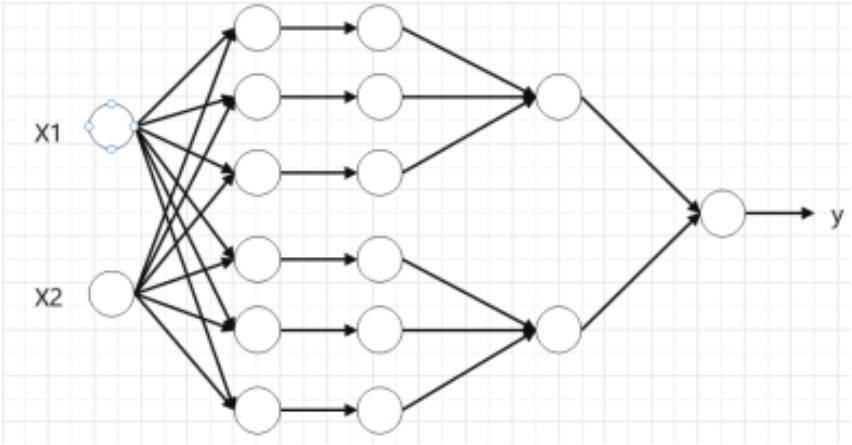

图 19. 三层神经网络模拟示意图

我们可以设置如图 18 所示的两层神经网络，输入为 x_1, x_2，输出为 y。当输出为正值，就说明落在决策面中。为了保证这一点，我们可以在第二层神经网络中，将 b 设置为-2.5。因为第二层一共有 3 个神经元，该层每个神经元的输出信号 (wX) 非 0 即 1，我们必须保证三个输出信号均取 1 时，输出才为正值。

例 2：如果将上例改为决策面为 n 个不同的三角形呢？

只需在例 1 的基础上，增加 n 组神经元，此时神经元数目为 3n。理论上每层神经元的数目可以无限大，因此该问题需要三层神经网络就可以解决。我们以 $n = 2$ 为例，建立如图 19 所示的模拟神经网络。

例 3：如果将上例改为决策面为圆形或不规则形状呢？

思路很简单，就是将圆形或不规则形状近似模拟为很多个直线围成的图形，其他同例 2，有多少条直线，就有多少个神经元（可以无限大）。解决该问题依然只用三层神经网络。

由此，从特殊到一般，可以说明定理的正确性。

7.2.2 误差逆传播（BP 算法）

误差逆传播算法（error BackPropagation , 简称 BP 算法）是迄今为止最著名的神经网络算法，其原理主要基于**梯度下降法**求局部极值，并从输出层神经元依次**从后向前链式求导**。

梯度下降法求局部极值的过程很好理解，想象一个置身于黑暗之中的人，想要下楼梯，他能做的就是探出自己的脚慢慢摸索，发现向下的台阶就移动。所以，这个人最终会到达一个他所能到最低点，即局部最低点，这个最低点不一定是整个函数的最低点（全局最低点）。能够得到什么样的局部最低点往往和初始下降位置有关。下面给出梯度下降法的训练过程：

Step1：找一个 w_0；

Step2：设 $k=0$，假设 $\frac{\partial f(w)}{\partial w}\Big|_{w_k}=0$，退出循环；否则 $w_{k+1}=w_k-\alpha\frac{\partial f(w)}{\partial w}\Big|_{w_k}$。

其中，α 称为学习率或步长，上述更新表达式可以用在 w 局部的泰勒展开式来证明：

$$\begin{aligned} f(w+\Delta w) &= f(w)+\frac{\partial f(w)}{\partial w}\Big|_{w}\cdot\Delta w+o(\Delta w)\\ f(w_{k+1}) &= f(w_k)+(\frac{\partial f(w)}{\partial w}\Big|_{w_k})(-\alpha\frac{\partial f(w)}{\partial w}\Big|_{w_k})+o(\Delta w)\\ &= f(w_k)-\alpha(\frac{\partial f(w)}{\partial w}\Big|_{w_k})^2+o(\Delta w)\\ &< f(w_k) \end{aligned} \tag{9}$$

即后一步永远比前一步小。

下面以图 17 所示的一个非常简单的两层神经网络模型案例来推导 BP 算法：

我们假设数据输入为：$\{(X_i,Y_i)\}_{i=1\sim N}$。我们现在要调节图中的所有的 w 与 b，使得输出 y 尽可能就是 Y_i。那么针对某一个输入 (X,Y)，有优化函数：

$$E=\frac{1}{2}(y-Y)^2 \tag{10}$$

结合梯度下降法写出流程为：

Step1：随机取（ $w_{11},w_{12},w_{21},w_{22},b_1,b_2,w_1,w_2,b$ ）；

Step2：求所有的 $\frac{\partial E}{\partial w}$与$\frac{\partial E}{\partial b}$；

Step3：

$$w^{new}=w^{old}-\alpha\frac{\partial E}{\partial w}\bigg|_{w_{old}}\quad b^{new}=b^{old}-\alpha\frac{\partial E}{\partial b}\bigg|_{b_{old}}\tag{11}$$

；

Step4：当所有 $\frac{\partial E}{\partial w}$与$\frac{\partial E}{\partial b}$ 都为零时，退出循环。

于是，我们可以从后向前链式求导：

$$\frac{\partial E}{\partial y}=y-Y\tag{12}$$

$$\begin{aligned}\frac{\partial E}{\partial a_1}&=\frac{\partial E}{\partial y}\cdot\frac{\partial y}{\partial z_1}\cdot\frac{\partial z_1}{\partial a_1}=(y-Y)w_1\varphi'(a_1)\\\frac{\partial E}{\partial a_2}&=\frac{\partial E}{\partial y}\cdot\frac{\partial y}{\partial z_2}\cdot\frac{\partial z_2}{\partial a_2}=(y-Y)w_2\varphi'(a_2)\end{aligned}\tag{13}$$

$$\frac{\partial E}{\partial b}=\frac{\partial E}{\partial y}\cdot\frac{\partial y}{\partial b}=y-Y\tag{14}$$

$$\begin{aligned}\frac{\partial E}{\partial w_1}&=\frac{\partial E}{\partial y}\cdot\frac{\partial y}{\partial w_1}=(y-Y)z_1\\\frac{\partial E}{\partial w_2}&=\frac{\partial E}{\partial y}\cdot\frac{\partial y}{\partial w_2}=(y-Y)z_2\end{aligned}\tag{15}$$

$$\begin{aligned}\frac{\partial E}{\partial w_{11}}&=\frac{\partial E}{\partial a_1}\cdot\frac{\partial a_1}{\partial w_{11}}=(y-Y)\cdot w_1\cdot\varphi'(a_1)\cdot X_1\\\frac{\partial E}{\partial w_{12}}&=\frac{\partial E}{\partial a_1}\cdot\frac{\partial a_1}{\partial w_{12}}=(y-Y)\cdot w_1\cdot\varphi'(a_1)\cdot X_2\end{aligned}\tag{16}$$

$$\frac{\partial E}{\partial b_1}=(y-Y)w_1\varphi'(a_1)\tag{17}$$

$$\begin{aligned}\frac{\partial E}{\partial a_1}&=\frac{\partial E}{\partial y}\cdot\frac{\partial y}{\partial z_1}\cdot\frac{\partial z_1}{\partial a_1}=(y-Y)w_1\varphi'(a_1)\\\frac{\partial E}{\partial a_2}&=\frac{\partial E}{\partial y}\cdot\frac{\partial y}{\partial z_2}\cdot\frac{\partial z_2}{\partial a_2}=(y-Y)w_2\varphi'(a_2)\end{aligned}\tag{18}$$

$$\frac{\partial E}{\partial b_2}=(y-Y)w_2\varphi'(a_2)\tag{19}$$

上面的式子中，非线性函数（激活函数）决定了最终的化简结果，下面列举几种最常用的非线性函数：

（1）$\varphi(x)=sigmoid(x)=\frac{1}{1+e^{-x}}$

$\varphi'(x)=\varphi(x)[1-\varphi(x)]$

（2）$\varphi(x)=\tanh(x)=\frac{e^x-e^{-x}}{e^x+e^{-x}}$

$\varphi'(x)=1-\varphi^2(x)$

在深度学习中，常用以下激活函数：

（3）$\varphi(x)=relu(x)=\max(0,x)$

$$\varphi'(x)=\begin{cases}1,x>0\\0,x\le 0\end{cases}$$

意义：压缩小于 0 的数，将其变成 0. 使该层神经网络中活跃的参数减少，降低训练复杂度。事实上，这种做法往往太过武断，实际常用 LeakRelu 函数。

（4）$\varphi(x) = \text{leakrelu}(x) = \begin{cases} x, & x > 0 \\ \varepsilon x, & x \leq 0 \end{cases}$

$\varphi'(x) = \begin{cases} 1, x > 0 \\ \varepsilon, x \leq 0 \end{cases}$

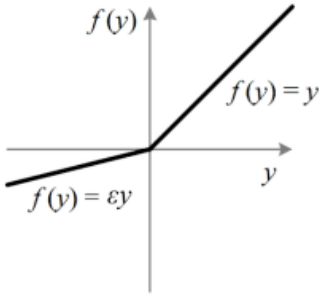

图 20. LeakRelu 函数图像

接下来，我们来推导一般形式的 BP 前馈神经网络的相关步骤。首先，将一个含有 l 层神经元的神经网络向量化，如图 21：

$$
\begin{aligned}
&\text{输入} X(a^{(0)}) \Rightarrow W^{(1)}a^{(0)} + b^{(1)} = z^{(1)} \xrightarrow{\varphi} a^{(1)} = \varphi(z^{(1)}) \\
&\Rightarrow z^{(2)} = W^{(2)}a^{(1)} + b^{(2)} \xrightarrow{\varphi} a^{(2)} = \varphi(z^{(2)}) \\
&\Rightarrow \ldots\ldots \\
&\Rightarrow z^{(l)} = W^{(l)}a^{(l-1)} + b^{(l)} \xrightarrow{\varphi} a^{(l)} = \varphi(z^{(l)}) \\
&\Rightarrow \text{输出} y
\end{aligned}
$$

图 21. 网络向量化

不妨设 $\delta_i^{(m)} = \frac{\partial E}{\partial z_i^{(m)}}$，用以表示求第 m 层 z 的第 i 个分量的偏导数。

Step1：

$$\delta_i^{(l)} = \frac{\partial E}{\partial z_i^{(l)}} = \frac{\partial E}{\partial y_i}\frac{\partial y_i}{\partial z_i^{(l)}} = (y_i - Y_i)\varphi'(z_i^{(l)}) \tag{20}$$

Step2：（递推公式）

$$\delta_i^{(m)} = \frac{\partial E}{\partial z_i^{(m)}} = \frac{\partial E}{\partial a_i^{(m)}} \cdot \frac{\partial a_i^{(m)}}{\partial z_i^{(m)}} = (\sum_{j=1}^{m-1} \delta_j^{(m+1)} \cdot W_{ij}) \cdot \varphi'(z_i^{(m)}) \tag{21}$$

Step3：

$$\begin{cases} \frac{\partial E}{\partial W_{ij}^{(m)}} = \delta_j^{(m)} a_i^{(m-1)} \\ \frac{\partial E}{\partial b_i^{(m)}} = \delta_i^{(m)} \end{cases} \tag{22}$$

Step4：联立上述三个式子可以求出所有的 w 与 b，此后更新参数：

$$w^{new} = w^{old} - \alpha \frac{\partial E}{\partial w}\bigg|_{w_{old}} \quad b^{new} = b^{old} - \alpha \frac{\partial E}{\partial b}\bigg|_{b_{old}} \tag{23}$$

综上所述，我们可以总结出 BP 算法的四个步骤如下：

Step1: 随机初始化（W 与 b）；

Step2：训练样本 (X, Y)，代入网络，可求出所有的 (z, a, y)；

Step3：链式法则求偏导：($\frac{\partial E}{\partial w}$与$\frac{\partial E}{\partial b}$)；

Step4：更新参数，循环，直至停止训练。

7.3 softmax 回归

我们从线性回归任务说起，从回归到多分类任务，回归任务通常将单连续数值作为输出、位于自然区间 R、将跟真实值的区别作为损失。而多分类任务通常有多个输出，输出 i 是预测为第 i 类的置信度。我们可以将线性回归任务单纯的理解为单层神经网络：

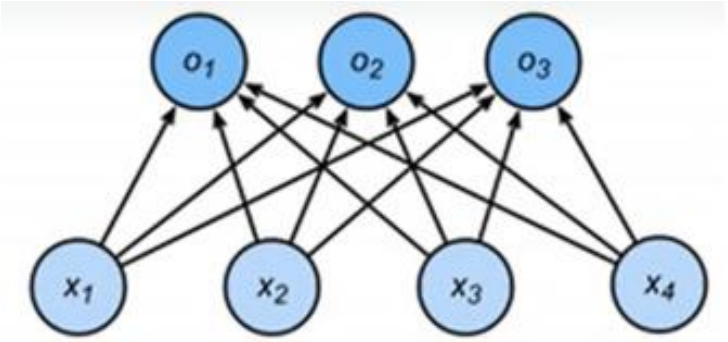

图 22. 单层神经网络

对类别进行一位有效编码 $y = [y_1, y_2, ..., y_n]^T$，其中

$$y_i = \begin{cases} 1, i = y \\ 0, else \end{cases} \tag{24}$$

使用**均方损失**训练，最大值即为预测：

$$\hat{y} = \arg\max_i o_i \tag{25}$$

为了达到将**输出直接视为概率**这一目标，我们需要**校验比例**，即将未规范化的预测变换为非负数并且总和为 1，同时让模型保持可导的性质。为此，引入 softmax 因子，按下式输出匹配概率，并将概率 y 和$\hat{y}$ 的区别作为损失：

$$\begin{aligned} \hat{y} &= \text{soft max}(o) \\ \hat{y}_i &= \frac{\exp(o_i)}{\sum_k \exp(o_k)} \end{aligned} \tag{26}$$

需要注意的是，softmax 运算不会改变未规范化的预测 o 之间的大小次序，只会确定分配给每个类别的概率。尽管 softmax 是一个非线性函数，但 softmax 回归的输出仍然由输入特征的仿射变换决定。因此，softmax 回归是一个线性模型（linear model）。

为了提高计算效率，我们会采取小批量样本进行矢量计算，于是可以得到 softmax 回归的非常漂亮的向量化表达形式：

$$\begin{aligned} O &= XW + b \\ \hat{Y} &= \text{softmax}(O) \end{aligned} \tag{27}$$

下面介绍**交叉熵损失**，交叉熵常用来衡量两个概率的区别：

$$H(p, q) = \sum_i -p_i \log(q_i) \tag{28}$$

将它作为损失函数：

$$l(y, \hat{y}) = -\sum_i y_i \log \hat{y}_i = -\log \hat{y}_y \tag{29}$$

其梯度是真实概率与预测概率的区别：

$$l'(y, \hat{y}) = \text{softmax}(o)_i - y_i \tag{30}$$

【总结】

1.softmax 回归实际上是一个多分类模型；

2. 使用 softmax 因子得到每个类的预测置信度；

3. 使用交叉熵来衡量预测和标号的区别。

7.4 Dropout

丢弃法（也称**暂退法**，Dropout）的思考动机是，一个好的模型需要对输入数据的扰动鲁棒，以期提高模型的泛化能力，使用有噪音的数据等价于 Tikhonov 正则。在训练过程中，在计算后续层之前向网络的每一层注入噪声。（即在层间加入噪音）。当训练一个有多层的深层网络时，注入噪声只会在输入—输出映射上增强平滑性。

无偏差的加入噪音，对 x 加入噪音 x'，我们希望：

$$E(x') = x \tag{31}$$

丢弃法对每个元素进行如下扰动：

$$x_i' = \begin{cases} 0, & withprobablityp \\ \frac{x_i}{1-p}, & else \end{cases} \tag{32}$$

实际使用中，通常将丢弃法作用在隐藏全连接层的输出上。

推理中，正则项只在训练中使用，他们影响模型参数的更新，丢弃法直接返回输入：

$$h = dropout(h) \tag{33}$$

这样也能保证确定性的输出。

总结一下，有以下三点需要注意：

（1）丢弃概率是控制模型复杂度的超参数；

（2）丢弃法将一些输出项随机置 0 来控制模型复杂度；

（3）常作用在多层感知机的隐藏层输出上。

7.5 数值稳定性

7.5.1 梯度消失

本节，我们就前文 BP 算法中的 (w, b) 初始化这一操作所引发的梯度消失进行讨论。如果 $w^T x + b$ 一开始很大或很小，那么梯度将趋近于 0，反向传播后前面与之相关的梯度也趋近于 0，导致训练缓慢。因此，我们需要使 $w^T x + b$ 一开始就在零附近。

一种比较简单有效的办法是：(w, b) 的初始化从区间 $(-\frac{1}{\sqrt{d}}, \frac{1}{\sqrt{d}})$ 均匀随机取值，其中 d 为 (w, b) 所在层的神经元个数。可以证明：如果 x 服从正态分布，均值为 0，方差为 1，且各个维度无关，而 (w, b) 是 $(-\frac{1}{\sqrt{d}}, \frac{1}{\sqrt{d}})$ 的均匀分布，则 $w^T x + b$ 是均值为 0，方差为 $\frac{1}{3}$ 的正态分布。

7.5.2 梯度爆炸

考虑有如下 d 层的神经网络：

$$h^t = f_t(h^{t-1}) \quad and \quad y = \ell \cdot f_d \cdot ... \cdot f_1(x) \tag{34}$$

计算损失 l 关于参数 W_t 的梯度：

$$\frac{\partial \ell}{\partial W_t} = \frac{\partial \ell}{\partial h_d} \cdot \frac{\partial h_d}{\partial h_{d-1}} \cdot ... \cdot \frac{\partial h_{t+1}}{\partial h_t} \cdot \frac{\partial h_t}{\partial W_t} \tag{35}$$

式（35）涉及向量求导，得到矩阵，因此会发生梯度爆炸的现象。解释是：如果使用线性修正单元作为激活函数，则有 $\prod_{i=t}^{d-1} \frac{\partial h^{i+1}}{\partial h^i} = \prod_{i=t}^{d-1} \mathrm{diag}\left(\sigma'(w^i h^{i-1})\right)(w^i)^T$ 的一些元素会来自于 $\prod_{i=t}^{d-1}(w^i)^T$。此时，如果 $(d-t)$ 很大，那么值将会很大。

梯度爆炸一般会导致两个问题：其一是值超出值域（infinity），对于 16 位浮点数尤为严重（数值区间 $6e^{-5}$~$6e^{4}$）；其二是对学习率敏感，如果学习率太大则造成大参数值，这就意味着造成更大的梯度；如果学习率太小通常训练无进展。由此，我们可能需要在训练过程不断调整学习率。

类似的，如果使用 Sigmoid 函数作为激活函数，就会导致上式中的元素值是 $(d-t)$ 个小数相乘，结果会非常小，接近于零，因此会发生梯度消失现象。这通常会带来三个问题：其一，梯度值变成 0，对 16 位浮点数尤为严重；其二，不论如何选择学习率，训

练都没有进展；其三，对于底部层尤为严重，仅仅顶部层训练得好，无法使神经网络更深。

总之，合理的权重初始值和激活函数的选取可以提升数值稳定性。

7.6 卷积神经网络与 CV

7.6.1 何谓卷积？

要学**卷积神经网络**（Convolutional Neural Networks, CNN），首当其冲的是理解卷积运算这个概念。卷积的计算公式很简洁：

$$(f^{*}g)(t)=\int f(\tau)\,g(t-\tau)\,d_{\tau} \tag{36}$$

乍一看，令人头疼而无从下手。事实上，它用了一个极简的数学公式漂亮地描述了一个动态过程。用一个生动形象的例子，便是火车进山洞的过程。

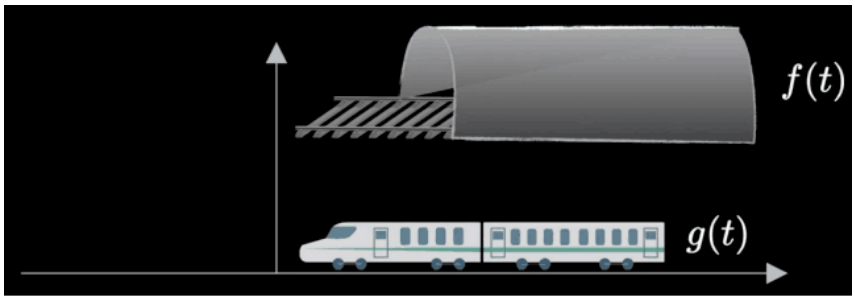

图 23. 理解卷积——火车进山洞

观察图 23，假设山洞是函数 $f(t)$，沿 y 轴逆向行驶的火车是函数 $g(t)$。想象一下，为使火车面向山洞，即在 y 轴左侧沿着正方向行驶，函数经过旋转后则变为 $g(t-\tau)$。此后火车开始运动，乘积 $f(\tau)g(t-\tau)$ 表示的是任一时刻 t 时，两者相对的位置。再将其求积分，就描述了**两者相互作用过程中重叠的部分**。这便是卷积的意义。

卷积有如下的性质：

性质 1：卷积函数既可以是连续的，也可以是离散的。

性质 2：**可交换性**，理解是火车进山洞的过程也可以理解为山洞进火车，运动是相对的。

性质 3：在深度学习领域，由于对称性，互相关运算可以等价于卷积运算。

在卷积的计算公式中，第一个参数 t 称为**输入**，第二个参数 τ 称为**核函数（与 SVM 里的不一样）**，卷积运算后的输出结果称为**特征映射**。现实中，核函数往往是高维的，以二维为例，卷积运算可以写为：

$$(I^{*}K)(i,j)=\sum\sum I(m,n)\,K(i-m,j-n) \tag{37}$$

交换后的式子是：

$$(K^{*}I)(i,j)=\sum_{m}\sum_{n} I(i-m,j-n)\,K(m,n) \tag{38}$$

图 24 是一个计算卷积的实际例子：

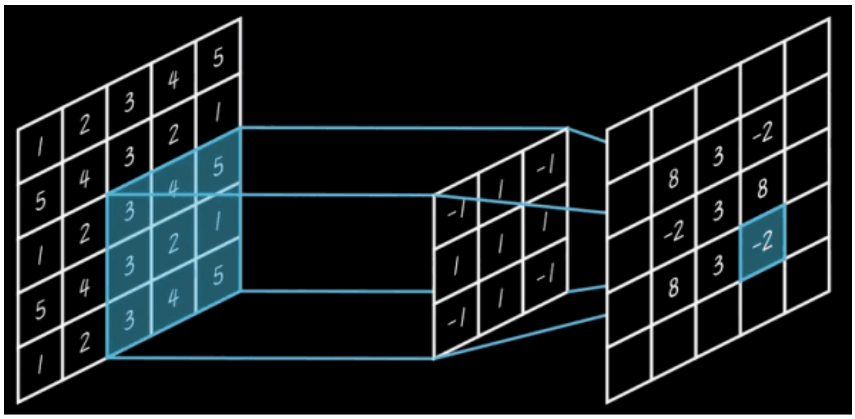

图 24. 卷积运算简单示例

要求计算特征映射矩阵中位于右下角的元素，便是左侧大矩阵中右下角的蓝色区域矩阵经过核函数矩阵作用后的输出，结果为将每个元素对应相乘并相加：

$$3\times(-1)+4\times1+5\times(-1)+3\times1+2\times1+1\times1+3\times(-1)+4\times1+5\times(-1)=-2 \quad (39)$$

7.6.2 超参数——填充与步幅

填充指的是在输入周围添加额外的行和列，这是为正确计算卷积而产生的操作。如下面的示例中，输入是一个 3×3 的矩阵，卷积核是一个 2×2 的矩阵，此时无法直接计算卷积的结果。因此，我们需要将输入周围一圈补上零，再进行正确的卷积计算。

$$\begin{bmatrix}0&0&0&0&0\\0&0&1&2&0\\0&3&4&5&0\\0&6&7&8&0\\0&0&0&0&0\end{bmatrix} * \begin{bmatrix}0&1\\2&3\end{bmatrix} = \begin{bmatrix}0&3&8&4\\9&19&25&10\\21&37&43&16\\6&7&8&0\end{bmatrix} \quad (40)$$

填充 p_h 行和 p_w 列，输出形状为：

$$(n_h-k_h+p_h+1)\times(n_w-k_w+p_w+1) \quad (41)$$

通常取 $p_h=k_h-1, p_w=k_w-1$，

（1）当为奇数时，在上下两侧填充 $\frac{p_h}{2}$；

（2）当为偶数时，在上侧填充 $\lceil p_h/2\rceil$，在下侧填充 $\lfloor p_h/2\rfloor$。

步幅是指行和列的滑动步长。因为填充减小的输出大小与层数呈线性相关，所以需要大量的计算才能够得到较小输出。例如给定输出大小 224×224 像素，在使用 5×5 的卷积核的情况下，需要 44 层将输出降低到 4×4 像素。在这一背景下，为了弱化计算量，因此产生了改动滑动步长的想法，即让每次卷积扫过的间隔变大。

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 & 0 \\ 0 & 3 & 4 & 5 & 0 \\ 0 & 6 & 7 & 8 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 0 & 1 \\ 2 & 3 \end{bmatrix} = \begin{bmatrix} 0 & 8 \\ 6 & 8 \end{bmatrix}$$

图 25. 计算卷积输出的过程，注意标红之处

例：用高度为 3，宽度为 2 的步幅计算下列输入的卷积输出。（图 25 所示的计算过程演示了输出中位于右上角的结果的来源）

给定高度为 s_h 和宽度为 s_w 的步幅，输出形状是：

$$\lfloor (n_h - k_h + p_h + s_h)/s_h \rfloor \times \lfloor (n_w - k_w + p_w + s_w)/s_w \rfloor \tag{42}$$

（1）如果 $p_h = k_h - 1, p_w = k_w - 1$，输出形状是：

$$\lfloor (n_h + s_h - 1)/s_h \rfloor \times \lfloor (n_w + s_w - 1)/s_w \rfloor \tag{43}$$

（2）如果输入高度和宽度可以被步幅整除，输出形状是：

$$(n_h/s_h) \times (n_w/s_w) \tag{44}$$

总结一下，填充和步幅都是卷积层的超参数。填充的主要作用是控制输出形状的减少量；步幅则是可以成倍的减少输出形状。

7.6.3 超参数——多输入多输出通道

卷积神经网络使用者往往会非常仔细的设置的超参数，便是多输入和多输出通道了。彩色图像可能有 RGB 三个通道，如果简单粗暴的直接采用单通道（转换为灰度图像）的话会丢失信息。每个通道都有一个卷积核，其运算结果是所有通道卷积结果的和。示例如图 26：

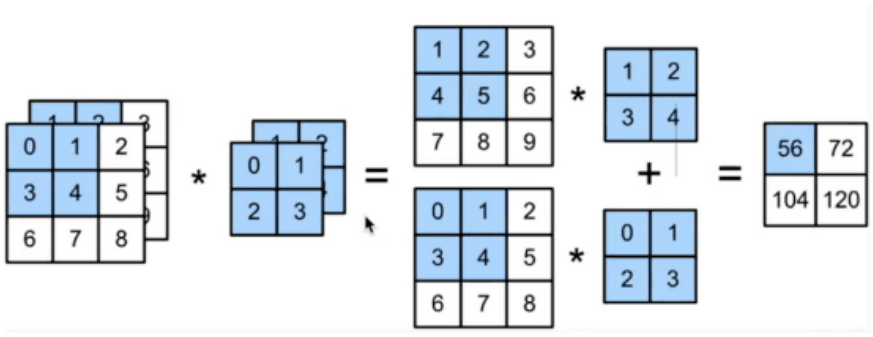

图 26. 通道示例

需要注意的是，无论有多少输入通道，到目前为止我们只用到单输出通道。我们可以有多个三维卷积核，每个核生成一个输出通道。每个输出通道可以识别特定模式，输入通道核识别并组合输入中的模式。

在实际任务中，1×1 的卷积层 $(k_h = k_w = 1)$ 是一个受欢迎的选择，它不识别空间模式，只是起到融合通道的作用，等价于一个全连接层。计算复杂度：

$$(\text{浮点计算数}FLOP)O(c_i c_0 k_h k_w m_h m_w) \tag{45}$$

举例：如果是 10 层，1M 样本，10PFLOPs，使用 CPU：0.15TF，需要计算 18 小时；使用 GPU：12TF，需要计算 14 分钟。

7.6.4 池化

从一个简单的例子说起，式（46）是一个完成检测垂直边缘的任务：

$$X: \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \end{bmatrix} * K: \begin{bmatrix} 1 & -1 \end{bmatrix} = Y: \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \tag{46}$$

这样，就实现了输出 Y 中的垂直边缘被检测出的效果。这个例子也反映了**卷积对位置敏感的特性**。在实际任务中，我们得到的图像往往不可能是完美的：照明条件、物体位置、比例、外观等因素因图像而异。因此我们在处理计算机视觉的任务时，往往需要一定的**平移不变性**，在发生一些扰动（一定范围内的变化时），我们也要有办法去应对，使输出不会产生明显的变化（基本不变）。

而**池化层**（pooling），也称为**汇聚层**便应运而生，其作用是降低卷积层对位置的敏感性，同时降低对空间降采样表示的敏感性。

最常见的两种简单池化方法是**最大池化法**与**平均池化法**。

以二维最大池化为例，就是返回滑动窗口中的最大值，示例如下：

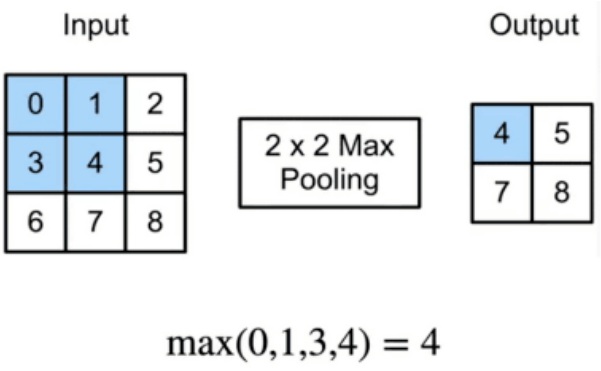

图 27. 二维最大池化简单示例

以上面的垂直边缘检测示例为例，如果经过 2×2 的最大池化后，输出如下：

$$\begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \end{bmatrix} \tag{47}$$

可以发现，相比原来单纯的卷积操作，经最大池化后的结果左右各可容 1 位像素移位。这样便提升了模型的容错率，增强了在实际中运用的效果。

（1）最大池化层：每个窗口中最强的模式信号；

（2）平均池化层：每个窗口中的模式信号取平均值。

下面做个简单的总结：

- 池化与卷积层类似，都具有填充、步幅以及窗口大小作为超参数；
- 池化层没有可学习的参数；
- 在每个输入通道应用池化层以获得相应的输出通道；
- 输出通道数 = 输入通道数

7.6.5 计算机视觉

60 多年来，科学家和工程师一直在尝试开发各种方法，让机器能够看到和理解视觉数据。在 1959 年的第一次实验中，神经生理学家向一只猫展示一组图像，试图唤起猫大脑的反应。他们发现猫会先对硬边缘或线条做出反应，从科学角度来说，这意味着图像处理从简单的形状开始，例如直边。

大约在同一时期，第一个计算机图像扫描技术成功地开发出来，使计算机能够将图像数字化并获取图像。1963 年，计算机能够将二维图像转换为三维形式，标志着第二个里程碑的实现。在 20 世纪 60 年代，人工智能作为一个学术域研究诞生了，同时也标志着人们开始探求依靠人工智能解决人类视觉问题的方法。

1974 年，光学字符识别 (OCR) 技术走向市场，它能够识别以任何字体或字型打印的文字。同样，智能字符识别 (ICR) 能够使用神经网络辨认手写文字。此后，OCR 和 ICR 广泛地运用到文件和发票处理、车牌识别、移动支付、机器翻译和其他常见领域。

1982 年，神经系统科学家 David Marr 证实了视觉分层工作原理，并推出了使机器能够检测边缘、角落、曲线和类似的基本形状的算法。与此同时，计算机科学家 Kunihiko Fukushima 开发了一个能够识别模式的细胞网络。这个网络称为 Neocognitron，它在一个神经网络中包含了多个卷积层。

到 2000 年，物体识别成为研究重点，2001 年，第一个实时人脸识别应用诞生。在 21 世纪初，逐渐形成了视觉数据集标记和注释的标准化实践。2010 年，ImageNet 数据集公开可用。该数据集包含上千种物体的数百万张标记的图像，为如今使用的 CNN 和深度学习模型奠定了基础。2012 年，来自多伦多大学的团队带着一个 CNN 模型参加了图像识别竞赛。这个名为 AlexNet 的模型显着降低了图像识别的错误率。在这一次突破后，错误率已经下降到仅仅百分之几的水平。

卷积网络发展很快，笔者仅介绍一个简单的卷积网络如下：LeNet 诞生于 20 世纪末期，是卷积神经网络中的早期成功代表，用于识别图像中的手写数字。LeNet 先使用卷积层来学习图片空间信息，再使用卷积层来降低敏感度，然后使用全连接层来转换到类别空间。LeNet 由两个部分组成：**卷积编码器**和**全连接层密集块**。

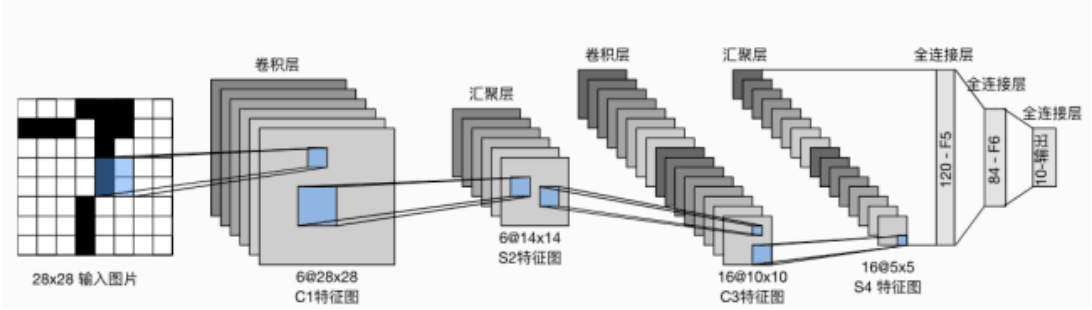

图 28. LeNet 空间结构示意图

7.7 循环神经网络与 NLP

7.7.1 循环神经网络导论

循环神经网络 (RNN) 是一种使用序列数据或时序数据的人工神经网络。这些深度学习算法常用于顺序或时间问题，如语言翻译、自然语言处理 (nlp)、语音识别、图像字幕等；它们包含在一些流行的应用中，比如 Siri、语音搜索和 Google Translate。与前馈神经网络和卷积神经网络 (CNN) 一样，循环神经网络利用训练数据进行学习。区别在于“记忆”，因为它从先前的输入中获取信息，以影响当前的输入和输出。虽然传统的深度神经网络假设输入和输出相互独立的，但循环神经网络的输出依赖于序列中先前的元素。尽管未来的活动也可能有助于确定特定序列的输出，但是单向循环神经网络无法在预测中说明这些事件。

让我们举个惯用语的例子来帮助解释循环神经网络 (RNN)：“feeling under the weather”（感觉身体不舒服），这通常是指某人病了。为了体现出这个惯用语的意思，必须按这个特定顺序进行表达。因此，循环神经网络需要考虑到该惯用语中每个单词的位置，并使用这些信息来预测序列中的下一个单词。

循环神经网络 (RNN) 的另一个显著特征是它们在每个网络层中共享参数。虽然前馈网络的每个节点都有不同的权重，但循环神经网络在每个网络层都共享相同的权重参数。尽管如此，这些权重仍可通过反向传播和梯度下降过程进行调整，以促进强化学习。

循环神经网络 (RNN) 利用随时间推移的反向传播 (BPTT) 算法来确定梯度，这与传统的反向传播略有不同，因为它特定于序列数据。BPTT 的原理与传统的反向传播相同，模型通过计算输出层与输入层之间的误差来训练自身。这些计算帮助我们适当地调整和拟合模型的参数。BPTT 与传统方法的不同之处在于，BPTT 会在每个时间步长对误差求和，而前馈网络则不需要对误差求和，因为它们不会在每层共享参数。

通过这个过程，循环神经网络 (RNN) 往往会产生两个问题，即梯度爆炸和梯度消失。这些问题由梯度的大小定义，也就是损失函数沿着错误曲线的斜率。如果梯度过小，它会更新权重参数，让梯度继续变小，直到变得可以忽略，即为 0。发生这种情况时，算法就不再学习。如果梯度过大，就会发生梯度爆炸，这会导致模型不稳定。在这种情况下，模型权重会变得太大，并最终被表示为 NaN。这些问题的一种解决方案就是减少神经网络中隐藏层的数量，以便消除循环神经网络 (RNN) 模型中的一些复杂性。

下面介绍几种常见的循环神经网络变种：

1. **双向循环神经网络**（BRNN)：这是循环神经网络 (RNN) 的网络架构变体。单向循环神经网络 (RNN) 只能从先前输入中抽取数据，做出有关当前状态的预测；而双向循环神经网络 (RNN) 还可以拉取未来的数据，从而提高预测的精度。回到前面那个“feeling under the weather”的例子，如果模型知道该序列中的最后一个单词是“weather”，就更有可能预测该词组中的第二个单词是“under”。

2. **长短期记忆** (LSTM)：这是一种比较流行的循环神经网络 (RNN) 架构，由 Sepp Hochreiter 和 Juergen Schmidhuber 提出，用于解决梯度消失问题。在他们的论文（链接位于 IBM 外部）中，他们着力解决长期依赖问题。也就是说，如果影响当前预测的先前状态不是最近发生的，那么循环神经网络 (RNN) 模型可能无法准确预测当前状态。例如，假设我们想要预测以下斜体句子“Alice is allergic to nuts. She can’t eat peanut butter.”（Alice 对坚果过敏。她不能吃花生酱）。“坚果过敏”上下文可以帮助我们预测不能吃的食物含有坚果。但是，如果上下文是之前的几句话，那么循环神经网络 (RNN) 就很难甚至无法连接信息。作为补救措施，LSTM 在神经网络的隐藏层中包含一些“元胞”(cell)，共有三个门：一个输入门、一个输出门和一个遗忘门。这些门控制着预测网络中的输出所需信息的流动。例如，如果在先前的句子中，性别代词（比如 she）重复出现了多次，那么可将其从元胞状态中排除。

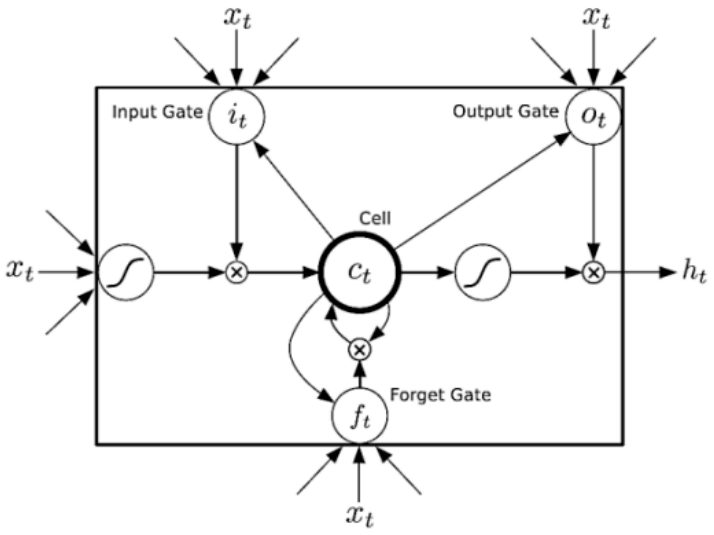

图 29. LSTM 网络框架图

如图 29，LSTM 网络主要由输入词 x_t，细胞状态 c_t，隐层状态 h_t，遗忘门 f_t，记忆门 i_t，输出门 o_t 六部分组成。其基本工作原理为：在细胞状态中进行信息遗忘以及记忆新的信息，使得有用的信息得以传递，无用的信息得以丢弃。下面将计算步骤分述如下：

Step1：输入词进入处理细胞中。

Step2：计算遗忘门：

$$f_t = \sigma\big(w_f \cdot \big[h_{t-1}, x_t\big] + b_f\big) \tag{48}$$

Step3：计算记忆门与当前时刻的细胞状态：

$$i_t = \sigma\left(w_i \cdot [h_{t-1}, x_t] + b_i\right) \tilde{c}_t = \tanh\left(w_c \cdot [h_{t-1}, x_t] + b_c\right) \tag{49}$$

Step4：计算当前时刻细胞状态：

$$c_t = f_t \cdot c_{t-1} + i_t \cdot \tilde{c}_t \tag{50}$$

Step5：计算输出门与当前时刻的隐层状态：

$$o_t = \sigma\big(w_o \cdot \big[h_{t-1}, x_t\big] + b_o\big) h_t = o_t \cdot \tanh\left(c_t\right) \tag{51}$$

Step6：最终得到与句子长度相同的隐层状态序列：

$$\{h_0, h_1, ..., h_{n-1}\} \tag{52}$$

双向长短期记忆网络（BiLSTM）是一种改进版的 LSTM，它加入了从后往前的逆向信息编码过程，能够更好的捕捉双向的语义依赖关系，这对于提高多分类任务的准确率有很大帮助。如分析语句“我不觉得你很差”时，“不”是对后面“差”的修饰，双重否定表示肯定，该句的情感极性分类为“褒义”。而一些简单的 NLP 模型捕捉到“不”等否定词时一般容易判断为负面的意思，因此，在处理较为细腻的情感分类任务时，BiLSTM 拥有较大的优势。

3. **门控循环单元** (GRU)：这种循环神经网络 (RNN) 变体类似于 LSTM，因为它也旨在解决 RNN 模型的短期记忆问题。但它不使用“元胞状态”来调节信息，而是使用隐藏状态；它不使用三个门，而是两个：一个重置门和一个更新门。类似于 LSTM 中的门，重置门和更新门控制要保留哪些信息以及保留多少信息。

7.7.2 注意力机制

Transformer 模型是 2017 年由 Google 提出的，其最引人入胜的一点是采用了自注意力机制取代了传统的循环神经网络。注意力机制的灵感来自于对人类日常行为中的“注意力”的研究，人类的注意力资源是有限的，我们在关注的目标区域投入更多的注意力资源，从而在目标区域获得更多的信息，抑制其他无用的信息。一言以蔽之，其目的就是为了在大量繁琐的信息中筛选出有价值的信息。

在注意力机制中有三个重要的向量，分别是查询向量（Query vector）、键向量（Key vector）、值向量（Value vector）。查询向量表示当前需要处理的信息，模型根据查询向

量在输入序列中查找相关信息；键向量是来自输入序列的一组表示向量，根据查询计算注意力权重，反映不同位置的输入数据与查询的相关性；值向量是来自输入序列的一组表示向量，用于根据注意力权重计算加权和，得到最终的注意力输出向量，包含了与查询最相关的输入信息。

注意力机制通过计算 Q 与各个 K 之间的相似性，为每个 V 分配一个权重。然后，将加权的值相加（每个 V 乘以对应的权重，即注意力分数），得到一个蕴含输入序列最相关信息的输出向量。这个输出向量的形状与 Q 相同，将用于计算下一步模型计算的输入。注意力机制可以描述为：

$$Attention\,(Q,K,V) = soft\max\left(\frac{QK^T}{\sqrt{d_k}}\right)\cdot V \tag{53}$$

7.7.3 Transformer

Transformer 模型可以简单的分为四个部分：输入词向量、N 层编码器（Encoder）、N 层解码器（Decoder）、输出。模型的网络结构如图 x，下面笔者将按照这四个组件简述工作机理。

1. 输入层

将需要处理的文本序列转换为一个输入词向量，然后为这些词向量添加位置编码，从而为模型提供位置信息，输出进入编码层。

位置编码的目的是让模型能够区分输入序列中不同位置的词，模型采用的是正弦位置编码是因为其具有平滑性和保留相对位置信息等优点，公式如下：

$$PE(pos,2i) = \sin\left(\frac{pos}{10000^{2i/d}}\right)\, PE(pos,2i+1) = \cos\left(\frac{pos}{10000^{2i/d}}\right) \tag{54}$$

2. 编码器

编码器是由多个相同的结构层堆叠而成，每一个层包含了两个主要部分: 多头自注意力和前馈神经网络。

词向量和位置编码将结合起来进入到编码器的第一层，首先进行多头自注意力计算。每个头都有自己的注意力权重，这些权重将被用来对输入序列的不同部分加权求和。多头自注意力的输出会进入残差连接，经过层归一化处理。这样做有助于稳定模型的训练过程，提高收敛速度。此后，进入到前馈神经网络，这个过程会重复执行 N 次，最终的输出将被传递给解码器。

3. 解码器

解码器的主要任务是基于编码器输出的上下文向量生成目标序列，它同样也由多个相同的结构层堆叠而成。每个层主要包含多头自注意力，编码器—解码器注意力机制，前馈神经网络。

首先进行多头自注意力计算，与编码系中的多头自注意力计算不同的是，解码器要遵循一个重要的原则：**只能关注已经生成的输出序列的位置，避免在生成新词时看到未来**。解码器可以捕捉目标序列内部和输入序列与目标序列之间的依赖关系，进一步增强

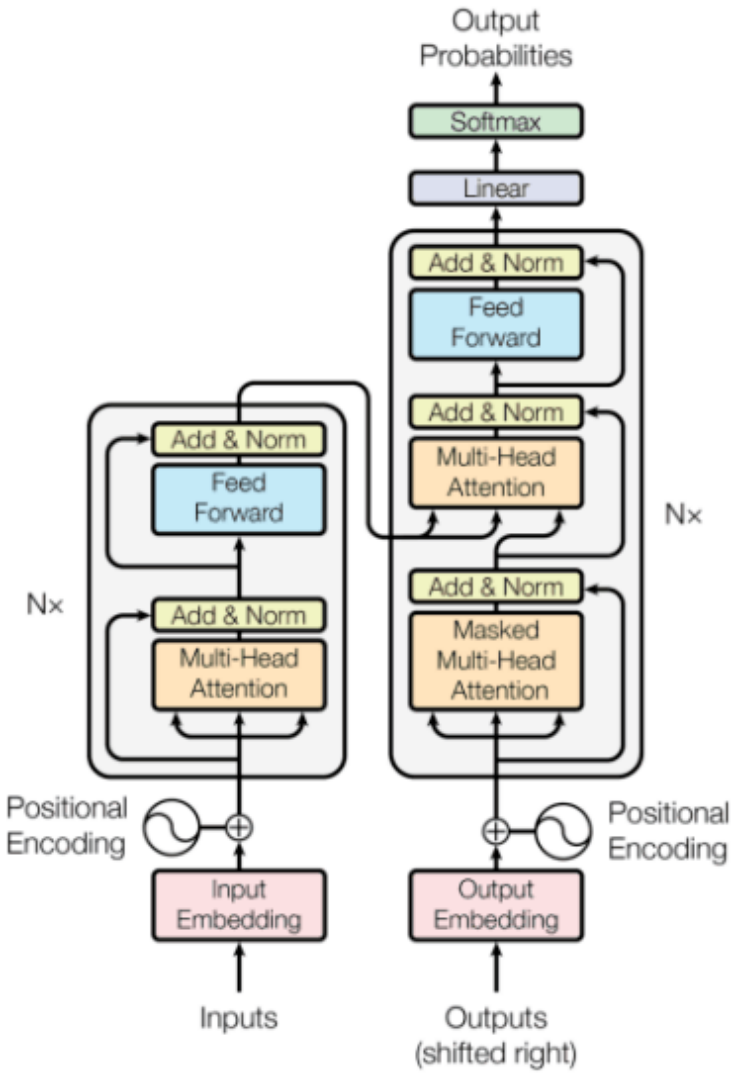

图 30. Transformer 网络全貌

模型的表达能力。接下来的步骤和编码器类似，最终的输出将用于预测目标序列。总而言之，解码器接收属于自己的输入，并结合编码器的输出（上下文向量）来生成目标序列预测值。

4. 输出层

解码器完成了所有层的处理后，将得到一个表示目标序列的向量，这个向量代表了 transformer 模型在序列中学习到的全部特征表示。接下来经过一个线性层和一个 softmax 层，将结果输出为概率，方便我们选择最有可能是结果的词。

关于 Transformer，大家可以参见黄佳（2023）《层峦叠翠上青天：搭建 GPT 核心组件——Transformer》。

如图 31 所示的 BERT 网络，便是利用了 Transformer。BERT 模型是 Google 团队于 2019 年提出的，其本质是一种预训练的深度学习模型。它基于 Transformer 架构，整个网络可以看作是多层自注意力的叠加。

BERT 模型具有两个重要的性质，其一是仅使用 Transformer 中的编码器架构，是

一个掩码语言模型；其二是具有双向性，编码器的双向结构使得 BERT 能够充分利用上下文信息，每个位置的向量表示都通过上下文信息一起来学习。综上所述，BERT 模型更加适合面向理解的情感分析任务。

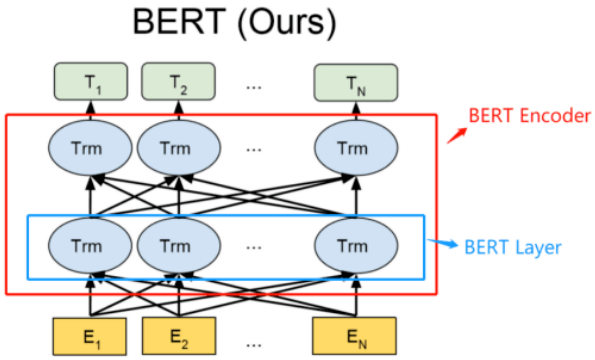

图 31. BERT 网络

7.7.4 自然语言处理与情感分析

情感分析是一种自然语言处理技术，旨在识别和理解文本中的情感和情绪倾向。通过使用机器学习和自然语言处理技术，情感分析可以自动分析文本，并将其归类为积极、消极或中性情感。这项技术在社交媒体挖掘、产品评论分析和舆情监控等领域得到广泛应用，帮助企业了解公众对其产品和服务的看法。本部分阐述目前深度学习领域中情感分析的研究进展与技术方法。

丁锋（2022）团队提出了 BiLSTM - CRF 模型并将其应用到基于方面的情感分析任务中。利用双向长短期记忆网络可以捕获长距离双向语义依赖关系并且能学习文本语义信息的特点，预测句子级别的全局最佳标签序列 [3]。Hu D. (2016) 等人提出了上下文推理网络进行对话情绪识别，从认知角度充分理解会话上下文。其还设计了多轮推理模块来提取和整合情绪线索。Luo W. (2022) 使用 Word2Vec 对语料库进行训练，得到文本词向量表示后使用引入注意力机制的 LSTM 模型进行文本特征提取，结合交叉熵训练模型并将模型应用到旅游问题文本分类方法中。RNN 只能进行顺序运算，不能并行运算。这一缺点使其在具有更深和更多参数的问题趋势中逐渐占据下风。

Yu S. (2019) 等人提出了一种基于 BERT 的文本分类模型 BERT4TC。该模型通过构造辅助句将分类任务转化为二元句对，旨在解决训练数据受限问题和任务感知问题。Hu Y. (2022) 则是指出 BERT 学习一种语言表示法，可以用于对许多 NLP 任务进行微调。主要方法是增加数据，提高计算能力，并设计训练程序以获得更好的结果。

高珊（2024）对于文本分类中的 BERT 模型做出了充分的综述 [4]。Liu Y. (2019) 等人在 BERT 的基础上进一步改进，将新模型命名为 RoBERTa。它采用动态掩蔽方法，每次生成掩蔽模式，并将序列送入模型；Lan Z. (2019) 通过减少了碎片向量的长度和与所有编码器共享参数的方式实现了 BERT 的参数数量的减少，并实现了跨层参数共享，并将该模型命名为 ALBERT (A Lite BERT) 。Su H. (2022) 提出的 RoCBert (Robust

Chinese Bert) 是一种经过训练的中文 BERT，旨在解决中文字形易受对抗攻击性的问题而提出的。Dai Y. (2022) 等人分别使用标准字符级掩码、全词掩蔽以及两者的组合来训练 3 个中文 BERT 模型。

今年上半年，笔者做了一个关于微博文本情感分析的任务，使用到的便是 BERT 以及 RoBERTa 模型。自然语言处博大精深，大家可以参见吴军《数学之美》。

8 贝叶斯分类器

8.1 贝叶斯思维简单解读

笔者第一次听到贝叶斯这个名字，是在高三复习期间，市面上的一些秒杀大招辅导资料中介绍的。第二次听到贝叶斯这个名字，是在大一暑假自学宋浩老师的《概率论与数理统计》课程。第三次听到贝叶斯这个名字，便是大二上自学西瓜书的贝叶斯决策论这一章了。当时宋浩老师强调了一句话，令我至今记忆犹新：**证据并不应该直接决定你的看法，而是更新你的看法**。

贝叶斯思想在统计学习中起着举足轻重的作用，也是经典机器学习的两大流派之一，其核心思想凝聚成了一个十分优美的公式——贝叶斯定理。

$$P(A \mid B) = \frac{P(B \mid A) \cdot P(A)}{P(B)} \tag{1}$$

机器学习可以理解为通过观测数据（X）推测结果（y）。同时学习推断规则（c），使得推断结果 $\hat{y} = c(X)$ 与真实值 y 误差尽可能的小。而贝叶斯思维又是什么呢？让我们从以下四个问题出发：

8.1.1 当我们一无所知时，如何进行推断？

假设我们第一次接触硬币正反的问题，什么都不知道。因为一无所知，只能预测正面和反面的概率都是 0.5。

假设我们虽然第一次接触硬币问题，但神告诉我们这种硬币出现正面的概率 $p(1) = 0.8$ ，反面的概率是 $p(0) = 0.2$。在这种情况下，我们就会倾向于预测结果是正面，且预测错误的概率是 $1 - p(1) = p(0) = 0.2$ 。

此处的 p(1)=0.8 和 p(0)=0.2 就叫做**先验概率**（prior probability），指的是在观测前我们就已知的结果概率分布 p(y)。此处我们不需要观测硬币尺寸，就可以大胆推测硬币的正反。

8.1.2 当我们有了更多信息，该如何利用它们？

显然，前文提到的估算方法是很不准确的，因为没有考虑到硬币的属性。而且现实情况中我们往往可以观测到硬币的一些属性，而非完全一无所知。因此，我们尝试回答：“当我观测到硬币大小时，它正面朝上的概率是多少？”用数学语言表达就是估计 $p(y = 1|X)$ 。相似的，$p(y = 0|X)$ 代表当我们观测到硬币属性 X 时，它被投掷后是反面的概率。

我们给 $p(y = 1|X)$ 和 $p(y = 0|X)$ 起了个名字，叫做**后验概率**（posterior probability），指的是在观测到 X 后我们对结果 y 的估计。和 p(y=1) 不同，它不是计算 $y = 1$ 的概率，而是计算仅当 X 被观测到时 $y = 1$ 的概率。

先验概率和后验概率之间的关系是什么？简单来看，后验概率可以被看做是对先验概率的一种“更加细致的刻画和更新”，因为此时我们观测到了 X，有了额外的信息。所以后验概率比先验概率更有意义，因为引入了额外的观测信息，所以预测的准确度得到了加强。而大部分机器学习模型尝试得到的，就是后验概率。

8.1.3 如何得到后验概率？为什么要引入贝叶斯公式？

后验概率无法直接获得，因此我们需要找到方法来计算它，而解决方法就是引入贝叶斯公式。后验概率这种表达叫做条件概率（conditional probability）。

贝叶斯决策的预测值是选取后验概率最大的结果。在二分类问题中，也就是对比 $p(y=1|X)$与$p(y=0|X)$ 的结果，取最大值。

8.1.4 为什么贝叶斯决策理论是“最优的”？

因为 $p(y=0|X)=1-p(y=1|X)$，也就是“观测到 X 并预测为反面的错误概率“就是“观测到 X 并预测为正面的后验概率”。

因此只要观测到 X，并选择后验概率最大的结果，就可以最小化预测的错误概率。这个结论对所有观测值 X 均成立，因此只要按照这个预测逻辑，那么就可以保证预测的错误概率最小化，所以才叫做“最优”。

8.2 贝叶斯决策论

贝叶斯决策论 (Bayesian Decision Theory) 是在概率框架下实施决策的基本理论。其思想已经简单的在上期做了介绍，本期主要呈现核心理论知识。

给定 N 个类别，令 λij 代表将第 j 类样本误分类为第 i 类所产生的损失，则基于后验概率将样本 x 分到第 i 类的条件风险为：

$$R(c_i \mid \boldsymbol{x})=\sum_{j=1}^{N}\lambda_{ij}P(c_j \mid \boldsymbol{x}) \tag{2}$$

贝叶斯判定准则 (Bayes decision rule)：

$$h^*(\boldsymbol{x})=\arg\min_{c\in\mathcal{Y}}R(c \mid \boldsymbol{x}) \tag{3}$$

式（3）中的 h^* 称为**贝叶斯最优分类器** (Bayes optimal classifier)，其总体风险称为**贝叶斯风险** (Bayes risk)，该式反映了学习性能的理论上限。

$P(c \mid x)$ 难以在现实任务中直接获得，从这个角度来看，机器学习所要实现的是基于有限的训练样本尽可能准确地估计出后验概率。通常有两种判别方式：

（1）**判别式 (Discriminative) 模型**，其思路为直接对 $P(c \mid x)$ 建模，代表有：决策树、BP 神经网络、SVM；

（2）**生成式 (Generative) 模型**，其思路是先对联合概率分布 $P(c \mid x)$ 建模，再由此获得 $P(c \mid x)$ 。获得方法：$P(c \mid x) = \frac{P(x,c)}{P(x)}$ 。代表是贝叶斯分类器。

这里需要强调的是: **贝叶斯分类器 ! = 贝叶斯学习**

下面我们来介绍一种常用的参数估计方法——**极大似然估计**。我们先假设某种概率分布形式，再基于训练样例对参数进行估计。假定 $P(x \mid c)$ 具有确定的概率分布形式，且被参数 θ_c 唯一确定，则任务就是利用训练集 D 来估计参数 θ_c 。

θ_c 对于训练集 D 中第 c 类样本组成的集合 D_c 的似然 (Likelihood) 为：

$$P(D_c \mid \theta_c) = \prod_{x \in D_c} P(x \mid \theta_c) \tag{4}$$

连乘易造成下溢，因此通常使用对数似然 (Log-Likelihood)：

$$LL(\theta_c) = \log P(D_c \mid \theta_c) = \sum_{x \in D_c} \log P(x \mid \theta_c) \tag{5}$$

于是, θ_c 的极大似然估计为：

$$\hat{\theta}_c = \arg\max_{\theta_c} LL(\theta_c) \tag{6}$$

由此观之，**估计结果的准确性严重依赖于所假设的概率分布形式是否符合潜在的真实分布!**

8.3 朴素贝叶斯分类器

我们直接由贝叶斯公式说开去：

$$P(c \mid x) = \frac{P(c)\boxed{P(x \mid c)}}{P(x)} \tag{7}$$

方框部分的获取难度很大，其主要障碍是：所有属性上的联合概率难以从有限训练样本估计获得；组合爆炸；样本稀疏。

基本思路：假设属性相互独立。

$$P(c \mid x) = \frac{P(c)P(x \mid c)}{P(x)} = \frac{P(c)}{P(x)} \prod_{i=1}^{d} P(x_i \mid c) \tag{8}$$

d 为属性数，x_i 为 x 在第 i 个属性上的取值 $P(X)$ 对所有类别相同，于是

$$h_{nb}(x) = \arg\max_{c \in \mathcal{Y}} P(c) \prod_{i=1}^{d} P(x_i \mid c) \tag{9}$$

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

估计 $P(X|c)$:

（1）对离散属性，令 D_{c,x_i} 表示 D_c 中在第 i 个属性上取值为 x_i 的样本组成的集合，则

$$P(x_i \mid c) = \frac{|D_{c,x_i}|}{|D_c|} \tag{10}$$

(2) 对连续属性，考虑概率密度函数，假定 $p(x_i \mid c) \sim \mathcal{N}(\mu_{c,i}, \sigma_{c,i}^2)$ 则有：

$$p(x_i \mid c) = \frac{1}{\sqrt{2\pi}\sigma_{c,i}} \exp\left(-\frac{(x_i - \mu_{c,i})^2}{2\sigma_{c,i}^2}\right) \tag{11}$$

若某个属性值在训练集中没有与某个类同时出现过，则直接计算会出现问题，因为概率连乘将“抹去”其他属性提供的信息。此时需要进行**拉普拉斯修正**。

令 N 表示训练集 D 中可能的类别数，N_i 表示第 i 个属性可能的取值数：

$$\hat{P}(c) = \frac{|D_c| + 1}{|D| + N} \tag{12}$$

$$\hat{P}(x_i \mid c) = \frac{|D_{c,x_i}| + 1}{|D_c| + N_i} \tag{13}$$

此法假设了属性值与类别的均匀分布，这是额外引入的 bias(偏置项, 也称函数的截距)。