3.线性分类

3.1 任务设定

输入：x1，x2....

输出：离散空间中的y，由k个类别构成。

二分类：y={-1,+1} or y={0,1}

多分类：y={1,2...K}对应K个类别

线性可分

如果满足下述条件，即为线性可分。

$$x^T w + w_0 \begin{cases} > 0 & \text{if x} \in \text{正类} \\ < 0 & \text{if x} \in \text{负类} \end{cases}$$

广义线性分类：

- 广义线性分类

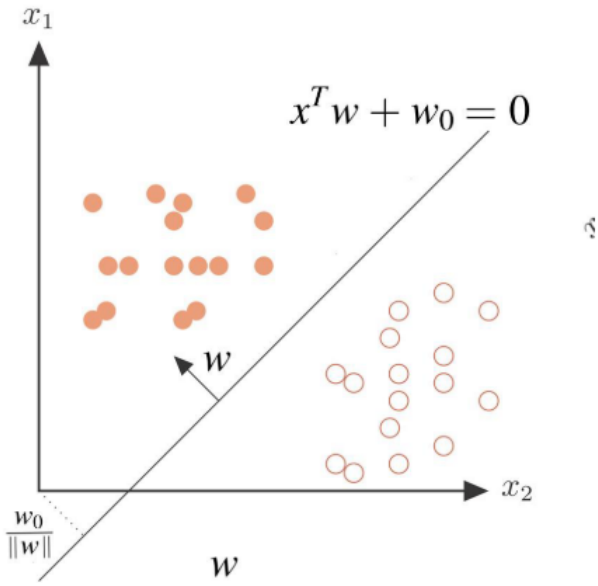
分类函数的输出需要映射到离散空间，e.g.，$\{-1,+1\}$所以需要引入一个额外的映射函数$g(\cdot)$来将线性函数的输出加工成目标输出

对于二分类问题，$g(\cdot)$可以是符号函数 *sign function:*

$$g\big(f(x)\big) = \text{sign}(x^T w + w_0)$$

其中 $w \in \mathbb{R}^d$, $w_0 \in \mathbb{R}$

$$\triangleq \begin{cases} +1 & \text{if } f(x) > 0 \\ -1 & \text{if } f(x) < 0 \end{cases}$$



可以使用最小二乘回归进行线性分类，但受到离群点的影响很大。

3.2 感知机算法

– 假设：
数据是线性可分的，即存在一个线性分类器能够将所有样本都分对

– 决策函数

$$y = f(x) = \text{sign}(x^T w)$$

– 损失函数

$\mathbb{1}$: indicate function，括号中成立为1，不成立为0

$$\mathcal{L} = -\sum_{i=1}^{n} (y_i \cdot x_i^T w)\mathbb{1}\{y_i \neq \text{sign}(x_i^T w)\}$$

注意到 $y \in \{-1, +1\}$, 所以有:

$$y_i \cdot x_i^T w \text{ 满足 } \begin{cases} > 0 \text{ if } y_i = \text{sign}(x_i^T w) \\ < 0 \text{ if } y_i \neq \text{sign}(x_i^T w) \end{cases}$$

可见最小化 $\mathcal{L}$ 对应于我们希望能够将样本全部分类正确.

难以解出损失函数导数为0的解，所以采用梯度下降法，用一个合适的步长迭代。

– 算法：

输入: 训练数据 $(x_1, y_1), \ldots, (x_n, y_n)$ 以及步长参数 η

1. 令 $w^{(1)} = \mathbf{0}$
2. **For** $t = 1, 2, \ldots$ **do**
   a) 遍历训练样本 $(x_i, y_i) \in \mathcal{D}$ 查看是否有错分即满足 $y_i \neq \text{sign}(x_i^T w^{(t)})$ 的样本
   b) 如果错分样本存在，则挑选其中的一个样本 (x_i, y_i) 并进行如下更新

$$w^{(t+1)} = w^{(t)} + \eta y_i x_i,$$

如果不存在错分样本，则返回能够分对所有训练样本的 $w^{(t)}$

限制：

1. 对于一个线性可分的训练集，感知器无法对能够正确分类的（无数多个）决策面进行质量上的区分（只能分开，不能保证分好）
2. 对于线性不可分的训练集，感知器的学习过程无法收敛

3.3 逻辑回归

https://blog.csdn.net/weixin_39445556/article/details/83930186讲的很明白

使用对数函数原因:

在线性回归中，我们用h(x)来代表我们的假设函数，这里输出就是我们想要的结果，在逻辑回归里，因为输出项只有0或者1，所以我们需要对这个函数做一下修改，使它产生的输出也是0到1。然后如果h(x)的值大于0.5，代表y为1，h(x)小于0.5，代表y为0。

$$h_w(x) = g(w^T X) \quad g(z) = \frac{1}{1+e^{-z}}$$

损失函数:

$$cost(h_w(x), y) = \begin{cases} -\ln(h_w(x)) & y = 1 \\ -\ln(1 - h_w(x)) & y = 0 \end{cases}$$

这样定义的原因：如果采用线性函数的定义方法，这将是一个非凸的函数，不能保证每次都能找到全局最优解，所以采用了这样略显奇怪的定义。

总损失函数:

$$J(w) = \frac{1}{m}\sum_{i=1}^{m} cost(y_i, h_w(x_i))$$

$$= -\frac{1}{m}[\sum_{i=1}^{m} y_i \ln h_w(x_i) + (1-y_i)\ln(1-h_w(x_i))]$$

$$\min_w J(w), h_w(x) = \frac{1}{1+e^{-w^T x}}$$

使用梯度下降法求解w，跟线性回归一样，只不过把线性函数换成h函数，推导过程如下：（把θ换成w看呜呜呜呜）

$$y^{(i)} \log\left(h_\theta\left(x^{(i)}\right)\right) + \left(1-y^{(i)}\right)\log\left(1-h_\theta\left(x^{(i)}\right)\right)$$
$$= y^{(i)} \log\left(\frac{1}{1+e^{-\theta^T x^{(i)}}}\right) + \left(1-y^{(i)}\right)\log\left(1-\frac{1}{1+e^{-\theta^T x^{(i)}}}\right)$$
$$= -y^{(i)} \log\left(1+e^{-\theta^T x^{(i)}}\right) - \left(1-y^{(i)}\right)\log\left(1+e^{\theta^T x^{(i)}}\right)$$

所以:

$$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{\partial}{\partial \theta_j}[-\frac{1}{m}\sum_{i=1}^{m}[-y^{(i)} \log\left(1+e^{-\theta^T x^{(i)}}\right) - \left(1-y^{(i)}\right)\log\left(1+e^{\theta^T x^{(i)}}\right)]]$$
$$= -\frac{1}{m}\sum_{i=1}^{m}[-y^{(i)}\frac{-x_j^{(i)}e^{-\theta^T x^{(i)}}}{1+e^{-\theta^T x^{(i)}}} - \left(1-y^{(i)}\right)\frac{x_j^{(i)}e^{\theta^T x^{(i)}}}{1+e^{\theta^T x^{(i)}}}]$$
$$= -\frac{1}{m}\sum_{i=1}^{m} y^{(i)}\frac{x_j^{(i)}}{1+e^{\theta^T x^{(i)}}} - \left(1-y^{(i)}\right)\frac{x_j^{(i)}e^{\theta^T x^{(i)}}}{1+e^{\theta^T x^{(i)}}}]$$
$$= -\frac{1}{m}\sum_{i=1}^{m}\frac{y^{(i)}x_j^{(i)} - x_j^{(i)}e^{\theta^T x^{(i)}} + y^{(i)}x_j^{(i)}e^{\theta^T x^{(i)}}}{1+e^{\theta^T x^{(i)}}}$$
$$= -\frac{1}{m}\sum_{i=1}^{m}\frac{y^{(i)}\left(1+e^{\theta^T x^{(i)}}\right) - e^{\theta^T x^{(i)}}}{1+e^{\theta^T x^{(i)}}}x_j^{(i)}$$
$$= -\frac{1}{m}\sum_{i=1}^{m}(y^{(i)} - \frac{e^{\theta^T x^{(i)}}}{1+e^{\theta^T x^{(i)}}})x_j^{(i)}$$
$$= -\frac{1}{m}\sum_{i=1}^{m}(y^{(i)} - \frac{1}{1+e^{-\theta^T x^{(i)}}})x_j^{(i)}$$
$$= -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)} - h_\theta\left(x^{(i)}\right)]x_j^{(i)}$$
$$= \frac{1}{m}\sum_{i=1}^{m}[h_\theta\left(x^{(i)}\right) - y^{(i)}]x_j^{(i)}$$

注：虽然得到的梯度下降算法表面上看上去与线性回归的梯度下降算法一样，但是这里的$h_\theta(x) = g\left(\theta^T X\right)$与线性回归中不同，所以实际上是不一样的。另外，在运行梯度下降算法之前，进行特征缩放依旧是非常必要的，下降收敛更快。

引入正则化：添加惩罚项，λ参数值越大，惩罚越厉害，模型越欠拟合。当模型参数w过多时，损失函数会很大，要减少w函数使损失函数最小。

$$L1\text{正则化}：J_{L1}(w)=J(w)+\frac{\lambda}{2m}\sum_{j=1}^{n}|w_j|$$

$$L2\text{正则化}：J_{L2}(w)=J(w)+\frac{\lambda}{2m}\sum_{j=1}^{n}w_j^2$$

下面是用概率模型解释的，使发生的概率最大

将对数几率同线性函数对应起来

$$\ln\frac{P(y=+1|x)}{P(y=-1|x)}=x^Tw+w_0 \quad p(y=-1|x)=1-p(y=1|x)$$

$$p(y=1|x)=\frac{exp\{x^Tw+w_0\}}{1+exp\{x^Tw+w_0\}}=\sigma(X^Tw+w_0)$$

我们可以得到观察变量 $y_1,\ldots,y_n$ 的联合似然为（记 $\sigma_i(w)=\sigma(x_i^Tw)$）

$$\begin{aligned}p(y_1,\ldots,y_n|x_1,\ldots,x_n,w) &= \prod_{i=1}^{n}p(y_i|x_i,w)\\ &= \prod_{i=1}^{n}\sigma_i(w)^{\mathbb{1}(y_i=+1)}\left(1-\sigma_i(w)\right)^{\mathbb{1}(y_i=-1)}\end{aligned}$$

我们可以推导出:

$$\underbrace{\frac{e^{y_ix_i^Tw}}{1+e^{y_ix_i^Tw}}}_{\sigma_i(y_i\cdot w)}=\left(\underbrace{\frac{e^{x_i^Tw}}{1+e^{x_i^Tw}}}_{\sigma_i(w)}\right)^{\mathbb{1}(y_i=+1)}\left(\underbrace{1-\frac{e^{x_i^Tw}}{1+e^{x_i^Tw}}}_{1-\sigma_i(w)}\right)^{\mathbb{1}(y_i=-1)}$$

这能够令我们将似然公式写的更漂亮：

$$p(y_1,\ldots,y_n|x_1,\ldots,x_n,w)=\prod_{i=1}^{n}\sigma_i(y_i\cdot w)$$

记最大似然 *maximum likelihood* 下w的解为

$$
\begin{aligned}
w_{\mathrm{ML}} &= \arg\max_{w} \sum_{i=1}^{n} \ln \sigma_i(y_i \cdot w) \\
&= \arg\max_{w} \mathcal{L}
\end{aligned}
$$

$$
\nabla_w \mathcal{L} = \sum_{i=1}^{n} \left(1 - \sigma_i(y_i \cdot w)\right) y_i x_i
$$

输入: 训练集$(x_1, y_i), \ldots, (x_n, y_n)$和步长$\eta > 0$

1. 令$w_{(1)} = \mathbf{0}$
2. **For step** $t = 1, 2, \ldots$ **do**

   更新 $w^{(t+1)} = w^{(t)} + \eta \sum_{i=1}^{n} \left(1 - \sigma_i(y_i \cdot w)\right) y_i x_i$

3.4 多分类

三种拆分策略：一对一 一对其余 多对多

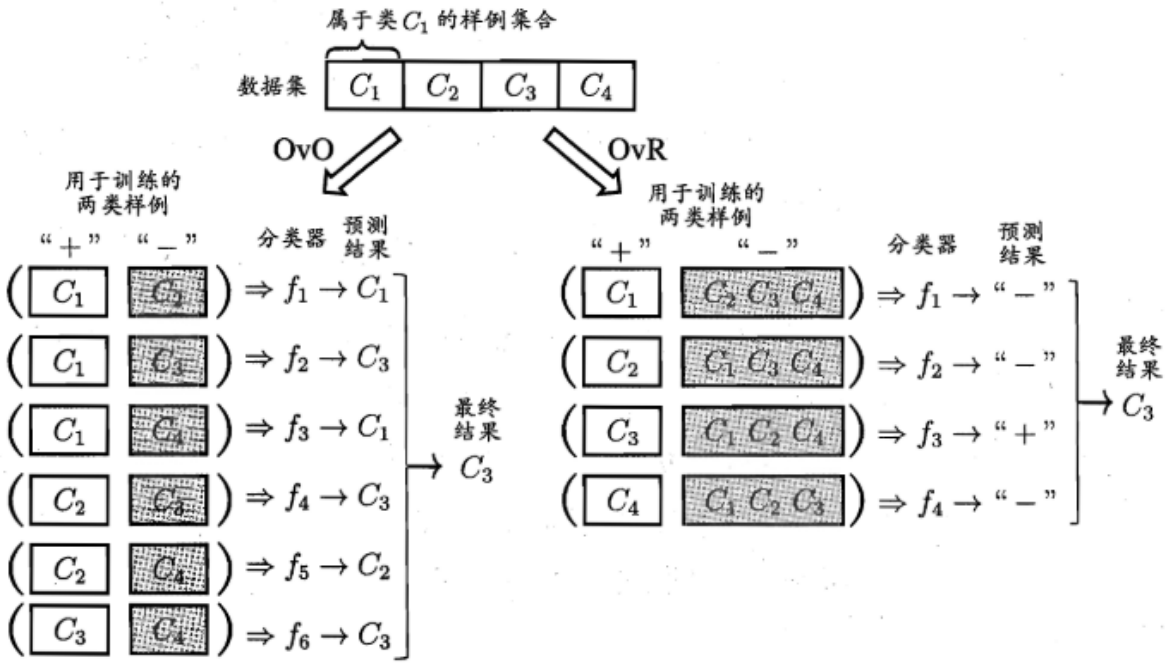


图 3.4 OvO 与 OvR 示意图

分类器的预测置信度, 选择置信度最大的类别标记作为分类结果.

容易看出, OvR 只需训练 N 个分类器, 而 OvO 需训练 $N(N-1)/2$ 个分类器, 因此, OvO的存储开销和测试时间开销通常比 OvR 更大. 但在训练时, OvR 的每个分类器均使用全部训练样例, 而 OvO 的每个分类器仅用到两个类的样例, 因此, 在类别很多时, OvO 的训练时间开销通常比 OvR 更小. 至于预测性能, 则取决于具体的数据分布, 在多数情形下两者差不多.

MvM 是每次将若干个类作为正类, 若干个其他类作为反类. 显然, OvO 和 OvR 是 MvM 的特例. MvM 的正、反类构造必须有特殊的设计, 不能随意选取. 这里我们介绍一种最常用的 MvM 技术: “纠错输出码” (Error Correcting Output Codes, 简称 ECOC).

ECOC [Dietterich and Bakiri, 1995] 是将编码的思想引入类别拆分, 并尽可能在解码过程中具有容错性. ECOC 工作过程主要分为两步:

- 编码: 对 N 个类别做 M 次划分, 每次划分将一部分类别划为正类, 一部分划为反类, 从而形成一个二分类训练集; 这样一共产生 M 个训练集, 可训练出 M 个分类器.
- 解码: M 个分类器分别对测试样本进行预测, 这些预测标记组成一个编码. 将这个预测编码与每个类别各自的编码进行比较, 返回其中距离最小的类别作为最终预测结果.