问题

机器学习中常用的损失函数总结

前言

我们经常听到**损失函数**、**代价函数**和**目标函数**这三种说法，这三种说法有什么联系和区别呢？这里明确下：

损失函数 Loss Function 通常是**针对单个训练样本而言的**，给定一个模型输出 $\hat{y}$ 和一个真实值 y ，损失函数输出一个实值损失 $L = f(y_i, \hat{y_i})$

代价函数 Cost Function 通常是**针对整个训练集**（或者是在使用 mini-batch gradient descent 时的一个 mini -batch）的总损失 $J = \sum_{i=1}^{N} f(y_i, \hat{y_i})$

目标函数 Objective Function 是**一个更通用的术语**，表示任意希望被优化的函数。

一句话总结三者的关系就是： A loss function is a part of a cost function which is a type of an objective function.

由于损失函数和代价函数只是在针对样本集上有区别，因此本文中统一使用损失函数这个术语。

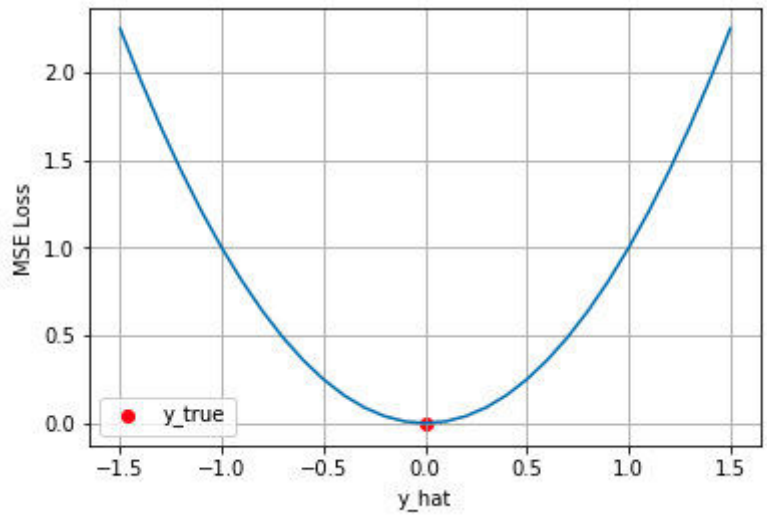
下面首先介绍几种适用于回归问题的损失函数

均方差损失 (MSE)

Mean Squared Error (MSE，得记住这个缩写，很多地方都直接用缩写的) 损失是回归任务中最常用的一种损失函数了，也叫 L2 Loss。其基本形式如下：

$$J_{MSE} = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y_i})^2 \tag{1}$$

当预测 $\hat{y_i}$ 等于真实值 y_i 时，误差最小为 0 ，不等时误差最大为无穷大。下图当真实值 y 为 0 时，随着预测值和真实值绝对误差 $|y - \hat{y}|$ 的增加，均方差损失呈二次方地增加。



其背后的假设

我们设真实值与预测值之间的误差为：

$$\varepsilon_i = y_i - \hat{y}_i \tag{2}$$

我们通常认为误差 ε 服从**标准正态分布**$(\mu = 0, \sigma^2 = 1)$，即给定一个 x_i，模型输出真实值为 y_i 的概率为：

$$p(y_i|x_i) = \frac{1}{\sqrt{2\pi}} * exp(-\frac{\varepsilon_i^2}{2}) \tag{3}$$

进一步我们假设数据集中N个样本点之间相互独立，则给定所有 x 输出所有真实值 y 的概率即似然Likeihood，为所有 $p(y_i|x_i)$ 的累乘：

$$L(x, y) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}} * exp[-\frac{\varepsilon_i^2}{2}] \tag{4}$$

取对数似然函数得：

$$log[L(x, y)] = -\frac{n}{2} log 2\pi - \frac{1}{2} \sum_{i=1}^{n} \varepsilon_i^2 \tag{5}$$

去掉与 $\hat{y}_i$ 无关的第一项，然后转化为最小化负对数似然：

$$neg\_log[L(x, y)] = \frac{1}{2} \sum_{i=1}^{n} \varepsilon_i^2 = \frac{1}{2} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{6}$$

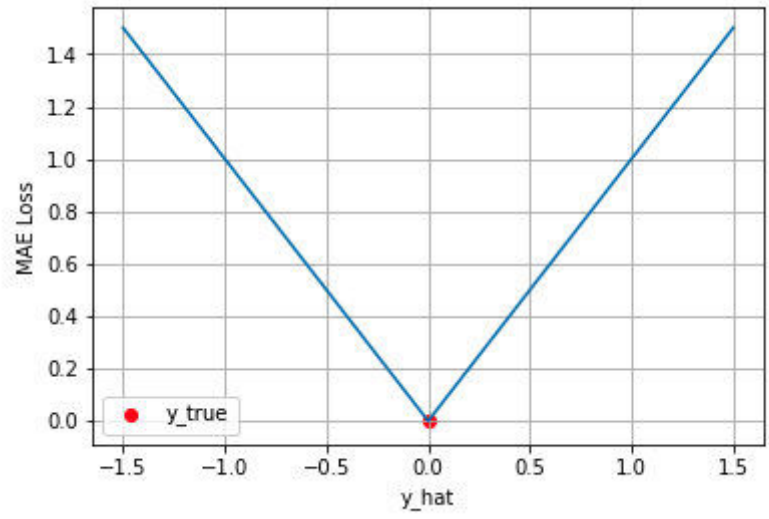
可以看到这个实际上就是均方差损失的形式。也就是说**在模型输出与真实值的误差服从高斯分布的假设下，最小化均方差损失函数与极大似然估计本质上是一致的**，因此在这个假设能被满足的场景中（比如回归），均方差损失是一个很好的损失函数选择；当这个假设没能被满足的场景中（比如分类），均方差损失不是一个好的选择，这也解释了为什么在分类问题中不使用均方误差作为损失函数而是使用交叉熵的问题。

平均绝对误差损失 (MAE)

Mean Absolute Error 也称为 L1 Loss。基本形式如下：

$$J_{MAE} = \frac{1}{N} \sum_{i=1}^{N} |y_i - \hat{y}_i| \tag{7}$$

当真实值 y 为 0 时，该损失函数可视化如下，横轴为预测值 $\hat{y}$，纵坐标为损失值。随着预测值与真实值绝对误差 $|y - \hat{y}|$ 的增加，MAE呈线性增长。



其背后的假设

与 MSE 类似，我们同样可以在一定的假设下通过最大化似然得到 MAE 的形式，假设模型预测值与真实值之间的误差服从**拉普拉斯分布** $(\mu = 0, \sigma = 1)$，拉普拉斯分布形式如下：

$$f(x) = \frac{1}{2\sigma} exp(-\frac{|x-\mu|}{\sigma}) \tag{8}$$

因此给定一个 x_i ，模型输出真实值 y_i 的概率为：

$$p(y_i|x_i) = \frac{1}{2} exp(-|y_i - \hat{y_i}|) \tag{9}$$

进一步我们假设数据集中 N 个样本点之间相互独立，则给定所有所有 x ，输出所有真实值 y 的概率，即似然 Likelihood 为所有 $p(y_i|x_i)$ 的累乘：

$$L(x,\ y) = \prod_{i=1}^{N} \frac{1}{2} exp(-|y_i - \hat{y_i}|) \tag{10}$$

为了计算方便，我们通常最大化对数似然 Log-Likelihood ：

$$LL(x, y) = log(L(x, y)) = -N * log(2) - \sum_{i=1}^{N} |y_i - \hat{y_i}| \tag{11}$$

去掉与 y_i 无关的第一项，然后转化为最小化负对数似然 Negative Log-Likelihood 得

$$NLL(x, y) = \sum_{i=1}^{N} |y_i - \hat{y_i}| \tag{12}$$

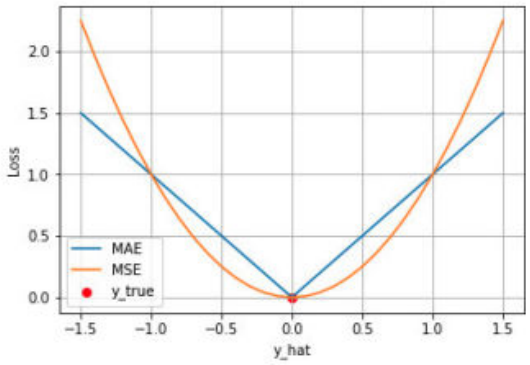
这样便得到了平均绝对值误差的形式。

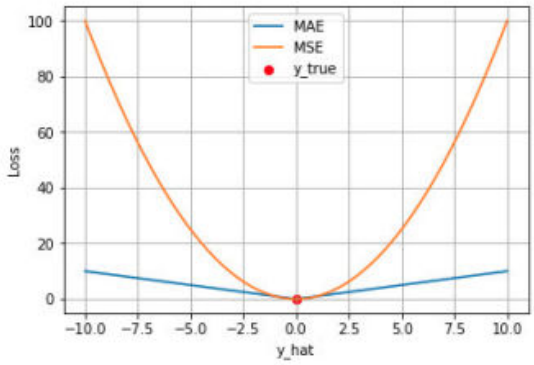
MSE 与 MAE 的区别

1. MSE 相比 MAE 通常可以更快地收敛
2. MAE 相比 MSE 对于离群点更加健壮，包容性更强，即更加不易受到离群点 outlier 的影响。

MSE 相比 MAE 通常可以更快地收敛：当使用梯度下降算法时，MSE 损失的梯度为 $-(y_i - \hat{y_i})$ ，而 MAE 损失的梯度为正负 1，即 MSE 的梯度的 scale 会随误差大小变化，而 MAE 的梯度的 scale 则一直保持为 1，即便在绝对误差 $|y_i - \hat{y_i}|$ 很小的时候 MAE 的梯度 scale 也同样为 1，这实际上是非常不利于模型的训练的。当然你可以通过在训练过程中动态调整学习率缓解这个问题，但是总的来说，损失函数梯度之间的差异导致了 MSE 在大部分时候比 MAE 收敛地更快。**这个也就是回归问题中一般采用 MSE 而不是 MAE 的原因。**

MAE 相比 MSE 对于离群点更加健壮：这点可以这样直观地理解，下图是 MAE 和 MSE 损失画到同一张图里面，由于MAE 损失与绝对误差之间是线性关系，MSE 损失与误差是平方关系，当误差非常大的时候，MSE 损失会远远大于 MAE 损失。因此当数据中出现一个误差非常大的 outlier 时，MSE 会产生一个非常大的损失，对模型的训练会产生较大的影响。





Huber Loss

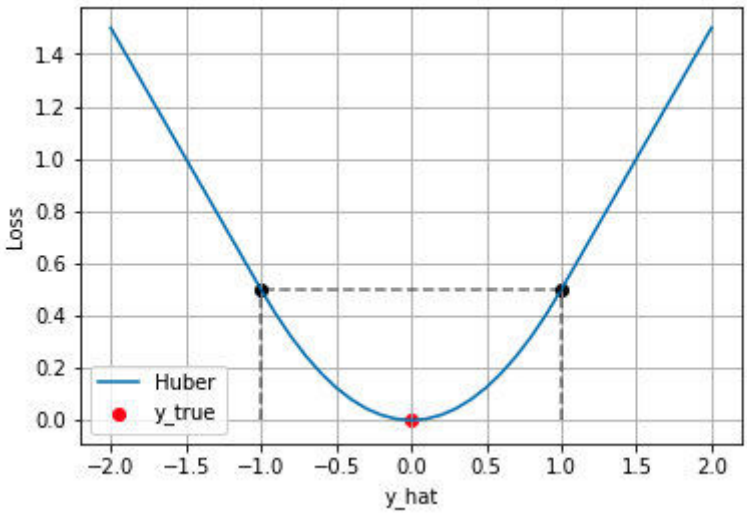
上文我们分别介绍了 MSE 和 MAE 损失以及各自的优缺点，MSE 损失收敛快但容易受 outlier 影响，MAE 对 outlier 更加健壮但是收敛慢，Huber Loss则是一种将 MSE 与 MAE 结合起来，取两者优点的损失函数，也被称作 Smooth Mean Absolute Error Loss 。其原理很简单，就是在误差接近 0 时使用 MSE，误差较大时使用 MAE，公式为：

$$J_{huber} = \begin{cases} \frac{1}{N}\sum_{i=1}^{N}\frac{(y_i-\hat{y}_i)^2}{2} & , \quad |y_i-\hat{y}_i| \le \delta \\ \frac{1}{N}\sum_{i=1}^{N}(\delta|y_i-\hat{y}_i| - \frac{1}{2}\delta^2) & , \quad |y_i-\hat{y}_i| > \delta \end{cases} \tag{13}$$

注意：目标检测中的坐标回归损失函数 Smooth L1 是 Huber Loss 的超参数 $\delta = 1$ 时的情况。

上式中，δ 是 Huber Loss 的一个超参数，δ 的值是 MSE与 MAE 两个损失函数连接的位置。分段函数中上部分是 MSE 项，下部分是 MAE 项。MAE 项的公式为 $\delta|y_i-\hat{y}_i| - \frac{1}{2}\delta^2$ ，是为了保证误差 $|y-\hat{y}| = \pm\delta$ 时 MAE 和 MSE 的取值一致，进而保证 Huber Loss 损失连续可导。

下图是 $\delta = 1$ 时的 Huber Loss，可以看到在 $[-\delta, \delta]$ 的区间内实际上就是 MSE 损失，在 $(-\infty, \delta)$ 和 $(\delta, +\infty)$ 区间内为 MAE 损失。

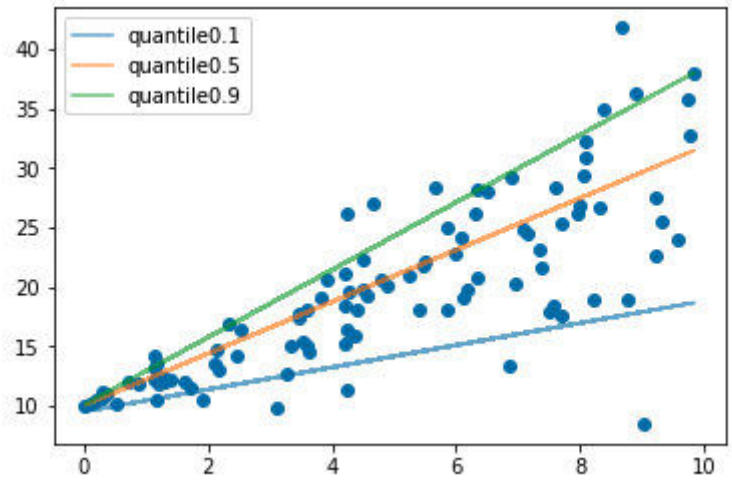


特点

Huber Loss 结合了 MSE 和 MAE 损失，在误差接近 0 时使用 MSE，使损失函数可导并且梯度更加稳定；在误差较大时使用 MAE 可以降低 outlier 的影响，使训练对 outlier 更加健壮。缺点是需要额外地设置一个超参数 δ 。

分位数损失 Quantile Loss

分位数回归是一类在实际应用中非常有用的回归算法，通常的回归算法是拟合目标值的期望或者中位数，而分位数回归可以通过给定不同的分位点，拟合目标值的不同分位数。例如我们可以分别拟合出多个分位点，得到一个置信区间，如下图所示：

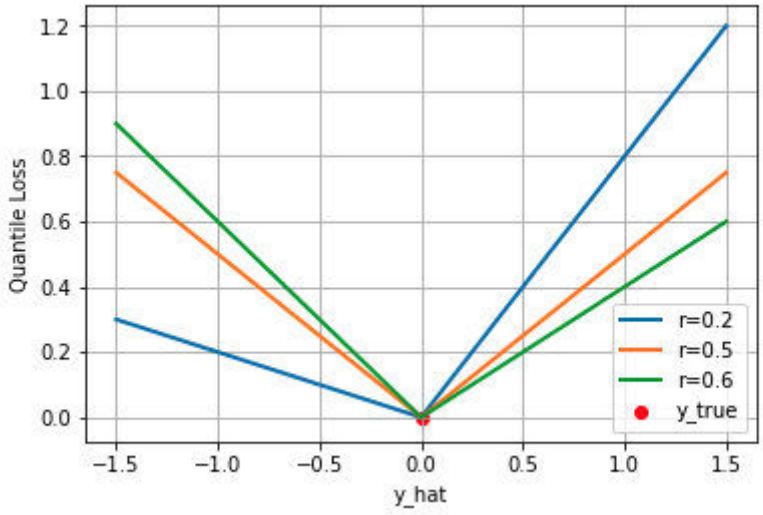


分位数回归是通过使用分位数损失 Quantile Loss 来实现这一点的，分位数损失形式如下，式中的 r 分位数系数。

$$J_{quant} = \begin{cases} \frac{1}{N}\sum_{i=1}^{N} r|y_i - \hat{y}_i| & , \quad \hat{y}_i < y_i \\ \frac{1}{N}\sum_{i=1}^{N} (1-r)|y_i - \hat{y}_i| & , \quad \hat{y}_i \geq y_i \end{cases} \tag{14}$$

我们如何理解这个损失函数呢？这个损失函数是一个分段的函数，将 $\hat{y}_i < y_i$（低估）和 $\hat{y}_i \geq y_i$（高估）两种情况分开来，并分别给予不同的系数。当 $r > 0.5$ 时，低估的损失要比高估的损失更大，反过来当 $r < 0.5$ 时，高估的损失比低估的损失大；分位数损失实现了**分别用不同的系数控制高估和低估的损失，进而实现分位数回归**。特别的，当 $r = 0.5$ 时，分位数损失退化为 MAE 损失，从这里可以看出 MAE 损失实际上是分位数损失的一个特例 — 中位数回归（这也可以解释为什么 MAE 损失对 outlier 更鲁棒：MSE 回归期望值，MAE 回归中位数，通常 outlier 对中位数的影响比对期望值的影响小）。

下图是取不同的分位点 0.2、0.5、0.6 得到的三个不同的分位损失函数的可视化，可以看到 0.2 和 0.6 在高估和低估两种情况下损失是不同的，而 0.5 实际上就是 MAE。



上面介绍的都是适用于回归问题的损失函数，对于分类问题，最常用的损失函数是交叉熵损失函数。

交叉熵损失 (Cross Entropy Loss)

二分类

首先考虑二分类，在在二分类中我们通常使用 Sigmoid 函数将模型的输出压缩到 (0, 1) 区间内 $\hat{y_i} \in (0,1)$ ，用来代表给定输入 x_i ，模型判断为正类的概率。由于只有正负两类，因此同时也得到了负类的概率。

$$
\begin{aligned}
p(y_i = 1|x_i) &= \hat{y_i} \\
p(y_i = 0|x_i) &= 1 - \hat{y_i}
\end{aligned} \tag{15}
$$

将两条式子合并成一条得

$$
p(y_i|x_i) = (\hat{y_i})^{y_i} * (1 - \hat{y_i})^{1-y_i} \tag{16}
$$

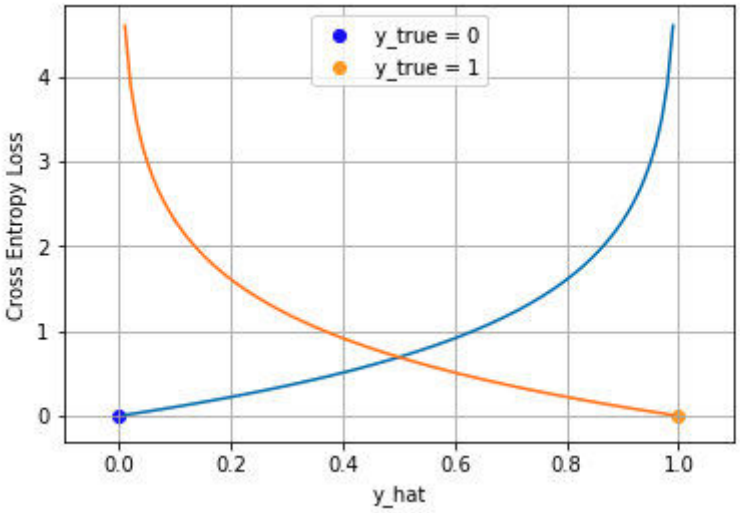
假设数据点之间独立同分布，则似然可以表示为：

$$
L(x, y) = \prod_{i=1}^{N} (\hat{y_i})^{y_i} * (1 - \hat{y_i})^{1-y_i} \tag{17}
$$

对以上似然取对数，然后加负号变成最小化负对数似然，即为交叉熵损失函数的形式：

$$
NLL(x, y) = J_{CE} = -\sum_{i=1}^{N} [y_i log(\hat{y_i}) + (1 - y_i) log(1 - \hat{y_i})] \tag{18}
$$

下图是对二分类的交叉熵损失函数的可视化，蓝线是目标值为 0 时输出不同输出的损失，黄线是目标值为 1 时的损失。可以看到约接近目标值损失越小，随着误差变大，损失呈指数增长。



多分类

在多分类的任务中，交叉熵损失函数的推导思路和二分类是一样的，变化的地方是真实值 y_i 现在是一个 One-hot 向量，同时模型输出的压缩由原来的 Sigmoid 函数换成 Softmax 函数。Softmax 函数将每个维度的输出范围都限定在 [0, 1] 之间，同时所有维度的输出和为 1，用于表示一个概率分布。

$$
p(y_i|x_i) = \prod_{k=1}^{K} (\hat{y_i^k})^{y_i^k} \tag{19}
$$

其中 $k \in K$ 表示 K 个类别中的一类，同样的假设数据点之间独立同分布，可得到负对数似然为：

$$
NLL(x, y) = J_{CE} = -\sum_{i=1}^{N} \sum_{k=1}^{K} y_i^k log(\hat{y_i^k}) \tag{20}
$$

由于 y_i 是一个 one-hot 向量，除了目标类为 1 之外其他类别上的输出都为 0，因此上式也可以写为：

$$J_{CE} = -\sum_{i=1}^{N} y_i^{c_i} log(\hat{y_i^{c_i}}) \tag{21}$$

其中 c_i 为样本 x_i 的目标类， 通常这个应用于多分类的交叉熵损失函数也被称为 Softmax Loss 或者 Categorical Cross Entropy Loss。

分类中为什么不用均方差损失？

分类中为什么不用均方差损失？**上文在介绍均方差损失的时候讲到实际上均方差损失假设了误差服从高斯分布，在分类任务下这个假设没办法被满足，因此效果会很差。**为什么是交叉熵损失呢？有两个角度可以解释这个事情，一个角度从最大似然的角度，也就是我们上面的推导；另一个角度是可以用信息论来解释交叉熵损失：

假设对于样本 x_i 存在一个最优分布 y_i^* 真实地表明了这个样本属于各个类别的概率，那么我们希望模型的输出 $\hat{y_i}$ 尽可能地逼近这个最优分布，在信息论中，我们可以使用 KL 散度来衡量两个分布的相似性。给定分布 p 和分布 q ，两者的 KL 散度公式如下：

$$KL(p,q) = \sum_{k=1}^{K} p^k log(p^k) - \sum_{k=1}^{K} p^k log(q^k) \tag{22}$$

其中的第一项为分布 p 的**信息熵**，第二项为分布 p 和分布 q 的**交叉熵**。将最优分布 y_i^* 和输出分布 $\hat{y_i}$ 代入 p 和 q 得到：

$$KL(y_i^*, \hat{y_i}) = \sum_{k=1}^{K} {y_i^*}^k log({y_i^*}^k) - \sum_{k=1}^{K} {y_i^*}^k log(\hat{y_i^k}) \tag{23}$$

由于我们希望两个分布尽量相近，因此我们最小化 KL 散度。同时由于上式第一项信息熵仅与最优分布本身相关，因此我们在最小化的过程中可以忽略掉，变成最小化

$$-\sum_{k=1}^{K} {y_i^*}^k log(\hat{y_i^k}) \tag{24}$$

我们并不知道最优分布 y_i^* ，但训练数据里面的目标值 y_i 可以看做是 y_i^* 的一个近似分布：

$$-\sum_{k=1}^{K} y_i^k log(\hat{y_i^k}) \tag{25}$$

这个是针对单个训练样本的损失函数，如果考虑整个数据集，则：

$$J_{CE} = -\sum_{i=1}^{N}\sum_{k=1}^{K} y_i^k log(\hat{y_i^k}) = -\sum_{i=1}^{N} y_i^{c_i} log(\hat{y_i^{c_i}}) \tag{26}$$

可以看到**通过最小化交叉熵的角度推导出来的结果和使用最大化似然得到的结果是一致的**。

合页损失 (Hinge Loss)

合页损失是另外一种二分类损失函数，适用于 maximum-margin 即最大间隔的分类，支持向量机 Support Vector Machine (SVM) 模型的损失函数本质上就是 Hinge Loss + L2 正则化。合页损失的公式如下：

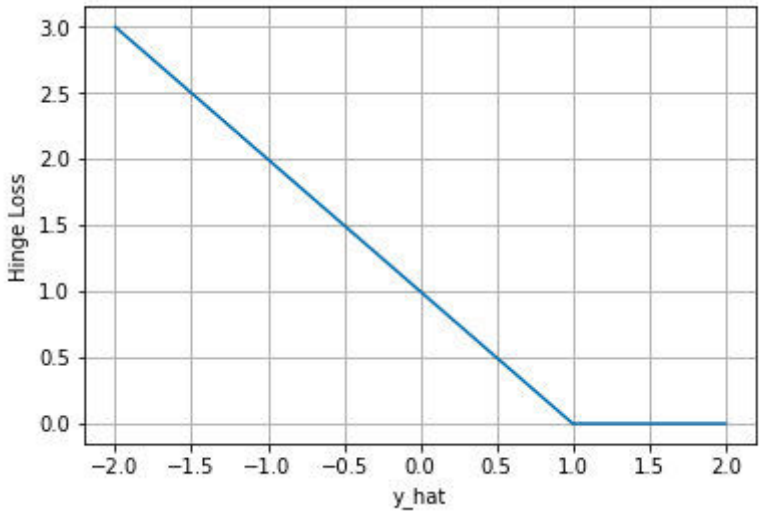
$$L(y_i) = max(0, 1 - y\hat{y}) \tag{27}$$

即如果 $y\hat{y} < 1$ ，损失为： $1 - y\hat{y}$
如果 $y\hat{y} \geq 1$ ，，则损失为： 0

以上是针对一个样本而言，针对整个数据集时合页损失为：

$$J_{hinge} = \sum_{i=1}^{N} max(0, 1 - y_i \hat{y}_i) \tag{28}$$

下图是 y 为正类时，不同输出的合页损失示意图：



可以看到当 y 为正类时，模型输出负值会有较大的惩罚，当模型输出为正值且在 (0, 1) 区间时还会有一个较小的惩罚。即合页损失不仅惩罚预测错的，并且对于预测对了但是置信度不高的也会给一个惩罚，只有置信度高的才会有零损失。使用合页损失直觉上理解是要**找到一个决策边界，使得所有数据点被这个边界正确地、高置信地被分类**。

0-1 损失

在二分类问题中，可以使用判别式的正负号来判断类别归属，判别式本身的绝对值大小并不是很重要。0-1损失函数比较的是预测值 $\hat{y}_i$ 与真实值 y_i 的符号是否相同，其具体形式如下：

$$J_{01} = \begin{cases} 0 & ,if \quad y_i \hat{y}_i \geq 0 \\ 1 & ,if \quad y_i \hat{y}_i < 0 \end{cases} \tag{29}$$

以上的函数等价与下述的函数：

$$\frac{1}{2}(1 - sign(y_i \hat{y}_i)) \tag{30}$$

其中 sign(x) 为符号函数，当 $x \geq 0$ 时，$sign(x) = 1$ ，当 $x < 0$ 时，$sign(x) = -1$

将 0-1 损失函数和上述的合页损失函数画在一起：

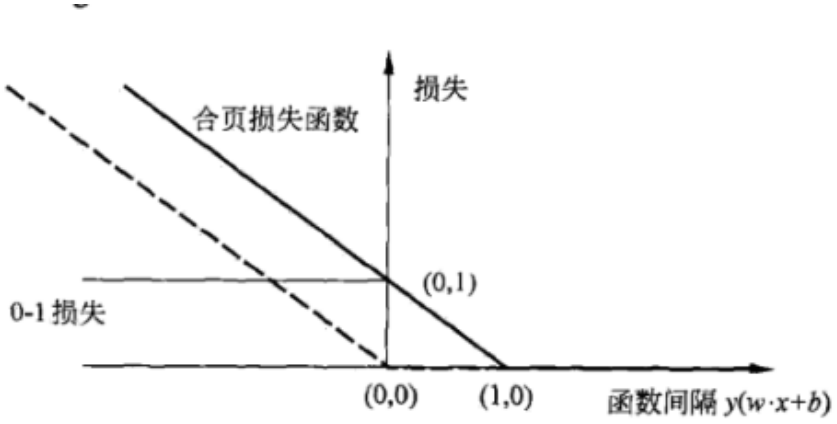


图 7.6　合页损失函数

由于 0-1 损失函数不是连续可导的，直接优化由其构成的目标函数比较困难，所以在支持向量机中是优化由 0-1 损失函数的上界（合页损失函数）构成的目标函数。

这里再啰嗦一下：图中的虚线表示的是感知机的损失函数 $[-y_i(w*x_i+b)]_+$ ，其中下标 $+$ 表示的是当中括号中的表达式的值大于 0 时，整体值就是表达式的值 ，而当中括号中的表达式的值小于等于 0 时，整体值为 0。

这时，当样本点 (x_i,y_i) 被正确分类时，损失为 0，否则损失为 $-y_i(w*x_i+b)$ ，相比之下，合页损失函数不仅要求要分类正确，而且确信度足够高时损失才是0，也就是说合页损失函数比感知机的损失函数以及 0-1 损失的要求都高。

指数损失

运用指数损失的典型分类器是 AdaBoost 算法。AdaBoost 可以看作是前向分步加法算法的特例，是由基本分类器组成的加法模型，损失函数为指数函数。

指数函数形式：

$$L(y,f(x))=exp(-yf(x)) \tag{31}$$

其中 y 为真实值，$f(x)$ 为预测值。

指数损失函数的含义：

> 对于样本 x 来说，$y\in\{-1,+1\}$ ，而 $f(x)$ 是预测得到的概率值，是一个实数；当 $f(x)$ 的符号与 y 一致时，$yf(x)>0$ ，因此 $e^{-yf(x)}<1$ ，且 $|f(x)|$ 越大指数损失 $e^{-yf(x)}$ 越小（这很合理，此时 $|f(x)|$ 越大意味着分类器本身对预测结果的信心越大，损失应该越小；若 $|f(x)|$ 在 0 附近，虽然预测正确，但表示分类器本身对预测结果信心很小，损失应该较大)。当 $f(x)$ 的符号与 y 不一致时的分析相似。

在 AdaBoost 中，经过 m 次迭代后，可以得到 $f_m(x)=f_{m-1}(x)+\alpha_m G_m(x)$

目标是使前向分步算法得到的 α_m 和 $G_m(x)$ 使 $f_m(x)$ 在训练集 T 上的指数损失最小，即：

$$(\alpha_m,G_m(x))=argmin_{\alpha,G}\sum_{i=1}^{N}exp[-y_i(f_{m-1}(x_i)+\alpha G(x_i))] \tag{32}$$

也可以表示为：

$$(\alpha_m,G_m(x))=argmin_{\alpha,G}\sum_{i=1}^{N}\hat{w}_{mi}exp[-y_i\alpha G(x_i)] \tag{33}$$

其中 $\hat{w}_{mi}=exp[-y_i f_{m-1}(x_i)]$ 。因为 $\hat{w}_{mi}$ 既不依赖于 α 也不依赖于 G ，所以与最小化无关。但 $\hat{w}_{mi}$ 依赖于 $f_{m-1}(x)$ ，随着每一轮迭代而发生改变。

可以证明使上面式子达到最小的 α_m^* 和 $G_m^*(x)$ 就是 AdaBoost 算法所得到的 α_m 和 $G_m(x)$ 。

参考资料

机器学习常用损失函数小结
统计学习方法