损失函数

# 1 什么是损失函数

机器学习，准确地说监督学习的本质是给定一系列训练样本(xi, yi)，尝试学习x -> y 的映射关系，使得给定一个新的x ，即便这个x不在训练样本中，也能够使模型的预测值尽量与真实值接近。

损失函数是用来估量模型的预测值与真实值之间的差距，给模型的优化指引方向。

损失函数（Loss Function）通常是针对单个训练样本而言，给定一个模型的预测值和一个真实值，损失函数输出一个实值损失。

代价函数（Cost Function）通常是针对整个训练集（或者在使用mini-batch gradient descent 时一个mini-batch）的总损失。

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，用于机器学习领域和非机器学习领域（比如运筹优化）。

可以认为代价函数是目标函数的一种表现形式。

# 2 回归问题中的损失函数

对回归问题而言，衡量模型预测的准确程度，靠的是考察模型预测值与样本真实值之间的差值。因此，回归问题的损失函数应该满足两个基本条件：

（1）应是的函数；

（2）整体上关于单调递增

其中，y是样本的真实值，是该样本的预测值。

## 2.1 平均绝对误差（Mean Absolute Error）

### 2.1.1 定义

平均绝对误差，是计算预测值和目标值差的绝对值，也称为L1 Loss。其基本形式如下：



### 2.1.2 梯度

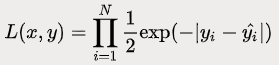


### 2.1.3 为什么

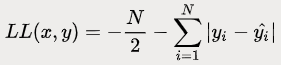
我们可以在一定的假设下通过最大化似然得到MAE损失函数的形式，假设模型预测值与真实值之间的误差服从拉普拉斯分布Laplace distribution（），则给定一个 输入xi模型输出真实值yi的概率为：



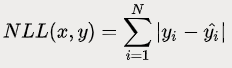
进一步，可以假设数据集中所有样本之间相互独立，则给定所有输入样本x，输出真实值y的概率，即似然函数Likelihood，即的乘积，公式如下：



为了计算方便，通常通过log来将比较难处理的连乘形式的似然函数转换成连加形式的对数似然函数，其单调性不变：



可以看到对数似然函数是负号的形式，而且第一项是一个常数，所以我们通过进一步的改写将最大化以上的对数似然函数变化为最小化以下的负对数似然函数（Negative Log-Likelihood）。



### 2.1.4 存在的问题

L1损失函数最大的问题是其梯度在零点处不连续，这会给基于梯度下降算法的优化算法带来不稳定行，为了应对这个问题，需要使用L2损失函数。

## 2.2 平均平方损失函数（Mean Square Error Loss）

### 2.2.1 定义

平均平方损失函数，是计算预测值和目标值差的平方的平均值，也被称为均方误差损失、L2 Loss，是机器学习和深度学习中常用的一种损失函数。



### 2.2.2 梯度



### 2.2.3 存在的问题

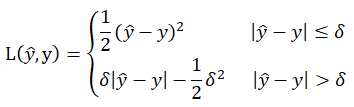
L2损失函数通过对L1 loss取平方，解决了L1损失函数在零点处梯度不连续的问题，但是也带来了另外一个问题，就是对异常点不够鲁棒（robust）。对异常点而言，一般会比较大，而因为L2损失函数引入的平方的作用，异常点的损失会被进一步放大。因为异常点是一种反常的行为，所以我们并不想让模型去过度学习这种行为。为了应对这一挑战，有了Huber损失函数。

## 2.3 Huber损失函数（SMAE）

### 2.3.1 定义

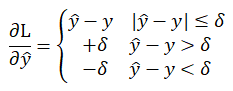
MSE损失收敛快但容易受异常点的影响，MAE对异常点更加健壮但是收敛慢，Huber Loss 则是一种将MSE与MAE结合起来，取两者优点的损失函数，也被称作Smooth Mean Absolute Error Loss。

其原理就是在误差接近0时使用MSE，误差较大时使用MAE，公式为：

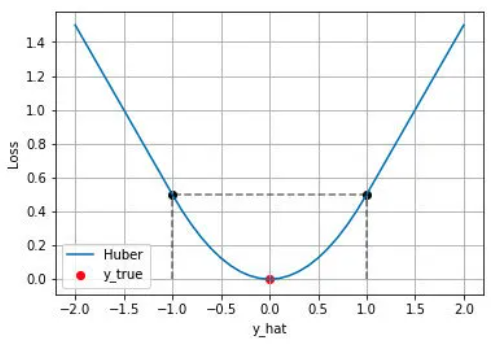


上式中是Huber Loss的一个超参数，的值是MSE和MAE两个损失连接的位置。等号右边第一项是MSE的部分，第二项是MAE部分，在MAE的部分公式为是为了保证误差时MAE和MSE的取值一致，进而保证Huber Loss损失连续可导。

### 2.3.2 梯度



下图是时的Huber Loss，可以看到在的区间内实际上就是MSE损失，在和区间内为MAE损失。



### 2.3.3 Huber Loss 的特点

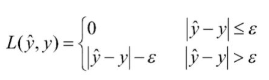
Huber Loss结合了MSE和MAE损失，在误差接近0时使用MSE，使损失函数可导并且梯度更加稳定；在误差较大时使用MAE可以降低异常点的影响，使训练对异常点更加健壮。缺点是需要额外地设置一个超参数。

### 2.3.4 存在的问题

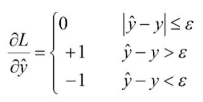
Huber损失函数几乎完美了，但是有人提出，想要得到一种基于训练集中的少数样本进行预测的模型。这就是Vapnik等人提出的SVM，对分类问题，下文提到的Hinge Loss会导出SVM；而对回归问题，导出SVM的是如下的。

## 

### 2.4.1 定义



### 2.4.2 梯度



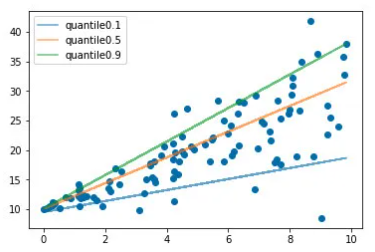
### 2.4.3 存在的问题

显然，对于那些预测值和实际值差别很小的样本，直接忽略掉了。这样做的结果是，整个样本训练出的模型，在进行预测的时候只有很少的样本起作用，这样既增加了模型的鲁棒性，也加快了模型预测时的计算。这少部分在预测中起作用的样本，也被称为支持向量。

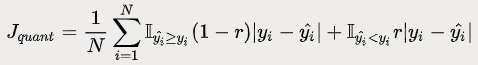
## 2.5 分位数损失（Quantile Loss）

分位数回归Quantile Regression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：



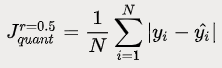
分位数回归是通过使用分位数损失Quantile Loss来实现这一点的，分位数损失形式如下，式中的r分位数系数。（注：下面的公式直接写了代价函数的形式）



这个损失函数是一个分段的函数 ，将（高估）和（低估）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。

特别地，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）。



# 3分类问题中的损失函数

对分类问题，其损失函数取决于样本label的编码方式。以二分类为例，我们可以将正类和负类分别编码为{1,0}，也可以分别编码为{1,-1}。

如果样本编码为{1,0}，我们可以设法使模型预测的结果介于0和1之间。这样就可以理解成一种概率，如果越大，对应样本属于1这类的概率越大；反之，越小，对应样本属于0这一类的概率越大，既然预测的是概率，当然可以用极大似然的框架求解模型参数，这时候的损失函数就是负的对数似然。

如果样本编码为{1,-1}，则通过模型预测的结果的正负进行分类，，分到+1对应的类，越大确信度越高；，分到-1对应的类，越小确信度越高。在这种情况下，表明了误分类的程度，损失函数自然应该是的函数，而且整体上应该是的单调递增函数。

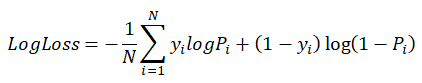
## 3.1 0-1损失函数（Zero-One Loss)



若绝对值在T以下，损失函数为0，否则为1。它是一个非凸函数不太适用。

## 3.2 对数损失函数

对数损失函数（LogLoss）是经常在离线评估中使用的指标，在一个二分类问题中，LogLoss的定义如下（注：下面的形式是代价函数的形式）：

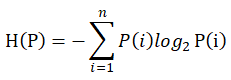


其中，为输入实例的真实类别，为预测输入实例是正样本的概率，N是样本总数。

## 3.3 交叉熵（Cross Entropy）损失函数

### 3.3.1 定义

熵(Entropy)源于信息学，用于度量信息的不确定度。熵越大，代表信息的不确定性越大，信息也就越大（一种类比的理解就是，对于十分稳定的物体来说，一直不发生变化是确定的，发生变化所蕴含的信息量就更大）。某个分布P(i)的熵定义为：



实际上，𝐻(𝑃)也可以使用其他底数的log函数计算。举个例子，对于 4 分类问题，如果某个样本的真实标签是第4类，那么标签的One-hot编码为[0,0,0,1]，即这张图片的分类是唯一确定的，它属于第4类的概率𝑃(𝑦=4|𝒙) = 1，不确定性为0，它的熵可以简单的计算为：



也就是说，对于确定的分布，熵为0，不确定性最低。如果它预测的概率分布是[0.1,0.1,0.1,0.7]，它的熵可以计算为：



这种情况比前面确定性类别的例子的确定性要稍微大点。

考虑随机分类器，它每个类别的预测概率是均等的：[0.25,0.25,0.25,0.25]，同样的方法，可以计算它的熵约为2，这种情况的不确定性略大于上面一种情况。

由于𝑃(𝑖)∈[0,1], log2 𝑃(𝑖) ≤ 0，因此熵𝐻(𝑃)总是大于等于0。当熵取得最小值0时， 不确定性为0。

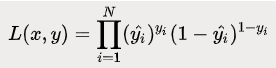
在二分类中，通常使用Sigmoid函数将模型的输出压缩到(0, 1)区间内，用来代表给定输入xi，模型判断为正类的概率。由于只有正负两类，因此同时也得到了负类的概率。



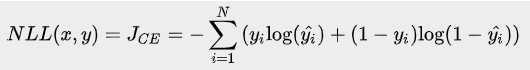
上面两个公式可以合并成一个公式：



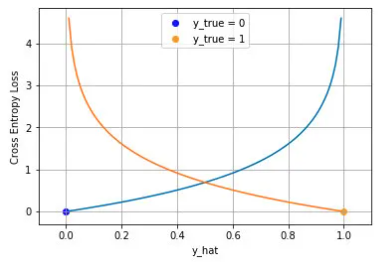
假设数据点之间独立同分布，则似然可以表示为



对似然取对数，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（注：下面是代价函数的形式）：



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。



可以看到越接近目标值损失越小，随着误差变差，损失呈指数增长。

### 3.3.2 梯度

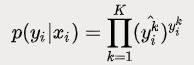


如果两个类分别编码为{0,1}，y=1可以看作样本属于1类的概率为1，属于0类的概率为0，这是样本的真实概率分布。则代表了模型预测的样本属于1类的概率。交叉熵表征了这两个概率分布之间的接近程度。

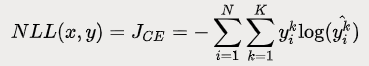
## 3.4 SoftMax损失函数

### 3.4.1 定义

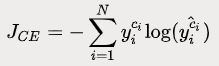
Softmax函数将每个维度的输出范围都限定在(0,1)之间，同时所有维度的输出和为1，用于表示一个概率分布。



其中表示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：

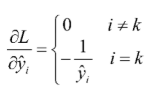


由于yi是一个one-hot向量，除了目标类为1之外其他类别上的输出都为0，因此上式也可以写为：



其中ci是样本yi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。

### 3.4.2 梯度



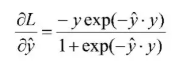
对比可以发现，Softmax损失和Cross Entropy损失是一回事。Softmax损失函数可以看作Cross Entropy在多分类问题上的扩展。

## 3.5 Log Loss

### 3.5.1 定义



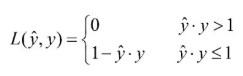
### 3.5.2 梯度



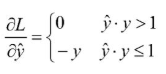
* 当，且值很大时，指数运算在计算时有益出风险，可能进行截断。
* Log Loss的计算梯度计算公式里，分子分母都有指数项，时，可以对分子分母同时放缩，然后计算（Soft Max也适用此技巧）

## 3.6 Hinge Loss

### 3.6.1 定义



### 3.6.2 梯度



对Hinge损失来说，如果样本被正确分类，且距离分类边界的距离超过了margin，则损失标记为0。因此，Hinge损失函数最小化的目标，是让样本尽量都被正确分类，且距离边界足够远，所以Hinge损失导出了分类的SVM。

## 3.7 指数损失函数（Exponential Loss）

### 3.7.1 定义



对噪声点，离群点非常敏感。

### 3.7.2 梯度



* 指数损失函数加上前向分布算法，可以导出Adaboost。
* 类别编码为{+1,-1}时，表明了样本误分类程度。而指数函数又对这一程度进行了缩放，因此Adaboot在训练中将重点放在了上一次被误分类的样本上。
* 指数函数比平方函数增长更快，因此指数损失函数相比L2损失，比异常点更加不鲁棒（robust）。

## 3.8 合页损失（Hinge Loss）

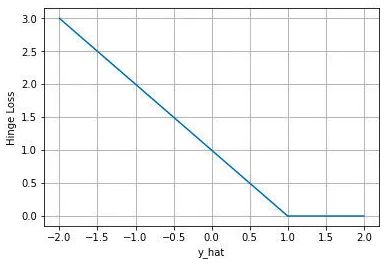
合页损失Hinge Loss是另外一种二分类损失函数，其公式如下：



如果被分类正确，损失为0，否则损失就为



下图是y为正类， 即y=1时，不同输出的合页损失示意图：

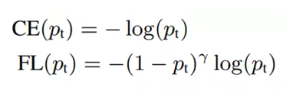


从上图可以看到，当y为正类时，模型输出负值会有较大的惩罚，当模型输出为正值且在(0,1)区间时还会有一个较小的惩罚。

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。

合页损失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。合页损失函数适用于maximum-margin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。

## 3.9 Focal Loss



以交叉熵为基础，使模型更关注较难的样本。

# 4 讨论

## 4.1 MAE与MSE区别

MAE和MSE作为损失函数的主要区别是：MSE损失相比MAE通常可以更快地收敛，但 MAE损失对于outlier更加健壮，即更加不易受到 outlier 影响。

MSE通常比MAE可以更快地收敛。当使用梯度下降算法时，MSE损失的梯度为，而MAE损失的梯度为，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。

这个也是 MSE 更为流行的原因。

MAE对于outlier更加robust。我们可以从两个角度来理解这一点：

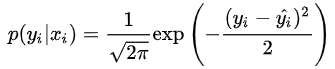
* 第一个角度是直观地理解，由于MAE损失与绝对误差之间是线性关系，MSE损失与误差是平方关系，当误差非常大的时候，MSE损失会远远大于MAE损失。因此当数据中出现一个误差非常大的outlier时，MSE会产生一个非常大的损失，对模型的训练会产生较大的影响。
* 第二个角度是从两个损失函数的假设出发，MSE假设了误差服从高斯分布，MAE假设了误差服从拉普拉斯分布。拉普拉斯分布本身对于outlier更加robust。

因此以拉普拉斯分布为假设的MAE对outlier比高斯分布为假设的MSE更加robust。

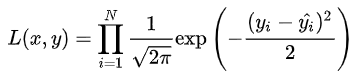
## 4.2 为什么回归任务常用均方误差损失

可以发现，在回归任务中，常常会用到均方误差损失函数，这是为什么呢？

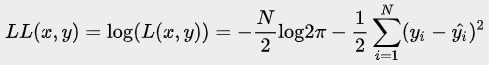
一般在生活中，如果没有系统误差，那么我们对一个数值进行估计或测量，很多情况下预测值和真实值之间的误差是服从高斯分布（均值为0，方差为1）。则给定一个输入样本xi，模型输出真实值yi的概率为：



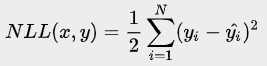
进一步，可以假设数据集中所有样本之间相互独立，则给定所有输入样本x，输出真实值y的概率，即似然函数Likelihood，即的乘积，公式如下：



为了计算方便，通常通过log来将比较难处理的连乘形式的似然函数转换成连加形式的对数似然函数，其单调性不变：



可以看到对数似然函数是负号的形式，而且第一项是一个常数，所以我们通过进一步的改写将最大化以上的对数似然函数变化为最小化以下的负对数似然函数（Negative Log-Likelihood）。



可以看到，这就是均方误差损失函数的形式。在回归任务中，真实值与预测值误差服从高斯分布的假设下，最大化似然函数和最小化均方损失函数是等价的。

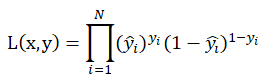
## 4.3 为什么分类任务用交叉熵作为损失函数

可以从最大似然的角度和信息论这两个角度来解释。

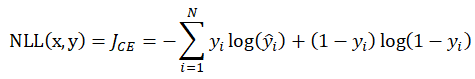
### 4.3.1 从最大似然的角度

交叉熵损失函数是我们在分类任务中经常用到的损失函数，我们可以来做类似的分析。我们将要求解的问题简化为二分类，即我们使用一个模型来预测某个样本是属于分类0还是分类1，一般做法我们会去预测一个在0到1内的值，如果大于0.5那么就是分类1，小于0.5就是分类0。

那么，在其中我们常常已经做了一个假设，即我们期望模型的输出服从一个伯努利分布。模型的输出就是预测分类为1的概率，即 P(y=1) = p, P(y=0) = 1 - p。我们将左边这个式子改写一下，在输入为x时，P(y|x) = py \* ( 1 - p)(1-y)。对于所有的样本来说，将p替换成模型的预测yi^，那么其似然函数如下：



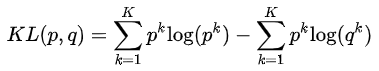
我们优化模型的过程就是极大化我们的似然函数，让模型的预测值出现的可能性最大。刚刚我们已经介绍过了，我们通过常规操作，可以将最大化似然函数的目标转化为最小化负对数似然函数。



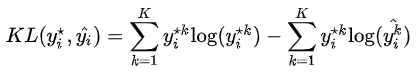
我们又能发现，这显然就是我们的交叉熵损失函数。所以，本质上对分类任务的极大似然估计和最小化交叉熵损失函数是一致的。所以，只要服从伯努利分布的假设，我们使用交叉熵处理分类任务就是很合理的。

### 4.3.2 从信息论的角度

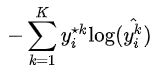
假设对于样本xi存在一个最优分布真实地表明了这个样本属于各个类别的概率，那么我们希望模型的输出尽可能地逼近这个最优分布，在信息论中，我们可以使用KL散度 Kullback–Leibler Divergence来衡量两个分布的相似性。给定分布p和分布q， 两者的 KL 散度公式如下：



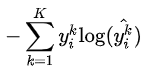
其中第一项为分布p的信息熵，第二项为分布p和q的交叉熵。将最优分布和输出分布带入p和q得到：



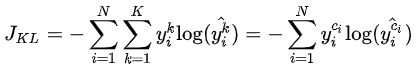
由于我们希望两个分布尽量相近，因此我们最小化KL散度。同时由于上式第一项信息熵仅与最优分布本身相关，因此我们在最小化的过程中可以忽略掉，变成最小化：



我们并不知道最优分布，但训练数据里面的目标值yi可以看做是的一个近似分布



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：



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。

参考：

龙书，TensorFlow深度学习。

<https://mp.weixin.qq.com/s/wGBmxfOeEHcx4r4wGgjSDw>

<https://mp.weixin.qq.com/s/7R6oLwIQ8Aa9aonWhM38Rg>

<https://mp.weixin.qq.com/s/kpMmXYi42ZuJb0e7DLgS1A>

<https://mp.weixin.qq.com/s/hStgf_6Nvul9DRYgsMLc_w>