损失函数

# 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，也能够使模型的输出y^，尽量与真实的y接近。

损失函数是用来估量模型的输出y^与真实值y之间的差距，给模型的优化指引方向。

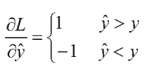
# 回归问题中的损失函数

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实际值之间的差值。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label，代表该样本的预测值）。满足要求且常用的损失函数如下：

## L1 损失函数



梯度：



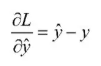
挑战：

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，L2损失函数。

## L2损失函数



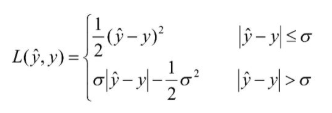
梯度：



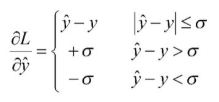
挑战：

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。

## Huber损失函数



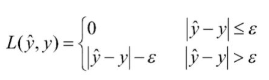
梯度：



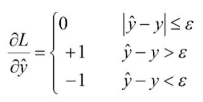
挑战：

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。

## 



梯度：



挑战：

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。

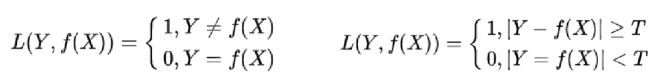
# 分类问题中的损失函数

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，>0，分到+1对应的类，越大确信度越高；<0，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。

## 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

## 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：

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。

## 交叉熵（Cross Entropy）损失函数

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为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。

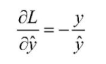
交叉熵(Cross Entropy)的定义如下：

𝐻(𝑝||𝑞) ≜ −∑𝑝(𝑖)log2 𝑞(𝑖)

通常我们把交叉熵写成下面的形式：



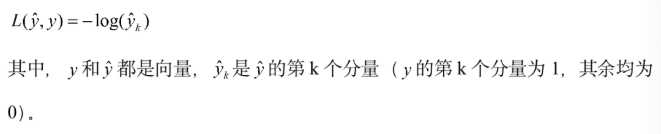
梯度：



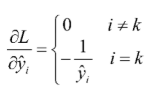
Note：

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。

## SoftMax损失函数



梯度：



Note：

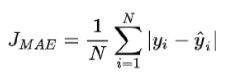
对比可以发现，Softmax损失和Cross Entropy损失是一回事。Softmax损失函数可以看作Cross Entropy在多分类问题上的扩展。

## 指数损失函数



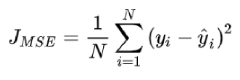
对噪声点，离群点非常敏感。

## 平均绝对误差损失（Mean Absolute Error Loss）



计算预测值和目标值差的绝对值，也称为MAE Loss、L1 Loss

## 平均平方误差损失（Mean Square Error Loss）



计算预测值和目标值差的平方，也称均方误差损失，MSE Loss，L2 Loss

机器学习、深度学习回归任务中最常用的一种损失函数

## 合页损失（Hinge Loss）

其公式如下：

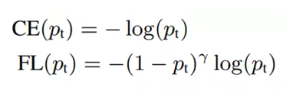


如果被分类正确，损失为0，否则损失就为



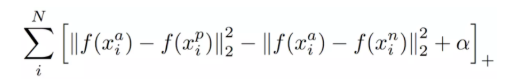
合页损失函数适用于maximum-margin的分类，支持向量机Support Vector Machine (SVM)模型的损失函数，本质上就是Hinge Loss + L2正则化。

## Focal Loss



以交叉熵为基础，使模型更关注较难的样本。

## Triplet Loss



a: anchor，p: positive，n: negative

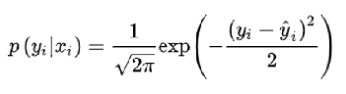
整体优化目标是拉近a, p的距离，拉远a, n的距离，达成分类效果

# 讨论

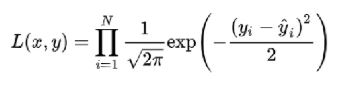
## 为什么回归任务常用均方误差损失

我们可以发现，在深度学习的回归任务中，常常会用到均方误差损失函数，这是为什么呢？

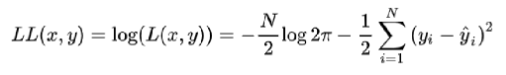
一般在生活中，如果没有系统误差，那么我们对一个数值进行估计或测量，很多情况下估计值和真值之间的误差是服从高斯分布的。我们一个模型对于目标标签的估计也可以这么认为。我们可以假设高斯分布的均值为0，方差为1。第i个样本的输入为xi , 模型的输出为yi^, 而它的标签为yi , 那么我们估计的yi的条件概率密度函数如下图所示。



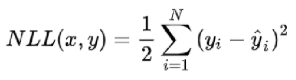
对于所有样本来说，我们可以计算出它的似然函数，我们对模型的训练就是要极大化这个似然函数。



常用的做法是通过log来将比较难处理的连乘形式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。

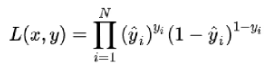


大家应该注意到了，这不就跟均方损失函数的形式几乎一样吗？正是如此，最大化似然函数和最小化均方损失函数是等价的。所以，在回归任务中，估计值和真值误差服从高斯分布的假设下，我们以均方误差作为损失函数来训练模型是合理的。

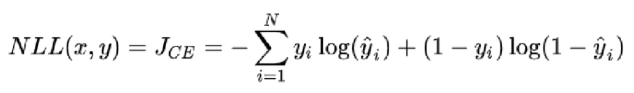
## 为什么分类任务用交叉熵作为损失函数

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yi^=p，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。

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>