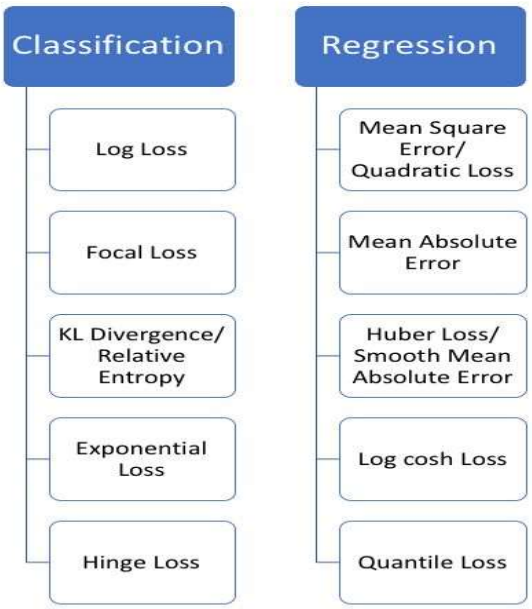
最常用的5个回归损失函数

L1、L2 损失、Huber 损失、Log-Cosh 损失、分位数损失

损失函数大致可分为两类：分类问题的损失函数和回归问题的损失函数。



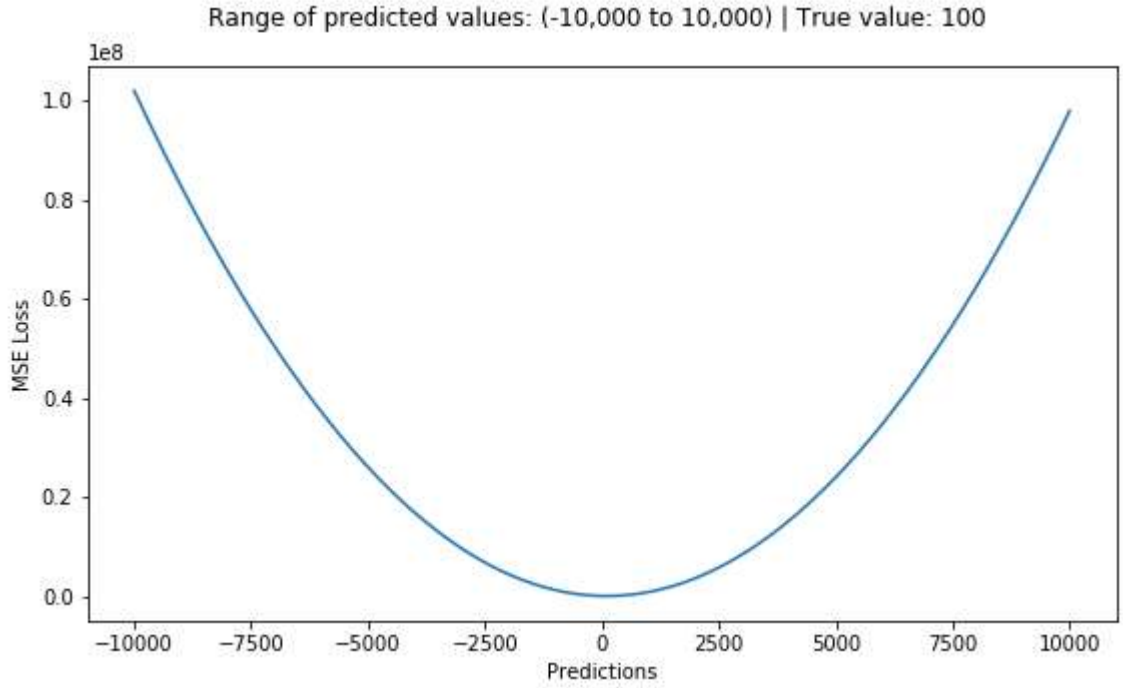
分类、回归问题损失函数对比

均方误差

$$MSE = \sum_{i=1}^{n} (y_i - y_i^p)^2$$

均方误差(MSE)是最常用的回归损失函数，计算方法是求预测值与真实值之间距离的平方和，公式如图。

下图是MSE函数的图像，其中目标值是100，预测值的范围从-10000到10000，Y轴代表的MSE取值范围是从0到正无穷，并且在预测值为100处达到最小。

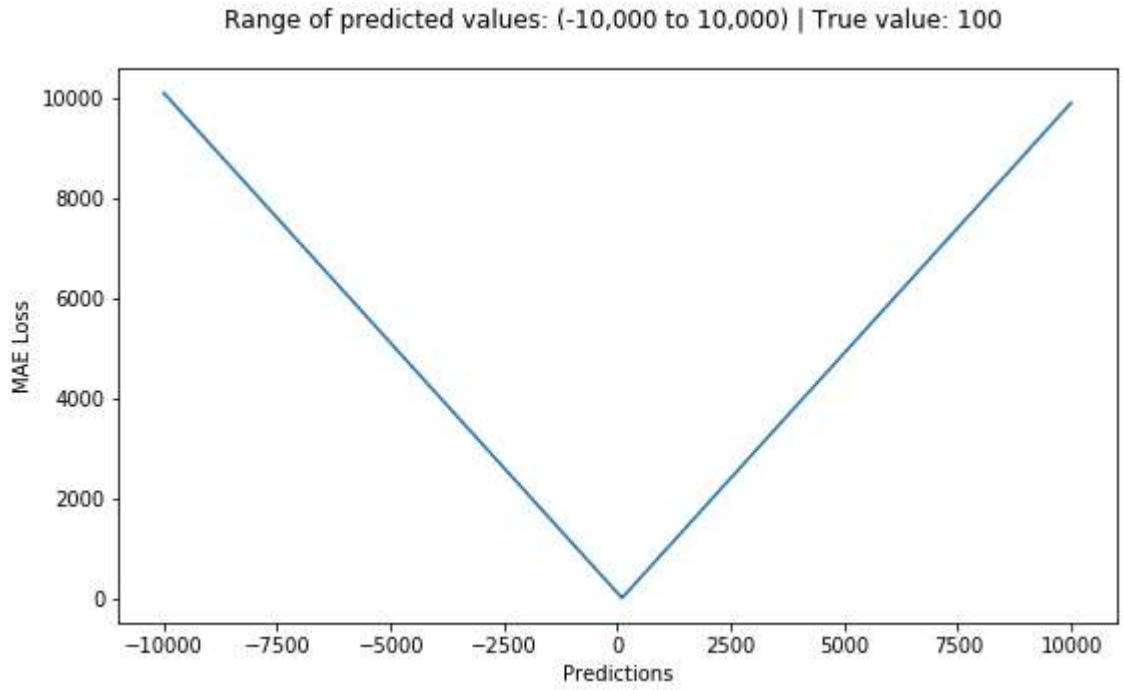


MSE损失（Y轴）-预测值（X轴）

平均绝对值误差（也称L1损失）

$$MAE = \sum_{i=1}^{n} |y_i - y_i^p|$$

平均绝对误差（MAE）是另一种用于回归模型的损失函数。MAE是目标值和预测值之差的绝对值之和。其只衡量了预测值误差的平均模长，而不考虑方向，取值范围也是从0到正无穷（如果考虑方向，则是残差/误差的总和——平均偏差（MBE））。



MAE损失（Y轴）-预测值（X轴）

MSE（L2损失）与MAE（L1损失）的比较

简单来说，MSE计算简便，但MAE对异常点有更好的鲁棒性。下面就来介绍导致二者差异的原因。

训练一个机器学习模型时，我们的目标就是找到损失函数达到极小值的点。当预测值等于真实值时，这两种函数都能达到最小。

下面是这两种损失函数的python代码。你可以自己编写函数，也可以使用sklearn内置的函数。

```
# true: Array of true target variable
# pred: Array of predictions
def mse(true, pred):
    return np.sum((true - pred)**2)
def mae(true, pred):
    return np.sum(np.abs(true - pred))

# also available in sklearn
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
```

下面让我们观察MAE和RMSE（即MSE的平方根，同MAE在同一量级中）在两个例子中的计

算结果。第一个例子中，预测值和真实值很接近，而且误差的方差也较小。第二个例子中，因为存在一个异常点，而导致误差非常大。

MAE vs. RMSE for cases with slight variance in data

ID	Error	|Error|	Error²
1	0	0	0
2	1	1	1
3	-2	2	4
4	-0.5	0.5	0.25
5	1.5	1.5	2.25

MAE: 1　　RMSE: 1.22

MAE vs. RMSE for cases with outliers in data

ID	Error	|Error|	Error²
1	0	0	0
2	1	1	1
3	1	1	1
4	-2	2	4
5	15	15	225

outlier

MAE: 3.8　　RMSE: 6.79

左图：误差比较接近 右图：有一个误差远大于其他误差

从图中可以知道什么？应当如何选择损失函数?

MSE对误差取了平方（令e=真实值-预测值），因此若e>1，则MSE会进一步增大误差。如果数据中存在异常点，那么e值就会很大，而e^2则会远大于|e|。

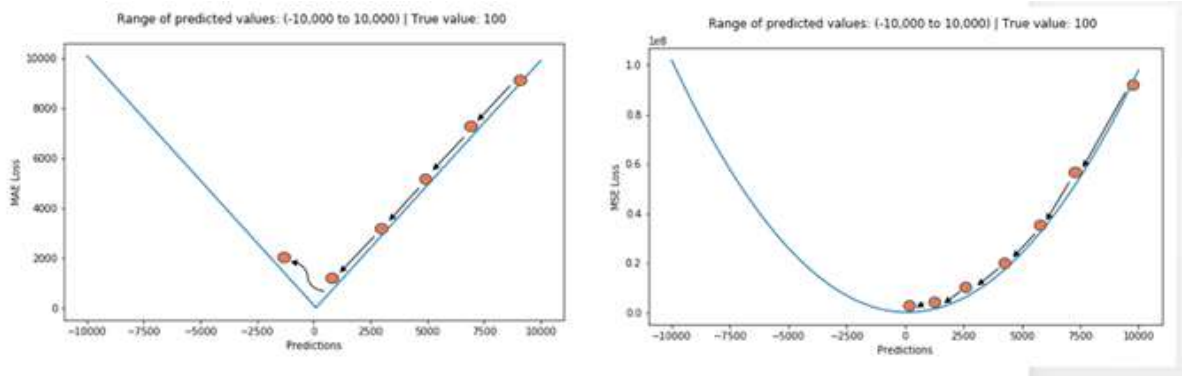
因此，相对于使用MAE计算损失，使用MSE的模型会赋予异常点更大的权重。在第二个例子中，用RMSE计算损失的模型会以牺牲了其他样本的误差为代价，朝着减小异常点误差的方向更新。然而这就会降低模型的整体性能。

如果训练数据被异常点所污染，那么MAE损失就更好用（比如，在训练数据中存在大量错误的反例和正例标记，但是在测试集中没有这个问题）。

直观上可以这样理解：如果我们最小化MSE来对所有的样本点只给出一个预测值，那么这个值一定是所有目标值的平均值。但如果是最小化MAE，那么这个值，则会是所有样本点目标值的中位数。众所周知，对异常值而言，中位数比均值更加鲁棒，因此MAE对于异常值也比MSE更稳定。

然而MAE存在一个严重的问题（特别是对于神经网络）：更新的梯度始终相同，也就是说，即使对于很小的损失值，梯度也很大。这样不利于模型的学习。为了解决这个缺陷，我们可以使用变化的学习率，在损失接近最小值时降低学习率。

而MSE在这种情况下的表现就很好，即便使用固定的学习率也可以有效收敛。MSE损失的梯度随损失增大而增大，而损失趋于0时则会减小。这使得在训练结束时，使用MSE模型的结果会更精确。



根据不同情况选择损失函数

如果异常点代表在商业中很重要的异常情况，并且需要被检测出来，则应选用MSE损失函数。相反，如果只把异常值当作受损数据，则应选用MAE损失函数。

推荐大家读一下这篇文章，文中比较了分别使用L1、L2损失的回归模型在有无异常值时的表现。

文章网址：

http://rishy.github.io/ml/2015/07/28/l1-vs-l2-loss/

这里L1损失和L2损失只是MAE和MSE的别称。

总而言之，处理异常点时，L1损失函数更稳定，但它的导数不连续，因此求解效率较低。L2损失函数对异常点更敏感，但通过令其导数为0，可以得到更稳定的封闭解。

二者兼有的问题是：在某些情况下，上述两种损失函数都不能满足需求。例如，若数据中90%的样本对应的目标值为150，剩下10%在0到30之间。那么使用MAE作为损失函数的模型

可能会忽视10%的异常点，而对所有样本的预测值都为150。

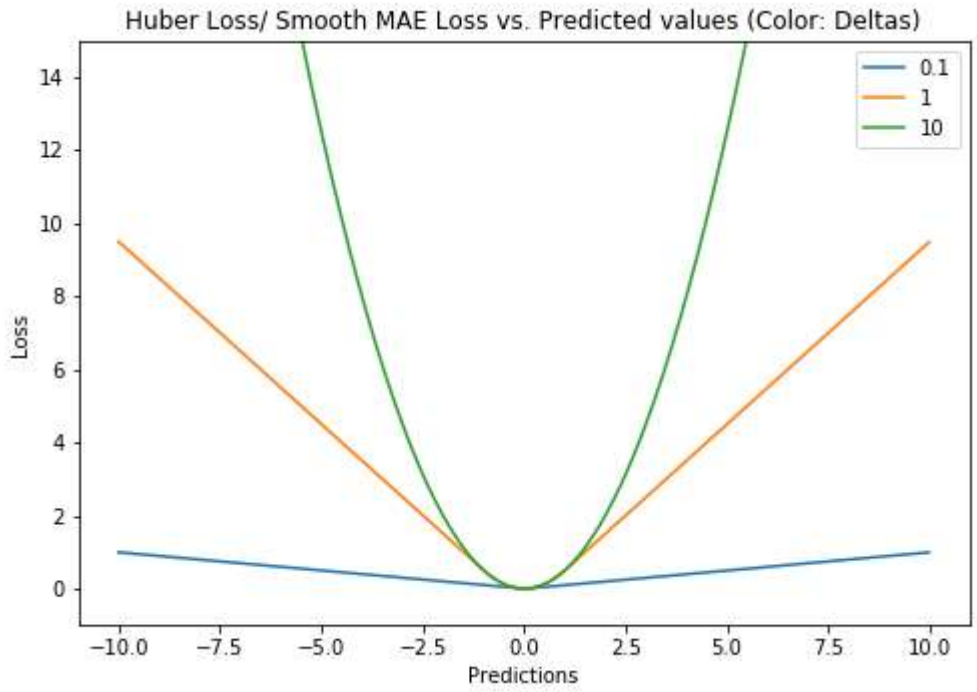
这是因为模型会按中位数来预测。而使用MSE的模型则会给出很多介于0到30的预测值，因为模型会向异常点偏移。上述两种结果在许多商业场景中都是不可取的。

这些情况下应该怎么办呢？最简单的办法是对目标变量进行变换。而另一种办法则是换一个损失函数，这就引出了下面要讲的第三种损失函数，即Huber损失函数。

Huber损失，平滑的平均绝对误差

Huber损失对数据中的异常点没有平方误差损失那么敏感。它在0也可微分。本质上，Huber损失是绝对误差，只是在误差很小时，就变为平方误差。误差降到多小时变为二次误差由超参数δ（delta）来控制。当Huber损失在[0-δ,0+δ]之间时，等价为MSE，而在[-∞,δ]和[δ,+∞]时为MAE。

$$L_\delta(y, f(x)) = \begin{cases} \frac{1}{2}(y - f(x))^2 & \text{for} |y - f(x)| \leq \delta, \\ \delta |y - f(x)| - \frac{1}{2}\delta^2 & \text{otherwise.} \end{cases}$$



Huber损失（Y轴）与预测值（X轴）图示。真值取0

这里超参数delta的选择非常重要，因为这决定了你对与异常点的定义。当残差大于delta，应当采用L1（对较大的异常值不那么敏感）来最小化，而残差小于超参数，则用L2来最小化。

为何要使用Huber损失?

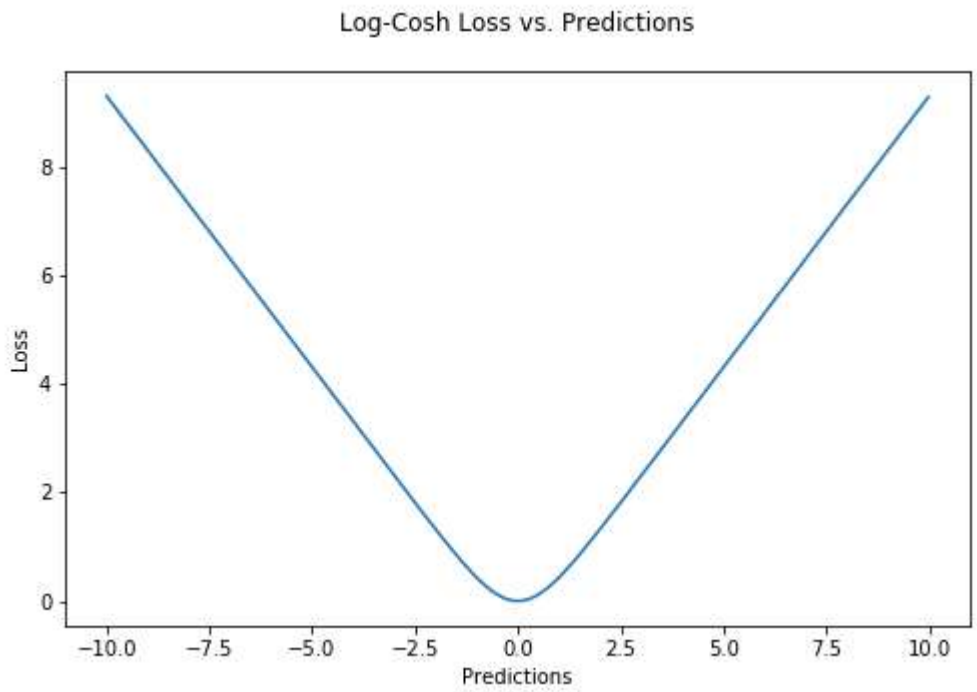
使用MAE训练神经网络最大的一个问题就是不变的大梯度，这可能导致在使用梯度下降快要结束时，错过了最小点。而对于MSE，梯度会随着损失的减小而减小，使结果更加精确。

在这种情况下，Huber损失就非常有用。它会由于梯度的减小而落在最小值附近。比起MSE，它对异常点更加鲁棒。因此，Huber损失结合了MSE和MAE的优点。但是，Huber损失的问题是我们可能需要不断调整超参数delta。

Log-Cosh损失

Log-cosh是另一种应用于回归问题中的，且比L2更平滑的的损失函数。它的计算方式是预测误差的双曲余弦的对数。

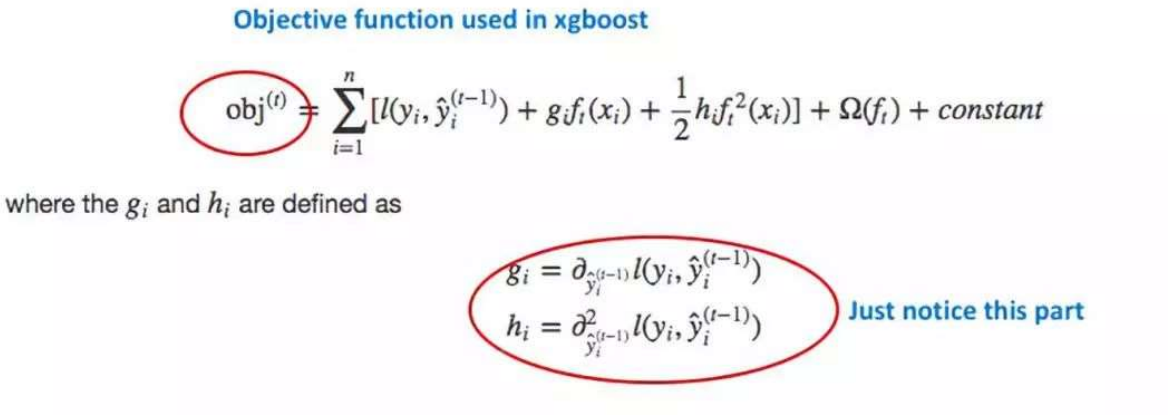
$$L(y, y^p) = \sum_{i=1}^{n} \log(\cosh(y_i^p - y_i))$$



Log-cosh损失（Y轴）与预测值（X轴）图示。真值取0

优点：对于较小的x，log(cosh(x))近似等于(x^2)/2，对于较大的x，近似等于abs(x)-log(2)。这意味着‘logcosh’基本类似于均方误差，但不易受到异常点的影响。它具有Huber损失所有的优点，但不同于Huber损失的是，Log-cosh二阶处处可微。

为什么需要二阶导数？许多机器学习模型如XGBoost，就是采用牛顿法来寻找最优点。而牛顿法就需要求解二阶导数（Hessian）。因此对于诸如XGBoost这类机器学习框架，损失函数的二阶可微是很有必要的。



XgBoost中使用的目标函数。注意对一阶和二阶导数的依赖性

但Log-cosh损失也并非完美，其仍存在某些问题。比如误差很大的话，一阶梯度和Hessian会变成定值，这就导致XGBoost出现缺少分裂点的情况。

Huber和Log-cosh损失函数的Python代码：

```
# huber loss
def huber(true, pred, delta):
    loss = np.where(np.abs(true-pred) < delta , 0.5*((true-pred)**2), 
delta*np.abs(true - pred) - 0.5*(delta**2))
    return np.sum(loss)

# log cosh loss
def logcosh(true, pred):
    loss = np.log(np.cosh(pred - true))
return np.sum(loss)
```

分位数损失

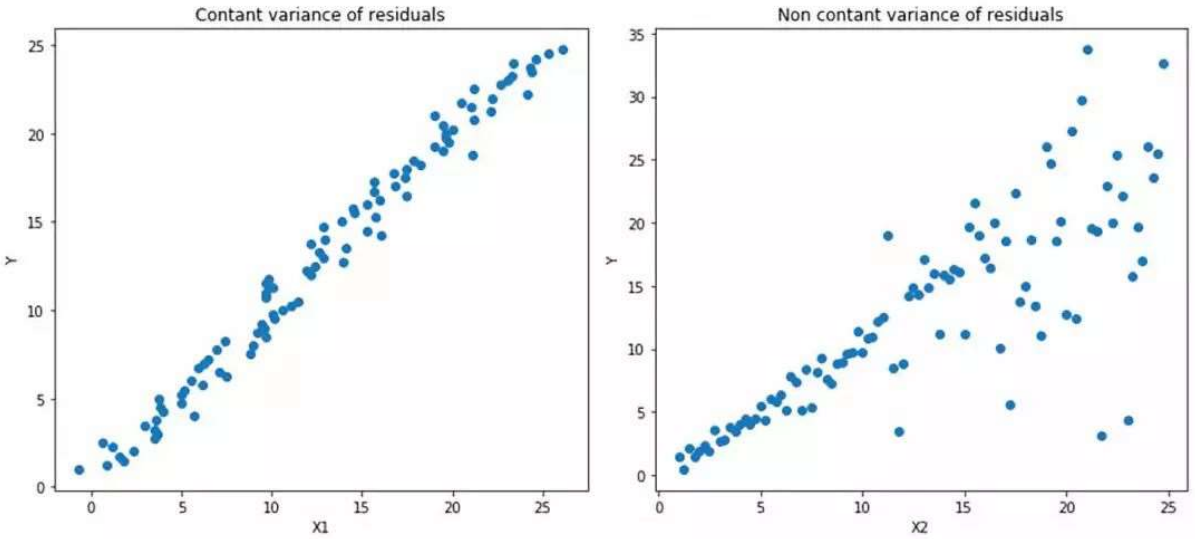
在大多数现实世界预测问题中，我们通常希望了解预测中的不确定性。清楚预测的范围而非仅是估计点，对许多商业问题的决策很有帮助。

当我们更关注区间预测而不仅是点预测时，分位数损失函数就很有用。使用最小二乘回归进行区间预测，基于的假设是残差（y-y_hat）是独立变量，且方差保持不变。

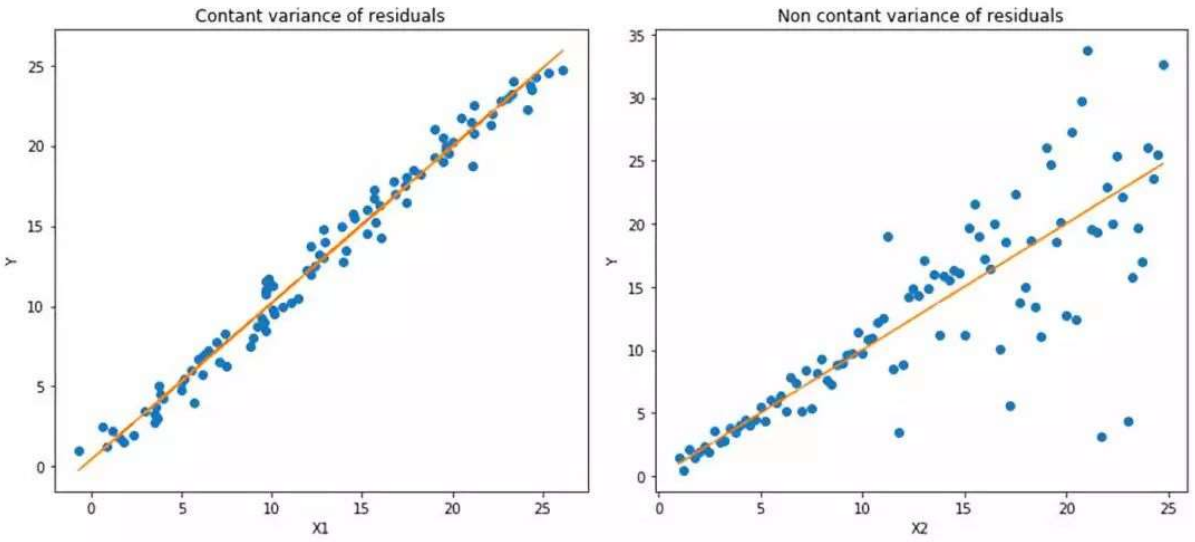
一旦违背了这条假设，那么线性回归模型就不成立。但是我们也不能因此就认为使用非线性函数或基于树的模型更好，而放弃将线性回归模型作为基线方法。这时，分位数损失和分位数回归就派上用场了，因为即便对于具有变化方差或非正态分布的残差，基于分位数损失的回归也能给出合理的预测区间。

下面让我们看一个实际的例子，以便更好地理解基于分位数损失的回归是如何对异方差数据起作用的。

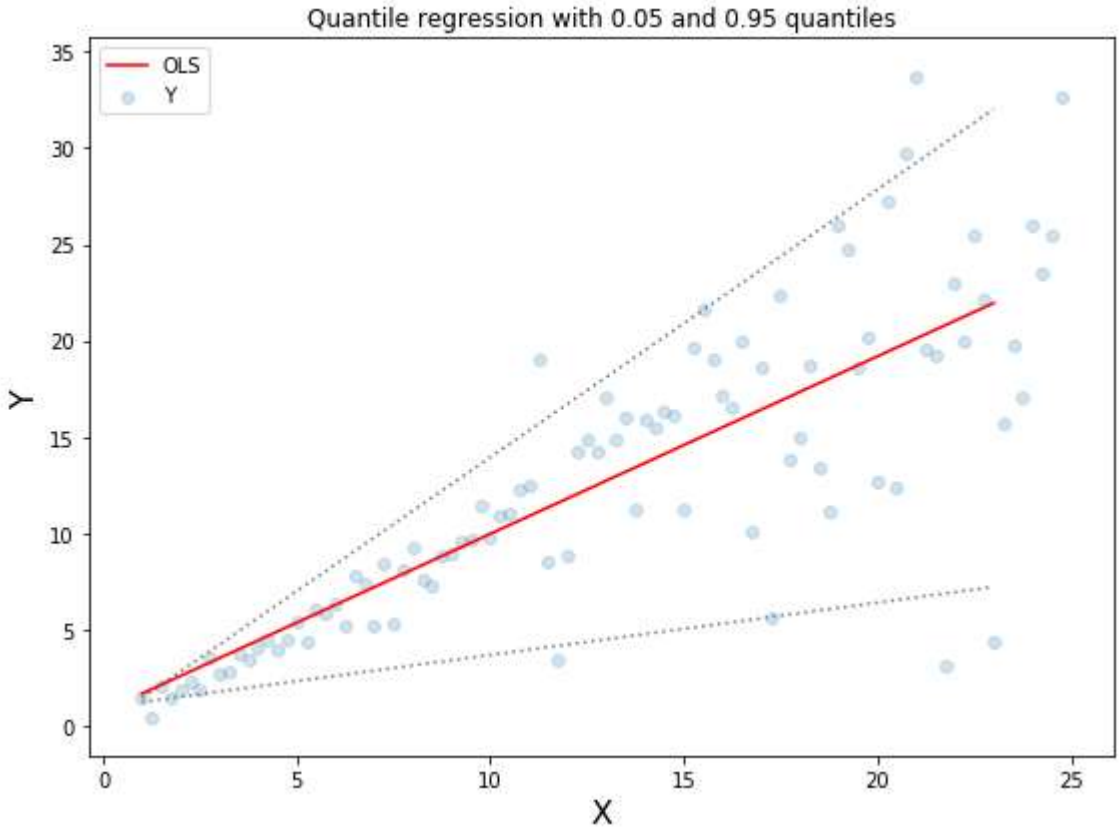
分位数回归与最小二乘回归



左：b/wX1和Y为线性关系。具有恒定的残差方差。右：b/wX2和Y为线性关系，但Y的方差随着X2增加。（异方差）



橙线表示两种情况下OLS的估值



分位数回归。虚线表示基于0.05和0.95分位数损失函数的回归

附上图中所示分位数回归的代码：

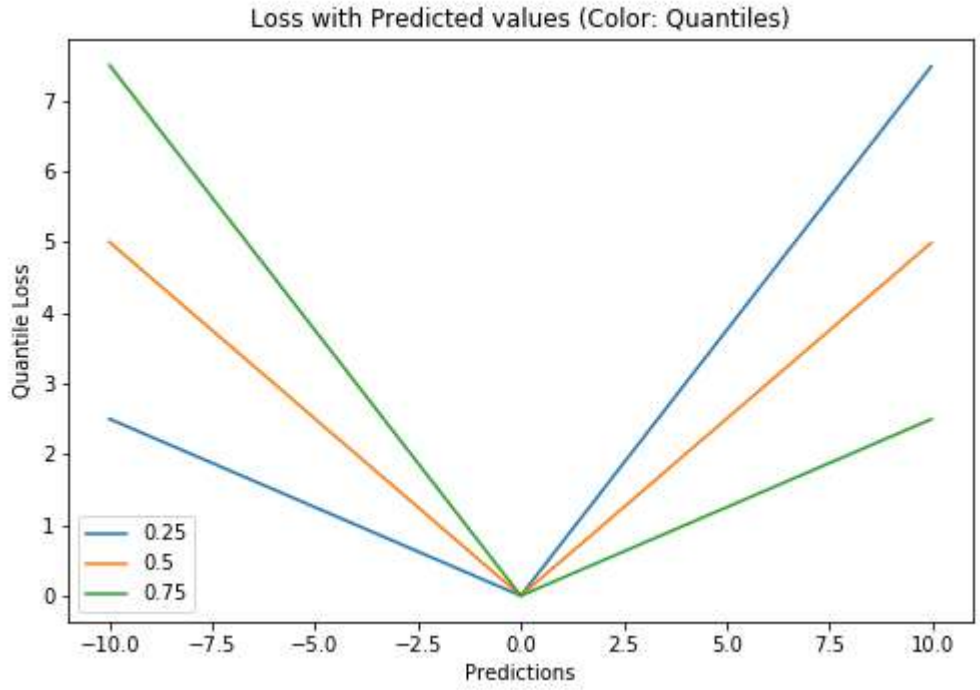
https://github.com/groverpr/Machine-Learning/blob/master/notebooks/09_Quantile_Regression.ipynb

理解分位数损失函数

如何选取合适的分位值取决于我们对正误差和反误差的重视程度。损失函数通过分位值（γ）对高估和低估给予不同的惩罚。例如，当分位数损失函数γ=0.25时，对高估的惩罚更大，使得预测值略低于中值。

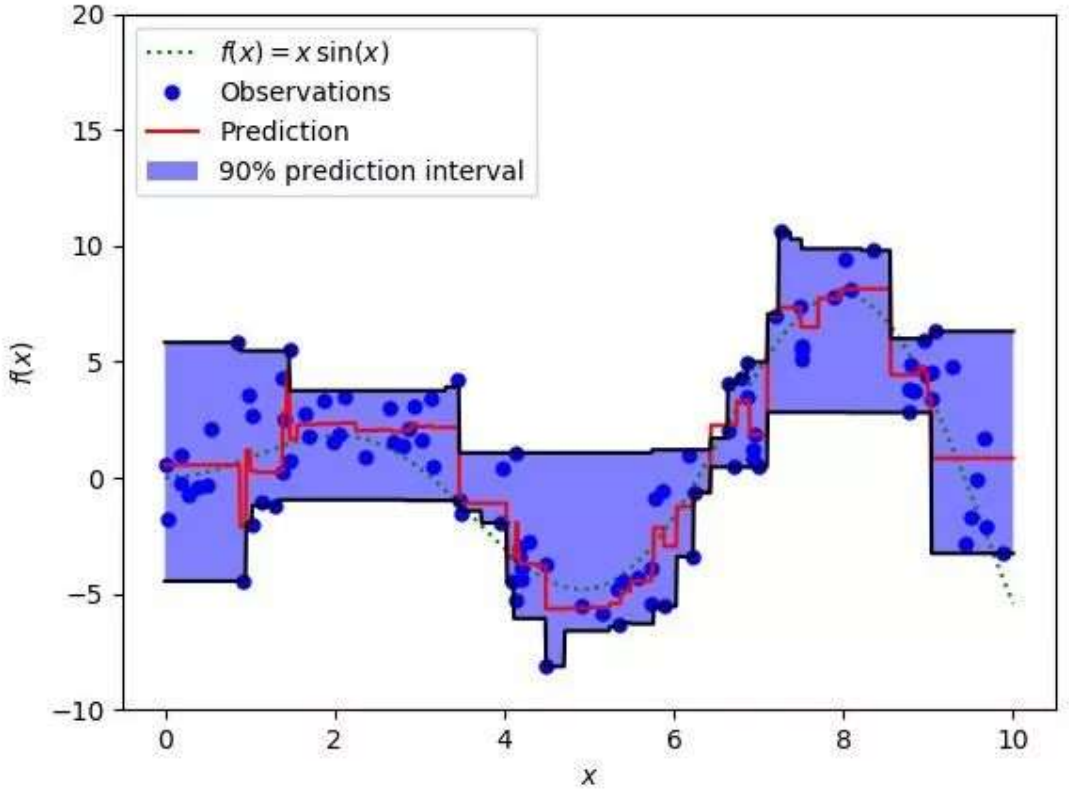
$$L_\gamma(y, y^p) = \sum_{i:y_i<y_i^p} (1-\gamma)|y_i - y_i^p| + \sum_{i:y_i \geq y_i^p} \gamma|y_i - y_i^p|$$

γ是所需的分位数，其值介于0和1之间。



分位数损失（Y轴）与预测值（X轴）图示。Y的真值为0

这个损失函数也可以在神经网络或基于树的模型中计算预测区间。以下是用Sklearn实现梯度提升树回归模型的示例。



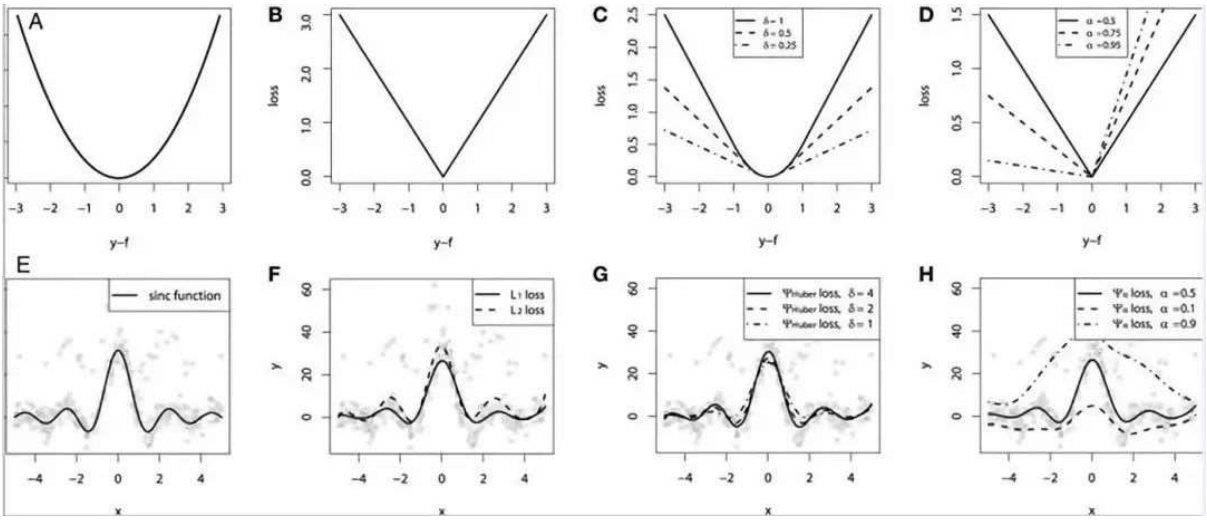
使用分位数损失（梯度提升回归器）预测区间

上图表明：在sklearn库的梯度提升回归中使用分位数损失可以得到90%的预测区间。其中上限为γ=0.95，下限为γ=0.05。

对比研究

为了证明上述所有损失函数的特点，让我们来一起看一个对比研究。首先，我们建立了一个从sinc（x）函数中采样得到的数据集，并引入了两项人为噪声：高斯噪声分量ε∽N（0，σ2）和脉冲噪声分量ξ∽Bern（p）。

加入脉冲噪声是为了说明模型的鲁棒效果。以下是使用不同损失函数拟合GBM回归器的结果。

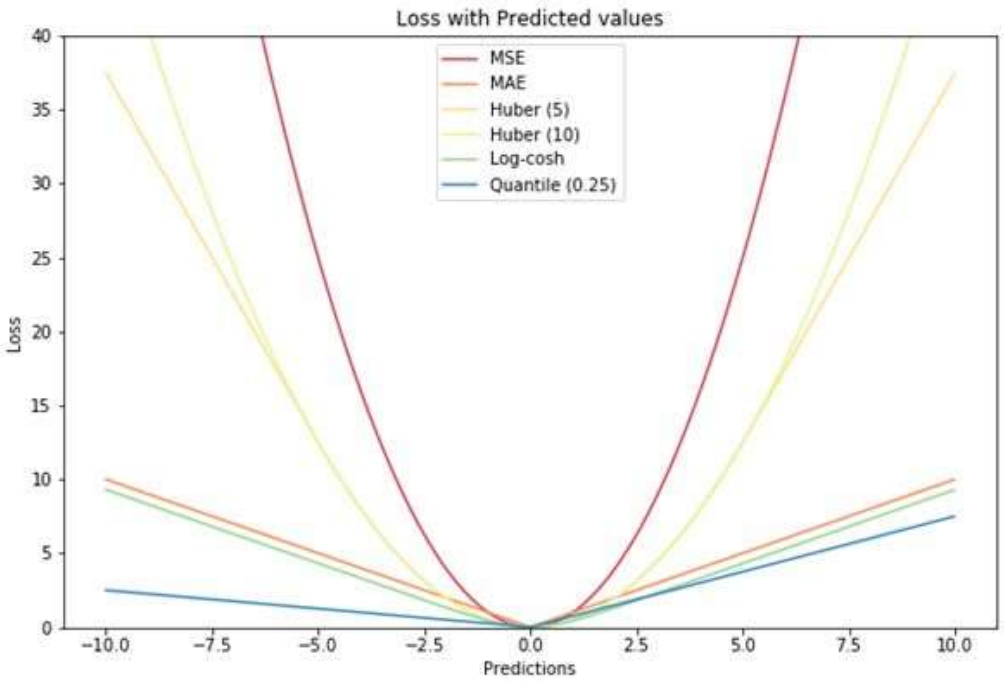


连续损失函数：（A）MSE损失函数；（B）MAE损失函数；（C）Huber损失函数；（D）分位数损失函数。将一个平滑的GBM拟合成有噪声的sinc（x）数据的示例：（E）原始sinc（x）函数；（F）具有MSE和MAE损失的平滑GBM；（G）具有Huber损失的平滑GBM，且δ={4,2,1}；（H）具有分位数损失的平滑的GBM，且α={0.5,0.1,0.9}。

仿真对比的一些观察结果：

- MAE损失模型的预测结果受脉冲噪声的影响较小，而MSE损失函数的预测结果受此影响略有偏移。
- Huber损失模型预测结果对所选超参数不敏感。
- 分位数损失模型在合适的置信水平下能给出很好的估计。

最后，让我们将所有损失函数都放进一张图，我们就得到了下面这张漂亮的图片！它们的区别是不是一目了然了呢~



本文出现的代码和图表保存位置：

https://nbviewer.jupyter.org/github/groverpr/Machine-Learning/blob/master/notebooks/05_Loss_Functions.ipynb

相关报道：

https://heartbeat.fritz.ai/5-regression-loss-functions-all-machine-learners-should-know-4fb140e9d4b0