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

<https://medium.com/human-in-a-machine-world/mae-and-rmse-which-metric-is-better-e60ac3bde13d>

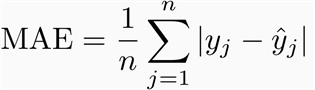
Mean Absolute Error (MAE) and Root mean squared error (RMSE) are two of the most common metrics used to measure accuracy for continuous variables. Not sure if I’m imagining it but I think there used to be a time when there were a lot more published MAE results. It seems that publications I come across now mostly use either RMSE or some version of R-squared.

Is RMSE actually better in most cases? When would it be better to use MAE? I wanted to dig into these two questions a bit because I find myself using RMSE often because it’s been programmed as the default modeling metric.

## **Definitions**

**Mean Absolute Error (MAE):** MAE measures the average magnitude of the errors in a set of predictions, without considering their direction. It’s the average over the test sample of the absolute differences between prediction and actual observation where all individual differences have equal weight.

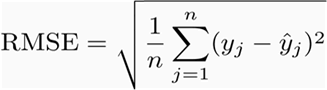




*If the absolute value is not taken (the signs of the errors are not removed), the average error becomes the Mean Bias Error (MBE) and is usually intended to measure average model bias. MBE can convey useful information, but should be interpreted cautiously because positive and negative errors will cancel out.*

**Root mean squared error (RMSE)**: RMSE is a quadratic scoring rule that also measures the average magnitude of the error. It’s the square root of the average of squared differences between prediction and actual observation.



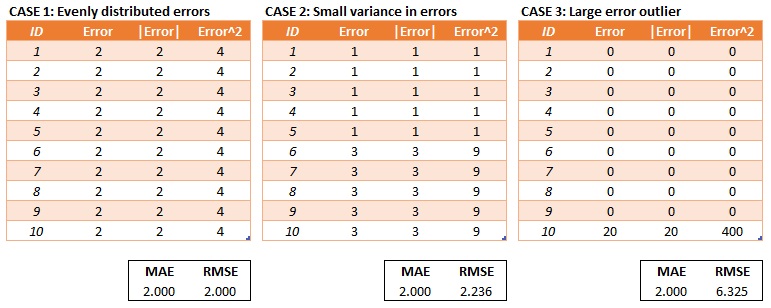


## **Comparison**

**Similarities**: Both MAE and RMSE express average model prediction error in units of the variable of interest. Both metrics can range from 0 to ∞ and are indifferent to the direction of errors. They are negatively-oriented scores, which means lower values are better.

**Differences**: Taking the square root of the average squared errors has some interesting implications for RMSE. Since the errors are squared before they are averaged, the RMSE gives a relatively high weight to large errors. This means the RMSE should be more useful when large errors are particularly undesirable. The three tables below show examples where MAE is steady and RMSE increases as the variance associated with the frequency distribution of error magnitudes also increases.





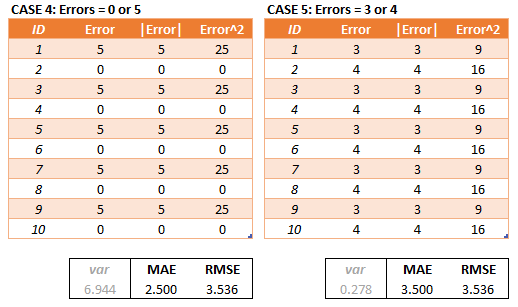
MAE and RMSE for cases of increasing error variance

The last sentence is a little bit of a mouthful but I think is often incorrectly interpreted and important to highlight.

*RMSE does not necessarily increase with the variance of the errors. RMSE increases with the variance of the frequency distribution of error magnitudes.*

To demonstrate, consider Case 4 and Case 5 in the tables below. Case 4 has an equal number of test errors of 0 and 5 and Case 5 has an equal number of test errors of 3 and 4. The variance of the errors is greater in Case 4 but the RMSE is the same for Case 4 and Case 5.





3,4,5 is a Pythagorean Triple

There may be cases where the variance of the frequency distribution of error magnitudes (still a mouthful) is of interest but in most cases (that I can think of) the variance of the errors is of more interest.

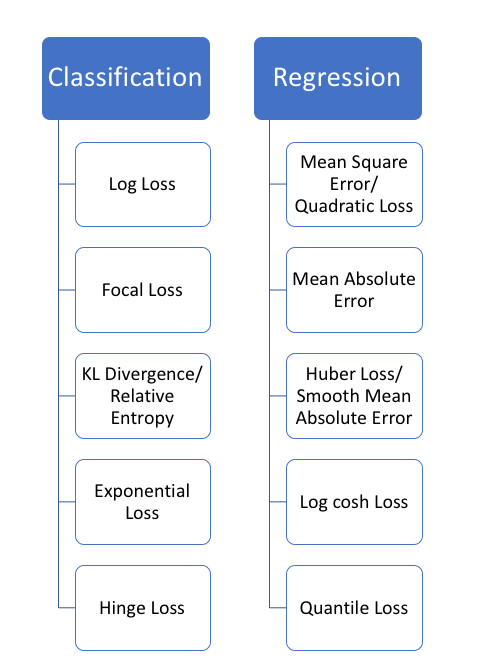
Another implication of the RMSE formula that is not often discussed has to do with sample size. Using MAE, we can put a lower and upper bound on RMSE.

1. **[MAE] ≤ [RMSE]**. The RMSE result will always be larger or equal to the MAE. If all of the errors have the same magnitude, then RMSE=MAE.
2. **[RMSE] ≤ [MAE \* sqrt(n)]**, where n is the number of test samples. The difference between RMSE and MAE is greatest when all of the prediction error comes from a single test sample. The squared error then equals to [MAE^2 \* n] for that single test sample and 0 for all other samples. Taking the square root, RMSE then equals to [MAE \* sqrt(n)].

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Loss functions can be broadly categorized into 2 types: **Classification and Regression Loss**. In this post, I’m focussing on regression loss. In future posts I cover loss functions in other categories. Please let me know in comments if I miss something. Also, all the codes and plots shown in this blog can be found in [this notebook.](https://nbviewer.jupyter.org/github/groverpr/Machine-Learning/blob/master/notebooks/05_Loss_Functions.ipynb)





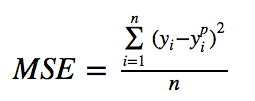
Regression functions predict a quantity, and classification functions predict a label.

# **Regression loss**

## **1. Mean Square Error, Quadratic loss, L2 Loss**

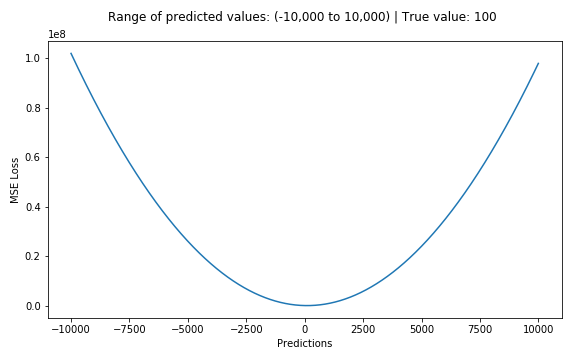
[Mean Square Error (MSE)](https://medium.freecodecamp.org/machine-learning-mean-squared-error-regression-line-c7dde9a26b93) is the most commonly used regression loss function. MSE is the sum of squared distances between our target variable and predicted values.





Below is a plot of an MSE function where the true target value is 100, and the predicted values range between -10,000 to 10,000. The MSE loss (Y-axis) reaches its minimum value at prediction (X-axis) = 100. The range is 0 to ∞.



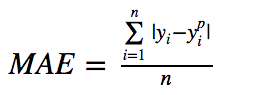


Plot of MSE Loss (Y-axis) vs. Predictions (X-axis)

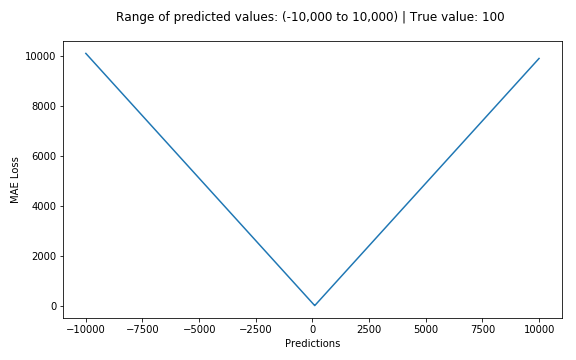
## **2. Mean Absolute Error, L1 Loss**

[Mean Absolute Error](https://medium.com/@ewuramaminka/mean-absolute-error-mae-sample-calculation-6eed6743838a) (MAE) is another loss function used for regression models. MAE is the sum of absolute differences between our target and predicted variables. So it measures the average magnitude of errors in a set of predictions, without considering their directions. (If we consider directions also, that would be called Mean Bias Error (MBE), which is a sum of residuals/errors). The range is also 0 to ∞.









Plot of MAE Loss (Y-axis) vs. Predictions (X-axis)

## **MSE vs. MAE (L2 loss vs L1 loss)**

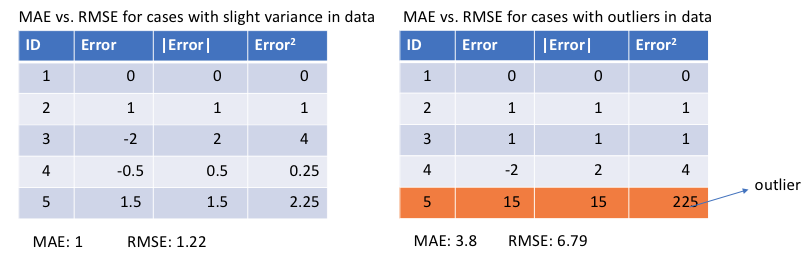
**In short,** **using the squared error is easier to solve, but using the absolute error is more robust to outliers. But let’s understand why!**

Whenever we train a machine learning model, our goal is to find the point that minimizes loss function. Of course, both functions reach the minimum when the prediction is exactly equal to the true value.

Here’s a quick review of python code for both. We can either write our own functions or use sklearn’s built-in metrics functions:

Let’s see the values of MAE and Root Mean Square Error (RMSE, which is just the square root of MSE to make it on the same scale as MAE) for 2 cases. In the first case, the predictions are close to true values and the error has small variance among observations. In the second, there is one outlier observation, and the error is high.





**Left:** Errors are close to each other **Right:** One error is way off as compared to others

**What do we observe from this, and how can it help us to choose which loss function to use?**

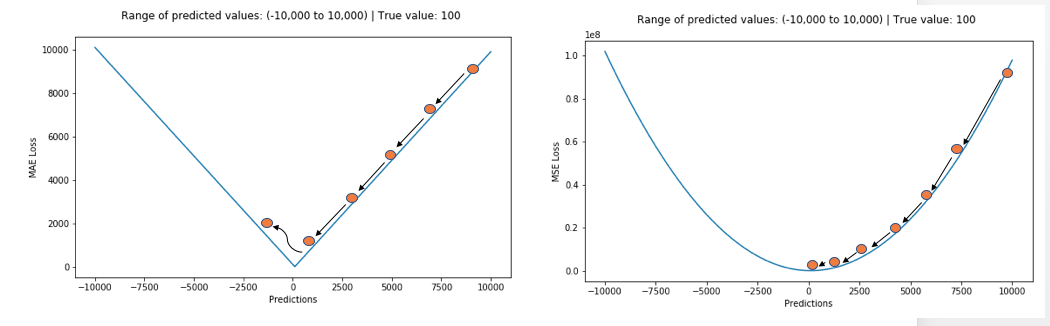
Since MSE squares the error (y — y\_predicted = e), the value of error (e) increases a lot if e > 1. If we have an outlier in our data, the value of e will be high and e² will be >> |e|. This will make the model with MSE loss give more weight to outliers than a model with MAE loss. In the 2nd case above, the model with RMSE as loss will be adjusted to minimize that single outlier case at the expense of other common examples, which will reduce its overall performance.

**MAE loss is useful** if the training data is corrupted with outliers (i.e. we erroneously receive unrealistically huge negative/positive values in our training environment, but not our testing environment).

Intuitively, we can think about it like this: If we only had to give one prediction for all the observations that try to minimize MSE, then that prediction should be the **mean** of all target values. But if we try to minimize MAE, that prediction would be the **median** of all observations. We know that median is more [robust to outliers](https://heartbeat.fritz.ai/how-to-make-your-machine-learning-models-robust-to-outliers-44d404067d07) than mean, which consequently makes MAE more robust to outliers than MSE.

**One big problem in using MAE loss** (for neural nets especially) is that its gradient is the same throughout, which means the gradient will be large even for small loss values. This isn’t good for learning. To fix this, we can use dynamic learning rate which decreases as we move closer to the minima. MSE behaves nicely in this case and will converge even with a fixed learning rate. The gradient of MSE loss is high for larger loss values and decreases as loss approaches 0, making it more precise at the end of training (see figure below.)





**Deciding which loss function to use**

If the outliers represent anomalies that are important for business and should be detected, then we should use MSE. On the other hand, if we believe that the outliers just represent corrupted data, then we should choose MAE as loss.

I recommend reading this post with a nice study [comparing the performance of a regression model using L1 loss and L2 loss](http://rishy.github.io/ml/2015/07/28/l1-vs-l2-loss/) in both the presence and absence of outliers. Remember, L1 and L2 loss are just another names for MAE and MSE respectively.

L1 loss is more robust to outliers, but its derivatives are not continuous, making it inefficient to find the solution. L2 loss is sensitive to outliers, but gives a more stable and closed form solution (by setting its derivative to 0.)

**Problems with both:** There can be cases where neither loss function gives desirable predictions. For example, if 90% of observations in our data have true target value of 150 and the remaining 10% have target value between 0–30. Then a model with MAE as loss might predict 150 for all observations, ignoring 10% of outlier cases, as it will try to go towards median value. In the same case, a model using MSE would give many predictions in the range of 0 to 30 as it will get skewed towards outliers. Both results are undesirable in many business cases.

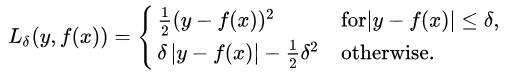
**What to do in such a case?** An easy fix would be to transform the target variables. Another way is to try a different loss function. This is the motivation behind our 3rd loss function, Huber loss.

The next evolution in machine learning will move models from the cloud to edge devices. But this process is tricky. [Luckily, Fritz AI has the developer tools you need to make this evolution possible.](https://www.fritz.ai/product/premium.html?utm_campaign=buildmodels3&utm_source=heartbeat)

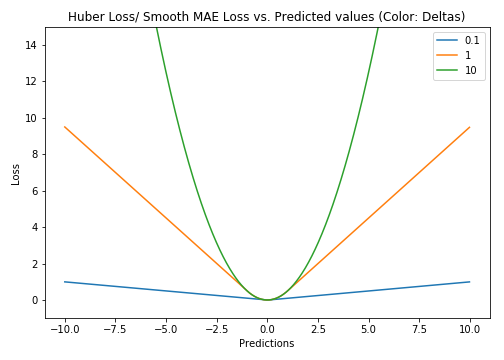
## **3. Huber Loss, Smooth Mean Absolute Error**

[Huber loss](https://en.wikipedia.org/wiki/Huber_loss) is less sensitive to outliers in data than the squared error loss. It’s also differentiable at 0. It’s basically absolute error, which becomes quadratic when error is small. How small that error has to be to make it quadratic depends on a hyperparameter, 𝛿 (delta), which can be tuned. Huber loss approaches **MAE when 𝛿 ~ 0 and MSE when 𝛿 ~ ∞ (large numbers.)**

****

****

****

****

Plot of Hoss Loss (Y-axis) vs. Predictions (X-axis). True value = 0

The choice of delta is critical because it determines what you’re willing to consider as an outlier. Residuals larger than delta are minimized with L1 (which is less sensitive to large outliers), while residuals smaller than delta are minimized “appropriately” with L2.

**Why use Huber Loss?**

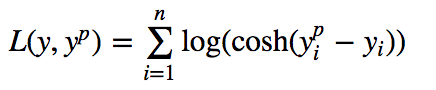
One big problem with using MAE for training of neural nets is its constantly large gradient, which can lead to missing minima at the end of training using gradient descent. For MSE, gradient decreases as the loss gets close to its minima, making it more precise.

Huber loss can be really helpful in such cases, as it curves around the minima which decreases the gradient. And it’s more robust to outliers than MSE. Therefore, it combines good properties from both MSE and MAE. However, the **problem with Huber loss** is that we might need to train hyperparameter delta which is an iterative process.

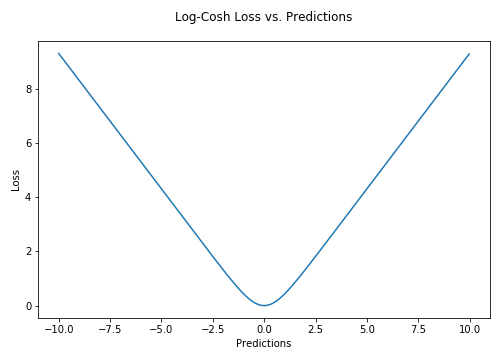
## **4. Log-Cosh Loss**

Log-cosh is another function used in regression tasks that’s smoother than L2. Log-cosh is the logarithm of the hyperbolic cosine of the prediction error.







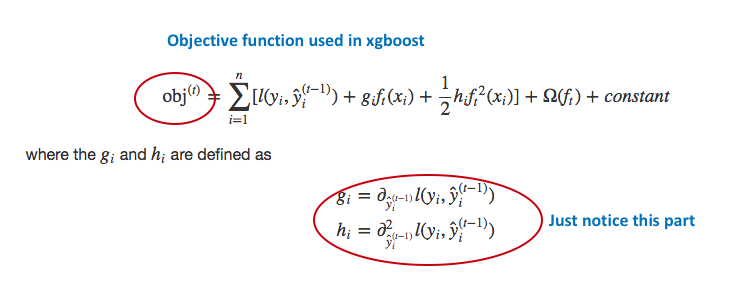


Plot of Log-cosh Loss (Y-axis) vs. Predictions (X-axis). True value = 0

**Advantage:** log(cosh(x)) is approximately equal to (x \*\* 2) / 2 for small x and to abs(x) - log(2) for large x. This means that 'logcosh' works mostly like the mean squared error, but will not be so strongly affected by the occasional wildly incorrect prediction. It has all the advantages of Huber loss, and it’s twice differentiable everywhere, unlike Huber loss.

**Why do we need a 2nd derivative?** Many ML model implementations like [XGBoost](https://heartbeat.fritz.ai/boosting-your-machine-learning-models-using-xgboost-d2cabb3e948f) use Newton’s method to find the optimum, which is why the second derivative (Hessian) is needed. For ML frameworks like XGBoost, twice differentiable functions are more favorable.





Objective function used in XgBoost. Notice dependency on both 1st and 2nd order derivative

But Log-cosh loss isn’t perfect. It still suffers from the problem of gradient and hessian for very large off-target predictions being constant, therefore resulting in the absence of splits for XGBoost.

Python code for Huber and Log-cosh loss functions:

Join more than 10,000 of your fellow machine learners and data scientists. [Subscribe to the premier newsletter for all things deep learning.](https://www.deeplearningweekly.com/newsletter?utm_campaign=dlweekly-newsletter-peers2&utm_source=heartbeat)

## **5. Quantile Loss**

In most of the real world prediction problems, we are often interested to know about the uncertainty in our predictions. Knowing about the range of predictions as opposed to only point estimates can significantly improve decision making processes for many business problems.