**Loss Function**

The loss function is a method of evaluating how well your machine learning algorithm models your featured data set. In other words, loss functions are a measurement of how good your model is in terms of predicting the expected outcome.

The cost function and loss function refer to the same context (i.e. the training process that uses backpropagation to minimize the error between the actual and predicted outcome). We calculate the cost function as the average of all loss function values whereas we calculate the loss function for each sample output compared to its actual value.

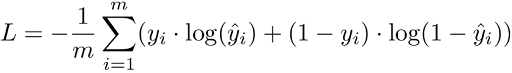
The loss function is directly related to the predictions of the model you’ve built. If your loss function value is low, your model will provide good results. The loss function (or rather, the cost function) you use to evaluate the model performance needs to be minimized to improve its performance.

Loss functions can be grouped into two major categories concerning the types of problems we come across in the real world: [**classification**](https://en.wikipedia.org/wiki/Loss_functions_for_classification) and **regression**. In classification problems, our task is to predict the respective probabilities of all classes the problem is dealing with. When it comes to regression, our task is to predict the continuous value concerning a given set of independent features to the learning algorithm.

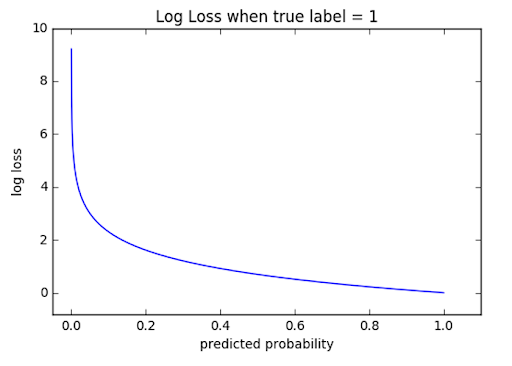
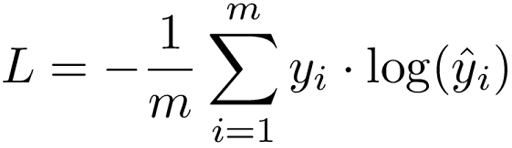
**1. BINARY CROSS-ENTROPY LOSS / LOG LOSS**

This is the most common loss function used in classification problems. The cross-entropy loss decreases as the predicted probability converges to the actual label. It measures the performance of a classification model whose predicted output is a probability value between 0 and 1.

When the number of classes is 2, it’s binary classification.



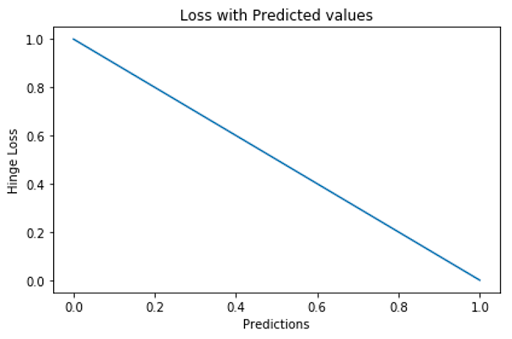
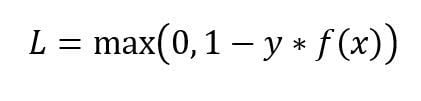
When the number of classes is more than 2, it’s multi-class classification.



We derive the cross-entropy loss formula from the regular likelihood function, but with logarithms added in.

**2. HINGE LOSS**

The second most common loss function used for classification problems and an alternative to the cross-entropy loss function is hinge loss, primarily developed for support vector machine (SVM) model evaluation.

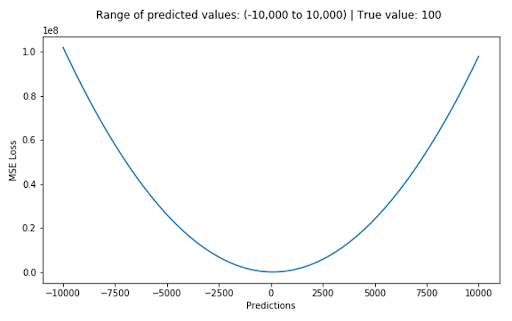
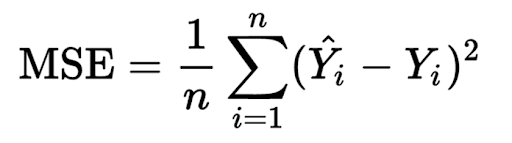


Hinge loss penalizes the wrong predictions and the right predictions that are not confident. It’s primarily used with SVM classifiers with class labels as -1 and 1. Make sure you change your malignant class labels from 0 to -1.

**Regression Losses**

**1. MEAN SQUARE ERROR / QUADRATIC LOSS / L2 LOSS**

We define MSE loss function as the average of squared differences between the actual and the predicted value. It’s the most used regression loss function.



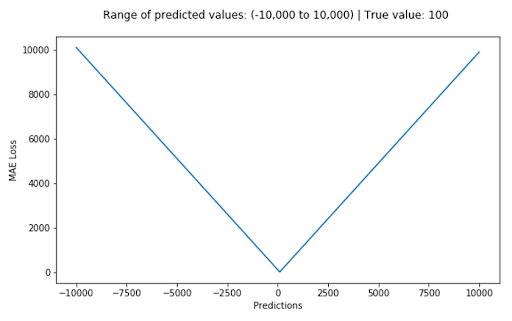
The corresponding cost function is the mean of these squared errors (MSE). The MSE loss function penalizes the model for making large errors by squaring them and this property makes the MSE cost function less robust to outliers. Therefore, you shouldn’t use it if the data is prone to many outliers.

**2. MEAN ABSOLUTE ERROR / L1 LOSS**

We define MAE loss function as the average of absolute differences between the actual and the predicted value. It’s the second most used regression loss function. It measures the average magnitude of errors in a set of predictions, without considering their directions.

Text

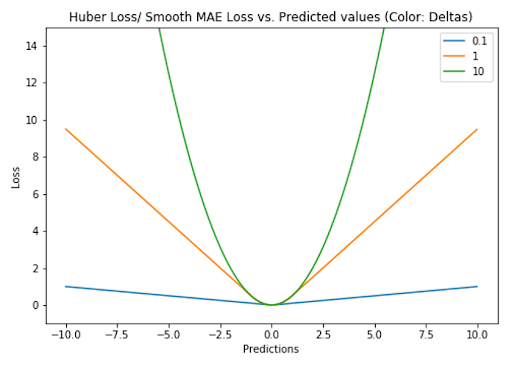
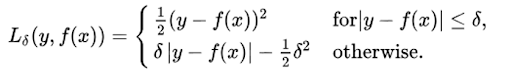
Description automatically generated with medium confidence



The corresponding cost function is the **mean** of these absolute errors (MAE)**. The MAE loss function is more robust to outliers compared to the MSE loss function.**Therefore, you *should* use it if the data is prone to many outliers.

**3. HUBER LOSS / SMOOTH MEAN ABSOLUTE ERROR**

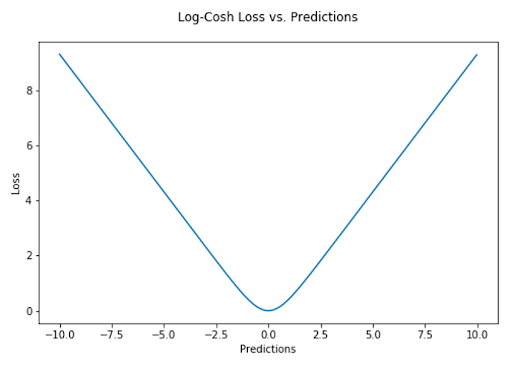
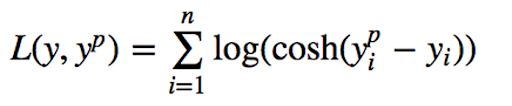
The Huber loss function is defined as the combination of MSE and MAE loss functions because it approaches MSE when 𝛿 ~ 0 and MAE when 𝛿 ~ ∞ (large numbers). It is mean absolute error, which becomes quadratic when the error is small. To make the error quadratic depends on how small that error could be, which is controlled by a hyperparameter, 𝛿 (delta) that you can tune.



The choice of the delta value is critical because it determines what you’re willing to consider an outlier. Hence, the Huber loss function could be less sensitive to outliers than the MSE loss function, depending on the hyperparameter value. Therefore, you can use the Huber loss function if the data is prone to outliers. In addition, we might need to train hyperparameter delta, which is an iterative process.

**4. LOG-COSH LOSS**

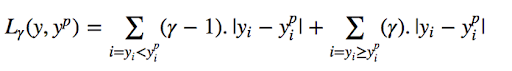
The log-cosh loss function is defined as the logarithm of the hyperbolic cosine of the prediction error. It’s another function used in regression tasks that’s much smoother than MSE loss. It has all the advantages of Huber loss because it’s twice differentiable everywhere, unlike Huber loss, because some learning algorithms like XGBoost use Newton’s method to find the optimum, and hence the second derivative (Hessian).

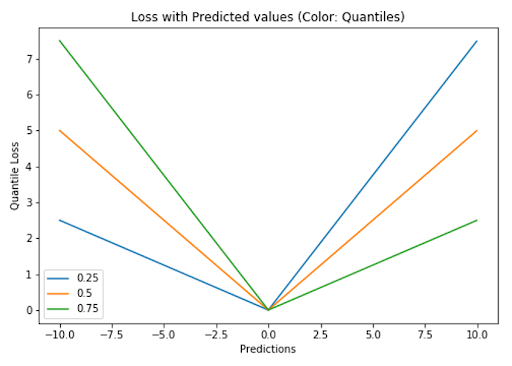


*“*[*Log(cosh(x))*](https://www.tensorflow.org/api_docs/python/tf/keras/losses/logcosh)*is approximately equal to (x \*\* 2) / 2 for small x and to abs(x) - log(2) for large x. This means that ‘logcosh’ works mostly like the mean squared error, but will not be so strongly affected by the occasional wildly incorrect prediction.”*

**5. QUANTILE LOSS**

A quantile is a value below which a fraction of samples in a group falls. Machine learning models work by minimizing (or maximizing) an objective function. As the name suggests, we apply the quantile regression loss function to predict quantiles. For a set of predictions, the loss will be its average.





[Quantile loss function](https://towardsdatascience.com/deep-quantile-regression-c85481548b5a) turns out to be useful when we’re interested in predicting an interval instead of only point predictions.