Loss Functions for Machine Learning

Function	Usage	Library / Implementation
Mean Absolute Error (MAE) **(L1 Norm Loss)** $$\mathrm{MAE} = \left(\frac{1}{n}\right) * \sum (y_i - \bar{y})$$	**REGRESSION** Less sensitive to outliers Easily interpretable Has constant gradient, slower convergence with gradient optimizing methods	**Scikit Learn :** scikit_learn.metrics.mean_absolute_error **PyTorch :** mae = torch.nn.L1Loss( )
Mean Squared Error (MSE) **(L2 Norm Loss)** $$\mathrm{MSE} = \left(\frac{1}{n}\right) * \sum (y_i - \bar{y})^2$$	**REGRESSION** Sensitive to large errors/outliers due to quadratic nature. Has smoother gradient.	**Scikit Learn :** scikit_learn.metrics.mean_squared_error **PyTorch :** mse_loss = torch.nn.MSELoss()
Smooth Mean Squared Error **(Huber Loss)** Large errors : $$L(\delta, y, f(x)) = \delta * \lvert f(x) - y \rvert - (\frac{1}{2}) * \delta^2$$ Small errors: $$L(\delta, y, f(x)) = \left(\frac{1}{2}\right) * (f(x) - y)^2$$	**REGRESSION** Combines advantages of MSE and MAE. Handles large and small errors differently based on parameter delta. Has medium impact with outliers.	**Scikit Learn :** from sklearn.linear_model import HuberRegressor **PyTorch :** huber_loss = torch.nn.SmoothL1Loss() **Tensorflow :** huber_loss = tf.keras.losses.Huber()
Binary Cross-Entropy Loss **(Log Loss)** $$L(y, f(x)) = -[y * log(f(x)) + (1 - y) * log(1 - f(x))]$$	**BINARY CLASSIFICATION** Mostly used for classifying elements into two classes. Ideal for models which output probabilities.	**Scikit Learn :** from sklearn.metrics import log_loss **PyTorch :** bce_loss = nn.BCELoss() **OR** bce_logits_loss = nn.BCEWithLogitsLoss() **Tensorflow :** bce = tf.keras.losses.BinaryCrossentropy()
Categorical Cross -Entropy Loss **(Softmax Loss)** $$Loss = -\sum_{i=1}^{n} \sum_{c=1}^{C} y_{i,c} \log(p_i, c)$$	**MULTI CLASS CLASSIFICATION** Ideal for models which output probabilities across various categories. Used for classifying elements into multiple	**Scikit Learn :** from sklearn.metrics import log_loss **PyTorch :** cross_entropy_loss = torch.nn.CrossEntropyLoss() **Tensorflow :** cross_entropy_loss = tf.keras.losses.CategoricalCrossentropy()

	classes, when labels are one-hot encoded.	
Sparse Categorical Cross -Entropy Loss $Loss = -\frac{1}{n}\sum_{i=1}^{n} log(p_i, y_i)$	**MULTI CLASS CLASSIFICATION** Used for classifying elements into multiple classes, when labels are integers, not one-hot encoded.	**PyTorch :** cross_entropy_loss = torch.nn.CrossEntropyLoss() **Tensorflow :** sparse_cross_entropy_loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
Hinge Loss $Loss(y, f(x)) = max(0, 1 - y.f(x))$	**CLASSIFICATION (ESPECIALLY SVMs)** Focuses on maximizing the margin between data points (classes) and decision boundary.	**Scikit Learn :** from sklearn.metrics import hinge_loss **PyTorch :** hinge_loss = nn.HingeEmbeddingLoss() **Tensorflow :** loss=tf.keras.losses.Hinge()
K-L Divergence Loss $D_{KL}(P\|\|Q) = \sum_i P_i.log(\frac{P(i)}{Q(i)})$	**FOR PROBABILITY DISTRIBUTIONS** Measure difference between two probability distributions. Used for variational autoencoders, etc.	**Scipy :** kl_divergence = numpy.sum(scipy.special.rel_entr(P, Q)) **PyTorch :** kl_divergence = torch.nn.Functional.kl_div(P.log(), Q) **Tensorflow :** from tensorflow.keras.losses import KLDivergence
Cosine Similarity Loss $L(a, b) = -\frac{a.b}{\|\|a\|\|\|\|b\|\|}$	**COMPARING SIMILARITY OF VECTORS** Aims to maximize the cosine similarity between predicted and target vectors. Used in natural language processing, recommendation	**Scikit Learn :** from sklearn.metrics.pairwise import cosine_similarity **PyTorch / TensorFlow :** Use custom function
Adversarial Loss	**FOR GENERATIVE ADVERSARIAL NETWORKS** Used for training GANs, where discriminator tries to maximize loss function and generator tries to minimize it.	**Defined specifically for the task.**