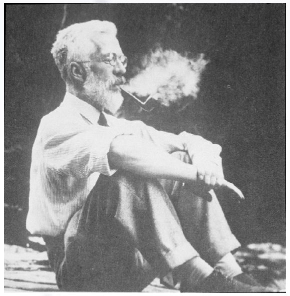
Applied ML

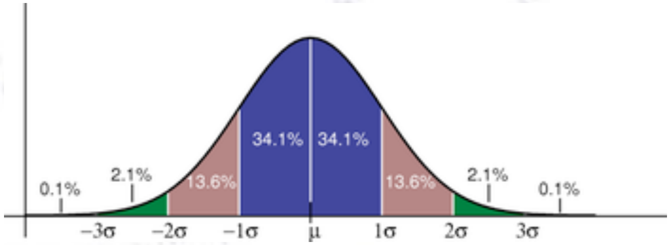
Loss Functions








Troels C. Petersen (NBI)



"Statistics is merely a quantisation of common sense - Machine Learning is a sharpening of it!"

What loss function to use?

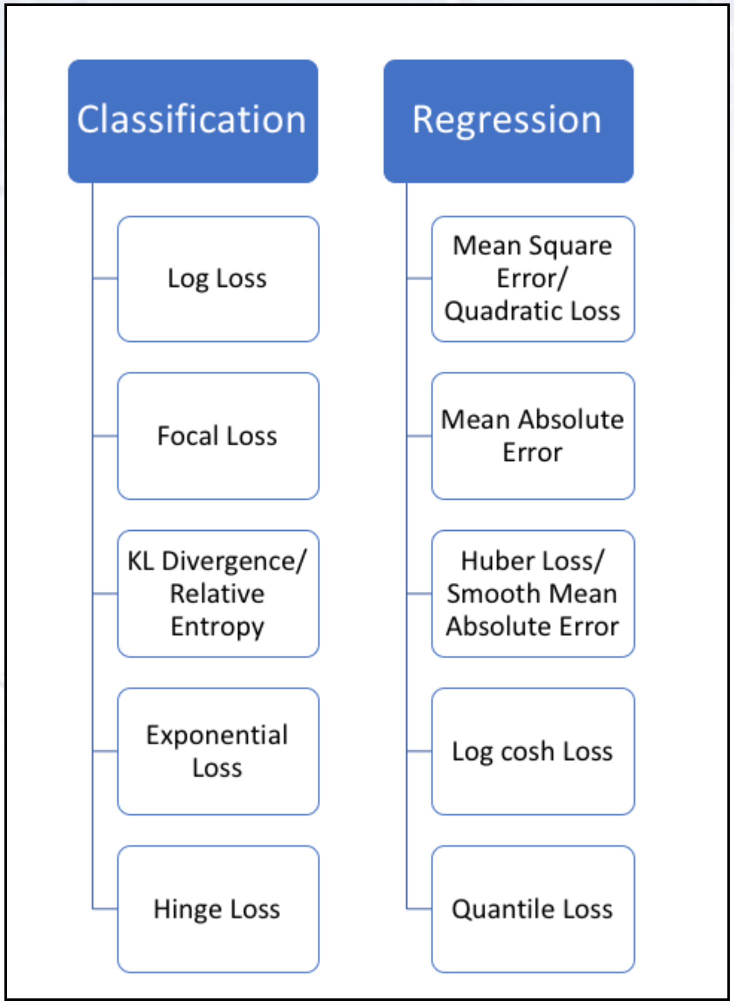
The choice of loss function depends on the problem at hand, and in particular what you find important!

<u>In classification:</u>

- Do you care how wrong the wrong are?
- Do you want pure signal or high efficiency?
- Does it matter what type of errors you make?

<u>In regression:</u>

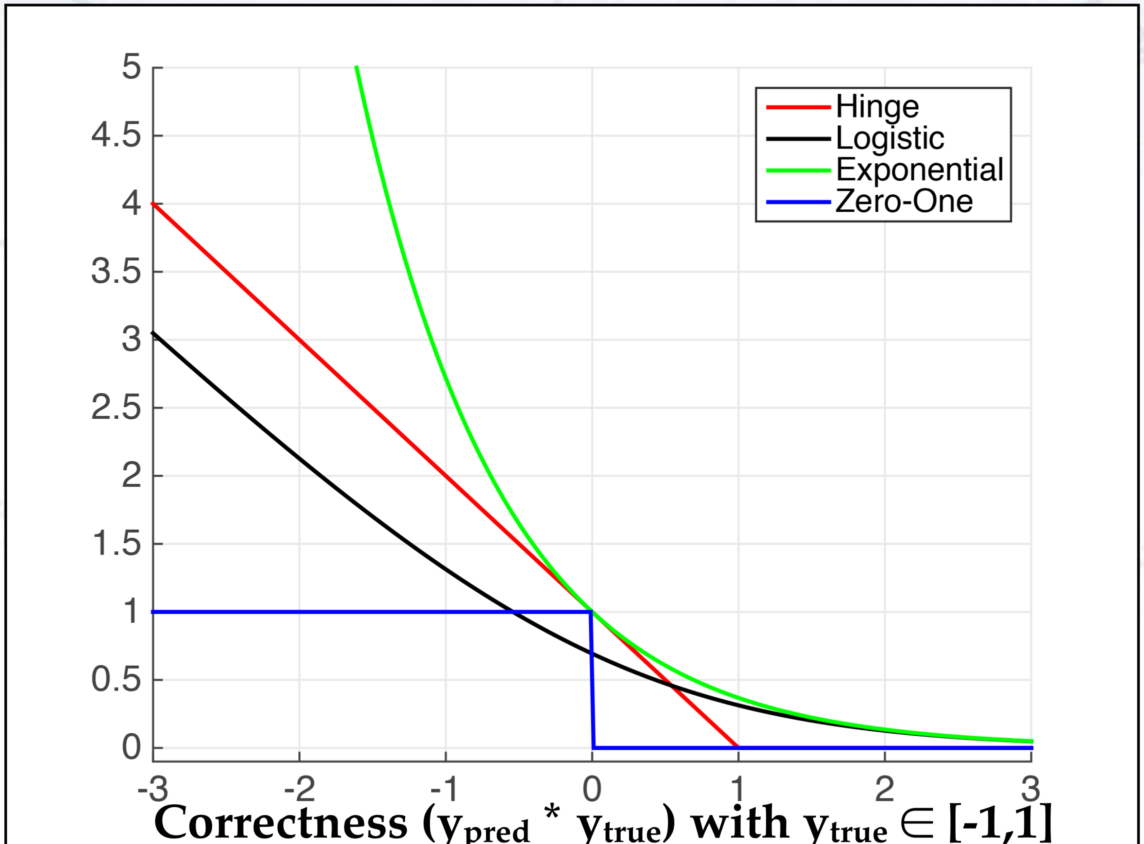
- Do you care about outliers?
- Do you care about size of outliers?
- Is core resolution vital?

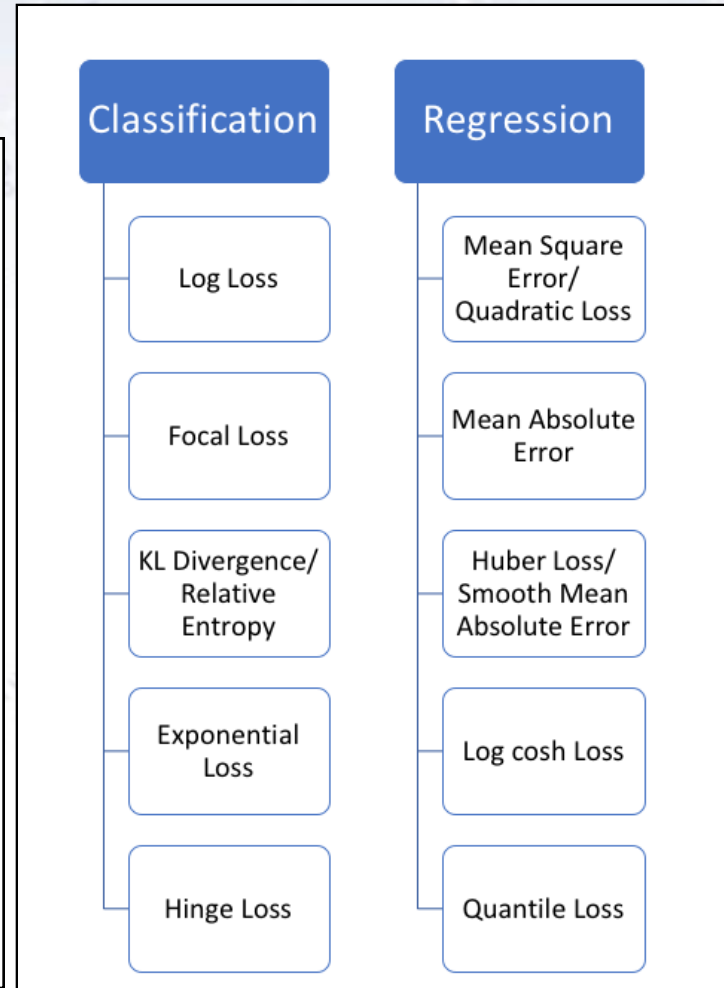


What loss function to use?

The choice of loss function depends on the problem at hand, and in particular what you find important!

Loss functions for classification

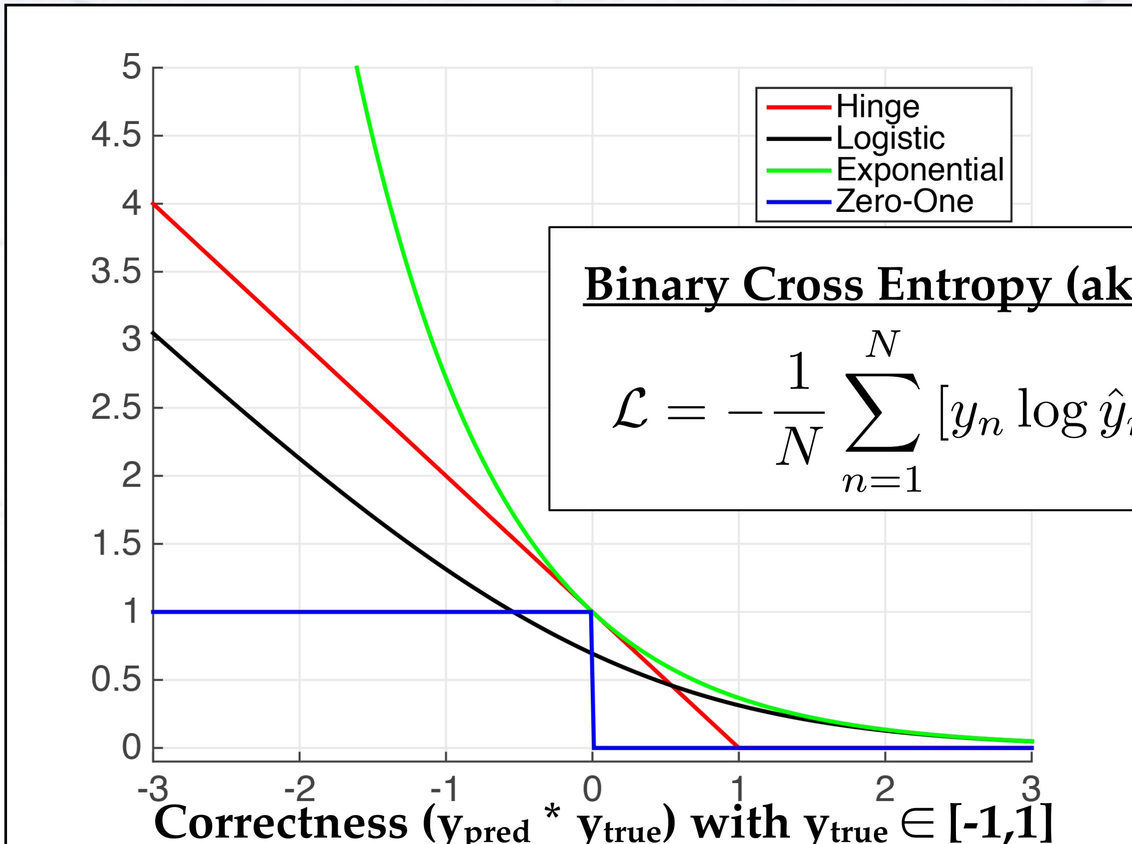


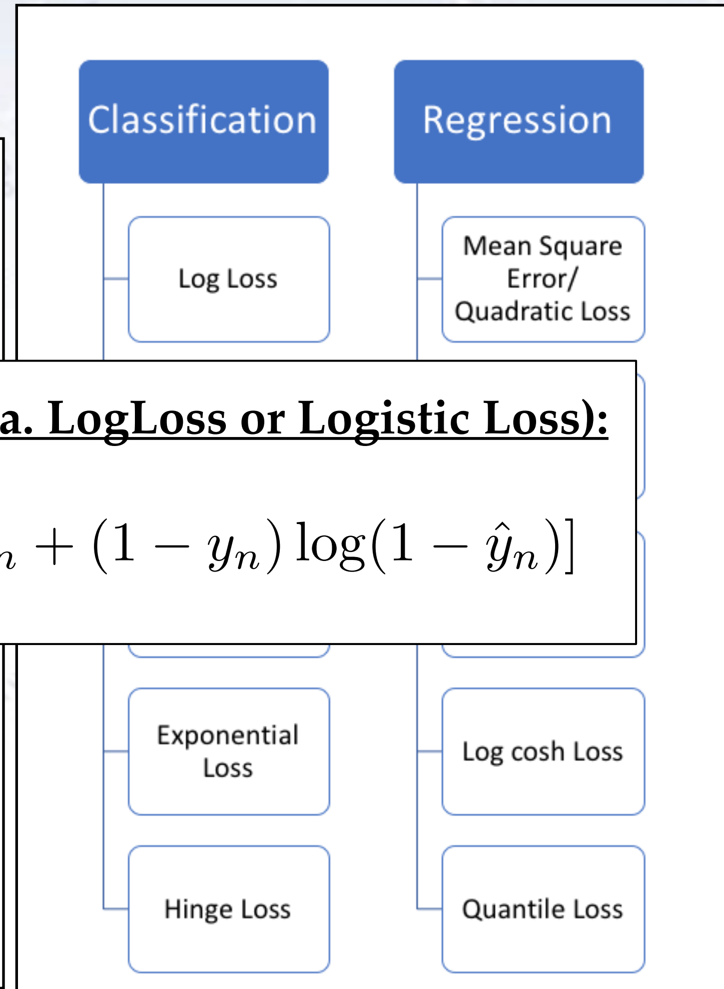


What loss function to use?

The choice of loss function depends on the problem at hand, and in particular what you find important!

Loss functions for classification





Binary Cross Entropy (aka. LogLoss or Logistic Loss):

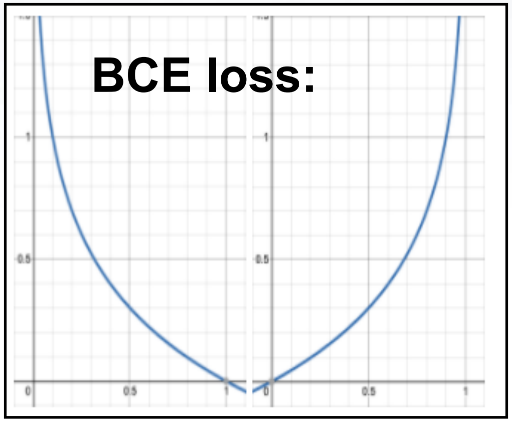
$$\mathcal{L} = -\frac{1}{N}\sum_{n=1}^{N}\left[y_n \log \hat{y}_n + (1-y_n)\log(1-\hat{y}_n)\right]$$

Unbalanced data

If the data is unbalanced, that is if one outcome/target is much more abundant than the alternative, case has to be taken.

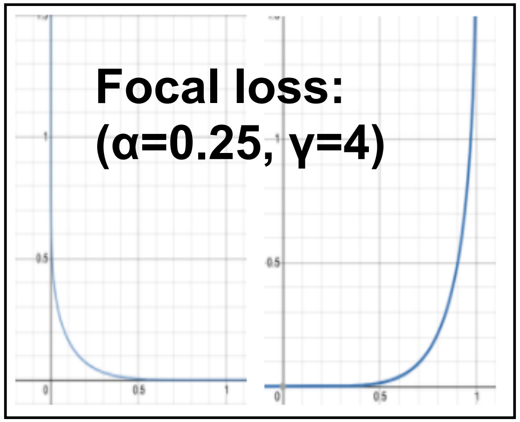
Example: You consider data with 19600 (98%) healthy and 400 (2%) ill patients. An algorithm always predicting "healthy" would get an accuracy score of 98%!

In this case, using Area Under Curve (AUC) or F1 for loss is better. An alternative is "focal loss", which focuses on the lesser represented cases:



Binary Cross Entropy loss:

$$\mathcal{L} = -\frac{1}{N}\sum_{n=1}^{N}\left[y_n \log \hat{y}_n + (1-y_n)\log(1-\hat{y}_n)\right]$$



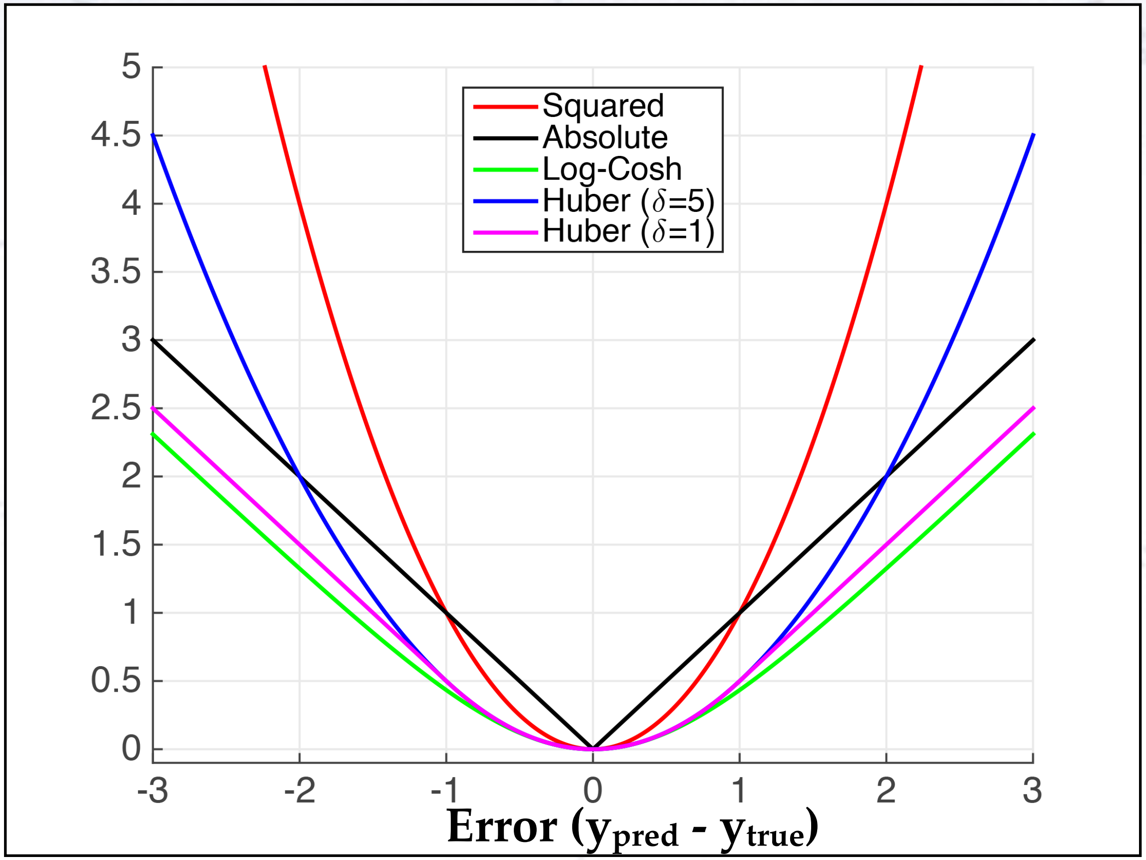
Focal loss:

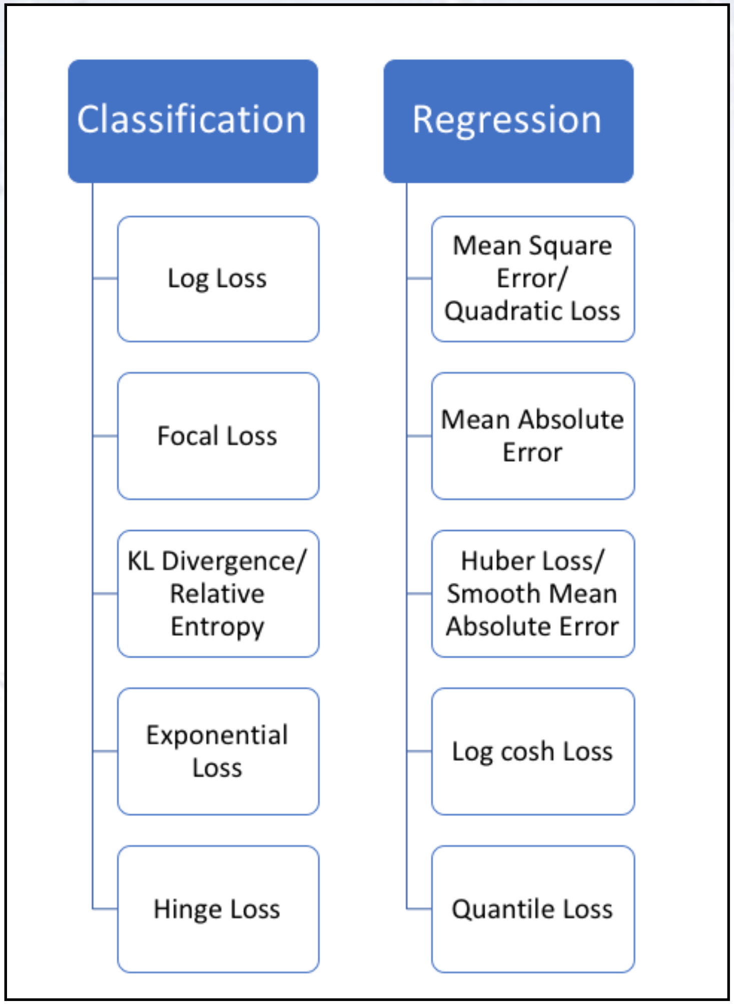
$$\mathcal{L} = -\frac{1}{N}\sum_{n=1}^{N}\left[(1-\alpha)y_n^{\gamma} \log \hat{y}_n + (1-y_n)^{\gamma}\log \alpha(1-\hat{y}_n)\right]$$

What loss function to use?

The choice of loss function depends on the problem at hand, and in particular what you find important!

Loss functions for regression



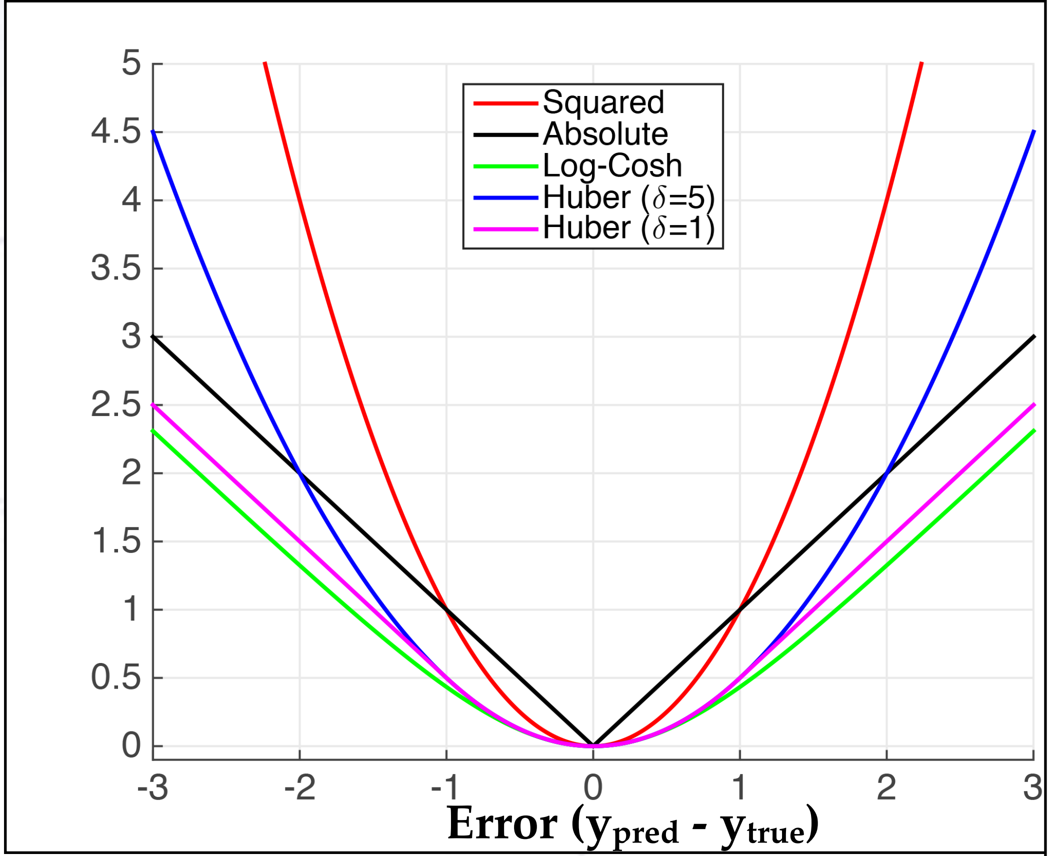


Discussion of regression loss functions

What loss function to use?

The choice of loss function depends on the problem at hand, and in particular what you find important!

Loss functions for regression



Squared Loss:

- Most popular regression loss function
- Estimates Mean Label
- ADVANTAGE: Differentiable everywhere
- DISADVANTAGE: Sensitive to outliers

Absolute Loss:

- Also a very popular loss function
- Estimates Median Label
- ADVANTAGE: Less sensitive to noise
- DISADVANTAGE: Not differentiable at 0

Huber Loss:

- ADVANTAGE: "Best of Both Worlds" of Squared and Absolute Loss.
- DISADVANTAGE: Only once-differentiable

LogCosh Loss:

- ADVANTAGE: "Best of Both Worlds" of Squared and Absolute Loss.
- ADVANTAGE: Similar to Huber Loss, but twice differentiable everywhere.

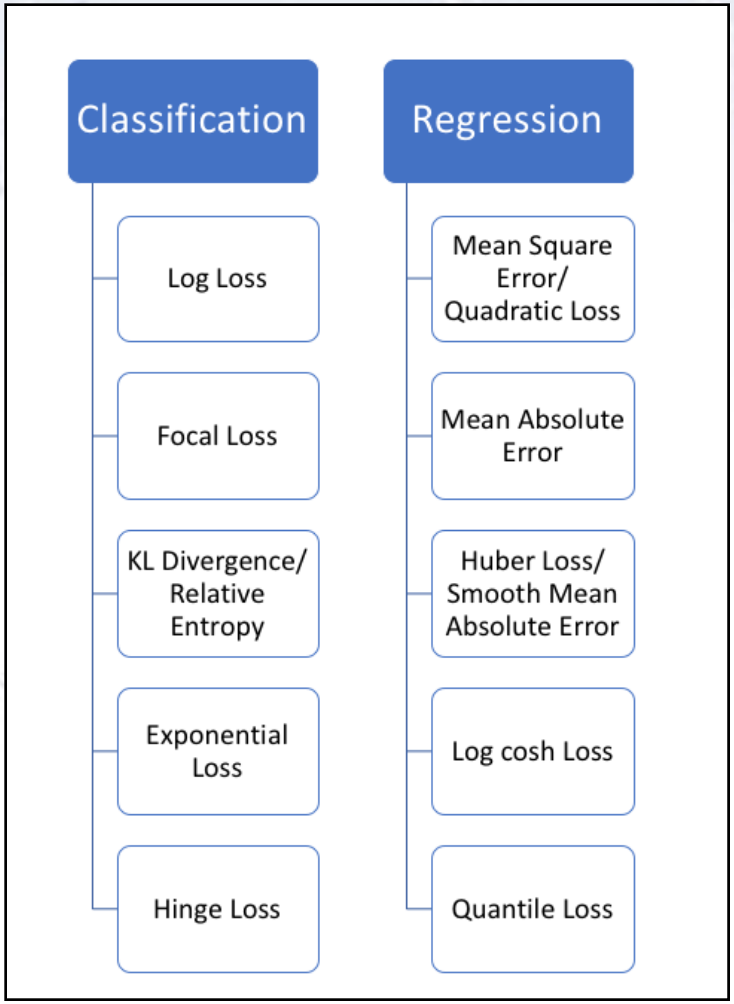
Discussion of regression loss functions

What loss function to use?

The choice of loss function depends on the problem at hand, and in particular what you find important!

<u>In classification:</u>
- Do you care how wrong the wrong are?
- Do you want pure signal or high efficiency?
- Does it matter what type of errors you make?

<u>In regression:</u>
- Do you care about outliers?
- Do you care about size of outliers?
- Is core resolution vital?



> Ultimately, the loss function should be tailored to match the wishes of the user. This is however not always that simple, as this might be hard to even know!

XGBoost algorithm

In order to "punish" complexity, the cost-function has a regularised term also:

$$\mathcal{L}(\phi) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k)$$

$$\text{where } \Omega(f) = \gamma T + \frac{1}{2}\lambda \|w\|^2$$

Table 1: Comparison of major tree boosting systems.

System	exact greedy	approximate global	approximate local	out-of-core	sparsity aware	parallel
XGBoost	yes	yes	yes	yes	yes	yes
pGBRT	no	no	yes	no	no	yes
Spark MLLib	no	yes	no	no	partially	yes
H2O	no	yes	no	no	partially	yes
scikit-learn	yes	no	no	no	no	no
R GBM	yes	no	no	no	partially	no

XGBoost algorithm

In order to "punish" complexity, the cost-function has a regularised term also:

$$\mathcal{L}(\phi) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k)$$

$$\text{where } \Omega(f) = \gamma T + \frac{1}{2}\lambda \|w\|^2$$

Table 1: Comparison of major tree boosting systems.

System	exact greedy	approximate global	approximate local	out-of-core	sparsity aware	parallel
XGBoo	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
pGBRT	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Spark M	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
H2O	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
scikit-learn	yes	no	no	no	no	no
R GBM	yes	no	no	no	partially	no

Generally, all constraints or priors should be included into the model through additions to the loss function.