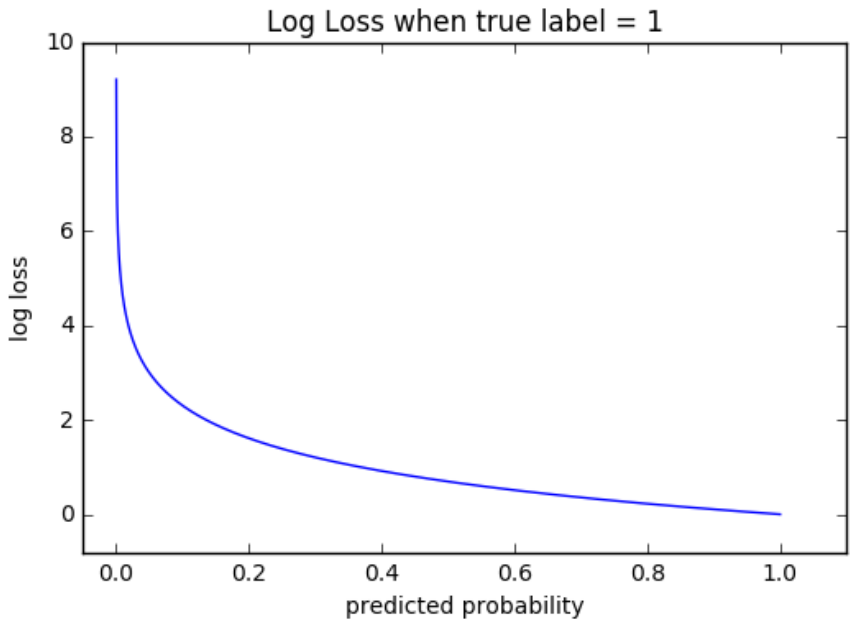
Cross Entropy

Cross-entropy loss, or log loss, measures the performance of a classification model whose output is a probability value between 0 and 1. Cross-entropy loss increases as the predicted probability diverges from the actual label. So predicting a probability of 0.012 when the actual observation label is 1 would be bad and result in a high loss value. A perfect model would have a log loss of 0.



The graph above shows the range of possible loss values given a true observation. As the predicted probability approaches 1, log loss slowly decreases. As the predicted probability decreases, however, the log loss increases rapidly. Log loss penalizes both types of errors, but especially those predictions that are confident and wrong.

Cross-entropy and log loss are slightly different depending on context, but in machine learning when calculating error rates between 0 and 1 they resolve to the same thing.

```
def CrossEntropy(y_hat, y):
    if y==1:
        return -log(y_hat)
    else:
        return -log(1-y_hat)
```

In binary classification, where the number of classes M equals 2, cross-entropy can be calculated as $-(y \cdot log(p) + (1-y) \cdot log(1-p))$ If M>2, we calculate a separate loss for each class label per observation and sum the result $-\sum_{c=1}^{M} y_{0,c} log(p_{0,c})$

Hinge Loss

This name comes from its shape of the figure. The equation is $L(m_i) = max(0, 1 - m_i(w))$, if it is classified correctly, the loss is zero, else the loss is $1 - m_i(w)$.

In machine learning, Hinge Loss can be used to solve Maximization of Margin. A representative problem is SVM. The initial SVM problem is as follows

$argmin_{w,\zeta} \frac{1}{2}||w||^2 + C\sum_i \zeta_i$ st. $\forall y_i w^T x_i \geq 1 - \zeta_i$ and $\zeta_i \geq 0$

We transform the constraint equation, and get $\zeta_i \geq 1 - y_i w^T x_i$. And we have $J(w) = \frac{1}{2}||w||^2 + C\sum_i max(0, 1 - y_i w^T x_i)$
$J(w) = \frac{1}{2}||w||^2 + C\sum_i max(0, 1 - m_i(w))$ $J(w) = \frac{1}{2}||w||^2 + C\sum_i L_{Hinge}(m_i)$

Thus, the loss function for SVM can be viewed as the summation of L2-norm and Hinge loss.

Squared Loss

Least Square Method is a kind of Linear Regression, and OLS transforms this problem to a convex optimization problem. In linear regression, it assumes the samples and noise are Gaussian distributed (Central Limit Theorem). Then, according to Maximum Likelihood Estimation, we can derive the least square equation. The principle of least square is: the best fitted curve should be to minimize the summation of distances from points to the regression curve, or say to minimize the sum of squares.

The standard for of Square Loss is $L(Y, f(X)) = (Y - f(x))^2$.

When the number of samples is n, the loss function is $L(Y, f(X)) = \sum_{i=1}^n (Y - f(X))^2$. $Y - f(X)$ is the residue, and the whole equation is the summation of square residue. The objective is to minimize the summation of square residue.

In practice, we use Mean Square Estimation, $MSE = \frac{1}{n}\sum_{i=1}^n (\tilde{Y}_i - Y_i)$

Exponentially Loss

The standard form of loss function is $L(Y, f(X)) = exp[-Y \cdot f(X)]$.

exp-loss is mainly used in Boosting algorithm. In AdaBoost algorithm, after m iterations, we have $f_m(x)$, $f_m(x) = f_{m-1}(x) + \alpha_m G_m(x)$

In each iteration, the objective of AdaBoost is to get the parameter α and G to minimize $argmin_{\alpha, G} = \sum_{i=1}^N exp[-y_i(f_{m-1}(x_i) + \alpha G(x_i))]$