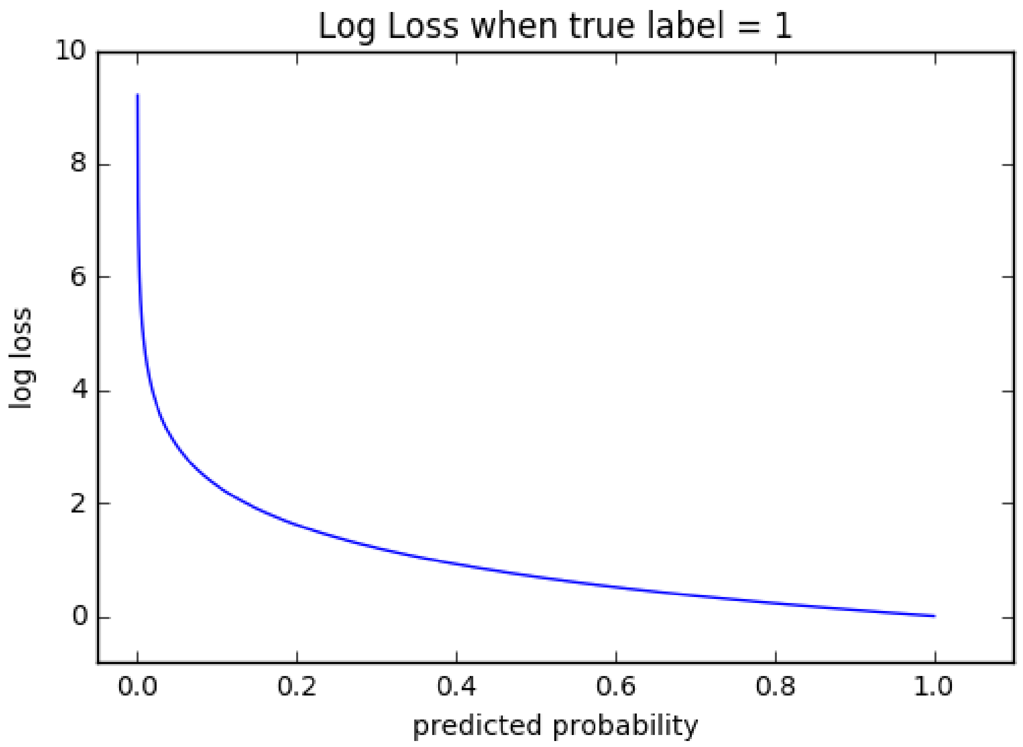
Loss Functions

ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html

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 .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 (isDog = 1). 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.

Code

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

Math

In binary classification, where the number of classes MM equals 2, cross-entropy can be calculated as:

−(ylog(p)+(1−y)log(1−p))−(ylog(p)+(1−y)log(1−p))
If M>2M>2 (i.e. multiclass classification), we calculate a separate loss for each class label per observation and sum the result.

−∑c=1Myo,clog(po,c)−∑c=1Myo,clog(po,c)
Note

- M - number of classes (dog, cat, fish)
- log - the natural log
- y - binary indicator (0 or 1) if class label cc is the correct classification for observation oo
- p - predicted probability observation oo is of class cc

Hinge

Used for classification.

Code

```
def Hinge(yHat, y):
    return np.max(0, 1 - yHat * y)
```

Huber

Typically used for regression. It’s less sensitive to outliers than the MSE as it treats error as square only inside an interval.

Lδ={12(y−y^)2δ((y−y^)−12δ)if|(y−y^)|<δotherwiseLδ={12(y−y^)2if|(y−y^)|<δδ((y−y^)−12δ)otherwise
Code

```
def Huber(yHat, y, delta=1.):
    return np.where(np.abs(y-yHat) < delta,.5*(y-yHat)**2 , delta*(np.abs(y-yHat)-0.5*delta))
```

Further information can be found at Huber Loss in Wikipedia.

Kullback-Leibler

Code

```
def KLDivergence(yHat, y):
    """
    :param yHat:
    :param y:
    :return: KLDiv(yHat || y)
    """
    return np.sum(yHat * np.log((yHat / y)))
```

MAE (L1)

Mean Absolute Error, or L1 loss. Excellent overview below [6] and [10].

Code

```
def L1(yHat, y):
    return np.sum(np.absolute(yHat - y))
```

MSE (L2)

Mean Squared Error, or L2 loss. Excellent overview below [6] and [10].

```
def MSE(yHat, y):
    return np.sum((yHat - y)**2) / y.size

def MSE_prime(yHat, y):
    return yHat - y
```