Log Loss

Logarithmic loss (related to cross-entropy) measures the performance of a classification model where the prediction input is a probability value between 0 and 1. The goal of our machine learning models is to minimize this value. A perfect model would have a log loss of 0. Log 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 log loss. There is a more detailed explanation of the justifications and math behind log loss here .

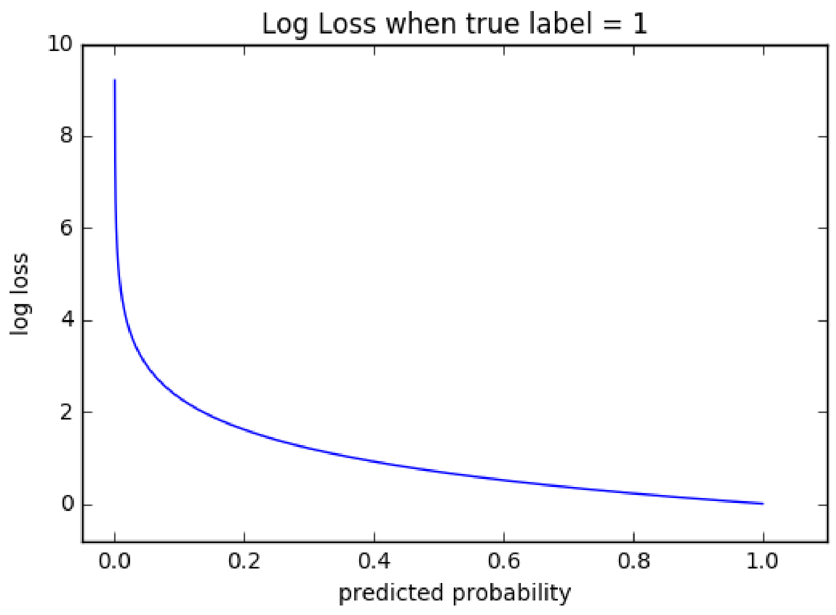
Contents [hide]

Log Loss vs Accuracy

- **Accuracy** is the count of predictions where your predicted value equals the actual value. Accuracy is not always a good indicator because of its yes or no nature.
- **Log Loss** takes into account the uncertainty of your prediction based on how much it varies from the actual label. This gives us a more nuanced view into the performance of our model.

Visualization

The graph below shows the range of possible log 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 predications that are confident and wrong!



Code

To calculate log loss from scratch, we need to include the MinMax function (see below). Numpy implements this for us with np.clip().

```
def logloss(true_label, predicted, eps=1e-15):
  p = np.clip(predicted, eps, 1 - eps)
  if true_label == 1:
    return -log(p)
  else:
    return -log(1 - p)
```

Scikit-learn provides a utility function for calculating log loss

```
from sklearn.metrics import log_loss

log_loss(y_true, y_pred, eps=1e-15)
```

Keras also provides a way to specify a loss function during model training. The calculation is run after every epoch. In our case we select categorical_crossentropy, which is another term for multi-class log loss.

```
model.compile(loss='categorical_crossentropy', optimizer='sgd')
```

Math

The equations below demonstrate how to calculate log loss for a single observation. When evaluating a model against a dataset, your log loss score is simply the average log loss across all observations.

Variables

- N - number of observations
- M - number of possible class labels (dog, cat, fish)
- log - the natural logarithm
- y - a binary indicator (0 or 1) of whether class label c is the correct classification for observation o
- p - the model's predicted probability that observation o is of class c

Binary Classification

In binary classification (M=2), the formula equals:

$$-(y\log(p) + (1-y)\log(1-p))$$

For example, given a class label of 1 and a predicted probability of .25, using the formula above we can calculate the log loss:

$$-(1\log(.25) + (1-1)\log(1-.25))$$
$$-(\log(.25) + 0\log(.75))$$
$$-\log(.25)$$

Similarly given a class label of 0 and a predicted probability of .25, we can calculate log loss as:

$$-(0\log(.25) + (1-0)\log(0-.25))$$
$$-(1\log(-.25))$$
$$-\log(-.25)$$

In Python we can express this even more simply:

```
def logloss(true_label, predicted_prob):
  if true_label == 1:
    return -log(predicted_prob)
  else:
    return -log(1 - predicted_prob)
```

Multi-class Classification

In multi-class classification (M>2), we take the sum of log loss values for each class prediction in the observation.

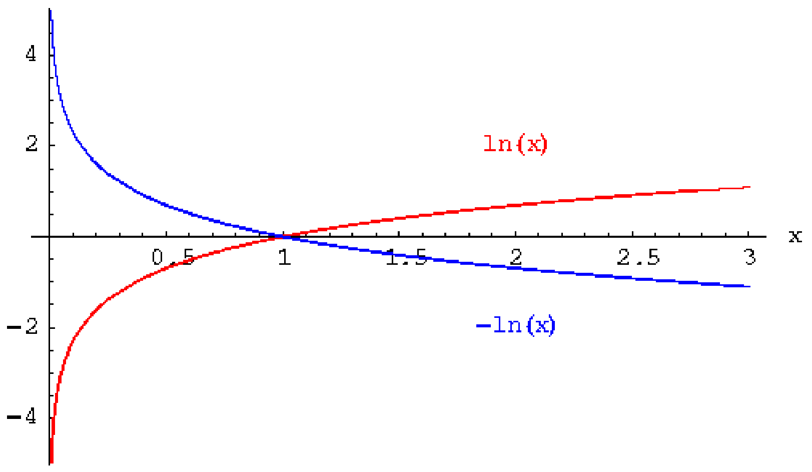
$$-\sum_{c=1}^{M} y_{o,c} \log(p_{o,c})$$

$\sum$ is shorthand for summation or in our case the sum of all log loss values across classes

$c = 1$ is the starting point in the summation (i.e. the first class)

Why the Negative Sign?

Log Loss uses negative log to provide an easy metric for comparison. It takes this approach because the positive log of numbers < 1 returns negative values, which is confusing to work with when comparing the performance of two models.



MinMax Rule

When calculating log loss using the formulas above, predicted input values of 0 and 1 are undefined. To avoid this problem, log loss functions typically adjust the predicted probabilities (p) by a small value (epsilon).

Formula

```
max(min(p, 1−10^−15), 10^−15)
```

Examples

p = 0

1. Apply min function (0 is smaller than 1 - 1e-15 --> 0)
2. Apply max function (1e-15 is larger than 0 --> 1e-15)
3. Thus, our submitted probability of 0 is converted to 1e-15 (~0.0000000000000001)

p = 1

1. Apply min function (1 - 1e-15 is smaller than 1 --> 1 - 1e-15 (~0.9999999999999999)
2. Apply max function (1 - 1e-15 is larger than 1e-15 —> 1 - 1e-15 (~0.9999999999999999))
3. Thus, our submitted probability of 1 is converted to 1 - 1e-15 (~0.9999999999999999)

Log Loss vs Cross-Entropy

Log loss and cross-entropy are slightly different depending on the context, but in machine learning when calculating error rates between 0 and 1 they resolve to the same thing. As a demonstration, where p and q are the sets p∈{y, 1−y} and q∈{ŷ, 1−ŷ} we can rewrite cross-entropy as:

$$H(p,q) = -\sum_i p_i \log q_i = -y \log \hat{y} - (1-y)\log(1-\hat{y})$$

- p = set of true labels
- q = set of prediction
- y = true label
- ŷ = predicted prob

Which is exactly the same as log loss!

Sources

- [1]
- [2]
- [3]
- [4]
- [5]