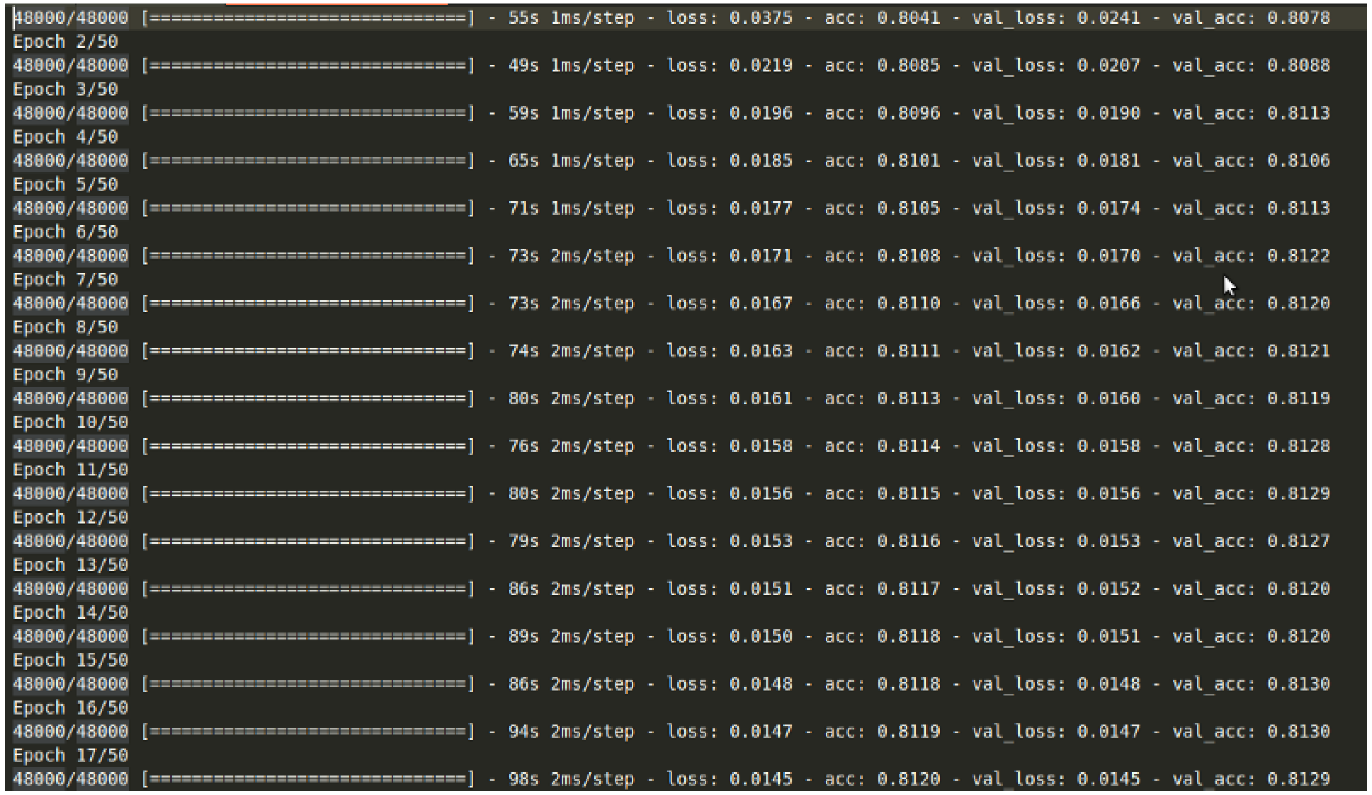
Metrics to Evaluate your Machine Learning Algorithm

Aditya Mishra Feb 24, 2018 · 7 min read



```
48000/48000 [==============================] - 55s 1ms/step - loss: 0.0375 - acc: 0.8041 - val_loss: 0.0241 - val_acc: 0.8078
Epoch 2/50
48000/48000 [==============================] - 49s 1ms/step - loss: 0.0219 - acc: 0.8085 - val_loss: 0.0207 - val_acc: 0.8088
Epoch 3/50
48000/48000 [==============================] - 59s 1ms/step - loss: 0.0196 - acc: 0.8096 - val_loss: 0.0190 - val_acc: 0.8113
Epoch 4/50
48000/48000 [==============================] - 65s 1ms/step - loss: 0.0185 - acc: 0.8101 - val_loss: 0.0181 - val_acc: 0.8106
Epoch 5/50
48000/48000 [==============================] - 71s 1ms/step - loss: 0.0177 - acc: 0.8105 - val_loss: 0.0174 - val_acc: 0.8113
Epoch 6/50
48000/48000 [==============================] - 73s 2ms/step - loss: 0.0171 - acc: 0.8108 - val_loss: 0.0170 - val_acc: 0.8122
Epoch 7/50
48000/48000 [==============================] - 73s 2ms/step - loss: 0.0167 - acc: 0.8110 - val_loss: 0.0166 - val_acc: 0.8120
Epoch 8/50
48000/48000 [==============================] - 74s 2ms/step - loss: 0.0163 - acc: 0.8111 - val_loss: 0.0162 - val_acc: 0.8121
Epoch 9/50
48000/48000 [==============================] - 80s 2ms/step - loss: 0.0161 - acc: 0.8113 - val_loss: 0.0160 - val_acc: 0.8119
Epoch 10/50
48000/48000 [==============================] - 76s 2ms/step - loss: 0.0158 - acc: 0.8114 - val_loss: 0.0158 - val_acc: 0.8128
Epoch 11/50
48000/48000 [==============================] - 80s 2ms/step - loss: 0.0156 - acc: 0.8115 - val_loss: 0.0156 - val_acc: 0.8129
Epoch 12/50
48000/48000 [==============================] - 79s 2ms/step - loss: 0.0153 - acc: 0.8116 - val_loss: 0.0153 - val_acc: 0.8127
Epoch 13/50
48000/48000 [==============================] - 86s 2ms/step - loss: 0.0151 - acc: 0.8117 - val_loss: 0.0152 - val_acc: 0.8120
Epoch 14/50
48000/48000 [==============================] - 89s 2ms/step - loss: 0.0150 - acc: 0.8118 - val_loss: 0.0151 - val_acc: 0.8120
Epoch 15/50
48000/48000 [==============================] - 86s 2ms/step - loss: 0.0148 - acc: 0.8118 - val_loss: 0.0148 - val_acc: 0.8130
Epoch 16/50
48000/48000 [==============================] - 94s 2ms/step - loss: 0.0147 - acc: 0.8119 - val_loss: 0.0147 - val_acc: 0.8130
Epoch 17/50
48000/48000 [==============================] - 98s 2ms/step - loss: 0.0145 - acc: 0.8120 - val_loss: 0.0145 - val_acc: 0.8129
```

Evaluating your machine learning algorithm is an essential part of any project. Your model may give you satisfying results when evaluated using a metric ***say accuracy_score*** but may give poor results when evaluated against other metrics such as ***logarithmic_loss*** or any other such metric. Most of the times we use classification accuracy to measure the performance of our model, however it is not enough to truly judge our model. In this post, we will cover different types of evaluation metrics available.

> *Classification Accuracy*
>
> *Logarithmic Loss*
>
> *Confusion Matrix*
>
> *Area under Curve*
>
> *F1 Score*
>
> *Mean Absolute Error*
>
> *Mean Squared Error*

Classification Accuracy

Classification Accuracy is what we usually mean, when we use the term accuracy. It is the ratio of number of correct predictions to the total number of input samples.

$$Accuracy = \frac{Number\ of\ Correct\ predictions}{Total\ number\ of\ predictions\ made}$$

It works well only if there are equal number of samples belonging to each class.

For example, consider that there are 98% samples of class A and 2% samples of class B in our training set. Then our model can easily get **98% training accuracy** by simply predicting every training sample belonging to class A.

When the same model is tested on a test set with 60% samples of class A and 40% samples of class B, then the **test accuracy would drop down to 60%.** Classification Accuracy is great, but gives us the false sense of achieving high accuracy.

The real problem arises, when the cost of misclassification of the minor class samples are very high. If we deal with a rare but fatal disease, the cost of failing to diagnose the disease of a sick person is much higher than the cost of sending a healthy person to more tests.

Logarithmic Loss

Logarithmic Loss or Log Loss, works by penalising the false classifications. It works well for multi-class classification. When working with Log Loss, the classifier must assign probability to each class for all the samples. Suppose, there are N samples belonging to M classes, then the Log Loss is calculated as below :

$$LogarithmicLoss = \frac{-1}{N} \sum_{i=1}^{N} \sum_{j=1}^{M} y_{ij} * \log(p_{ij})$$

where,

y_ij, indicates whether sample i belongs to class j or not

p_ij, indicates the probability of sample i belonging to class j

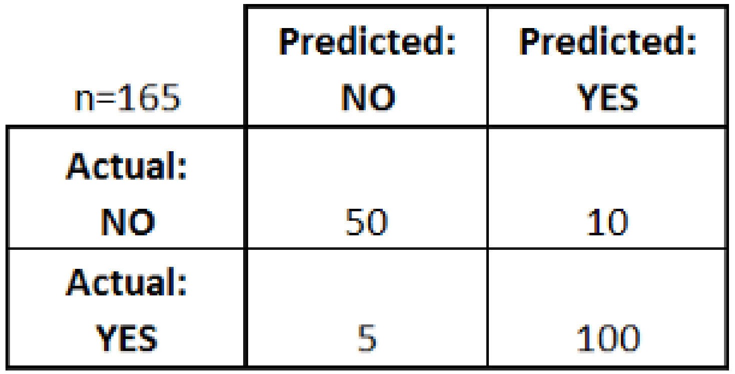
Log Loss has no upper bound and it exists on the range [0, ∞). Log Loss nearer to 0 indicates higher accuracy, whereas if the Log Loss is away from 0 then it indicates lower accuracy.

In general, minimising Log Loss gives greater accuracy for the classifier.

Confusion Matrix

Confusion Matrix as the name suggests gives us a matrix as output and describes the complete performance of the model.

Lets assume we have a binary classification problem. We have some samples belonging to two classes : YES or NO. Also, we have our own classifier which predicts a class for a given input sample. On testing our model on 165 samples ,we get the following result.



n=165	Predicted: NO	Predicted: YES
Actual: NO	50	10
Actual: YES	5	100

Confusion Matrix

There are 4 important terms :

- **True Positives** : The cases in which we predicted YES and the actual output was also YES.

- **True Negatives** : The cases in which we predicted NO and the actual output was NO.
- **False Positives** : The cases in which we predicted YES and the actual output was NO.
- **False Negatives** : The cases in which we predicted NO and the actual output was YES.

Accuracy for the matrix can be calculated by taking average of the values lying across the **“main diagonal”** i.e

$$Accuracy = \frac{TruePositive + TrueNegative}{TotalSample}$$

$$\therefore Accuracy = \frac{100 + 50}{165} = 0.91$$

Confusion Matrix forms the basis for the other types of metrics.

Area Under Curve

Area Under Curve(AUC) is one of the most widely used metrics for evaluation. It is used for binary classification problem. *AUC* of a classifier is equal to the probability that the classifier will rank a randomly chosen positive example higher than a randomly chosen negative example. Before defining *AUC*, let us understand two basic terms :

- **True Positive Rate (Sensitivity)** : True Positive Rate is defined as *TP/ (FN+TP)*. True Positive Rate corresponds to the proportion of positive data points that are correctly considered as positive, with respect to all positive data points.

$$TruePositiveRate = \frac{TruePositive}{FalseNegative + TruePositive}$$

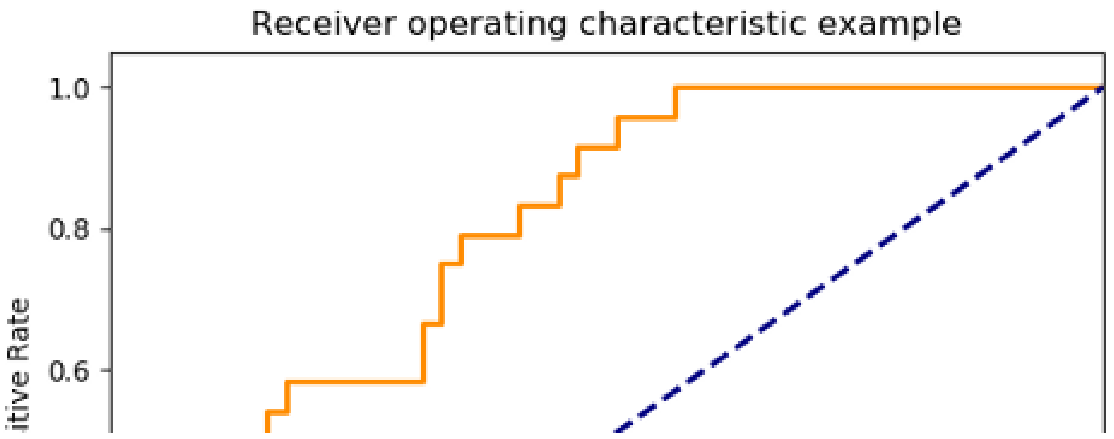
- **True Negative Rate (Specificity)** : True Negative Rate is defined as *TN / (FP+TN)*. False Positive Rate corresponds to the proportion of negative data points that are correctly considered as negative, with respect to all negative data points.

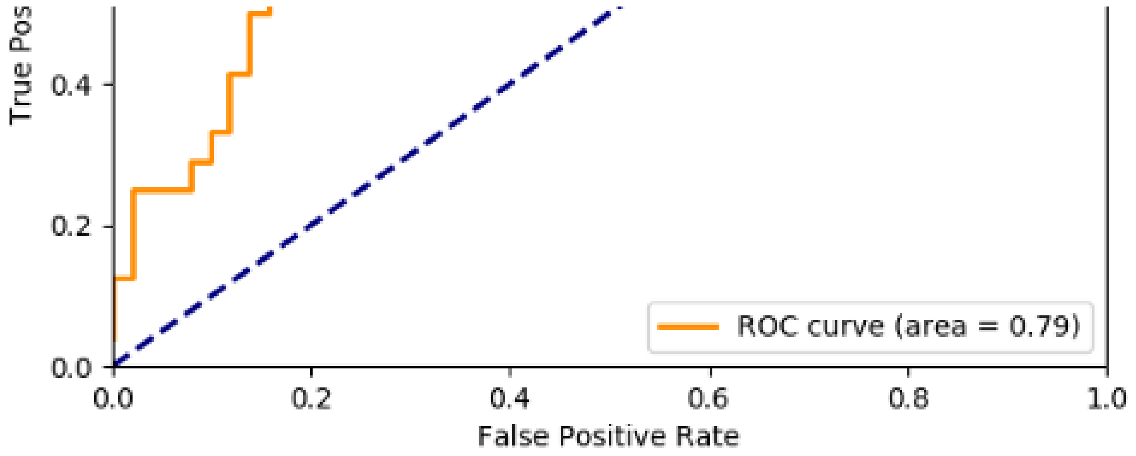
$$TrueNegativeRate = \frac{TrueNegative}{TrueNegative + FalsePositive}$$

- **False Positive Rate** : False Positive Rate is defined as *FP / (FP+TN)*. False Positive Rate corresponds to the proportion of negative data points that are mistakenly considered as positive, with respect to all negative data points.

$$FalsePositiveRate = \frac{FalsePositive}{TrueNegative + FalsePositive}$$

False Positive Rate and *True Positive Rate* both have values in the range **[0, 1]**. *FPR* and *TPR* both are computed at varying threshold values such as (0.00, 0.02, 0.04, …., 1.00) and a graph is drawn. *AUC* is the area under the curve of plot *False Positive Rate vs True Positive Rate* at different points in **[0, 1]**.





As evident, *AUC* has a range of [0, 1]. The greater the value, the better is the performance of our model.

F1 Score

> *F1 Score is used to measure a test's accuracy*

F1 Score is the Harmonic Mean between precision and recall. The range for F1 Score is [0, 1]. It tells you how precise your classifier is (how many instances it classifies correctly), as well as how robust it is (it does not miss a significant number of instances).

High precision but lower recall, gives you an extremely accurate, but it then misses a large number of instances that are difficult to classify. The greater the F1 Score, the better is the performance of our model. Mathematically, it can be expressed as :

$$F1 = 2 * \frac{1}{\frac{1}{precision} + \frac{1}{recall}}$$

F1 Score

F1 Score tries to find the balance between precision and recall.

- **Precision :** It is the number of correct positive results divided by the number of positive results predicted by the classifier.

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives}$$

Precision

- **Recall :** It is the number of correct positive results divided by the number of ***all*** relevant samples (all samples that should have been identified as positive).

$$Precision = \frac{TruePositives}{TruePositives + FalseNegatives}$$

Recall

Mean Absolute Error

Mean Absolute Error is the average of the difference between the Original Values and the Predicted Values. It gives us the measure of how far the predictions were from the actual output. However, they don’t gives us any idea of the direction of the error i.e. whether we are under predicting the data or over predicting the data. Mathematically, it is represented as :

$$MeanAbsoluteError = \frac{1}{N}\sum_{j=1}^{N}|y_j - \hat{y}_j|$$

Mean Squared Error

Mean Squared Error(MSE) is quite similar to Mean Absolute Error, the only difference being that MSE takes the average of the **square** of the difference between the original values and the predicted values. The advantage of MSE being that it is easier to compute the gradient, whereas Mean Absolute Error requires complicated linear programming tools to compute the gradient. As, we take square of the error, the effect of larger errors become more pronounced then smaller error, hence the model can now focus more on the larger errors.

$$MeanSquaredError = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

Mean Squared Error

That's it.

Thanks for reading. For any suggestion or queries, leave your comments below.

Edit

As many have pointed out, there were few errors in some of the terminologies. Guess, I should have double read the article before publishing it. Cheers!

References

Making Sense of Logarithmic Loss

Logarithmic Loss, or simply Log Loss, is a classification loss function often used as an evaluation metric in kaggle...

www.exegetic.biz

Simple guide to confusion matrix terminology

A confusion matrix is a table that is often used to describe the performance of a classification model (or...

www.dataschool.io

API Reference - scikit-learn 0.19.1 documentation

This is the class and function reference of scikit-learn. Please refer to the full user guide for further details, as...

scikit-learn.org

How to interpret F-measure values?

I would like to know how to interpret a difference of f-measure values. I know that f-measure is a balanced mean...

stats.stackexchange.com

What does AUC stand for and what is it?

Searched high and low and have not been able to find out what AUC, as in related to prediction, stands for or means.

stats.stackexchange.com

If you liked the article, please hit the 👏 icon to support it. This will help other Medium users find it. Share it, so that others can read it.

Machine Learning Sklearn Metrics Statistics