Documentation for Evaluation Metrics in Supervised Machine Learning Problems

---

Table of Content

Evaluation Metrics

An Overview

The idea of building ML models works on a constructive feedback principle. You build a model, get feedback from metrics, make improvements and continue until you achieve a desirable accuracy. Evaluation metrics explain the performance of a model. An important aspect of evaluation metrics is their capability to discriminate among model results.

Plenty of analysts and aspiring data scientists don't even bother to check how robust their model is. Once they are finished building a model, they hurriedly map predicted values on unseen data. This is an incorrect approach.

Simply building a predictive model is not our motive. It’s about creating and selecting a model which gives high accuracy on out of sample data. Hence, it is crucial to check the accuracy of your model prior to computing predicted values.

Now, when we talk about predictive models (or supervised ML problems), we are talking either about a regression model (continuous output) or a classification model (distinct output). The evaluation metrics used in each of these models are different.

Evaluation metrics for classification model

Confusion matrix

A confusion matrix is an N X N matrix, where N is the number of classes(distinct outputs) being predicted. The matrix compares the actual target values with those predicted by the machine learning model. This gives us a holistic view of how well our classification model is performing and what kinds of errors it is making. For a binary classification problem, we would have a 2 x 2 matrix.

Eg. For the infamous titanic dataset, the value of N will be 2 (survived: 1 or 0). The value of N may be more than 2 for some classification problems.

Let us visualise the confusion matrix with another example to better understand the concept. Suppose we had a classification dataset with 1000 data points. We fit a classifier on it and get the below confusion matrix:

		Predicted values	
		Positive	**Negative**
Actual values	**Positive**	560	60
	Negative	50	330

The values here are read as follows:

- True Positive (TP) = 560; meaning 560 positive class data points were correctly classified by the model
- True Negative (TN) = 330; meaning 330 negative class data points were correctly classified by the model
- False Negative (FN) = 60; meaning 60 actually positive class data points were incorrectly classified as belonging to the negative class by the model
- False Positive (FN) = 50; meaning 50 actually negative class data points were incorrectly classified as belonging to the positive class by the model

Now, For a multi-class classification problem, the no. of classes will be more than 2. In that case, the confusion matrix would look something like this:

		Predicted values		
		Class 1	Class 2	Class 3
Actual values	Class 1	cell 1	cell 2	cell 3
	Class 2	cell 4	cell 5	cell 6
	Class 3	cell 7	cell 8	cell 9

For this case, you can calculate the terminologies for each class as follows

	Class 1	Class 2	Class 3
TP	cell 1	cell 5	cell 9
FP	cell 4 + cell 7	cell 2 + cell 8	cell 3 + cell 6
TN	cell 5 + cell 6 + cell 8 + cell 9	cell 1 + cell 3 + cell 7 + cell 9	cell 1 + cell 2 + cell 4 + cell 5
FN	cell 2 + cell 3	cell 4 + cell 6	cell 7 + cell 8

Accuracy

It is the ratio of the number of correct predictions to the total number of input samples.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

It works well only if there are almost an 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.

On the contrary, It is best suited when the number of samples in each class are nearly the same.

Precision

It is the number of correct positive results divided by the number of positive results predicted by the classifier.

$$Precision = \frac{TP}{TP + FP}$$

Precision tells us how many of the positive predicted cases actually turned out to be positive. Let us take an example.

Suppose, you are a detective and you have arrived at a party. Now, you want to predict a group of criminals attending that party which is full of high-valued VIPs. If you wrongly select a VIP as a criminal, it is very bad for your image. So, it is very important that you rightfully select ‘only’ the criminal. This is the case where we require higher precision value.

So, we need to minimise the FP (I.e., arresting a VIP). This also means that there is a high chance of FN (I.e., criminals escaping).

Hence, Precision is a useful metric in cases where False Positive is a higher concern than False Negatives. I.e., when we cannot afford to have higher FP

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).

$$Recall = \frac{TP}{TP + FN}$$

Recall tells us how many of the actual positive cases we were able to predict correctly with our model. Let us take an example.

Suppose, at an airport metal detector, you(the security officer), has to check every person for which the metal detector rings. Now, whether that person is actually carrying any weapon or not, no matter what, if the detector rings for him, you check him. And the required case is that, no weapons should pass further from the metal detector. So, this is the case where we require higher recall value

So, we need to minimise the FN (I.e., undetected weapon passing further). This also means that there is a high chance of FP (I.e., checking actual innocent people for weapons).

Hence, Recall is a useful metric in cases where False Negative is a higher concern than False Positive. I.e., when we cannot afford to have higher FN

F1-score

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 :

$$F1 - score = \frac{2}{\frac{1}{Recall} + \frac{1}{Precision}}$$

F1 Score tries to find the balance between precision and recall. So, if you are unable to decide whether your model evaluation needs precision or recall, F1-score is the metric to go with.

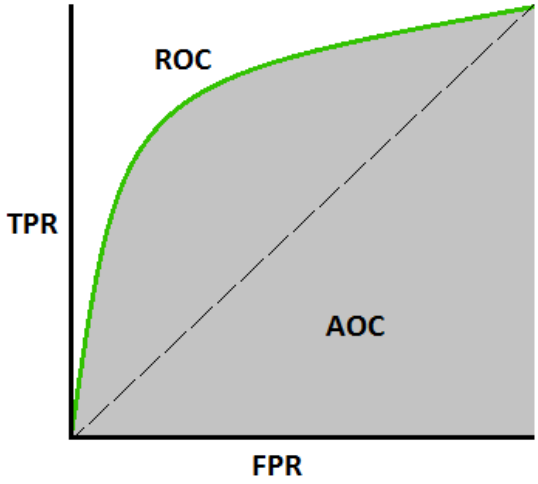
Since, F1-score is a harmonic mean of Precision and Recall, it gives a combined idea about these two metrics. It is maximum when Precision is equal to Recall.

But there is a catch here. The interpretability of the F1-score is quite poor. This means that we don't know what our classifier is maximizing – precision or recall? So, we use it in combination with other evaluation metrics which gives us a complete picture of the result.

AUC-ROC

AUC (Area Under The Curve) ROC (Receiver Operating Characteristics) curve. It is one of the most important evaluation metrics for checking any classification model's performance. It is also written as AUROC (Area Under the Receiver Operating Characteristics). It tells how much the model is capable of distinguishing between classes. Higher the AUC, the better the model is at predicting the classes correctly.

The ROC curve is plotted with TPR (True Positive Rate, TPR = TP / (TP + FN)) against the FPR (False Positive Rate, FPR = FP / (FP + TN)) where TPR is on the y-axis and FPR is on the x-axis.



An excellent model has AUC near to the 1 which means it has a good measure of separability. A poor model has AUC near to the 0 which means it has the worst measure of separability. In fact, it means it is reciprocating the result. I.e., It is predicting 0s as 1s and 1s as 0s. And when AUC is 0.5, it means the model has no class separation capacity whatsoever.

Note: This metric is only applicable for binary classification (i.e., when the number of classes is 2). If the target variable column is of object dtype (categorical column), then in order to view this metric, make sure that it is encoded (converted to numerical column) before making predictions.

Log-loss

Log-loss is one of the major metrics to assess the performance of a classification problem. Log-loss is indicative of how close the prediction probability is to the corresponding actual/true value (0 or 1 in case of binary classification). The more the predicted probability diverges from the actual value, the higher is the log-loss value.

$$Logloss = -\frac{1}{N}\sum_{i=1}^{N}[y_i \, ln \, p_i + (1 - y_i) \, ln(1 - p_i)]$$

where i is the given observation/record, N is total number of observations, y is the actual/true value, p is the prediction probability, and ln refers to the natural logarithm (logarithmic value using base of e) of a number.

A model with perfect skill has a log-loss score of 0. In other words, the perfect model predicts each observation's probability as the actual value. So, lesser the log-loss value, better the model performance. Note that, This metric is only applicable for binary classification (i.e., when the number of classes is 2)

Evaluation metrics for regression model

Mean Absolute Error (MAE)

An error basically is the absolute difference between the actual or true values and the values that are predicted. Absolute difference means that if the result has a negative sign, it is ignored. MAE takes the average of this error from every sample in a dataset and gives the output.

$$\text{MAE} = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j|$$

But this value might not be the relevant aspect that can be considered while dealing with a real-life situation because the data we use to build the model as well as evaluate it is the same, which means the model has no exposure to real, never-seen-before data. So, it may perform extremely well on seen data but might fail miserably when it encounters real, unseen data.
It is not very sensitive to outliers in comparison to MSE since it doesn't punish huge errors. It is usually used when the performance is measured on continuous variable data. It gives a linear

value, which averages the weighted individual differences equally. The lower the value, the better is the model's performance.

Mean Squared Error (MSE)

MSE is calculated by taking the average of the square of the difference between the original and predicted values of the data. The squaring also has the effect of inflating or magnifying large errors. That is, the larger the difference between the predicted and expected values, the larger the resulting squared positive error. This has the effect of “punishing” models more for larger errors when MSE is used as a loss function. It also has the effect of “punishing” models by inflating the average error score when used as a metric.

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

In most of the regression problems, mean squared error is used to determine the model's performance.

It is one of the most commonly used metrics, but least useful when a single bad prediction would ruin the entire model's predicting abilities, i.e when the dataset contains a lot of noise. It is most useful when the dataset contains outliers, or unexpected values (too high or too low values). The lower the value, the better is the model's performance.

Root Mean Squared Error (RMSE)

This is the same as MSE (Mean Squared Error) but the root of the value is considered while determining the accuracy of the model. The lower the value, the better is the model's performance.

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}$$

Importantly, the square root of the error is calculated, which means that the units of the RMSE are the same as the original units of the target value that is being predicted.

For example, if your target variable has the units “dollars,” then the RMSE error score will also have the unit “dollars” and not “squared dollars” like the MSE.

As such, it may be common to use MSE loss to train a regression predictive model, and to use RMSE to evaluate and report its performance.

RMSLE

In case of RMSLE, you take the log of the predictions and actual values. So basically, what changes is the variance that you are measuring. RMSLE is usually used when you don't want to penalize huge differences in the predicted and the actual values when both predicted and true values are huge numbers. The lower the value, the better is the model's performance.

$$\text{RMSLE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\log(y_i+1)-\log(\hat{y}_i+1))^2}$$

RMSLE has the meaning of a relative error, while RMSE is an absolute error. Choosing one depends on the nature of your problem. Imagine that the target spans values from around 1 to around 100. Is predicting *y=2* for true *y=1* as bad as *y=102* for *y=101* ? If not, then RMSLE might be a good choice. With RMSE, on the other hand, predicting *y=2* for *y=1* is as bad as *y=102* for *y=101*.

R2 (R-squared)

Alone, MAE or RMSE are not completely intuitive because we don’t have a benchmark to compare with. So, we take the R2 score. It is the coefficient of determination, scaled between 0 and 1. Here, SSR is nothing but the MSE of the current model whereas SST is the MSE of the baseline or benchmark model. In a baseline or benchmark model, every prediction is the mean value of all target values.

$$R^2 = 1 - \frac{SS_{Regression}}{SS_{Total}} = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}$$

R-squared is simply the fraction of response variance that is captured by the model. If R-squared = 1, means the model fits the data perfectly. It directly measures the goodness of fit in capturing the variance in training data.

For example : if R2=0.7, it says that with this model, we can explain 70% of what is going on in the real data, while the rest 30% can't be explained.

Say, if your R2 is in the range 0.35, then the model explains only 35% of the variance. This is not a good fit. Something in the range 0.7-0.8, is a good model.

But, the R-squared value alone isn't perfect. In fact, it suffers from a major flaw. Its value never decreases no matter the number of variables we add to our regression model. That is, even if we are adding redundant variables to the data, the value of R-squared does not decrease. It either remains the same or increases with the addition of new independent variables. This clearly does not make sense because some of the independent variables might not be useful in determining the target variable. Adjusted R-squared deals with this issue.

Adjusted R2

The Adjusted R-squared takes into account the number of independent variables(predictors) used for predicting the target variable. In doing so, we can determine whether adding new variables to the model actually increases the model fit.

$$R^2_{adjusted} = 1 - \frac{(1 - R^2)(N - 1)}{N - p - 1}$$

where

R^2 = sample R-square

p = Number of predictors

N = Total sample size.

So, if R-squared does not increase significantly on the addition of a new independent variable, then the value of Adjusted R-squared will actually decrease.

On the other hand, if on adding the new independent variable we see a significant increase in R-squared value, then the Adjusted R-squared value will also increase.

Clearly, it is better to use Adjusted R-squared when there are multiple variables in the regression model. This would allow us to compare models with differing numbers of independent variables.

*****