Authors : Dheeraj Poonia, Louis Paulet, Karim Djedidi

Validation Measures for Supervised Learning

Classification

- Accuracy

It is the ratio between correct predictions and total predictions.
This score is fast and easy to compute but it can be falsely good if there is an imbalance between classes.
ex : 99% classe A and 1% class B mix => if we fail to predict B but get a correct prediction for A most of the time, then the accuracy will be very high.

This is a score that you want to maximize.

- Logarithmic Loss

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

(with yij the correctness of the classification and pij the probability of being correct)

This score is very similar to the accuracy, but with the major difference being that it penalises wrong classifications based on their respective probability.

If we take the example of a mix of class A and B with a respective distribution of 99-1, detecting an error on a class B prediction will result in a 99% greater penalty than a class A false prediction (false positive or false negative).

This is a score that you want to minimize.

- Confusion Matrix

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

Example of Confusion Matrix

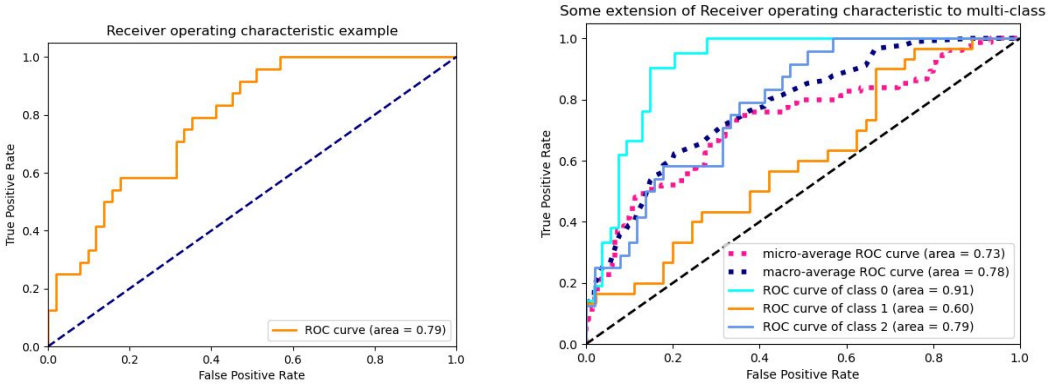
This is not a score but a table representing the distribution of predictions 2 axis : prediction of class A or B (here Yes or No), and what the reality is (should it have been an A class or a B class, here Yes and No).

In this validation method, you want to maximize the number of predictions located on the diagonal (true positives and negatives). You want to avoid being in the upper right hand corner (false positives) and lower left hand corner (false negatives) as these predictions are false.

You can also derive a score from this table by computing the average of the diagonal cells in relations to total predictions (although displaying

- Area under Curve (AUC)

This score is mostly used with binary classifiers as it relies on the ratio of true positives and true negatives, although it is possible to convert each label detection into a binary output and compare all the curves generated.



Single-class vs Multi-class AUC (source : https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html)

To obtain these graphs, and measure the area under the yellow curve and the default line of a random classifier (blue line), you can compute 3 metrics : sensitivity (rate of positive-class predictions), specificity (rate of negative-class predictions), and False Positive Rate.

For example, sensitivity will be the number of true positives divided by the sum of all positive predictions (true or false ones). Sam principle applies to the other curves.

The score is given by the area under the plotted curve relative to the diagonal line, which is the random limit (the performance of a classifier predicting with a 50% accuracy over 2 classes).

You want to maximize this score.

- Precision

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

This metric is an indication of how accurate a classifier is.
You want to maximize this score, as this means that the number of False positives will diminue.

- Recall

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

This score gives you an indication of robustness.
You want to maximize this score, as this means that the number of false Negatives will go down.

- F1 Score

This score is related to the previous 2 scores (recall and precision).

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

This kind of operation between precision and recall is called a harmonic mean (https://en.wikipedia.org/wiki/Harmonic_mean) and has the advantage of punishing the outlier values. This score is a good balance between precision and recall but doesn't take into account the ratio of False Negatives, so it has the same shortcomings as the Accuracy Score (what happens when the classes are strongly imbalanced?).

This score also needs to be maximized.

- Mean Squared Error (MSE)

$$MeanSquaredError = \frac{1}{N}\sum_{j=1}^{N}(y_j - \hat{y}_j)^2$$

To compute this score, you measure the difference between predicted class and true class, then square it, then compute the arithmetic mean. There is no difference between this score and the Mean Absolute Error (another score which uses an absolute value instead of a square) but the gradient can be computed faster so it is the most prevalent of the two methods.

One big disadvantage of this score is that the false-positive and false-negative errors are mixed together (so it might be a good idea to use a F1 score in addition).

You want to minimize this score as it gives you a value that increases with the number of wrong predictions.

Regression

- RMSE

Root Mean Square Error is the quadratic mean between expected predictions and actual predictions. It is also called RMSD (Root Mean Square Deviation) as the errors are deviations from the hypothesis.

$$\mathrm{RMSD} = \sqrt{\frac{\sum_{t=1}^{T}(\hat{y}_t - y_t)^2}{T}}.$$

Formula of RMSD, with T being the number of samples

RMSE gives more importance to large errors as the errors are squared. Another drawback of the squared value is that the positive and negative false predictions can cancel out.

We want to lower this score to achieve the highest accuracy, as it is the sum of errors.

- MAE

MAE stands for Mean Absolute Error. It is the absolute value of the error (or deviation, which is the delta between expected and predicted values) summed over all samples in the dataset.

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

MAE Formula, with N the number of samples

As the absolute value is taken, positive and negative errors are considered equal in this metric, and can cancel out (like the RMSE metric).
The advantage of this metric over RMSE is that the errors will damage the score linearly (an error that is at a distance x from the hypothesis will lower the score by x, when the RMSE will be impacted by a factor of x).

We want to lower this score to achieve the highest accuracy, as it is the sum of errors.

- MAPE

MAPE stands for Mean Absolute Percentage Error. The computing method is subtracting the true value by the predicted value, dividing by the true value, and computing the mean of abolutes.

$$\mathrm{M} = \frac{1}{n}\sum_{t=1}^{n} \left| \frac{A_t - F_t}{A_t} \right|$$

MAPE Formula, with n the number of samples

From the formula, we can see that the predicted values must be strictly different from 0 (otherwise a division by 0 would happen). This score penalises negative predictions over positive ones.

We want to lower this score (lowest percentage) to achieve the highest accuracy, as it is the sum of errors.

- R^2 validation

$$R^2 = 1 - \frac{SS_{\mathrm{res}}}{SS_{\mathrm{tot}}} \quad SS_{\mathrm{res}} = \sum_i (y_i - f_i)^2 = \sum_i e_i^2 \quad \text{with} \quad SS_{\mathrm{tot}} = \sum_i (y_i - \bar{y})^2$$

The coefficient of determination is a validation score that is roughly the residual sum of squares (SSres, the sum of the distance between each prediction and the truth) divided by the variance (how the data varies from record to record).

You want to maximize this score ($R^2 = 1$), as it means that the predictions match perfectly the input data (SSres = 0). If you have a negative result, you probably chose a wrong model because a simple hyperplane would have a better score.

- Out-of-sample Evaluation (or cross validation)

Train the same model over a mix of samples and test the error rate (with simple accuracy, or R^2 validation) on values outside of the training set. If the error rate depends on the sample used, the model is unstable. If the error rate of training data is much lower than the error rate on test data, it can reveal an overfitting problem.
There are two main methods for cross-validation :

- Hold-Out : cut the data in two : one training set, and one testing set. Randomly assign samples to one group of the other (with a ratio of 70/30 for training and testing respectively)
- K-Fold : cut the data into k-groups and randomly assign one of the groups to the testing phase.

Method 1 or 2 can be used to create training and testing groups. Once a model has been trained several times using the group creation method 1 or 2, you can analyse the error rate. If the error rate is consistent from one training to the other and the error rate between training set and test set is consistent, your model is good. If the error rate changes from one training to the other, the model is too dependent on the data, it can mean that the dataset is too small, or the model is too complex. If the training set over-performs the test set by a big margin, it reveals a problem of overfitting.

Experiments

Experiments have been coded (and run) the experiments on Google Colab because there is no environment installation needed and it is easy to share exactly output what we see. Just follow the links below to take a look. In case of permission problems, I(Louis Paulet) also have included PDFs of the whole page (code + results + analysis), but they lack syntaxic coloring.

Experiments in Python:

- Link to Mushroom SVM on Google Colab :

https://colab.research.google.com/drive/1fcEsMXGQ_f7_JD2-tlTIQBzDgMG4V_Hb?usp=sharing

- Link to Mushroom Tree Classifier on Google Colab :

https://colab.research.google.com/drive/1v9hmFMrh2P5_eBIFonX92sAkKuPBayB1?usp=sharing

- Link to simple Linear Regression with R^2 and MSE validation on Google Colab :

https://colab.research.google.com/drive/1bEQxYtpw-6ccrCyL5MFIQaD43atJBW8o?usp=sharing

Experiments in R

- Link to UNcloud folder containing the R notebooks (and pdfs with Plots and Graphs) for the diabetes dataset with Tree Classifier, SVM :

https://uncloud.univ-nantes.fr/index.php/s/fKsiATcXCbW5qLZ