**Evaluation metrics for Classification**

**Introduction**

**Evaluation metrics** is a generic term used to refer to a set of techniques to evaluate a machine learning model. It can differ from one task to another. In other words, based on whether you are dealing with the regression task or a classification task, the choice of the appropriate metric may differ.

Even within a given task, we may have different sort of metric which also may have different advantages and disadvantages basis on the nature of the data we are having.

Often, the metrics of regression tasks are much simpler to understand and implement compared to regression. We have already seen one metric for regression **(I.e. RMSE)** and one for classification **(I.e. Accuracy score)** in last chapters. But there are other metrics also and we need to have a separate discussion to choose appropriate metric for the classification problems.

There won’t be any CRISP-DM process for this chapter as we won’t be solving any problem in this chapter, but rather we will be looking at applying different metrics to the same problem we were tackling in the last chapter I.e. to predict the probability of a patient having a stroke.

I hope you are excited. So, let us get started.

**Objective of this chapter**

In this chapter, we are going see the various metrics for classification problems. We will see why accuracy as a metric can’t be relied upon too much and how can we improve our ability to evaluate our classification models.

Before we proceed further, open [**this**](https://mybinder.org/v2/gh/fenago/MLWorkshop/HEAD)link. This link will open an **online hosted Jupyter Lab environment that** can be used to run the codes discussed in this lab and try the exercises. You might have to wait a few minutes for it to fully load. This will have the labs, exercises and project notebooks of all the chapters in this book.

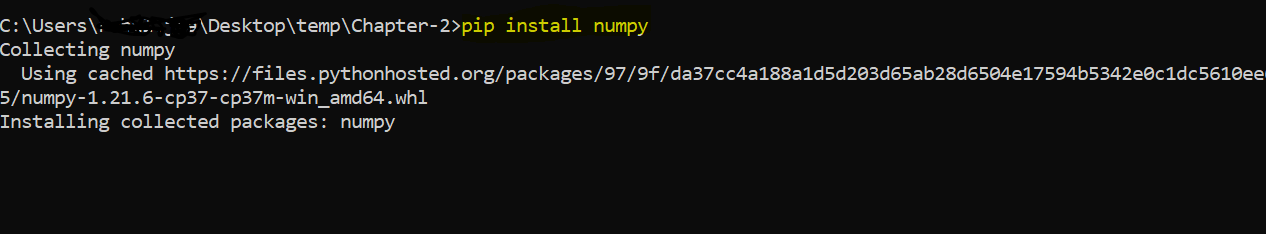
Link: **https://mybinder.org/v2/gh/fenago/MLWorkshop/HEAD**

We will use the data used in the last chapter about the probability of a patient to get a stroke and the same can be found [**here**](https://www.kaggle.com/fedesoriano/stroke-prediction-dataset). It is also made available on the online hosted environment which we provided for you. So, you don’t need to download this data as everything is available on the online hosted lab.

Data Link: **https://www.kaggle.com/fedesoriano/stroke-prediction-dataset**

We will be using the Python programming language to do this guided project and the libraries you will need are “Numpy”, “Skearn” and “Pandas”. **You don’t need to install anything as we will be providing you with a hosted notebook on binder which you can run to execute codes as we move further into this chapter**.

But if you are working on your local system, then make sure that you have numpy installed on your system. If it is not already installed, open a command prompt and enter the following command to install numpy:



**Accuracy Score**

Accuracy score is one of the simplest metrics out there to evaluate a classification problem. It is just an average number of places (rows) where our model predicted correctly.

**Formula**

**Example**

So, suppose we have a data of 2000 rows and we made correct predictions for 1800 of those rows, then our accuracy would be:

It is just that simple. We have already seen how to compute accuracy score both using NumPy and using sklearn’s inbuilt function.

Let us quickly review the calculation of accuracy score:



Please note that we are assuming that models has been fit and it has been used to generate the predictions for test data. This initial code will be provided to you in each exercise.

**Problems with Accuracy Score**

Accuracy score should never be used with a classification data which is **imbalanced in the class (target)**.

Because if it is so, then just predicting all the observations with the majority class will give us the same performance as the accuracy score. We will see this on our stroke data in the exercise.

**A note on threshold**

Sklearn offers two types of methods for making predictions on our test data. Almost always, these two methods are called as:



The first method by default uses a threshold of 0.5.

If you remember from last chapter, we discussed that logistic regression applies a sigmoid transformation on the linear regression output. That output lies between 0 and 1. So, to convert these probabilities back into labels, **we use the concept of threshold**. So, we take any row with a prediction of greater than 0.5 as 1 (as positive class) and otherwise 0 (negative class).

But we may want to, depending on the situation, use a different threshold value. Basis that, we can have different value of accuracy. So, to do that, we use the second function which gives us probabilities for each observation (row) and for each class (two in our case). We take these probabilities and apply a appropriate threshold value.

In the exercise, we will calculate accuracy for different threshold value.

**Exercise 4.1**

Link: **https://mybinder.org/v2/gh/fenago/MLWorkshop/HEAD**

Open the Exercise 4.1 file from the lab [**hosted here**](https://mybinder.org/v2/gh/fenago/MLWorkshop/HEAD) (navigate to chapter 4) and do the following:

1. Read the starter code. It will make the necessary splits for you.
2. Compute the accuracy score on the test and train data using the following code:



1. Compare the train and test accuracy score. What different do you? What is your first guess about the model? Is it a good model?
2. Now, make a new variable call random\_predictions will all hold only 0 (no stroke)



How is the result? Is it significantly different than the earlier result? If no what was the point of building the model when predicting 0 for all the observations is giving us just as good result than before.

1. Calculate accuracy score for different threshold values. We first get some threshold values starting from 0 all the way to 1, stepping up by 0.1. Then, we use the predict\_proba function to get the probabilities and note that we only take the second one (which is the probability associated with the positive class, 1)



**Confusion Matrix**

An alternative to accuracy score is to look at the confusion matrix. Although it can be created for non-binary classification problems as well but for binary classification problems, terminology turns out to be good which otherwise would be difficult to have for a non-binary problem.

So, in any binary classification problem, we may have two types of class:

1. **Positive class** (The class of importance, in our project, it may be a patient who are getting a stroke)
2. **Negative class** (the complement of the positive class)

Based on this definition, the confusion matrix computes the following four metrics:

1. **True Positives (TP)**: Our model predicts positive and actual observation is also a positive.
2. **True Negative (TN)**: Our model predicts negative and actyual observation is also negative.
3. **False Positive (FP)**: Our model predicts the observation as positive but it was actually a negative.
4. **False Negative (FN)** : Our model predicted a negative but the observations was actually a Positive.

Base on these definitions, we may observe the following formulae:

A confusion matrix is just a 2 by 2 matrix which displays this information. In the exercise, we will calculate this matrix using numpy as well using the sklearn’s implementation.

**Exercise 4.2**

Link: **https://mybinder.org/v2/gh/fenago/MLWorkshop/HEAD**

Open the Exercise 4.2 file from the lab [**hosted here**](https://mybinder.org/v2/gh/fenago/MLWorkshop/HEAD) (navigate to chapter 4) and do the following:

1. Read the starter code. It will make the necessary splits for you.
2. Implementing the confusion matrix method using numpy:
   1. First, we must create a True and False array depending upon whether the prediction is positive (stroke) or negative (no stroke):



* 1. Then, we just compare them. These are four vectors, and we will take all unique pairs of these vectors and apply the “and” operator:



* 1. Now, we are ready to create a matrix. We can display it as we wish, but to make sure you notice the change, we are doing it differently than the sklearn’s implementation:



Here, we are assuming that each row represents the row of actual and column is for predicted. So, you can think of this as:

|  |  |  |
| --- | --- | --- |
|  | **Predicted as Stroke** | **Predicted as “No Stroke”** |
| **Actual Stroke** | TP | FN |
| **Actual No Stroke** | FP | TN |

1. Using sklearn to create the same table but represented differently:



It just takes two lines of code to create the same matrix but represented as follows:

|  |  |  |
| --- | --- | --- |
|  | **Predicted as NoStroke** | **Predicted as Stroke** |
| **Actual No Stroke** | TN | FP |
| **Actual Stroke** | FN | TP |

**Precision, Recall and F1 Score**

A data scientist always looks at a way to capture the performance of the model through one number. We just looked at one number called “accuracy score” and we found out that it is not a good measure to evaluate the model. We saw that a random model gives us just as much accuracy score as the actual model.

So, clearly our model is not good, but we want a single number which would tell us this.

Then, we looked at confusion matrix which clearly captured the poor performance of our model but then this gives us four numbers and we want to somehow create one number summary from these four numbers to be able to better explain the poor performance of the model.

Let us define two such measure and they are called “Precision” and “Recall”. They are also called by different names such as sensitivity and specificity but let us try to look at the general definition and try to understand its meaning from that angle.

**Precision**

Precision is proportion of total number of predicted as positives. This is the denominator. The numerator is correct number of predictions as positives.

So, it can be calculated based on the values of the confusion matrix as follows:

So, consider a model where we want to predict cancer as benign or malignant. Suppose our data consists of 100 cases of malignant while 900 are benign. Here, if I have a model which predicts all 1000 observations as having a malignant tumour. Let us look at the different measures:

Where as precision is:

So, we can see that our model’s is not precise here. But now let us look at one more metric which is Recall.

**Recall**

Recall considers the actual number of positives. So, the denominator is actual number of positives and numerator is how many our model got correct. It can also be calculated from the values of the confusion matrix:

In our example, this will turn out to be:

So, based on the recall, this model performs best but at the same time, we can see that precision is poor. So, we want to have some way to take both into consideration. Because again (just like in confusion matrix) we have a choice of two numbers.

**F1 Score**

F1-score is one intelligent way to combine precision and recall producing one number summary for the classification mode. It can be computed as:

This is nothing but the geometric mean of the precision and recall. For the above example, the F1 Score will turn out to be:

This low value clearly states that the given classification model is not good.

**Exercise 4.3**

Link: **https://mybinder.org/v2/gh/fenago/MLWorkshop/HEAD**

Open the Exercise 4.3 file from the lab [**hosted here**](https://mybinder.org/v2/gh/fenago/MLWorkshop/HEAD) (navigate to chapter 4) and do the following:

1. Read the starter code. This has the confusion matrix code.
2. Now, compute the Precision and Recall using the numpy:



1. Compute the same two measures using sklearn and compare both the answers:



1. Now compute the F1 Score from the precision and recall:



**ROC curve and AUC metric**

**What was one thing common in all the metrics we discussed so far?**

All these metrics are defined for a given threshold. So, the success of these metric depends on the threshold we provide (by default it would be 0.5).

***So, is there a single metric which provides an overall performance of the classification model across all the possible thresholds values?***

The answer is yes. The metric is known as AUC matrix, which stands for area under curve.

To understand this metric clearly, we need to understand how to construct ROC curve.

Let us look at the steps to create a ROC curve.

**How to construct ROC curve?**

To construct a ROC curve, the following steps can be taken:

* 1. Make a list of all the thresholds
  2. Compute True positive rate and False positive rate for each threshold.
  3. Plot False positive rate on x-axis and True positive rate on Y-axis.
  4. The resulting plot is called the ROC plot.

**How do we interpret the resulting plot?**

First, let us define True Positive Rate and False positive rate. True positive rate is just the proportion of true positive among all positives. It is just another name of Recall. We want to increase this as much as possible.

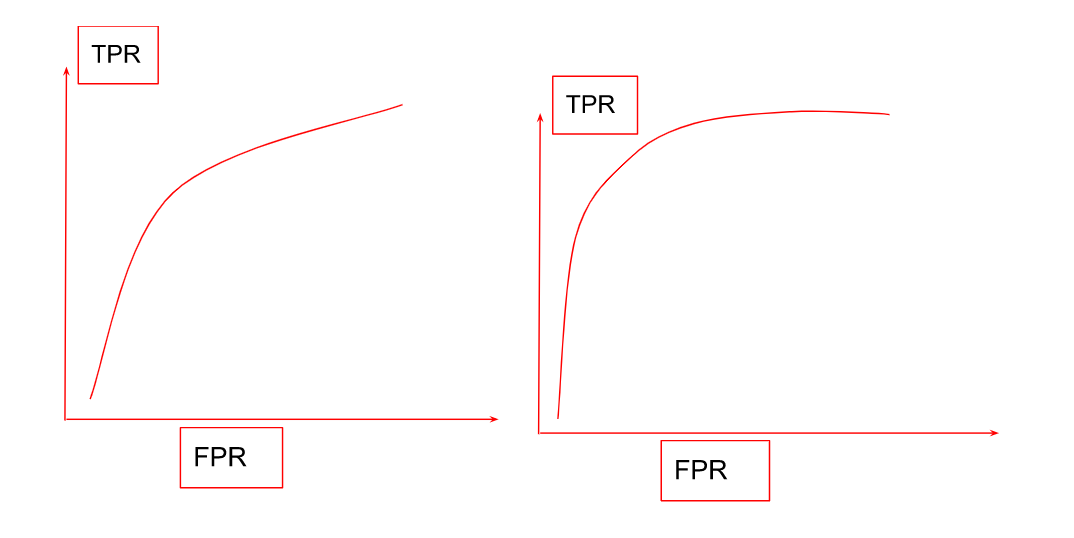
False positive rate is the fraction of all false positives among all negatives examples. We want this be as small as possible.

So, what could this curve look like? Remember we are plotting FPR on x-axis while TRP on y-axis. So, we want as low value for the x-axis while as high a value for y-axis.

We have the following formulae for the TPR and FPR:

**Hypothetical ROC curve**

Consider the following two plots:



One question is that: which one of these plots do you feel are better?

The answer should be right one. Why is this so? Because this curve is nearer to Y-axis (which has the TPR). We want to have TPR as large as possible. We want to have a ROC curve which is more towards the y-axis.

So, we would like all our thresholds to have TPR large while FPR as small as possible.

So, now how can I numerically represent this “**near to y-axis**” thing? Here comes the AUC (area under curve). It computes the area under the ROC curve which will be large if this curve is nearer to y-axis.

Implementing AUC is difficult from scratch, so we will take help of sklearn’s inbuilt library for that. But we can make ROC curve using numpy and matplotlib. We will do that in the next exercise.

**Exercise 4.4**

Link: **https://mybinder.org/v2/gh/fenago/MLWorkshop/HEAD**

Open the Exercise 4.4 file from the lab [**hosted here**](https://mybinder.org/v2/gh/fenago/MLWorkshop/HEAD) (navigate to chapter 4) and do the following:

1. Run the starter code. It will create some variables and import required packages.
2. In the confusion matrix exercise, we computed TP, FN, TN, FP for only one threshold (default one). Let us now take that code and put that into a for loop which does that thing repeatedly for all the thresholds we define.



Most of the calculations are the just like the ones we did earlier but now we are also calculating true\_positive\_rate and false\_positive\_rate . Then, we append these metrics in an empty list.

1. Let us now plot these two and make the ROC curve. Note that, currently the list scores1 cannot be indexed as it is not a numpy list. So, we convert the list to a numpy array and then Import matplotlib.



We can see that; it results in a plot which is just slightly tilted towards y-axis.

1. Let us do this same for a random model. Run the same for loop as before, but except for using the model to make predictions, we generated random predictions using numpy’s random module.



Please note the difference between the earlier loop and this one. There are only two differences:

* + 1. Current\_preds line is different
    2. Scores2 list is being used to append instead of scores1

1. Now let us plot this random model and see if our model performs any better than the random model.



This produces the following plot:



1. We can clearly see that our model is nearer to y-axis than random model. Let us check it numerically using the AUC (area under curve). This metric is difficult to obtain. So, we will use the sklearn’s implementation.



1. We can see that random model’s AUC is about 0.5 while our model gives us a AUC of 0.73 which is okay, not a very great score but still it says that our model is at least better than a random model.

**Parameter Tuning**

All machine learning algorithms have so called hyperparameters which a data scientist need to set while starting the training process. There are no one set of parameters which will work well in all problem statements. This is something which needs to be arrived at by experimentation. So, this is not something which machine learning figure out on its own while training. A good choice of hyper parameters can be a game changer in your machine learning model.

**How do we tune these parameters?**

There are several ways of tuning these hyperparameters, but one very simple way is to try our all bunch of possible values (for that hyperparameter) and see which one works best. Just a for loop through those values might do the job.

**One alternative method of evaluation of models**

So, far we have looked at various metrics all of which were used on the test data. Usually, this practice is not recommended. The recommended practice is to have a validation data and once you are satisfied with the validation score, you can finally test your model for the last time on the test data (which a data which was not seen in earlier steps, so we can get a completely unbiased results).

Having a validation set is not always possible because of data limitation. So, there is an alternative which we will discuss in the next topic.

**K-fold cross validation**

This is a method of validating the model without using the test data. The idea is to split your data in some folds (K-fold). Fold is just a fancy way of saying parts. So, we split our data into K parts and train K different models. In each training, we train on the K-1 folds while test that model on the remaining fold.

So, if we have a data with 30000 rows and we have decided to have 10 folds, then we will fit 10 different models and for each fit, the training will take place on the 27000 while the testing will happen on the remaining 3000 rows. In this way, instead of getting one single metric, we will have a collection of 10 metric which may give us an idea of the possible distribution of the given metric. If there is a distribution, we can compute:

* 1. Mean of these values to get an overall metric
  2. Standard division to get a sense of how much deviation can we expect if we deploy this model.

Let us practice tuning of parameters and K-fold validation.

**Exercise 4.5**

Link: **https://mybinder.org/v2/gh/fenago/MLWorkshop/HEAD**

Open the Exercise 4.5 file from the lab [**hosted here**](https://mybinder.org/v2/gh/fenago/MLWorkshop/HEAD) (navigate to chapter 4) and do the following:

1. Run the starter code. It will create some variables and import required packages.
2. Let us find the best C for the logistic regression model. It is a parameter which species the regularization. Don’t worry about this in this chapter. Just remember that this should a positive float.



It just loops through all the cs and fits model. For each model, it computes train and test AUC and prints those metrics. The output clearly shows that there is not much of difference but for C = 10, the AUC in both train and test are maximum.

1. Let us perform cross validation using K-fold and measure our model’s performance using this alternative way. This code will make 10 splits of our training data (which we also discussed in the example before). We will get 10 scores after this procedure and these scores will be on different validation sets. Hence, they provide us a way to get the distribution of these scores.



1. Let us compute the mean and standard deviso of these scores:



**Activity**

Open the Activity 4.ipynb file from the lab [**hosted here**](https://mybinder.org/v2/gh/fenago/MLWorkshop/HEAD) (navigate to chapter 4) and do the following. The objective is to predict water portability based on various feature .

1. The initial starter code is available. Explore the data using the steps we discussed in this chapter.
2. Build a classification model using one hot encoding for the categorical variables. Replace all the missing values with the appropriate values which we discussed in the last lecture.
3. Compute the following metrics:
   1. Accuracy score
   2. Confusion matrix
   3. Precision and Recall
   4. F1 Score
4. Compute accuracy for different thresholds.
5. Perform K fold cross validation using 20 folds (compute AUC) and compute the mean and standard deviation of these 20 AUC scores.

**Project**

Navigate to the Kaggle link [**here**](https://www.kaggle.com/datasets/pavansubhasht/ibm-hr-analytics-attrition-dataset) and build a classification model to predict employee attrition. Try to include all the steps that we followed in this chapter including the missing value treatment and including categorical variables in your model.

Evaluate your model with the f1 score and perform k fold cross validation.

**Summary**

Evaluate In this chapter, we looked at various evaluation metrics for classification problems. We started this discussion with some problems that accuracy score faces for a unbalanced data and how this can be solved using the confusion matrix.

Then we looked at how confusion matrix gives us four numbers, but they don’t let us evaluate immediately as to how our model is performing at an overall level. To overcome this, we can use precision, recall and f1-score.

All these metrics works for a given threshold value. We wanted one single metric which considers all the different thresholds and gives us one single number which can tell us about the performance of the model at an overall level.

So, we looked at AUC metric which is nothing but area under the ROC curve and how it correctly captures the overall performance of the model.

Then, finally we looked at an alternative strategy to evaluate our model and that is to use some holdout data from the training data itself to validate the model.

Last but not the least, we looked at hyperparameter tuning briefly to get the best parameters, C , for the logistic regression.