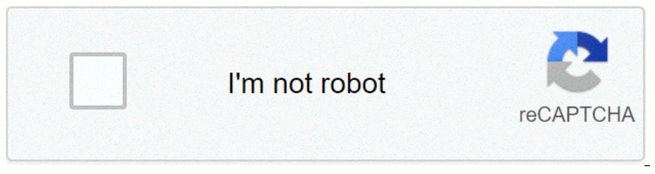

Continue

Sklearn classification report explanation

In this tutorial, we'll go over several classification metrics in scikit-learn python and write our own features from scratch to understand the math behind a few of them. One of the main areas of predictive modeling in data science is classification. Classification consists in trying to predict which class a particular sample comes from. For example, if we try to predict whether a particular patient will be hospitalized again, the two possible classes are hospitals (positive) and non-hospitalized (negative). The classification model then tries to predict whether each patient will be hospitalized or not hospitalized. In other words, the classification simply tries to predict which basket (predicted positive vs. predicted negative) should be a specific sample from the population located, as shown below. When you train a classification predictive model, you'll want to assess how good it is. Interestingly, there are many different ways to evaluate performance. Most data scientists who use Python for predictive modeling use a Python package called scikit-learn. Scikit-learn includes many built-in features for analyzing model performance. In this tutorial, we'll go through a few of these metrics and write our own features from scratch to understand the math behind a few of them. If you'd rather just read about performance metrics, see my previous post on here. This tutorial will cover the following metrics from sklearn.metrics :confusion_matrixaccuracy_scorerecall_scoreprecision_scoref1_scoreroc_curveroc_auc_scoreetting StartedFor the sample dataset and jupyter notebook, visit my github here. We will write our own functions from scratch provided two classification classes. Note that you will need to fill in the sections marked as #your code hereLet load the sample dataset that has the actual labels (actual_label) and probability predictions for the two models (model_RF and model_LR). Here the probabilities are the probability that classes 1.In most data science projects, you define a threshold to define which prediction probabilities are marked as predicted positive vs. predicted negative. For now, let's assume that the threshold is 0.5. We'll add two more columns to convert the probabilities to the labels.confusion_matrixGiven the actual label and the predicted label. The first thing we can do is divide our samples into 4 buckets: True positive - real = 1, predicted = 1False positive - real = 0, predicted = 1False negative - real = 1, predicted = 0True negative — real = 0, predicted = 0These buckets can be represented with the following image (original source Precisionrecall.svg) and we will refer to this image in many calculations below. These buckets can also be displayed using the confusion matrix as shown below: We can get matrix confusion (like a 2x2 field) from a scikit-learn that takes as inputs actual labels and assumed labels, where there were 5047 actual positives, 2360 false positives, 2832 false negatives and 5519 actual negatives. Let's define our own features to confusion_matrix. Note that I filled in the first and you need to fill in the next 3.You can check your results correspondLet is to write a function that will count all four of them for us, and other functions duplicate confusion_matrixCheck your results match withPlace of manual comparison, let's verify that our function worked using Python is built in assert and numpy's array_equal functionZimat these four buckets (TP , FP, FN, TN), we can calculate many other performance metrics.accuracy_scoreThe the most common metric for classification is accuracy, which is metrics.accuracy_scoreThe fraction of the samples correctly predicted as shown below:We can get an accuracy score from scikit-learn, which takes as inputs the actual labels and predicted labelsYus response should be 0.6705165630156111Define custom function that duplicates accuracy_score, using the above formula. Using precision as a performance metric, the rf model is more accurate (0.67) than the LR model (0.62). So we should stop here and say the RF model is the best model? No! Accuracy is not always the best metric to use to assess classification models. For example, let's say we're trying to predict something that only happens to 1 in 100. We could create a model that gets 99% accuracy by saying that the event never happened. However, we catch 0% of the events we care about. The 0% rate here is another performance metric known as recall.recall_scoreRecall (also known as sensitivity) is a fraction of the positive events that you correctly predicted, as shown below: We can get an accuracy score from scikit-learn, which takes as inputs actual labels and assumed labelsDefine custom function that duplicates recall_score, using the above formula. One method for increasing returns is to increase the number of samples that you define as predicted positive by lowering the threshold for predicted positives. Unfortunately, this will also increase the number of false positives. Another performance metric called accuracy takes it to account.precision_scorePrecision is a fraction of the anticipated positive events that are actually positive, as shown below:We can get an accuracy score from scikit-learn, which takes as inputs actual labels and predicted labelsdefine a custom function that duplicates precision_score, using the formula above. In this case, it looks like the RF model is better at both recall and accuracy. But what would you do if one model was better in appeal and the other was better in accuracy. One method used by some data scientists is called F1, and an F1 score is score.f1_scoreThe harmonic average of recall and accuracy with a higher score as a better model. The F1 score is calculated using the following formula:We can get an F1 score from scikit-learn, which takes as inputs labels and assumed labelsDefine a custom function that duplicates f1_score using the formula above. So far, we have assumed that we have defined a threshold of 0.5 for the selection of samples that are predicted to be positive. If we change this threshold, the performance metrics will change. As below:How do we rate a model if we have not chosen a threshold? One of the very common methods is to use receiver operating characteristics (ROC) curve.roc_curve and roc_auc_scoreROC curves are very helpful in understanding the balance between true-positive rates and false positive rates. Sci-kit learn has built-in features for ROC curves and for their analysis. The inputs for these functions (roc_curve and roc_auc_score) are actual labels and assumed probabilities (not assumed labels). Both roc_curve and roc_auc_score are both complex features, so we won't have you writing these features from scratch. Instead, we'll show you how to use sci-kit learn features to explain key points. Let's start roc_curve the internet to make an ROC conspiracy. roc_curve returns three lists:thresholds = all unique prediction probabilities in descending order fpr = false positive rate (FP / (FP + TN)) for each thresholdtpr = actual positive rate (TP / (TP + FN)) for each thresholdWe can plot the ROC curve for each model, as shown below. There are a few things that we can observer from this image: a model that randomly guesses a label will result in a black line and you want to have a model that has a curve above this black ROC line that is further away from the black line is better, so rf (red) looks better than LR (blue)Even if not visible directly, the high threshold results in a point in the lower left corner and a low threshold results in a point in the upper right corner. This means that when you lower the threshold, you get higher TPR at the cost of a higher FPRChacete if you analyze performance, we use the area-below-curve metric. As you can see, the area below the curve for the RF model (AUC = 0.738) is better than LR (AUC = 0.666). When I render the ROC curve, I like to add AUC to the legend as shown below. Overall, in this example, the RF model wins with each performance metric. ConclusionIn predictive analysis, it is important to select one performance metric when deciding between two models. As you can see here, there are many that you can choose from (accuracy, appeal, accuracy, f1-score, AUC, etc.). Finally, you should use the performance metric that is best suited for the business issue. Many data scientists prefer to use AUC to analyze the performance of each model because it does not require threshold selection and helps balance true positive speed and false positive speed. Please leave a comment if you have any suggestions on how to improve this tutorial. A classification overview is a key metric in a classification problem. You will have accuracy, appeal, f1-scores and support for every class you are trying to find. Teh means how much of this class you will find throughout the entire number of elements of this accuracy class will be how much is correctly classified among this class The f1 score is the harmonic average between accuracy & recall Support is the number of occurrences of that class in your dataset (so you have 37.5K class 0 and 37.5K class 1 , which is a really well balanced set of data. The point is, accuracy and revocation are very much used for unbalanced datasets, because in a highly unbalanced dataset, 99% accuracy can be meaningless. I would say that you don't really need to look at these metrics for this issue if a given class should be absolutely correctly established. To answer the next question, you cannot compare accuracy and revocation across two classes. This just means that you are a classifier it is better to find class 0 above class 1. The accuracy and sklearn.metrics.precision_score of recall_score or the device should not differ. But as long as the code is not available, it is impossible to determine the root cause of this. This.