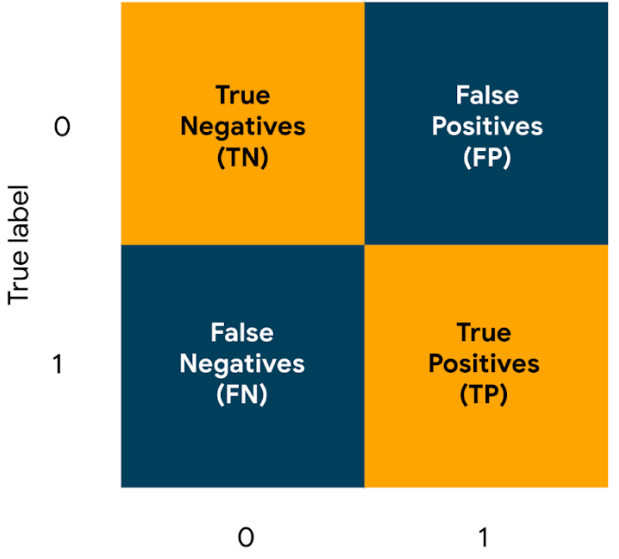
Common logistic regression metrics in Python

Logistic regression is a powerful technique for categorical prediction tasks in data science. Data professionals often use metrics such as precision, recall, and accuracy, as well as visualizations such as ROC curves, to gauge the performance of their logistic regression models. It is important to evaluate the performance of a model, as this shows how well the model can make predictions. The results from applying metrics can be used to report how well a model performs to relevant stakeholders.

In this reading, you will review parts of a confusion matrix and understand how to compute and visualize metrics for evaluating logistic regression through code in Python.

Parts of a confusion matrix

A confusion matrix helps summarize the performance of a classifier. The components of a confusion matrix are used to compute metrics for evaluating logistic regression classifiers.



The four key parts of a confusion matrix, in the context of binary classification, are the following:

1. **True negatives:**

The count of observations that a classifier correctly predicted as False (0)

2. **True positives:**

The count of observations that a classifier correctly predicted as True (1)

3. **False positives:**

The count of observations that a classifier incorrectly predicted as True (1)

4. **False negatives:**

The count of observations that a classifier incorrectly

predicted as False (0)

These counts are useful in computing metrics such as precision, recall, accuracy, and ROC for evaluating logistic regression classifiers.

Precision

One of the major metrics for evaluating a logistic regression classifier is **precision**. Precision measures the proportion of data points predicted as True that are actually True. Imagine that you have built a logistic regression classifier for email spam detection, trained this classifier on a relevant dataset, and used this classifier to generate predictions for a set of emails. The predictions consist of True and False values. True represents an email predicted as spam, and False represents an email predicted as not spam. The precision for this classifier would convey the proportion of emails that are actually spam, out of all the emails that have been predicted as spam.

The formula for precision is as follows:

Precision=True Positives + False PositivesTrue Positives

To compute precision in Python, you can use the *precision_score()* function from the *metrics* module in the *sklearn* library. You can start with the following import

statement.

import sklearn.metrics as metrics

The *precision_score()* function takes in true values and predicted values as arguments and returns the precision score. Assume that *y_test* and *y_pred* are variables that contain true values and predicted values respectively. You can use the following code to compute the precision.

metrics.precision_score(y_test,y_pred)

In the context of email spam detection, if precision_score() returns 0.91, that means 91% of the emails predicted as spam are indeed spam.

Recall

Another major metric for evaluating a logistic regression classifier is **recall**. Recall measures the proportion of data points that are predicted as True, out of all the data points that are actually True. Imagine that you have built a logistic regression classifier for fraud detection and generated predictions. In the predictions, True represents a credit card transaction predicted as fraudulent, and False represents a credit card transaction predicted as not fraudulent. The recall for this classifier would convey the proportion of fraudulent credit card transactions that the classifier

correctly identified as such.

The formula for recall is as follows:

Recall=True Positives + False NegativesTrue Positives

To compute recall in Python, you can use the *recall_score()* function from the metrics module. The function takes in true values and predicted values as arguments and returns the recall score. You can use the following code to compute the recall.

metrics.recall_score(y_test,y_pred)

In the context of fraud detection among credit card transactions, if the *recall_score()* function returns 0.87, that means 87% of the fraudulent credit card transactions are correctly detected as fraudulent.

Accuracy

Another important metric for evaluating logistic regression is **accuracy**. Accuracy measures the proportion of data points that are correctly classified. Imagine that you have built a logistic regression classifier for loan approval prediction. In the predictions, True represents a prediction that the loan will be approved, and False represents a prediction that the loan will not be approved. The accuracy score for this classifier would convey the proportion of loans that have

been correctly classified.

The formula for accuracy is as follows:

Accuracy=Total PredictionsTrue Positives+True Negatives

To compute accuracy in Python, you can use the *accuracy_score()* function from the metrics module. The function takes in true values and predicted values as arguments and returns the accuracy score. You can use the following code to compute the accuracy.

metrics.accuracy_score(y_test,y_pred)

In the context of loan approval prediction, if the *accuracy_score()* function returns 0.90, that means 90% of the loans are correctly predicted, with the predictions being either will be approved or will not be approved.

ROC curves

An **ROC curve** helps in visualizing the performance of a logistic regression classifier. ROC curve stands for receiver operating characteristic curve. To visualize the performance of a classifier at different classification thresholds, you can graph an ROC curve. In the context of binary classification, a classification threshold is a cutoff for differentiating the positive class from the negative class.

An ROC curve plots two key concepts

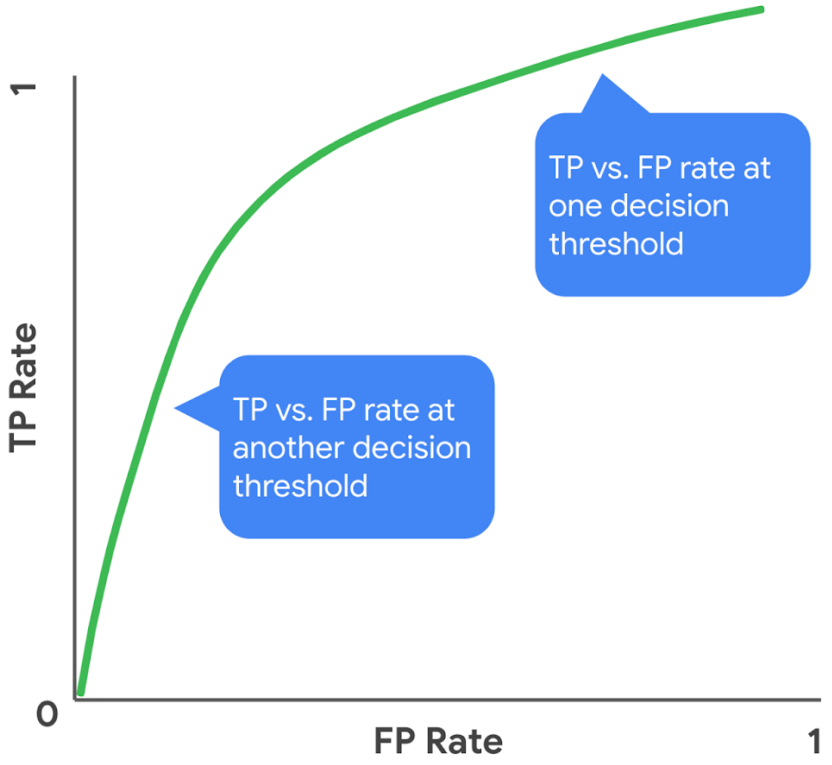
1. **True Positive Rate:** equivalent to **Recall.** The formula for True Positive Rate is as follows:

True Positive Rate=True Positives + False NegativesTrue Positives

2. **False Positive Rate:** The ratio between the False Positives and the total count of observations that should be predicted as False. The formula for False Positive Rate is as follows:

False Positive Rate=False Positives + True NegativesFalse Positives

For each point on the curve, the x and y coordinates represent the False Positive Rate and the True Positive Rate respectively at the corresponding threshold.



You can examine an ROC curve to observe how the False Positive Rate and True Positive Rate change together over the different thresholds. In the ROC curve for an ideal model, there would exist a threshold at which the True Positive Rate is high and the False Positive Rate is low. The more that the ROC curve hugs the top left corner of the plot, the better the model does at classifying the data.

You can use the following steps to graph an ROC curve in Python.

Start by importing the necessary modules as follows.

import matplotlib.pyplot as plt

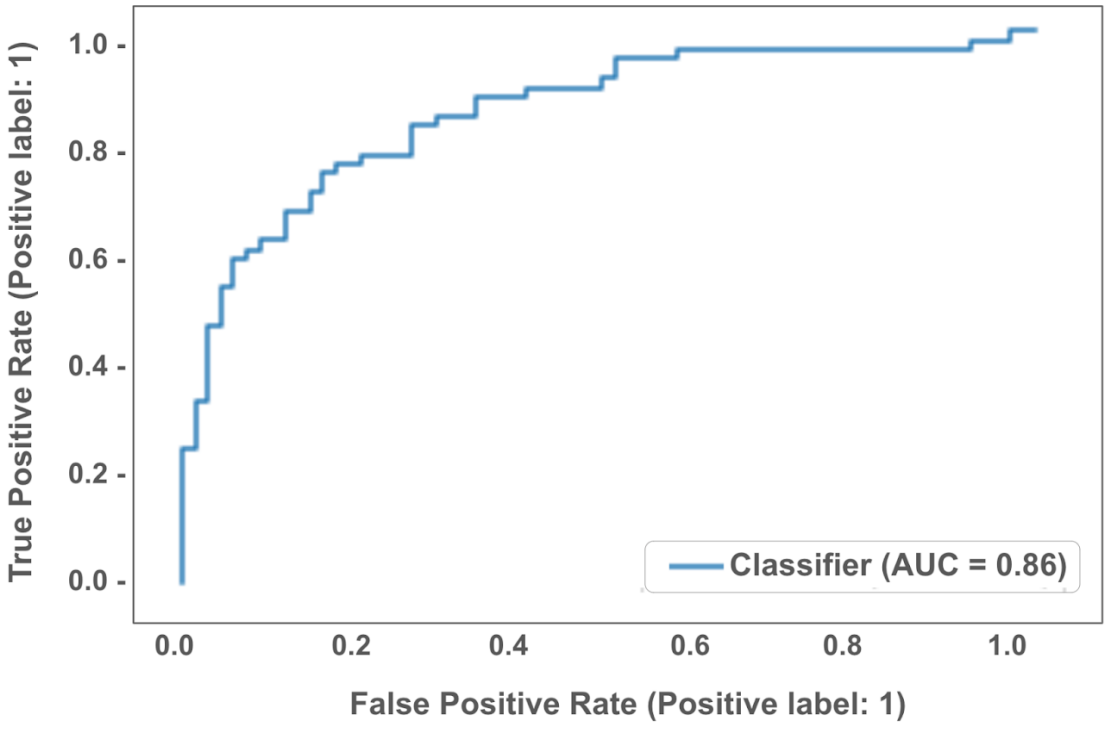
from sklearn.metrics import RocCurveDisplay

Then use the following code to plot the ROC curve.

RocCurveDisplay.from_predictions(y_test, y_pred)

plt.show()

Using these steps to generate an ROC curve could result in a graph.



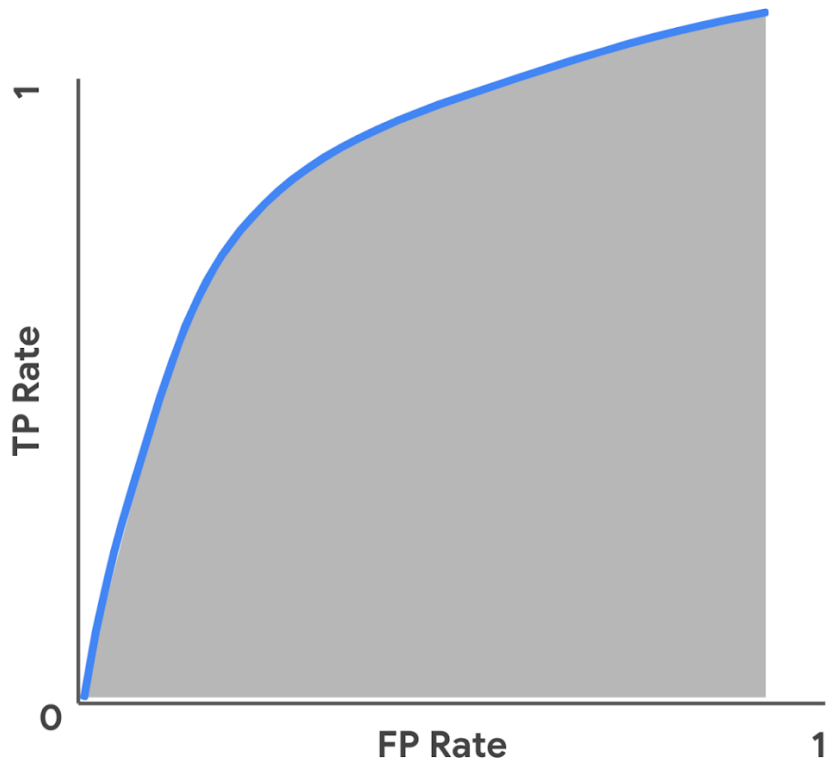
In this graph, the ROC curve indicates that the corresponding classifier performs decently well.

AUC

AUC stands for area under the ROC curve. AUC provides an aggregate measure of performance across all possible classification thresholds. AUC ranges in value from 0.0 to 1.0. A model whose predictions are 100% wrong has an AUC of

0.0, and a model whose predictions are 100% correct has an AUC of 1.0. An AUC smaller than 0.5 indicates that the model performs worse than a random classifier (i.e. a classifier that randomly assigns each example to True or False), and an AUC larger than 0.5 indicates that the model performs better than a random classifier.

In the following visualization, AUC is the area of the shaded region.



To compute AUC in Python, you can use the roc_auc_score() function from the metrics module. The function takes in true values and predicted values as arguments and returns the accuracy score. You can use the following code to compute the AUC.

metrics.roc_auc_score(y_test,y_pred)

For example, in the context of a logistic regression classifier for email spam detection, if the *roc_auc_score()* function returns 0.99, that means 99% of the classifier's predictions are correct across all classification thresholds.

Key takeaways

- Precision, Recall, and Accuracy are common metrics for evaluating the performance of a logistic regression classifier.
- You can use functions from the metrics module in the sklearn library to compute these metrics in Python.
- Graphing an ROC curve helps visualize how a classifier performs across different classification thresholds.
- Computing AUC helps aggregate a classifier's performance across thresholds into one measure.

Resources for more information

- precision_score: Documentation on the precision_score() function
- recall_score: Documentation on the recall_score() function

- accuracy_score: Documentation on the accuracy_score() function
- RocCurveDisplay.from_predictions: Documentation on the RocCurveDisplay.from_predictions() function
- roc_auc_score: Documentation on the roc_auc_score() function