17.8.3

Fit the Model, Make Predictions, and Evaluate Results

Now that you have prepared the data, you will put the random forest classifier model to practice, then evaluate the results.

Now that we have preprocessed the data into training and testing data for both features and target sets, we can fit the random forest model, make predictions, and evaluate the model.

Fit the Random Forest Model

Before we fit the random forest model to our `X_train_scaled` and `y_train` training data, we'll create a random forest instance using the random forest classifier, `RandomForestClassifier()`.

```
# Create a random forest classifier.
rf_model = RandomForestClassifier(n_estimators=128, random_state=78)
```

The `RandomForestClassifier` takes a variety of parameters, but for our purposes we only need the `n_estimators` and the `random_state`.

NOTE

Consult the sklearn documentation for additional information about the `RandomForestClassifier` and the parameters it takes.

The `n_estimators` will allow us to set the number of trees that will be created by the algorithm. Generally, the higher number makes the predictions stronger and more stable, but can slow down the output because of the higher training time allocated. The best practice is to use between 64 and 128 random forests, though higher numbers are quite common despite the higher training time. For our purposes, we'll create 128 random forests.

After we create the random forest instance, we need to fit the model with our training sets.

```
# Fitting the model
rf_model = rf_model.fit(X_train_scaled, y_train)
```

Make Predictions Using the Testing Data

After fitting the model, we can run the following code to make predictions using the scaled testing data:

```
# Making predictions using the testing data.
predictions = rf_model.predict(X_test_scaled)
```

The output will be similar as when the predictions were determined for the decision tree.

```
predictions
```

```
array([1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0,
       1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1,
       0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0,
       0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0])
```

Evaluate the Model

After making predictions on the scaled testing data, we analyze how well our random forest model classifies loan applications by using the `confusion_matrix`.

```
# Calculating the confusion matrix.
cm = confusion_matrix(y_test, predictions)

# Create a DataFrame from the confusion matrix.
cm_df = pd.DataFrame(
    cm, index=["Actual 0", "Actual 1"], columns=["Predicted 0", "Predicted 1"])

cm_df
```

	Predicted 0	Predicted 1
Actual 0	51	33
Actual 1	23	18

Based on the results in the confusion matrix DataFrame, how many true negatives are there? That is, how many were correctly predicted to belong to category 1?

- ○ 51
- ○ 23
- ○ 33
- ○ 18

Check Answer

Finish ▶

These results are relatively the same as the decision tree model. To improve our predictions, let's increase the `n_estimators` to 500. After running all the code again, after changing the `n_estimators` to 500, our confusion matrix DataFrame is about the same as before.

	Predicted 0	Predicted 1
Actual 0	50	34
Actual 1	26	15

Using the equation `(TP + TN) / Total`, we can determine our accuracy (determine how often the classifier predicts correctly) by running the following code. For this model, our accuracy score is 0.520:

```
# Calculating the accuracy score.
acc_score = accuracy_score(y_test, predictions)
```

Lastly, we can print out the above results along with the classification report for the two classes:

```
# Displaying results
print("Confusion Matrix")
display(cm_df)
print(f"Accuracy Score : {acc_score}")
print("Classification Report")
print(classification_report(y_test, predictions))
```

Confusion Matrix

	Predicted 0	Predicted 1
Actual 0	50	34
Actual 1	26	15

Accuracy Score : 0.52

Classification Report

	precision	recall	f1-score	support
0	0.66	0.60	0.62	84
1	0.31	0.37	0.33	41
accuracy			0.52	125
macro avg	0.48	0.48	0.48	125
weighted avg	0.54	0.52	0.53	125

From the confusion matrix results, the precision for the bad loan applications is low, indicating a large number of false positives, which indicates an unreliable positive classification. The recall is also low for the bad loan applications, which is indicative of a large number of false negatives. The F1 score is also low (33).

In summary, this random forest model is not good at classifying fraudulent loan applications because the model's accuracy, 0.520, and F1 score are low.

Rank the Importance of Features

One nice byproduct of the random forest algorithm is to rank the features by their importance, which allows us to see which features have the most impact on the decision.

To calculate the feature importance, we can use the `feature_importances_` attribute with the following code:

```
# Calculate feature importance in the Random Forest model.
importances = rf_model.feature_importances_
importances
```

The output from this code returns an array of scores for the features in the `X_test` set, whose sum equals 1.0:

```
importances
```

```
array([0.05454782, 0.07997292, 0.43280448, 0.32973986, 0.01887172,
       0.02110219, 0.00271658, 0.02151063, 0.01887818, 0.01985562])
```

To sort the features by their importance with the column in the `X_test` set, we can modify our code above as follows:

```
# We can sort the features by their importance.
sorted(zip(rf_model.feature_importances_, X.columns), reverse=True)
```

In the code, the sorted function will sort the zipped list of features with their column name (`X.columns`) in reverse order—more important features first—with `reverse=True`.

Running this code will return the following output:

```
[(0.43280447750315343, 'age'),
 (0.32973986443922343, 'month_num'),
 (0.07997292251445517, 'term'),
 (0.05454782107242418, 'amount'),
 (0.021510631303272416, 'education_college'),
 (0.021102188881175144, 'education_High School or Below'),
 (0.01985561654170213, 'gender_male'),
 (0.018878176828577283, 'gender_female'),
 (0.018871722006693077, 'education_Bachelor'),
 (0.002716578909323729, 'education_Master or Above')]
```

Now we can clearly see which features, or columns, of the loan application are more relevant. The `age` and `month_num` of the loan application are the more relevant features.

To improve this model, we can drop some of the lower ranked features.

Drop some of the lower ranked features, like education and/or gender. Does dropping these features improve our random forest model?

- ◯ Yes
- ◯ No

Check Answer

Finish ▶