17.8.2

Predict Loan Applications

Jill now asks you to run a random forest model to make classifications. As you have done before, the first step is to prepare the data for the random forest classifier model.

When we imported our dependencies to create the decision tree in the previous example, we use the "tree" module from the sklearn library, `from sklearn import tree`.

For the random forest model, we'll use the "ensemble" module from the sklearn library. All the remaining dependencies will be the same. In the dependencies, replace `from sklearn import tree` with `from sklearn.ensemble import RandomForestClassifier` so that our dependencies look like the following.

```
# Initial imports.
import pandas as pd
from path import Path
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
```

Next, read in your `loans_data_encoded.csv` file from previous exercises.

```
# Loading data
file_path = Path("../Resources/loans_data_encoded.csv")
df_loans = pd.read_csv(file_path)
df_loans.head()
```

After the data has been loaded, we're going to preprocess data just like we did for the decision tree model.

Preprocess the Data

Now, we're going to walk through the preprocessing steps for the loan applications' encoded data so that we can fit our training and testing sets with the random forest model.

If you do not quite remember the steps for preprocessing, add the blocks of code in your Jupyter Notebook as follows.

1. First, we define the features set.

```
# Define the features set.
X = df_loans.copy()
X = X.drop("bad", axis=1)
X.head()
```

2. Next, we define the target set. Here, we're using the `ravel()` method, which performs the same procedure on our target set data as the `values` attribute.

```
# Define the target set.
y = df_loans["bad"].ravel()
y[:5]
```

3. Now, we split into the training and testing sets.

```
# Splitting into Train and Test sets.
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=78)
```

4. Lastly, we can create the `StandardScaler` instance, fit the scaler with the training set, and scale the data.

```
# Creating a StandardScaler instance.
scaler = StandardScaler()
# Fitting the Standard Scaler with the training data.
X_scaler = scaler.fit(X_train)

# Scaling the data.
X_train_scaled = X_scaler.transform(X_train)
X_test_scaled = X_scaler.transform(X_test)
```

If you were able to do these steps without having to follow along, congratulations!