17.7.1 Overview of Decision Trees

1. Summary
   1. Decision trees encode a series of true/false questions that are represented by a series of if/else statements.
   2. Decision trees are one of the most interpretable models, as they provide a clear representation of how a model works.
   3. Decision trees are natural ways in which you can classify or label objects by asking a series of questions designed to zero in on the true answer.
2. Parts
   1. Root node/ Parent node
      1. The root node represents the entire population.
   2. Child nodes
      1. When we split the root node into two subnodes, they are called child nodes.
3. Steps (from the homework example)
   1. Import the dependencies

# Import the dependencies

import pandas as pd

from path import Path

from sklearn import tree

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import confusion\_matrix, accuracy\_score, classification\_report

* 1. Read in your csv file
     1. What is our goal?
     2. What are we trying to predict?
     3. Code

# Loading data

file\_path = Path(“../Resources/<filename.csv>”)

df = pd.read\_csv(file\_path)

df.head()

* 1. Define the features set (x)

X = df.copy()

X = X.drop(“<whatever will be y>”, axis=1)

X.head()

* 1. Define the target set (y)

y = df[“<whatever y is>”].values

y

* 1. Split the data into training and test sets to train our model
     1. This will help determine the relationships between each feature in the features training set and the target training set.
     2. Code

X\_train, x\_test, y\_train, y\_test = train\_test\_split(X, y, random\_state = 78)

# Determine the shape of our training and testing sets

print (X\_train.shape)

print (X\_test.shape)

print(y\_train.shape)

print(y\_test.shape)

* 1. Scale the data using Scikit-learn’s StandardScaler
     1. The standard scaler standardizes the data. Which means that each feature will be rescaled so that its mean is 0 and its standard deviation is 1.
     2. Typically, models that compute distances between data points, such as SVM, require scaled data. Although decision trees don't require scaling the data, it can be helpful when comparing the performances of different models.
     3. Code

# Creating a StandardScaler instance

scaler = StandardScaler()

# Fitting the StandardScaler with training data

X\_scaler = scalar.fit(X\_train)  
# Scaling the data

X\_train\_scaled = X\_scalar.transform(X\_train)

X\_test\_scaled = X\_scalar.transform(X\_test)

* 1. Fit the Decision Tree Model

# Create the decision tree classifier instance  
model = tree.DecisionTreeClassifier()

# Then train or fit the "model" with the scaled training data  
model = model.fit(X\_train\_scaled, y\_train)

* 1. Make predictions using the testing data

predictions = model.predict (X\_test\_scaled)

* 1. Evaluate the model
     1. Determine how well our model does what we wanted
        1. Ex. Classifies loan applications
     2. Code

# Calculate 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

# Calculate the accuracy score

acc\_score = accuracy\_score(y\_test, predictions)

# Display the results

print (“Confusion Matrix”)

display cm\_df

print (f”Accuracy Score: {acc\_score}”)

print (“Classification Report”)

print (classification\_report(ytest, predictions)

1. Additional documentation
   1. [sklearn.model\_selection.train\_test\_split — scikit-learn 1.0.2 documentation](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html)
   2. [sklearn.metrics.precision\_recall\_fscore\_support — scikit-learn 1.0.2 documentation](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_recall_fscore_support.html)