21.2 Lesson Summary - Classification

Classification is a critical function of Machine Learning. Being able to properly identify the kind of data you are working with is extremely important. There are a number of different algorithms that allow you to classify your data.

Concept: **Logistic Regression** is a statistical technique used for the binary categorization of data. You can utilize Logistic Regressions in your code by importing sklearn's LogisticRegression function. To create an object to classify your data, validate its ability to classify a data set, and predict the classification of some *test\_data* you could use the following code:

*from sklearn.linear\_model import LogisticRegression*

*classifier = LogisticRegression()*

*classifier.fit(X, y)*

*print(classifier.score(X, y))*

*predictions = classifier.predict(new\_data)*

*print(predictions)*

* Activity: 01-Ins\_Logistic\_Regression, 02-Stu\_Voice\_Recognition

Concept: Certain datasets can best be categorized by **tree-based models**. Tree models operate by routing classification through a series of if-then statements, arriving at classification after traversing a series of logical conditions. If you were describing a set of birds including a cardinal, blue jay, and a blue bird you could classify the bird as a cardinal if it is red. If it is not red you could categorize it as a blue bird if it weighs less than 2 ounces. For these kinds of datasets, you can use **Decision Tree** and **Random Forest** Models. Decision Tree models can overfit the data and not generalize well. To create a decision tree model, you could use the following code:

*from sklearn import tree*

*clf = tree.DecisionTreeClassifier()*

*clf = clf.fit(X, y)*

*clf.score(X\_test, y\_test)*

Random Forest models rely on creating a sequence of smaller trees that combine to approximate the description of a Decision Tree. However, because the Random Forest is not a large, single tree it is more flexible in its ability to describe data. To create a Random Forest you can use the following code:

*from sklearn.ensemble import RandomForestClassifier*

*rf = RandomForestClassifier(n\_estimators=200)*

*rf = rf.fit(X, y)*

*rf.score(X\_test, y\_test)*

* Activity: 03-Ins\_Trees, 04-Stu\_Trees

Concept: **K Nearest Neighbor** (**KNN**) is a classification algorithm that assigns a classification to a data point by establishing the closest data and applying the classification of the closest data points to the data point in question. This algorithm is best applied using a loop to examine a variety of closest neighbors and assign a classification based on the classifications established during the loop. An odd number of data points should be compared so as not to allow a tie. If we wanted to fit a model with X and y classification data and get a score for the model to indicate how good it is, we could use the following code:

*my\_k = 10*

*knn = KNeighborsClassifier(n\_neighbors=my\_k)*

*knn.fit(X\_train\_scaled, y\_train)*

*print('k=9 Test Acc: %.3f' % knn.score(X\_test\_scaled, y\_test))*

If we wanted to test to see what the best k value is for a dataset, we could loop through different k values and record the scores of each set to find the highest stable score. For example:

*train\_scores = []*

*test\_scores = []*

*for k in range(1, 20, 2):*

*knn = KNeighborsClassifier(n\_neighbors=k)*

*knn.fit(X\_train\_scaled, y\_train)*

*train\_score = knn.score(X\_train\_scaled, y\_train)*

*test\_score = knn.score(X\_test\_scaled, y\_test)*

*train\_scores.append(train\_score)*

*test\_scores.append(test\_score)*

*print(f"k: {k}, Train/Test Score: {train\_score:.3f}/{test\_score:.3f}")*

* Activity: 05-Ins\_KNN, 06-Stu\_KNN

Concept: **Support Vector Machine** (**SVM**) is an algorithm that attempts to find a linear boundary between two classifications in a dataset. The boundary can be used to distinguish between the different classifications of data. To create a SVM linear classifier for a *X* and *y* dataset we could use the following code:

*from sklearn.svm import SVC*

*model = SVC(kernel='linear')*

*model.fit(X, y)*

To evaluate the efficacy of the model you can use the *classification\_report* function from the sklearn library. For example:

*from sklearn.metrics import classification\_report*

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

*print(classification\_report(y\_test, predictions,*

*target\_names=target\_names))*

* Activity: 07-Ins\_SVM, 08-Stu\_SVM