1. What is the estimated depth of a Decision Tree trained (unrestricted) on a one million instance training set?

~ 20

2. Is the Gini impurity of a node usually lower or higher than that of its parent? Is it always lower/greater, or is it usually lower/greater?

A node's Gini impurity is generally **lower than** that of its parent as the CART training algorithm cost function splits each of the nodes in a way that minimizes the weighted sum of its children's Gini impurities.

3. Explain if its a good idea to reduce max depth if a Decision Tree is overfitting the training set?

No by reducing max depth is not solution of overfitting.

By tuning the max depth parameters, we can overcome the overfitting problem in decision tree.

4. Explain if its a good idea to try scaling the input features if a Decision Tree underfits the training set?

NO

5. How much time will it take to train another Decision Tree on a training set of 10 million instances if it takes an hour to train a Decision Tree on a training set with 1 million instances?

As we know that the computational complexity of training a Decision Tree is given by **O(n × m log(m))**. So, when we multiplied the size of the training set by 10, then the training time will be multiplied by some factor, say K.

For 10 million instances i.e., m = 106, then we get the value of K ≈ 11.7. Therefore, we can expect the training time to be **roughly 11.7 hours**.

6. Will setting presort=True speed up training if your training set has 100,000 instances?

Presorting the training set speeds up training only if the dataset is smaller than a few thousand instances. If it contains 100,000 instances, setting **presort=True will considerably slow down training**.

7. Follow these steps to train and fine-tune a Decision Tree for the moons dataset:

a. To build a moons dataset, use make moons(n samples=10000, noise=0.4).

**from** sklearn.datasets **import** make\_moons

**from** sklearn.tree **import** DecisionTreeClassifier

X, y **=** make\_moons(n\_samples**=**10000, noise**=**0.4, random\_state**=**42)

b. Divide the dataset into a training and a test collection with train test split().

**from** sklearn.model\_selection **import** train\_test\_split

X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(X, y, test\_size**=**0.2, random\_state**=**42)

c. To find good hyperparameters values for a DecisionTreeClassifier, use grid search with cross-validation (with the GridSearchCV class). Try different values for max leaf nodes.

**from** sklearn.model\_selection **import** GridSearchCV

params **=** {'max\_leaf\_nodes': list(range(2, 100)), 'min\_samples\_split': [2, 3, 4]}

grid\_search\_cv **=** GridSearchCV(DecisionTreeClassifier(random\_state**=**42), params, n\_jobs**=-**1, verbose**=**1, cv**=**3)

grid\_search\_cv**.**fit(X\_train, y\_train)

grid\_search\_cv**.**best\_estimator\_

d. Use these hyperparameters to train the model on the entire training set, and then assess its output on the test set. You can achieve an accuracy of 85 to 87 percent.

**from** sklearn.metrics **import** accuracy\_score

y\_pred **=** grid\_search\_cv**.**predict(X\_test)

accuracy\_score(y\_test, y\_pred)

Accuracy Score: 0.8695

8. Follow these steps to grow a forest:

a. Using the same method as before, create 1,000 subsets of the training set, each containing 100 instances chosen at random. You can do this with Scikit-ShuffleSplit Learn's class.

**from** sklearn.model\_selection **import** ShuffleSplit

**import** numpy **as** np

n\_trees **=** 1000

n\_instances **=** 100

mini\_sets **=** []

rs **=** ShuffleSplit(n\_splits**=**n\_trees, test\_size**=**len(X\_train) **-** n\_instances, random\_state**=**42)

**for** mini\_train\_index, mini\_test\_index **in** rs**.**split(X\_train):

X\_mini\_train **=** X\_train[mini\_train\_index]

y\_mini\_train **=** y\_train[mini\_train\_index]

mini\_sets**.**append((X\_mini\_train, y\_mini\_train))

b. Using the best hyperparameter values found in the previous exercise, train one Decision Tree on each subset. On the test collection, evaluate these 1,000 Decision Trees. These Decision Trees would likely perform worse than the first Decision Tree, achieving only around 80% accuracy, since they were trained on smaller sets.

**from** sklearn.base **import** clone

forest **=** [clone(grid\_search\_cv**.**best\_estimator\_) **for** \_ **in** range(n\_trees)]

accuracy\_scores **=** []

**for** tree, (X\_mini\_train, y\_mini\_train) **in** zip(forest, mini\_sets):

tree**.**fit(X\_mini\_train, y\_mini\_train)

y\_pred **=** tree**.**predict(X\_test)

accuracy\_scores**.**append(accuracy\_score(y\_test, y\_pred))

np**.**mean(accuracy\_scores)

Accuracy score: 0.805449

c. Now the magic begins. Create 1,000 Decision Tree predictions for each test set case, and keep only the most common prediction (you can do this with SciPy's mode() function). Over the test collection, this method gives you majority-vote predictions.

Y\_pred **=** np**.**empty([n\_trees, len(X\_test)], dtype**=**np**.**uint8)

**for** tree\_index, tree **in** enumerate(forest):

Y\_pred[tree\_index] **=** tree**.**predict(X\_test)

**from** scipy.stats **import** mode

y\_pred\_majority\_votes, n\_votes **=** mode(Y\_pred, axis**=**0)

d. On the test range, evaluate these predictions: you should achieve a slightly higher accuracy than the first model (approx 0.5 to 1.5 percent higher). You've successfully learned a Random Forest classifier!

accuracy\_score(y\_test, y\_pred\_majority\_votes**.**reshape([**-**1]))

accuracy score: 0.872