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

Answer:---->Thus, if the training set contains one million instances, the Decision Tree will have a depth of log2(106) ≈ 20 (actually a bit more since the tree will generally not be perfectly well balanced).We can set the maximum depth of our decision tree using the max_depth parameter. The more the value of max_depth, the more complex your tree will be. The training error will off-course decrease if we increase the max_depth value but when our test data comes into the picture, we will get a very bad accuracy.It can also be described as the length of the longest path from the tree root to a leaf. The root node is considered to have a depth of 0. The Max Depth value cannot exceed 30 on a 32-bit machine.

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?

Answer:---->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.The lower the impurity measure the better the split is. If you look at the graph you will notice that a lower impurity measure is better. When the probability of the observation being class 1 is zero (all the way to the left of the graph) then that means it will always be class 2, and the impurity measure is zero.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?

Answer:---->If a Decision Tree is overfitting the training set, it may be a good idea to decrease max_depth, since this will constrain the model, regularizing it.It can also be described as the length of the longest path from the tree root to a leaf. The root node is considered to have a depth of 0. The Max Depth value cannot exceed 30 on a 32-bit machine. The default value is 30.In decision trees, over-fitting occurs when the tree is designed so as to perfectly fit all samples in the training data set. Thus it ends up with branches with strict rules of sparse data. Thus this effects the accuracy when predicting samples that are not part of the training set

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

Answer:----->If a Decision Tree is underfitting the training set, is it a good idea to try scaling the input features? Decision Trees don't care whether or not the training data is scaled or centered; scaling the input features will just be a waste of time.Decision trees and ensemble methods do not require feature scaling to be performed as they are not sensitive to the the variance in the data.The decision tree splits a node on a feature that increases the homogeneity of the node. This split on a feature is not influenced by other features. Hence, there is virtually no effect of the remaining features on the split. This is what makes them invariant to the scale of the features.

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?

Answer:--->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.Thus, if the training set contains one million instances, the Decision Tree will have a depth of log2(106) ≈ 20 (actually a bit more since the tree will generally not be perfectly well balanced).The things we need while training a decision tree are the nodes which are typically stored as if-else conditions. Test time complexity would be O(depth) since we have to move from root to a leaf node of the decision tree

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

Answer:---->If your training set contains 100,000 instances, will setting presort=True speed up training? 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.Thus, if the training set contains one million instances, the Decision Tree will have a depth of $\log_2(10^6) \approx 20$ (actually a bit more since the tree will generally not be perfectly well balanced).

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).

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

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.

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.

Answer:--->Continuing the previous exercise, generate 1,000 subsets of the training set, each containing 100 instances selected randomly. Hint: you can use Scikit-Learn's ShuffleSplit class for this. Train one Decision Tree on each subset, using the best hyperparameter values found in the previous exercise. Evaluate these 1,000 Decision Trees on the test set. Since they were trained on smaller sets, these Decision Trees will likely perform worse than the first Decision Tree, achieving only about 80% accuracy. Now comes the magic. For each test set instance, generate the predictions of the 1,000 Decision Trees, and keep only the most frequent prediction (you can use SciPy's mode() function for this). This approach gives you majority-vote predictions over the test set. Evaluate these predictions on the test set: you should obtain a slightly higher accuracy than your first model (about 0.5 to 1.5% higher). Congratulations, you have trained a Random Forest classifier!

Here is some example code to follow the steps you outlined:

Copy code

a. Generate the moons dataset

from sklearn.datasets import make_moons X, y = make_moons(n_samples=10000, noise=0.4)

b. Split the dataset into training and test sets

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)

c. Use grid search with cross-validation to find good hyperparameters for the Decision Tree

from sklearn.model_selection import GridSearchCV from sklearn.tree import DecisionTreeClassifier

Define the parameter grid for the grid search

param_grid = {'max_leaf_nodes': [10, 20, 30, 40, 50]}

Create the grid search object

grid_search = GridSearchCV(DecisionTreeClassifier(), param_grid, cv=5)

Fit the grid search object to the training data

grid_search.fit(X_train, y_train)

Get the best hyperparameters from the grid search

best_max_leaf_nodes = grid_search.best_params_['max_leaf_nodes']

d. Train the model on the entire training set using the best hyperparameters, and assess its output on the test set

model = DecisionTreeClassifier(max_leaf_nodes=best_max_leaf_nodes) model.fit(X_train, y_train)

Calculate the accuracy on the test set

accuracy = model.score(X_test, y_test)

print(f'Test accuracy: {accuracy:.2f}') This should allow you to train and fine-tune a Decision Tree for the moons dataset and achieve an accuracy of around 85 to 87 percent.

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.

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.

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.

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!

Answer:----->In summary, the steps to grow a random forest classifier are as follows:

1.Create 1,000 subsets of the training set, each containing 100 instances chosen at random.

2.Train one Decision Tree on each subset using the best hyperparameter values found in the previous exercise.

3.Evaluate the 1,000 Decision Trees on the test set.

4.Create 1,000 predictions for each test set case using the Decision Trees and keep only the most common prediction.

5.Evaluate the majority-vote predictions on the test set and compare the accuracy to the first model.

A random forest classifier is an ensemble model that combines the predictions of multiple decision tree models to make a more accurate prediction. The decision tree models are trained on different subsets of the training data and their predictions are combined using a majority vote. This helps to reduce the overfitting that can occur when using a single decision tree model.