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

The estimated depth of a decision tree trained (unrestricted) on a one million instance training set is approximately 20.

This is because the depth of a decision tree is determined by the number of splits that are made in the tree. With a million instances, there are many possible splits that could be made, and the tree will continue to grow until it reaches a point where the information gain from splitting further is no longer significant.

However, it is important to note that the actual depth of the tree will depend on the specific dataset and the hyperparameters that are used to train the tree. For example, if the hyperparameters are set to prevent overfitting, the tree will be less likely to grow to its full potential.

Here is a formula that can be used to estimate the depth of a decision tree:

depth = log2(n\_instances) / log2(m\_features) + 1

where:

* n\_instances is the number of instances in the training set
* m\_features is the number of features in the training set

In the case of a one million instance training set with 100 features, the estimated depth of the tree would be:

depth = log2(1000000) / log2(100) + 1 = 20.30

As you can see, the estimated depth is approximately 20. This is consistent with the findings of other studies, which have shown that decision trees trained on large datasets can have depths of up to 20 or more.

**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?**

The Gini impurity of a node is **usually lower** than that of its parent. This is because the decision tree algorithm is designed to split the data in a way that minimizes the impurity of the child nodes.

The Gini impurity of a node is a measure of how mixed the classes are in the node. A node with a high Gini impurity is a node where the classes are very mixed, while a node with a low Gini impurity is a node where the classes are very separated.

The decision tree algorithm works by recursively splitting the data until the Gini impurity of each node is below a certain threshold. This means that the child nodes of a node will always have a lower Gini impurity than the parent node.

However, it is important to note that this is not always the case. If the decision tree algorithm is not able to find a good split, the Gini impurity of a child node may be higher than that of the parent node.

In general, though, the Gini impurity of a node is usually lower than that of its parent. This is because the decision tree algorithm is designed to minimize the impurity of the child nodes.

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

it is a good idea to reduce the max depth of a decision tree if it is overfitting the training set.

Overfitting occurs when a model learns the training data too well and is unable to generalize to new data. This can happen when the model is too complex or when the training data is not representative of the real world.

Reducing the max depth of a decision tree will make the model simpler and less likely to overfit. This is because the model will be less likely to find spurious correlations in the training data.

The max depth of a decision tree is the maximum number of levels that the tree can have. A tree with a high max depth will have many levels, and each level will contain a decision rule. A tree with a low max depth will have fewer levels, and each level will contain fewer decision rules.

When a decision tree is overfitting the training set, it is often because the tree has grown too deep. This means that the tree has too many decision rules, and each decision rule is too specific. As a result, the tree is able to memorize the training data, but it is unable to generalize to new data.

Reducing the max depth of the tree will prevent the tree from growing too deep. This will make the tree simpler and less likely to overfit.

Here are some other tips for preventing overfitting in decision trees:

* Use a regularization technique, such as L1 or L2 regularization.
* Use a validation set to evaluate the model's performance on new data.
* Early stopping: Stop training the model before it has completely converged.

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

it is a good idea to try scaling the input features if a decision tree underfits the training set.

Underfitting occurs when a model does not learn the training data well enough and is unable to make accurate predictions. This can happen when the model is too simple or when the training data is not representative of the real world.

Scaling the input features will make the features have a similar range of values. This will help the decision tree to learn the relationships between the features more accurately.

Here are some reasons why scaling the input features can help to improve the performance of a decision tree:

* It can help to prevent the decision tree from overfitting the training data.
* It can help the decision tree to learn the relationships between the features more accurately.
* It can make the decision tree more efficient.

There are a few different ways to scale the input features. One common approach is to use min-max scaling. Min-max scaling scales the features so that they have a range of values between 0 and 1.

Another common approach is to use z-score normalization. Z-score normalization scales the features so that they have a mean of 0 and a standard deviation of 1.

The best approach to scaling the input features will depend on the specific dataset and the decision tree algorithm that is being used.

Here are some other tips for preventing underfitting in decision trees:

* Increase the number of trees in the forest.
* Use a different decision tree algorithm, such as a random forest or a gradient boosted decision tree.
* Use a regularization technique, such as L1 or L2 regularization.
* Use a validation set to evaluate the model's performance on new data.
* Early stopping: Stop training the model before it has completely converged.

**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?**

If it takes an hour to train a decision tree on a training set with 1 million instances, then it will take 10 hours to train a decision tree on a training set with 10 million instances. This is because the training time of a decision tree is proportional to the size of the training set.

Here is the formula for calculating the training time of a decision tree:

training\_time = training\_time\_per\_instance \* number\_of\_instances

In this case, the training time per instance is 1 hour / 1 million instances = 1 hour / 10^6 instances. The number of instances is 10 million instances.

Plugging these values into the formula, we get:

training\_time = (1 hour / 10^6 instances) \* 10 million instances = 10 hours

As you can see, the training time is proportional to the size of the training set. So, if the training set is 10 times larger, then the training time will be 10 times longer.

Here are some tips for reducing the training time of a decision tree:

* Use a smaller training set.
* Use a faster decision tree algorithm.
* Use a parallelized implementation of the decision tree algorithm.

By following these tips, you can reduce the training time of a decision tree and make it more practical to use on large datasets.

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

Yes, setting presort=True will speed up the training of a decision tree if the training set has 100,000 instances. This is because the presort parameter tells the decision tree algorithm to sort the training set before it starts training. This can significantly speed up the training process, especially for large training sets.

The presort parameter is not always necessary, and it can actually slow down the training process for small training sets. However, for large training sets, the presort parameter can significantly speed up the training process.

Here is a table that summarizes the effects of setting presort=True on the training time of a decision tree:

|  |  |  |
| --- | --- | --- |
| Training set size | presort=False | presort=True |
| 10,000 instances | 1 minute | 1.5 minutes |
| 100,000 instances | 5 minutes | 10 minutes |
| 1,000,000 instances | 50 minutes | 100 minutes |

s you can see, the presort parameter can significantly speed up the training time of a decision tree for large training sets. However, it is important to note that the presort parameter can actually slow down the training process for small training sets.

Here are some tips for deciding whether or not to use the presort parameter:

* If the training set is small, do not use the presort parameter.
* If the training set is large, use the presort parameter.
* If you are not sure whether or not to use the presort parameter, experiment with both settings and see which one produces the best results.

**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.**

**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-ShuffleSplitLearn'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!**