**MACHINE LEARNING ASSIGNMENT\_21**

**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 on a one million instance training set depends on various factors, such as the complexity of the data, the number of features, and the distribution of the target variable.

In general, a decision tree trained on a large dataset is likely to be deeper than one trained on a small dataset, as the tree needs to capture more complex patterns and interactions in the data. However, without more information about the specific characteristics of the dataset, it is difficult to estimate the depth of a decision tree.

In practice, it is common to limit the depth of decision trees to prevent overfitting, especially when dealing with large datasets. This can be done by setting a maximum depth or a minimum number of samples per leaf node. Alternatively, ensembling techniques such as Random Forest or Gradient Boosted Trees can be used to improve the performance of decision trees while avoiding overfitting.

**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 or equal to that of its parent node. This is because a decision tree algorithm seeks to split the data in a way that reduces the impurity of the resulting nodes. The impurity measure, such as Gini impurity or entropy, quantifies the degree of heterogeneity of the target variable in a given node.

When splitting a node into two child nodes, the algorithm evaluates different splitting criteria to find the one that leads to the highest reduction in impurity. If the impurity of a child node is lower than that of its parent, it means that the split has successfully partitioned the data into more homogeneous subsets.

However, it is possible that a split may increase the impurity of a child node compared to its parent. This can occur when the split separates a small but highly homogeneous subset of the data from a larger but more heterogeneous subset. In such cases, the algorithm may still choose to split the node if the overall reduction in impurity is deemed significant.

Overall, while the impurity of a node is usually lower than or equal to that of its parent, it is not always guaranteed to be so, as the optimal splitting criterion depends on the specific characteristics of the data.

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

Reducing the maximum depth of a decision tree can be a good idea if the tree is overfitting the training set. Overfitting occurs when a model is too complex and captures the noise in the training data, leading to poor generalization performance on new, unseen data.

A decision tree with a deeper maximum depth can capture more complex patterns and interactions in the training data, but it is also more prone to overfitting. By reducing the maximum depth, the decision tree becomes less complex and less likely to overfit.

However, reducing the maximum depth too much can also lead to underfitting, where the model is too simple and fails to capture important patterns in the data. Therefore, it is important to strike a balance between model complexity and generalization performance.

In practice, the maximum depth of a decision tree is often tuned using techniques such as cross-validation or grid search, which allow us to find the optimal value that balances model complexity and performance on new data.

Overall, reducing the maximum depth of a decision tree can be an effective way to prevent overfitting and improve generalization performance, but it is important to carefully balance the trade-off between model complexity and simplicity to avoid underfitting.

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

Scaling the input features of a decision tree may not be a good idea if the tree is underfitting the training set.

Decision trees are based on the partitioning of the feature space into rectangular regions, and the decision of how to split each region is based on the values of the individual features. The feature values themselves do not affect the decision directly, only the relative ordering of the values within each feature.

Therefore, the decision tree algorithm is typically insensitive to the scale of the input features, as long as the relative ordering of the feature values is preserved. In fact, scaling the input features may even introduce noise or distortions in the data that could worsen the performance of the decision tree.

Underfitting occurs when a model is too simple and fails to capture important patterns in the data. This can happen with decision trees when the tree is not deep enough or the number of training instances is too small. In such cases, scaling the input features is unlikely to improve the performance of the decision tree, and it may be necessary to use a more powerful algorithm or increase the complexity of the model.

Overall, scaling the input features is not usually necessary or helpful for decision trees, and other approaches should be considered if the tree is underfitting the training set.

**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?**

It's difficult to provide an exact time estimate for how long it will take to train another Decision Tree on a training set of 10 million instances, as the training time can depend on a variety of factors such as the computational resources available, the complexity of the Decision Tree model, and the specific implementation of the algorithm.

However, we can make a rough estimate based on the assumption that the training time is roughly proportional to the size of the training set. If it takes an hour to train a Decision Tree on a training set with 1 million instances, we might expect it to take around 10 hours to train a Decision Tree on a training set with 10 million instances, assuming that all other factors remain constant.

It's worth noting that some Decision Tree algorithms can be parallelized to take advantage of multiple processors or even distributed computing, which can significantly reduce the training time for large datasets. Additionally, some Decision Tree algorithms have been optimized for scalability and can handle very large datasets more efficiently. Therefore, the actual training time may be faster or slower than our rough estimate, depending on the specific implementation and the resources available.

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

Setting the presort parameter to True in a decision tree algorithm can speed up the training process for small datasets, but it can slow down the training process for large datasets.

The presort parameter controls whether the training data should be presorted before building the decision tree. If presort is set to True, the algorithm will sort the data based on the feature values for each node, which can make it easier to find the best split. However, sorting the data can be very time-consuming for large datasets, and the overhead of sorting can outweigh any potential speedup from using presorting.

In general, it is recommended to set presort to True only for small datasets with less than a few thousand instances. For larger datasets, the overhead of presorting may slow down the training process, and it may be more efficient to leave presort set to False.

Therefore, in the case of a training set with 100,000 instances, setting presort=True is unlikely to speed up the training process and may actually slow it down. It is recommended to leave presort set to the default value of False for larger datasets like this.

**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.**

Here are the steps to train and fine-tune a Decision Tree for the moons dataset:

**a.** Use the make\_moons function from scikit-learn's datasets module to generate the moons dataset with 10,000 samples and a noise level of 0.4:

from sklearn.datasets import make\_moons

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

**b.** Split the dataset into a training set and a test set using train\_test\_split from scikit-learn's model\_selection module:

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.** Use GridSearchCV from scikit-learn's model\_selection module to perform a grid search with cross-validation to find the best hyperparameters for a DecisionTreeClassifier. In this case, we will vary the max\_leaf\_nodes hyperparameter to find the optimal value:

from sklearn.tree import DecisionTreeClassifier

from sklearn.model\_selection import GridSearchCV

param\_grid = {'max\_leaf\_nodes': [2, 4, 6, 8, 10, 12, 14, 16]}

tree\_clf = DecisionTreeClassifier(random\_state=42)

grid\_search = GridSearchCV(tree\_clf, param\_grid, cv=5, scoring='accuracy')

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

print("Best hyperparameters:", grid\_search.best\_params\_)

**d.** Use the best hyperparameters found in the previous step to train a DecisionTreeClassifier on the entire training set and evaluate its performance on the test set:

from sklearn.metrics import accuracy\_score

tree\_clf = DecisionTreeClassifier(max\_leaf\_nodes=8, random\_state=42)

tree\_clf.fit(X\_train, y\_train)

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

print("Test set accuracy:", accuracy\_score(y\_test, y\_pred))

With these steps, we can train and fine-tune a DecisionTreeClassifier for the moons dataset and achieve an accuracy of around 85-87% on the test set.

**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**