# Assignment 21

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

**Answer:**

The estimated depth of a decision tree trained on a one million instance training set can vary based on several factors, including the complexity of the data, the number of features, and the nature of the problem being solved. However, it's important to note that an unrestricted decision tree can potentially grow to its maximum depth, which is equal to the number of instances in the training set.

In theory, if you have a one million instance training set and there are no early stopping criteria or constraints on the tree growth, the decision tree can reach a depth of one million. Each split in the tree divides the dataset into two subsets, and if the tree is allowed to keep splitting until each instance is in its own leaf node, the depth will be equal to the number of instances.

However, in practice, it is not common to have decision trees of such depth due to computational limitations, overfitting concerns, and the desire for interpretability. Typically, pruning techniques, regularization parameters, or other stopping criteria are applied to control the depth and prevent overfitting. The actual depth of a decision tree trained on a one million instance training set will depend on these factors and the specific implementation used.

##### 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:**

The Gini impurity of a node in a decision tree is typically lower than or equal to the Gini impurity of its parent. The Gini impurity measures the degree of impurity or uncertainty in a node, where a lower value indicates a purer node with more homogeneous class distribution.

When a decision tree is constructed, the splitting criterion (such as Gini impurity or entropy) is used to determine the best feature and threshold for splitting the data at each node. The goal is to reduce the impurity or increase the purity of the resulting child nodes compared to the parent node.

In most cases, the splitting criterion aims to minimize the impurity or maximize the purity when choosing the best split. Therefore, it is expected that the Gini impurity of the child nodes will be lower than or equal to the Gini impurity of the parent node. The splitting process seeks to improve the homogeneity of the data within each child node by finding the optimal split that reduces impurity.

However, it's important to note that in some cases, due to randomness or limitations of the dataset, the impurity of a child node may be slightly higher than its parent node. This can occur if the split does not result in a significant improvement in impurity reduction or if the split introduces some noise or variability. Nonetheless, the overall trend is for the Gini impurity to decrease as the tree grows and the algorithm finds the best splits to separate the classes effectively.

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

**Answer:**

Yes, it is generally a good idea to reduce the maximum depth of a decision tree if it is overfitting the training set. Overfitting occurs when a decision tree learns the training data too well and captures the noise or random variations in the data, resulting in poor generalization to unseen data.

By reducing the maximum depth of the decision tree, you limit the number of splits and, therefore, the complexity of the tree. This helps to control the tree's ability to memorize the training data and increases its generalization capability. Here are a few reasons why reducing the maximum depth can help combat overfitting:

1. Simplicity: A shallower tree is generally simpler and less complex, making it less prone to capturing noise or random variations in the training data. It allows the tree to focus on the more important patterns and relationships.
2. Avoiding Overly Specific Rules: Deep trees can create very specific rules that are tailored to the training data but may not generalize well to new instances. By reducing the depth, the tree is forced to capture more general and relevant patterns.
3. Avoiding Small Sample Sizes: Deep trees can have leaf nodes with very few instances, leading to overfitting as the model tries to make predictions based on a limited sample. Restricting the depth helps to ensure that each leaf node has a sufficient number of instances to make reliable predictions.
4. Improved Generalization: A shallower tree is more likely to generalize well to new, unseen data because it focuses on the dominant patterns and avoids overemphasizing noise or outliers in the training set.

However, it's essential to find the right balance when reducing the maximum depth. If the depth is too low, the decision tree may become too simple and may underfit the training data, resulting in poor performance. It is often recommended to tune the maximum depth parameter through techniques like cross-validation or using other regularization methods to find the optimal depth that minimizes overfitting without sacrificing too much performance on unseen data.

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

**Answer:**   
Scaling the input features is not typically necessary or beneficial for addressing underfitting in a decision tree. Decision trees are not sensitive to the scale of the input features because they make binary splits based on the feature values.

Underfitting occurs when a decision tree is too simple and fails to capture the underlying patterns and relationships in the training data. It can happen if the maximum depth of the tree is too low or if the tree is not allowed to grow to its full complexity.

To address underfitting in a decision tree, it is generally more effective to consider the following steps:

1. Increase Maximum Depth: If the tree is too shallow, allowing it to grow deeper can capture more complex patterns in the data. This can be done by increasing the maximum depth parameter.
2. Reduce Regularization: Decision trees often have regularization parameters that control the complexity of the tree. For example, in scikit-learn's DecisionTreeClassifier, the "min\_samples\_split" and "min\_samples\_leaf" parameters can be adjusted to allow more flexibility in the splits and leaf nodes.
3. Feature Selection: Ensure that the relevant features are included in the training set. If some features are not informative or contribute to the underfitting, they can be removed from the input.
4. Ensemble Methods: Ensemble methods like Random Forests or Gradient Boosted Trees can be used to combine multiple decision trees and improve the overall predictive performance.

Scaling the input features, which is commonly done for models that rely on distance metrics or have different scales of features, is not necessary for decision trees. Decision trees split the data based on feature thresholds, and the scale of the features does not impact their relative ordering or decision-making process. Therefore, scaling features is unlikely to address underfitting in a decision tree and is not considered a typical step for improving its performance.

##### 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:**

The training time for a decision tree can vary depending on several factors, including the complexity of the dataset, the implementation of the decision tree algorithm, the hardware configuration, and the specific software used. However, if we assume a linear relationship between the training set size and training time, we can estimate the time it would take to train a decision tree on a larger dataset.

Given that it takes an hour to train a decision tree on a training set with 1 million instances, we can estimate the time it would take to train another decision tree on a training set with 10 million instances by using the ratio of the training set sizes:

Training time for 10 million instances = (Training time for 1 million instances) \* (10 million / 1 million)

Training time for 10 million instances = (1 hour) \* (10)

Therefore, it would take approximately 10 hours to train a 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. This is a rough estimate, and the actual training time can vary depending on the factors mentioned earlier.

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

**Answer:**

The presort parameter in scikit-learn's DecisionTreeClassifier controls whether to presort the data for faster tree building. By default, presort is set to False, which means the data is not presorted before training the decision tree.

Setting presort=True can potentially speed up training if the number of instances in the training set is relatively small, such as 100,000 instances. Presorting the data involves sorting it based on each feature, which can be time-consuming for large datasets but can offer performance improvements for smaller datasets.

However, it's important to note that the benefits of using presort=True may diminish as the size of the training set increases. Sorting the data for each feature can become computationally expensive and may result in longer training times for larger datasets. Therefore, it's recommended to consider the size of the dataset and the available computational resources before enabling presort=True.

In summary, enabling presort=True can potentially speed up training for a training set with 100,000 instances. However, for larger datasets, it may be more efficient to leave presort as the default False value. It's always a good idea to experiment with different settings and measure the actual training time to determine the most suitable approach for your specific dataset and computational resources.

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

1. To build a moons dataset, use make moons(n samples=10000, noise=0.4).
2. Divide the dataset into a training and a test collection with train test split().
3. 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.
4. 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:**

Here's an example code that follows the steps you mentioned to train and fine-tune a Decision Tree for the moons dataset using scikit-learn:

from sklearn.datasets import make\_moons

from sklearn.model\_selection import train\_test\_split, GridSearchCV

from sklearn.tree import DecisionTreeClassifier

# Step 1: Build the moons dataset

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

# Step 2: Split the dataset into training and test sets

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

# Step 3: Perform grid search with cross-validation to find optimal hyperparameters

param\_grid = {'max\_leaf\_nodes': [None, 5, 10, 15, 20]} # Try different values for max\_leaf\_nodes

grid\_search = GridSearchCV(DecisionTreeClassifier(random\_state=42), param\_grid, cv=5)

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

best\_params = grid\_search.best\_params\_

print("Best Hyperparameters:", best\_params)

# Step 4: Train the model with the best hyperparameters on the entire training set

best\_model = DecisionTreeClassifier(random\_state=42, \*\*best\_params)

best\_model.fit(X\_train, y\_train)

# Evaluate the model on the test set

accuracy = best\_model.score(X\_test, y\_test)

print("Test Set Accuracy:", accuracy)

This code generates a moons dataset with 10,000 samples and a noise level of 0.4. It then splits the dataset into training and test sets using an 80:20 ratio. Grid search with cross-validation is performed to find the best hyperparameters for the Decision Tree model, specifically exploring different values for max\_leaf\_nodes. The model is then trained on the entire training set using the best hyperparameters, and its accuracy is evaluated on the test set.

By adjusting the param\_grid, you can explore different hyperparameters and fine-tune the model to achieve higher accuracy on the test set. The accuracy achieved in the range of 85 to 87 percent mentioned in the description can be influenced by various factors, including the noise level in the dataset and the specific hyperparameters explored in the grid search.

##### 8. Follow these steps to grow a forest:

1. 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.
2. 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.
3. 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.
4. 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:**

Here's an example code that follows the steps you mentioned to grow a Random Forest classifier using the scikit-learn library:

from sklearn.datasets import make\_moons

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

from sklearn.tree import DecisionTreeClassifier

import numpy as np

from scipy.stats import mode

# Step 1: Create subsets of the training set

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

subsets = []

shuffle\_split = ShuffleSplit(n\_splits=1000, test\_size=0.99, random\_state=42)

for train\_index, \_ in shuffle\_split.split(X):

subsets.append((X[train\_index], y[train\_index]))

# Step 2: Train Decision Trees on each subset

decision\_trees = []

best\_params = {'max\_leaf\_nodes': 10} # Use the best hyperparameters found in the previous exercise

for subset in subsets:

model = DecisionTreeClassifier(random\_state=42, \*\*best\_params)

model.fit(subset[0], subset[1])

decision\_trees.append(model)

# Step 3: Make predictions using the ensemble of Decision Trees

predictions = np.zeros((len(X), len(decision\_trees)))

for i, model in enumerate(decision\_trees):

predictions[:, i] = model.predict(X)

ensemble\_predictions, \_ = mode(predictions, axis=1)

# Step 4: Evaluate the ensemble predictions on the test set

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

ensemble\_accuracy = np.sum(ensemble\_predictions.reshape(-1) == y\_test) / len(y\_test)

print("Random Forest Accuracy:", ensemble\_accuracy)

In this code, we start by creating 1,000 subsets of the training set using the ShuffleSplit class from scikit-learn. Each subset contains 100 randomly chosen instances. Then, we train a Decision Tree on each subset using the best hyperparameters found in the previous exercise.

Next, we make predictions using all the Decision Trees and store them in a matrix called predictions. We use the mode() function from SciPy to find the most common prediction for each test case, resulting in the ensemble predictions.

Finally, we evaluate the ensemble predictions on the test set and calculate the accuracy. The accuracy of the Random Forest classifier is expected to be slightly higher than that of the individual Decision Tree model.

By adjusting the parameters such as the number of subsets and the size of each subset, you can experiment with different configurations and further optimize the Random Forest classifier's performance.