**1**

The depth of a Decision Tree trained without restrictions on a dataset with 1 million instances can be approximated by the formula for the depth of a binary tree.

d≈log2(n)d \approx \log\_2(n)d≈log2​(n)

For a dataset with 1 million instances (n=1,000,000n = 1,000,000n=1,000,000):

d≈log2(1,000,000)d \approx \log\_2(1,000,000)d≈log2​(1,000,000)

Let's calculate this:

Log(1,000,000)≈19.93\log\_2(1,000,000) \approx 19.93log2​(1,000,000)≈19.93

**2**

**Gini Impurity and Decision Trees**

* **Gini impurity** measures the likelihood of an incorrect classification of a randomly chosen element if it was randomly labeled according to the distribution of labels in the node.
* The Decision Tree algorithm selects splits that minimize the weighted average impurity of the child nodes.

**General Behavior**

* **Generally Lower**: During the training process, the algorithm aims to reduce the impurity, so the Gini impurity of child nodes is typically less than that of the parent node.
* **Not Always Lower**: While the algorithm generally reduces impurity, there might be cases where impurity remains the same or even increases slightly in some child nodes due to the nature of the splits and the data distribution. However, the overall weighted average impurity of the child nodes combined will still be lower than the parent's impurity.
* **Why Not Always Lower?**
* **Splits and Data Distribution**: The exact split depends on the data distribution. If the data is such that a perfect split isn't possible, some child nodes might end up with a slightly higher impurity.
* **Weighted Average**: The goal is to minimize the weighted average impurity of the child nodes, not necessarily each individual child node. So, some child nodes might have higher impurity as long as the overall weighted average impurity is reduced.

**3**

**Simpler Model**: By reducing the maximum depth, you limit the number of splits the tree can make. This results in a simpler model that is less likely to capture noise and more likely to generalize well to unseen data.

**Reduced Variance**: A shallower tree has fewer splits, which generally reduces the model's variance. High variance is a key symptom of overfitting, where the model performs well on training data but poorly on validation or test data.

**Improved Generalization**: A less complex tree (with reduced depth) can better capture the underlying patterns in the data without memorizing the training data, thus improving its performance on new data.

**4**

Scaling the input features is generally not the primary method for addressing underfitting in Decision Trees. Unlike many other machine learning algorithms (e.g., linear regression, SVMs, k-nearest neighbors), Decision Trees are not sensitive to the scale of the input features. This is because Decision Trees operate by making splits based on the feature values and not on their magnitude.

**5**

Training a Decision Tree can be computationally intensive, and the time complexity depends on various factors including the number of instances (n) and the number of features (m). The time complexity for training a Decision Tree is generally considered to be O(nlog⁡n⋅m)O(n \log n \cdot m)O(nlogn⋅m), where:

* nnn is the number of instances.
* mmm is the number of features.

Given this complexity, let's reason through the problem:

1. **Current Training Time**: 1 hour for 1 million instances.
2. **Target Training Set Size**: 10 million instances, which is 10 times the original size.

**Time Complexity Analysis**

* The time complexity for training a Decision Tree is approximately O(nlog⁡n)O(n \log n)O(nlogn) for a given number of features.
* When the number of instances increases from n to 10n10n10n, the training time will increase roughly by a factor of 10log(10n)/log(n)10 \log(10n) / log(n)10log(10n)/log(n).

**6**

**Impact on Training Time**

1. **Small Datasets**: For small datasets, presorting can speed up the training because the overhead of sorting the data is outweighed by the speedup in finding the best splits.
2. **Large Datasets**: For larger datasets, the overhead of presorting the data can become significant, and this can actually slow down the training process.

**Training Set with 100,000 Instances**

With 100,000 instances, the dataset is relatively large. In this case:

* **Overhead of Sorting**: Sorting 100,000 instances initially can be computationally expensive.
* **Benefit of Presorting**: The benefit of having the data presorted for finding splits might not outweigh the initial sorting cost.

**7 (a)**

from sklearn.datasets import make\_moons

import matplotlib.pyplot as plt

# Generate moons dataset

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

# Plot the dataset

plt.figure(figsize=(10, 6))

plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Paired, s=10)

plt.title("Moons Dataset with 10,000 samples and noise=0.4")

plt.xlabel("Feature 1")

plt.ylabel("Feature 2")

plt.show()

**7(b)**

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

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

**# Generate moons dataset**

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

**# Split the dataset into training and testing sets**

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

**# Print the shapes of the resulting datasets**

**print("X\_train shape:", X\_train.shape)**

**print("X\_test shape:", X\_test.shape)**

**print("y\_train shape:", y\_train.shape)**

**print("y\_test shape:", y\_test.shape)**

**7(c)**

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

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

**from sklearn.tree import DecisionTreeClassifier**

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

**# Generate moons dataset**

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

**# Split the dataset into training and testing sets**

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

**# Define the parameter grid**

**param\_grid = {**

**'max\_leaf\_nodes': [None, 5, 10, 20, 50, 100, 200]**

**}**

**# Initialize the DecisionTreeClassifier**

**dt\_clf = DecisionTreeClassifier(random\_state=42)**

**# Initialize GridSearchCV**

**grid\_search = GridSearchCV(estimator=dt\_clf, param\_grid=param\_grid, cv=5, scoring='accuracy', verbose=1)**

**# Fit GridSearchCV**

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

**# Print the best parameters and best score**

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

**print("Best Cross-validation Accuracy:", grid\_search.best\_score\_)**

**# Evaluate the best model on the test set**

**best\_clf = grid\_search.best\_estimator\_**

**y\_pred = best\_clf.predict(X\_test)**

**test\_accuracy = accuracy\_score(y\_test, y\_pred)**

**print("Test set accuracy of the best model:", test\_accuracy)**

**7(d)**

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

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

**from sklearn.tree import DecisionTreeClassifier**

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

**# Generate moons dataset**

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

**# Split the dataset into training and testing sets**

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

**# Define the parameter grid**

**param\_grid = {**

**'max\_leaf\_nodes': [None, 5, 10, 20, 50, 100, 200]**

**}**

**# Initialize the DecisionTreeClassifier**

**dt\_clf = DecisionTreeClassifier(random\_state=42)**

**# Initialize GridSearchCV**

**grid\_search = GridSearchCV(estimator=dt\_clf, param\_grid=param\_grid, cv=5, scoring='accuracy', verbose=1)**

**# Fit GridSearchCV**

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

**# Get the best model**

**best\_clf = grid\_search.best\_estimator\_**

**# Train the best model on the full training set**

**best\_clf.fit(X\_train, y\_train)**

**# Predict on the test set**

**y\_pred = best\_clf.predict(X\_test)**

**# Evaluate accuracy on the test set**

**test\_accuracy = accuracy\_score(y\_test, y\_pred)**

**print("Test set accuracy:", test\_accuracy)**

**8(a)**

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

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

**import numpy as np**

**# Generate moons dataset**

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

**# Split the dataset into training and testing sets**

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

**# Initialize ShuffleSplit to generate 1,000 subsets**

**shuffle\_split = ShuffleSplit(n\_splits=1000, train\_size=100, random\_state=42)**

**# Lists to store subsets**

**subsets\_X = []**

**subsets\_y = []**

**# Generate subsets**

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

**X\_subset = X\_train[train\_index]**

**y\_subset = y\_train[train\_index]**

**subsets\_X.append(X\_subset)**

**subsets\_y.append(y\_subset)**

**# Convert lists to numpy arrays**

**subsets\_X = np.array(subsets\_X)**

**subsets\_y = np.array(subsets\_y)**

**# Print shapes of subsets**

**print("Shape of subsets\_X:", subsets\_X.shape)**

**print("Shape of subsets\_y:", subsets\_y.shape)**

**8(b)**

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

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

**from sklearn.tree import DecisionTreeClassifier**

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

**import numpy as np**

**# Generate moons dataset**

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

**# Split the dataset into training and testing sets**

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

**# Define the best hyperparameters found previously**

**best\_params = {'max\_leaf\_nodes': 50} # Replace with your best params from GridSearchCV**

**# Initialize DecisionTreeClassifier with best hyperparameters**

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

**# Initialize ShuffleSplit to generate 1,000 subsets**

**shuffle\_split = ShuffleSplit(n\_splits=1000, train\_size=100, random\_state=42)**

**# Lists to store predictions on test set**

**y\_pred\_list = []**

**# Train Decision Trees on subsets and evaluate on test set**

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

**X\_subset = X\_train[train\_index]**

**y\_subset = y\_train[train\_index]**

**# Train Decision Tree on the subset**

**dt\_clf\_subset = DecisionTreeClassifier(\*\*best\_params, random\_state=42)**

**dt\_clf\_subset.fit(X\_subset, y\_subset)**

**# Predict on the test set and store predictions**

**y\_pred = dt\_clf\_subset.predict(X\_test)**

**y\_pred\_list.append(y\_pred)**

**# Convert list of predictions to numpy array**

**y\_pred\_array = np.array(y\_pred\_list)**

**# Compute accuracy for each Decision Tree and average accuracy**

**accuracies = [accuracy\_score(y\_test, y\_pred) for y\_pred in y\_pred\_array]**

**average\_accuracy = np.mean(accuracies)**

**print(f"Average Test set accuracy of 1,000 Decision Trees: {average\_accuracy:.4f}")**

**8(c)**

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

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

**from sklearn.tree import DecisionTreeClassifier**

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

**import numpy as np**

**from scipy.stats import mode**

**# Generate moons dataset**

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

**# Split the dataset into training and testing sets**

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

**# Define the best hyperparameters found previously**

**best\_params = {'max\_leaf\_nodes': 50} # Replace with your best params from GridSearchCV**

**# Initialize DecisionTreeClassifier with best hyperparameters**

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

**# Initialize ShuffleSplit to generate 1,000 subsets**

**shuffle\_split = ShuffleSplit(n\_splits=1000, train\_size=100, random\_state=42)**

**# Lists to store predictions on test set**

**y\_pred\_list = []**

**# Train Decision Trees on subsets and store predictions on test set**

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

**X\_subset = X\_train[train\_index]**

**y\_subset = y\_train[train\_index]**

**# Train Decision Tree on the subset**

**dt\_clf\_subset = DecisionTreeClassifier(\*\*best\_params, random\_state=42)**

**dt\_clf\_subset.fit(X\_subset, y\_subset)**

**# Predict on the test set and store predictions**

**y\_pred = dt\_clf\_subset.predict(X\_test)**

**y\_pred\_list.append(y\_pred)**

**# Convert list of predictions to numpy array**

**y\_pred\_array = np.array(y\_pred\_list)**

**# Compute majority-vote predictions over the test set**

**y\_pred\_majority, \_ = mode(y\_pred\_array, axis=0)**

**# Calculate accuracy of majority-vote predictions**

**test\_accuracy\_majority = accuracy\_score(y\_test, y\_pred\_majority.flatten())**

**print(f"Test set accuracy with majority-vote predictions: {test\_accuracy\_majority:.4f}")**

**8(d)**

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

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

**from sklearn.tree import DecisionTreeClassifier**

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

**import numpy as np**

**from scipy.stats import mode**

**# Generate moons dataset**

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

**# Split the dataset into training and testing sets**

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

**# Define the best hyperparameters found previously**

**best\_params = {'max\_leaf\_nodes': 50} # Replace with your best params from GridSearchCV**

**# Initialize DecisionTreeClassifier with best hyperparameters**

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

**# Initialize ShuffleSplit to generate 1,000 subsets**

**shuffle\_split = ShuffleSplit(n\_splits=1000, train\_size=100, random\_state=42)**

**# Lists to store predictions on test set**

**y\_pred\_list = []**

**# Train Decision Trees on subsets and store predictions on test set**

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

**X\_subset = X\_train[train\_index]**

**y\_subset = y\_train[train\_index]**

**# Train Decision Tree on the subset**

**dt\_clf\_subset = DecisionTreeClassifier(\*\*best\_params, random\_state=42)**

**dt\_clf\_subset.fit(X\_subset, y\_subset)**

**# Predict on the test set and store predictions**

**y\_pred = dt\_clf\_subset.predict(X\_test)**

**y\_pred\_list.append(y\_pred)**

**# Convert list of predictions to numpy array**

**y\_pred\_array = np.array(y\_pred\_list)**

**# Compute majority-vote predictions over the test set**

**y\_pred\_majority, \_ = mode(y\_pred\_array, axis=0)**

**# Calculate accuracy of majority-vote predictions**

**test\_accuracy\_majority = accuracy\_score(y\_test, y\_pred\_majority.flatten())**

**print(f"Test set accuracy with majority-vote predictions: {test\_accuracy\_majority:.4f}")**