Assignment 21 Solutions ¶

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

In [ ]:
```
The estimated depth of a Decision Tree trained on a one million instance trair
the data and the specific algorithm used. However, it is generally expected th
deep, potentially reaching several levels, to capture the intricacies of the c
```

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 typically lower than or equal to that of its
parent node. It is not always lower or always greater,
as it depends on the specific splitting criterion and the distribution of
classes within the node. However, in most cases, the Gini
impurity tends to decrease as we move from the parent node to its child
nodes, as the goal of splitting is to create subsets with
more homogeneous class distributions.
```

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
model is overfitting the training set.
Overfitting occurs when the tree becomes too complex, capturing noise and
outliers in the data instead of general patterns.
By reducing the maximum depth, we limit the tree's ability to create complex
and detailed splits, which helps prevent overfitting.

A shallower tree with a reduced maximum depth tends to have higher bias and
lower variance. It may sacrifice some of the model's
ability to capture intricate patterns in the training data but can improve
its ability to generalize to unseen data.
It can result in a simpler and more interpretable model that avoids
overfitting and performs better on test data or new observations.

However, the optimal maximum depth depends on the specific dataset and
problem. It is important to find the right balance between
underfitting and overfitting by tuning the hyperparameters and evaluating
the model's performance on a validation or test set.
```

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 is generally not necessary or beneficial for a
Decision Tree if it is underfitting the training set.
```

```
Decision Trees are not sensitive to the scale of the input features because
they make decisions based on feature thresholds rather
than the actual feature values.

Underfitting occurs when the Decision Tree is too simple and fails to
capture the underlying patterns in the data.
Scaling the input features does not address this issue because it does not
affect the structure or complexity of the tree.

Instead of scaling the features, other approaches should be considered to
address underfitting in a Decision Tree.
Some possible solutions include increasing the maximum depth of the tree,
allowing more splits and complexity, adding
more relevant features, or using ensemble techniques such as Random Forests
to combine multiple trees.

However, it's important to note that if there are other algorithms or models
in the pipeline that rely on scaled features,
then scaling may be necessary for consistency.
```

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 ?

```
Assuming that the time to train a Decision Tree is directly proportional to
the size of the training set,
we can estimate the time it will take to train a Decision Tree on a training
set of 10 million instances.

If it takes 1 hour to train a Decision Tree on a training set of 1 million
instances, we can use the concept of proportionality
to estimate the time for a 10 million-instance training set.

The ratio between the training set sizes is 10 million / 1 million = 10. So,
we can assume that training a Decision Tree on a 10
million-instance training set will take approximately 10 times longer than
training on a 1 million-instance training set.

Therefore, it can be estimated that training another Decision Tree on a
training set of 10 million instances would take approximately
10 hours. However, it's important to note that this is a rough estimate, and
the actual training time can be influenced by various
factors such as hardware capabilities, optimization techniques, and
complexity of the data and model.
```

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

```
Setting presort=True in a Decision Tree algorithm specifies that the
training data should be presorted to improve the speed of training.
However, whether it will actually speed up training depends on the specific
characteristics of the dataset and the implementation of
the algorithm.
```

```
For smaller datasets, such as the one with 100,000 instances mentioned in
the question, the overhead of presorting the data may outweigh
the potential speed improvements. Presorting requires additional
computational resources and time upfront to sort the data,
which may not be efficient for smaller datasets.

In general, the decision to use presort=True should be based on the size and
complexity of the dataset, the available computational
resources, and the specific implementation of the algorithm. It is
recommended to try training with and without presorting and compare
the training times to determine the effectiveness of using presort=True for
a particular dataset.
```

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.

In [1]:
```
from sklearn.datasets import make_moons
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeClassifier

# Step a: Create the moons dataset
X, y = make_moons(n_samples=10000, noise=0.4)

# Step b: 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, randc

# Step c: Perform grid search to find the best hyperparameters
param_grid = {
    'max_leaf_nodes': [None, 5, 10, 20, 50]
}

grid_search = GridSearchCV(DecisionTreeClassifier(), param_grid, cv=5)
grid_search.fit(X_train, y_train)

best_params = grid_search.best_params_

# Step d: Train the model with the best hyperparameters on the entire training
clf = DecisionTreeClassifier(**best_params)
clf.fit(X_train, y_train)

# Evaluate the model on the test set
accuracy = clf.score(X_test, y_test)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.842
```

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!

In [3]:
```
import numpy as np
from sklearn.datasets import make_moons
from sklearn.model_selection import ShuffleSplit
from sklearn.tree import DecisionTreeClassifier
from scipy.stats import mode

# Step a: Create subsets of the training set
X, y = make_moons(n_samples=10000, noise=0.4)
rs = ShuffleSplit(n_splits=1000, train_size=100, random_state=42)
subsets = []

for train_index, _ in rs.split(X):
    X_subset, y_subset = X[train_index], y[train_index]
    subsets.append((X_subset, y_subset))

# Step b: Train Decision Trees on each subset and evaluate on the test set
dtrees = []
accuracies = []

for X_subset, y_subset in subsets:
    dtree = DecisionTreeClassifier(**best_params)
    dtree.fit(X_subset, y_subset)
    dtrees.append(dtree)
    accuracy = dtree.score(X_test, y_test)
    accuracies.append(accuracy)

# Step c: Make predictions using the ensemble of Decision Trees
predictions = np.array([dtree.predict(X_test) for dtree in dtrees])
ensemble_predictions = mode(predictions, axis=0)[0]

# Step d: Evaluate the ensemble predictions on the test set
ensemble_accuracy = np.mean(ensemble_predictions == y_test)
improvement = ensemble_accuracy - accuracy

print("Ensemble Accuracy:", ensemble_accuracy)
print("Accuracy Improvement:", improvement)
```

```
Ensemble Accuracy: 0.85
Accuracy Improvement: 0.051999999999999935

C:\Users\bhave\AppData\Local\Temp\ipykernel_24776\2696967645.py:29: FutureWar
ning: Unlike other reduction functions (e.g. `skew`, `kurtosis`), the default
behavior of `mode` typically preserves the axis it acts along. In SciPy 1.11.
0, this behavior will change: the default value of `keepdims` will become Fal
se, the `axis` over which the statistic is taken will be eliminated, and the
value None will no longer be accepted. Set `keepdims` to True or False to avo
id this warning.
  ensemble_predictions = mode(predictions, axis=0)[0]
```

In [ ]: