CHAPTER 6

DECISION TREES

JULY 26, 2024

HANDS-ON-MACHINE-LEARNING-WITH-SCIKIT-LEARN-KERAS-AND-TENSORFLOW

By Aurelien-Geron

1. **Decision Trees Overview**
    - Decision Trees can perform classification, regression, and multi-output tasks.
    - They are powerful and can fit complex datasets but are prone to overfitting.
2. **Training and Visualizing**
    - Use DecisionTreeClassifier to train a Decision Tree on the Iris dataset.
    - Visualize the tree using Graphviz.
3. **Predictions**
    - The tree makes predictions by traversing from the root to a leaf node.
    - Decision Trees require little data preparation, like feature scaling.
4. **Attributes and Impurity Measures**
    - Nodes have attributes like samples, value, and gini.
    - Gini impurity and entropy are common measures for node impurity.
5. **CART Algorithm**
    - Scikit-Learn uses the CART algorithm for Decision Trees.
    - The algorithm is greedy and creates binary trees.
6. **Computational Complexity**
    - Prediction complexity is O(log(m)).
    - Training complexity is O(n * m log(m)).
7. **Regularization**
    - To avoid overfitting, restrict tree growth using parameters like max_depth and min_samples_split.
8. **Regression**
    - Decision Trees can also be used for regression tasks with DecisionTreeRegressor.
9. **Limitations**
    - Decision Trees are sensitive to training data variations and orthogonal decision boundaries.

EXERCISE

1. What is the approximate depth of a Decision Tree training (without restrictions) on a training set with 1 million instances?

Answer: The approximate depth of a Decision Tree trained on 1 million instances is around log2(1,000,000), which is approximately 20. This is because a balanced binary tree with mmm leaves has a depth of about log⁡2(m)\log_2(m)log2(m).

2. Is a node's Gini impurity generally lower or greater than its parent's? Is it generally lower/greater, or always lower/greater?

Answer: A node's Gini impurity is generally lower than its parent's. This is always the case because the tree splits in a way that reduces impurity, leading to child nodes with lower Gini impurity than their parent.

3. If a Decision Tree is overfitting the training set, is it a good idea to try decreasing max_depth?

Answer: Yes, it is a good idea to try decreasing max_depth if a Decision Tree is overfitting the training set. Reducing the maximum depth of the tree helps in limiting its complexity and hence, reduces overfitting.

4. If a Decision Tree is underfitting the training set, is it a good idea to try scaling the input features?

Answer: No, scaling the input features is not generally necessary for Decision Trees. Instead, you should try increasing max_depth, min_samples_split, or min_samples_leaf to allow the tree to capture more complex patterns in the data.

5. If it takes one hour to train a Decision Tree on a training set containing 1 million instances, roughly how much time will it take to train another Decision Tree on a training set containing 10 million instances?

Answer: The training time complexity of a Decision Tree is O(n·mlog(m)). If it takes one hour to train on 1 million instances, training on 10 million instances would roughly take 10 times longer, so approximately 10 hours, assuming linear scalability.

6. If your training set contains 100,000 instances, will setting presort=True speed up training?

Answer: Setting presort=True can speed up training for smaller datasets because it allows the algorithm to sort data once rather than repeatedly. However, for larger datasets, it can actually slow down training due to the overhead of sorting. For a dataset with 100,000 instances, it might not be beneficial.

7. Train and fine-tune a Decision Tree for the moons dataset.

a. Generate a moons dataset using make_moons(n_samples=10000, noise=0.4).

Answer:

```
from sklearn.datasets import make_moons

X, y = make_moons(n_samples=10000, noise=0.4)
```

b. Split it into a training set and a test set using train_test_split().

Answer:

```
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 grid search with cross-validation (with the help of the GridSearchCV class) to find good hyperparameter values for a DecisionTreeClassifier. Hint: try various values for max_leaf_nodes.

Answer:

```
from sklearn.model_selection import GridSearchCV

from sklearn.tree import DecisionTreeClassifier


param_grid = {'max_leaf_nodes': [10, 20, 30, 40, 50]}

grid_search = GridSearchCV(DecisionTreeClassifier(), param_grid, cv=5, scoring='accuracy')

grid_search.fit(X_train, y_train)

best_params = grid_search.best_params_

best_params
```

d. Train it on the full training set using these hyperparameters and measure your model's performance on the test set. You should get roughly 85% to 87% accuracy.

Answer:

```
best_model = DecisionTreeClassifier(max_leaf_nodes=best_params['max_leaf_nodes'])
best_model.fit(X_train, y_train)
accuracy = best_model.score(X_test, y_test)
accuracy
```

8. Grow a forest.

a. Continuing the previous exercise, generate 1,000 subsets of the training set, each containing 100 instances selected randomly. Hint: you can use Scikit-Learn's ShuffleSplit class for this.

Answer:

```
from sklearn.model_selection import ShuffleSplit


n_trees = 1000
n_instances = 100
mini_sets = []


shuffle_split = ShuffleSplit(n_splits=n_trees, test_size=n_instances, random_state=42)
for mini_train_index, _ in shuffle_split.split(X_train):
    X_mini_train = X_train[mini_train_index]
    y_mini_train = y_train[mini_train_index]
    mini_sets.append((X_mini_train, y_mini_train))
```

b. Train one Decision Tree on each subset, using the best hyperparameter values found above. Evaluate these 1,000 Decision Trees on the test set. Since they were trained on smaller sets, these Decision Trees will likely perform worse than the first Decision Tree you trained on the full training set.

Answer:

```
from sklearn.metrics import accuracy_score

import numpy as np


forest = [DecisionTreeClassifier(max_leaf_nodes=best_params['max_leaf_nodes']) for _ in range(n_trees)]


accuracy_scores = []


for tree, (X_mini_train, y_mini_train) in zip(forest, mini_sets):

    tree.fit(X_mini_train, y_mini_train)

    y_pred = tree.predict(X_test)

    accuracy_scores.append(accuracy_score(y_test, y_pred))


np.mean(accuracy_scores)
```

Advantages and disadvantages of using Decision Trees.

- **Advantages**: Simple to understand and interpret, requires little data preprocessing, handles both numerical and categorical data.
- **Disadvantages**: Prone to overfitting, sensitive to noisy data, can create biased trees if some classes dominate.