Assignment 21 Solutions

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

Ans: The depth of a well-balanced binary tree containing m leaves is equal to $\log2(m)^3$, rounded up. A binary Decision Tree (one that makes only binary decisions, as is the case of all trees in Scikit-Learn) will end up more or less well balanced at the end of training, with one leaf per training instance if it is trained without restrictions. Thus, if the training set contains one million instances, the Decision Tree will have a depth of $\log2(10^6) \approx 20$ (actually a bit more since the tree will generally not be perfectly well balanced).

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 ?

Ans: A node's Gini impurity is generally lower than its parent's. This is ensured by the CART training algorithm's cost function, which splits each node in a way that minimizes the weighted sum of its children's Gini impurities. However, if one child is smaller than the other, it is possible for it to have a higher Gini impurity than its parent, as long as this increase is more than compensated for by a decrease of the other child's impurity. For example, consider a node containing four instances of class A and 1 of class B. Its Gini impurity is $1 - (1/5)^2 - (4/5)^2 = 0.32$. Now suppose the dataset is one-dimensional and the instances are lined up in the following order: A, B, A, A, A. You can verify that the algorithm will split this node after the second instance, producing one child node with instances A, B, and the other child node with instances A, A, A. The first child node's Gini impurity is $1 - (1/2)^2 - (1/2)^2 = 0.5$, which is higher than its parent. This is compensated for by the fact that the other node is pure, so the overall weighted Gini impurity is $2/5 \times 0.5 + 3/5 \times 0 = 0.2$, which is lower than the parent's Gini impurity.

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

Ans: If a Decision Tree is overfitting the training set, it may be a good idea to decrease max depth, since this will constrain the model, regularizing it.

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

Ans: Decision Trees don't care whether or not the training data is scaled or centered; that's one of the nice things about them. So if a Decision Tree underfits the training set, scaling the input features will just be a waste of time.

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 ?

Ans: The computational complexity of training a Decision Tree is O(n × m log(m)). So if you multiply the training set size by 10, the training time will be multiplied by K = (n × 10m × log(10m)) / (n × m × log(m)) = 10 × log(10m) / log(m). If m = 10^6, then K ≈ 11.7, so you can expect the training time to be roughly 11.7 hours.

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

Ans: Presorting the training set speeds up training only if the dataset is smaller than a few thousand instances. If it contains 100,000 instances, setting presort=True will considerably slow down the training.

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

Ans:

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.

8. Follow these steps to grow a forest:

Ans:

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 [ ]:

```
1
```