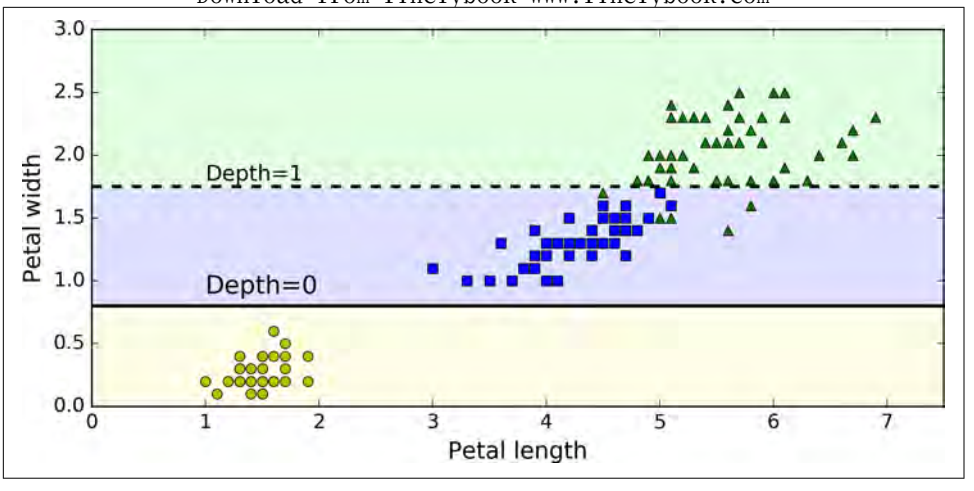

Figure 6-8. Sensitivity to training set details

Random Forests can limit this instability by averaging predictions over many trees, as we will see in the next chapter.

Exercises

1. What is the approximate depth of a Decision Tree trained (without restrictions) on a training set with 1 million instances?
2. Is a node's Gini impurity generally lower or greater than its parent's? Is it *generally* lower/greater, or *always* lower/greater?
3. If a Decision Tree is overfitting the training set, is it a good idea to try decreasing `max_depth`?
4. If a Decision Tree is underfitting the training set, is it a good idea to try scaling the input features?
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?
6. If your training set contains 100,000 instances, will setting `presort=True` speed up training?
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)`.
   b. Split it into a training set and a test set using `train_test_split()`.

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`.

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.

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.

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, achieving only about 80% accuracy.

c. Now comes the magic. For each test set instance, generate the predictions of the 1,000 Decision Trees, and keep only the most frequent prediction (you can use SciPy's `mode()` function for this). This gives you *majority-vote predictions* over the test set.

d. Evaluate these predictions on the test set: you should obtain a slightly higher accuracy than your first model (about 0.5 to 1.5% higher). Congratulations, you have trained a Random Forest classifier!

Solutions to these exercises are available in Appendix A.

CHAPTER 7

Ensemble Learning and Random Forests

Suppose you ask a complex question to thousands of random people, then aggregate their answers. In many cases you will find that this aggregated answer is better than an expert's answer. This is called the *wisdom of the crowd*. Similarly, if you aggregate the predictions of a group of predictors (such as classifiers or regressors), you will often get better predictions than with the best individual predictor. A group of predictors is called an *ensemble*; thus, this technique is called *Ensemble Learning*, and an Ensemble Learning algorithm is called an *Ensemble method*.

For example, you can train a group of Decision Tree classifiers, each on a different random subset of the training set. To make predictions, you just obtain the predictions of all individual trees, then predict the class that gets the most votes (see the last exercise in Chapter 6). Such an ensemble of Decision Trees is called a *Random Forest*, and despite its simplicity, this is one of the most powerful Machine Learning algorithms available today.

Moreover, as we discussed in Chapter 2, you will often use Ensemble methods near the end of a project, once you have already built a few good predictors, to combine them into an even better predictor. In fact, the winning solutions in Machine Learning competitions often involve several Ensemble methods (most famously in the Netflix Prize competition).

In this chapter we will discuss the most popular Ensemble methods, including *bagging*, *boosting*, *stacking*, and a few others. We will also explore Random Forests.

Voting Classifiers

Suppose you have trained a few classifiers, each one achieving about 80% accuracy. You may have a Logistic Regression classifier, an SVM classifier, a Random Forest classifier, a K-Nearest Neighbors classifier, and perhaps a few more (see Figure 7-1).