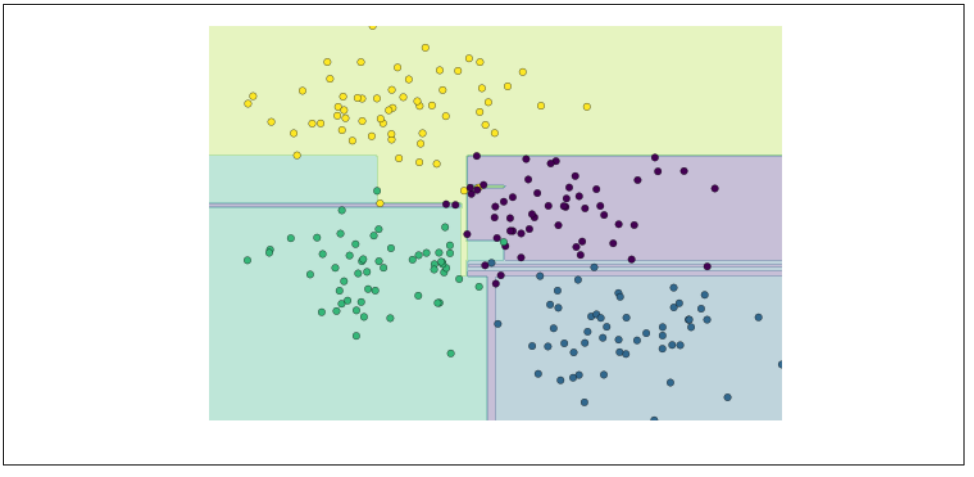

Figure 5-73. First frame of the interactive randomized decision tree widget; for the full version, see the online appendix

Just as using information from two trees improves our results, we might expect that using information from many trees would improve our results even further.

Ensembles of Estimators: Random Forests

This notion—that multiple overfitting estimators can be combined to reduce the effect of this overfitting—is what underlies an ensemble method called *bagging*. Bagging makes use of an ensemble (a grab bag, perhaps) of parallel estimators, each of which overfits the data, and averages the results to find a better classification. An ensemble of randomized decision trees is known as a *random forest*.

We can do this type of bagging classification manually using Scikit-Learn's `Bagging Classifier` meta-estimator as shown here (Figure 5-74):

```
In[8]: from sklearn.tree import DecisionTreeClassifier
       from sklearn.ensemble import BaggingClassifier

       tree = DecisionTreeClassifier()
       bag = BaggingClassifier(tree, n_estimators=100, max_samples=0.8,
                               random_state=1)

       bag.fit(X, y)
       visualize_classifier(bag, X, y)
```

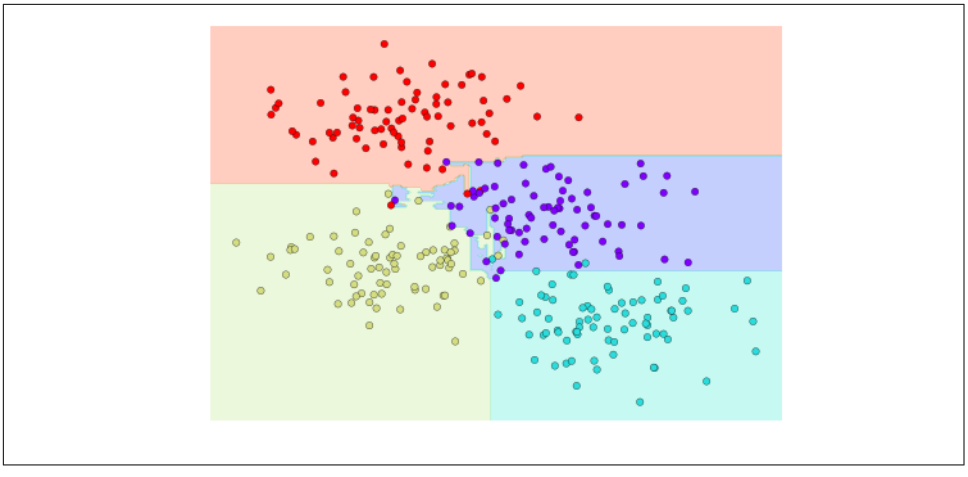

Figure 5-74. Decision boundaries for an ensemble of random decision trees

In this example, we have randomized the data by fitting each estimator with a random subset of 80% of the training points. In practice, decision trees are more effectively randomized when some stochasticity is injected in how the splits are chosen; this way, all the data contributes to the fit each time, but the results of the fit still have the desired randomness. For example, when determining which feature to split on, the randomized tree might select from among the top several features. You can read more technical details about these randomization strategies in the Scikit-Learn documentation and references within.

In Scikit-Learn, such an optimized ensemble of randomized decision trees is implemented in the `RandomForestClassifier` estimator, which takes care of all the randomization automatically. All you need to do is select a number of estimators, and it will very quickly (in parallel, if desired) fit the ensemble of trees (Figure 5-75):

```
In[9]: from sklearn.ensemble import RandomForestClassifier

       model = RandomForestClassifier(n_estimators=100, random_state=0)
       visualize_classifier(model, X, y);
```

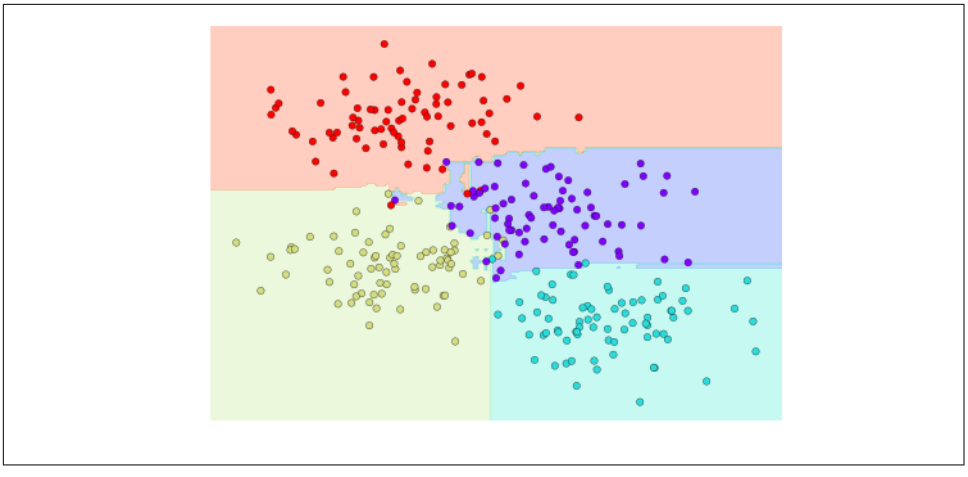

Figure 5-75. Decision boundaries for a random forest, which is an optimized ensemble of decision trees

We see that by averaging over 100 randomly perturbed models, we end up with an overall model that is much closer to our intuition about how the parameter space should be split.

Random Forest Regression

In the previous section we considered random forests within the context of classification. Random forests can also be made to work in the case of regression (that is, continuous rather than categorical variables). The estimator to use for this is the `RandomForestRegressor`, and the syntax is very similar to what we saw earlier.

Consider the following data, drawn from the combination of a fast and slow oscillation (Figure 5-76):

```
In[10]: rng = np.random.RandomState(42)
        x = 10 * rng.rand(200)

        def model(x, sigma=0.3):
            fast_oscillation = np.sin(5 * x)
            slow_oscillation = np.sin(0.5 * x)
            noise = sigma * rng.randn(len(x))

            return slow_oscillation + fast_oscillation + noise

        y = model(x)
        plt.errorbar(x, y, 0.3, fmt='o');
```