```
>>> bag_clf.fit(X_train, y_train)
>>> bag_clf.oob_score_
0.93066666666666664
```

According to this oob evaluation, this `BaggingClassifier` is likely to achieve about 93.1% accuracy on the test set. Let's verify this:

```
>>> from sklearn.metrics import accuracy_score
>>> y_pred = bag_clf.predict(X_test)
>>> accuracy_score(y_test, y_pred)
0.93600000000000005
```

We get 93.6% accuracy on the test set—close enough!

The oob decision function for each training instance is also available through the `oob_decision_function_` variable. In this case (since the base estimator has a `predict_proba()` method) the decision function returns the class probabilities for each training instance. For example, the oob evaluation estimates that the second training instance has a 60.6% probability of belonging to the positive class (and 39.4% of belonging to the positive class):

```
>>> bag_clf.oob_decision_function_
array([[ 0.        ,  1.        ],
       [ 0.60588235,  0.39411765],
       [ 1.        ,  0.        ],
       ...
       [ 1.        ,  0.        ],
       [ 0.        ,  1.        ],
       [ 0.48958333,  0.51041667]])
```

Random Patches and Random Subspaces

The `BaggingClassifier` class supports sampling the features as well. This is controlled by two hyperparameters: `max_features` and `bootstrap_features`. They work the same way as `max_samples` and `bootstrap`, but for feature sampling instead of instance sampling. Thus, each predictor will be trained on a random subset of the input features.

This is particularly useful when you are dealing with high-dimensional inputs (such as images). Sampling both training instances and features is called the *Random Patches* method.[7] Keeping all training instances (i.e., `bootstrap=False` and `max_samples=1.0`) but sampling features (i.e., `bootstrap_features=True` and/or `max_features` smaller than 1.0) is called the *Random Subspaces* method.[8]

7 "Ensembles on Random Patches," G. Louppe and P. Geurts (2012).

8 "The random subspace method for constructing decision forests," Tin Kam Ho (1998).

Sampling features results in even more predictor diversity, trading a bit more bias for a lower variance.

Random Forests

As we have discussed, a Random Forest[9] is an ensemble of Decision Trees, generally trained via the bagging method (or sometimes pasting), typically with `max_samples` set to the size of the training set. Instead of building a `BaggingClassifier` and passing it a `DecisionTreeClassifier`, you can instead use the `RandomForestClassifier` class, which is more convenient and optimized for Decision Trees[10] (similarly, there is a `RandomForestRegressor` class for regression tasks). The following code trains a Random Forest classifier with 500 trees (each limited to maximum 16 nodes), using all available CPU cores:

```
from sklearn.ensemble import RandomForestClassifier

rnd_clf = RandomForestClassifier(n_estimators=500, max_leaf_nodes=16, n_jobs=-1)
rnd_clf.fit(X_train, y_train)

y_pred_rf = rnd_clf.predict(X_test)
```

With a few exceptions, a `RandomForestClassifier` has all the hyperparameters of a `DecisionTreeClassifier` (to control how trees are grown), plus all the hyperparameters of a `BaggingClassifier` to control the ensemble itself.[11]

The Random Forest algorithm introduces extra randomness when growing trees; instead of searching for the very best feature when splitting a node (see Chapter 6), it searches for the best feature among a random subset of features. This results in a greater tree diversity, which (once again) trades a higher bias for a lower variance, generally yielding an overall better model. The following `BaggingClassifier` is roughly equivalent to the previous `RandomForestClassifier`:

```
bag_clf = BaggingClassifier(
        DecisionTreeClassifier(splitter="random", max_leaf_nodes=16),
        n_estimators=500, max_samples=1.0, bootstrap=True, n_jobs=-1
    )
```

9 "Random Decision Forests," T. Ho (1995).

10 The `BaggingClassifier` class remains useful if you want a bag of something other than Decision Trees.

11 There are a few notable exceptions: `splitter` is absent (forced to `"random"`), `presort` is absent (forced to `False`), `max_samples` is absent (forced to `1.0`), and `base_estimator` is absent (forced to `DecisionTreeClassifier` with the provided hyperparameters).