Download from finelybook www.finelybook.com
default hyperparameter values (which was 52,634). Congratulations, you have successfully fine-tuned your best model!

Don't forget that you can treat some of the data preparation steps as hyperparameters. For example, the grid search will automatically find out whether or not to add a feature you were not sure about (e.g., using the `add_bedrooms_per_room` hyperparameter of your `CombinedAttributesAdder` transformer). It may similarly be used to automatically find the best way to handle outliers, missing features, feature selection, and more.

Randomized Search

The grid search approach is fine when you are exploring relatively few combinations, like in the previous example, but when the hyperparameter *search space* is large, it is often preferable to use `RandomizedSearchCV` instead. This class can be used in much the same way as the `GridSearchCV` class, but instead of trying out all possible combinations, it evaluates a given number of random combinations by selecting a random value for each hyperparameter at every iteration. This approach has two main benefits:

- If you let the randomized search run for, say, 1,000 iterations, this approach will explore 1,000 different values for each hyperparameter (instead of just a few values per hyperparameter with the grid search approach).
- You have more control over the computing budget you want to allocate to hyperparameter search, simply by setting the number of iterations.

Ensemble Methods

Another way to fine-tune your system is to try to combine the models that perform best. The group (or "ensemble") will often perform better than the best individual model (just like Random Forests perform better than the individual Decision Trees they rely on), especially if the individual models make very different types of errors. We will cover this topic in more detail in Chapter 7.

Analyze the Best Models and Their Errors

You will often gain good insights on the problem by inspecting the best models. For example, the `RandomForestRegressor` can indicate the relative importance of each attribute for making accurate predictions:

```
>>> feature_importances = grid_search.best_estimator_.feature_importances_
>>> feature_importances
array([  7.14156423e-02,   6.76139189e-02,   4.44260894e-02,
```

Download from finelybook www.finelybook.com

```
1.66308583e-02,   1.66076861e-02,   1.82402545e-02,
1.63458761e-02,   3.26497987e-01,   6.04365775e-02,
1.13055290e-01,   7.79324766e-02,   1.12166442e-02,
1.53344918e-01,   8.41308969e-05,   2.68483884e-03,
3.46681181e-03])
```

Let's display these importance scores next to their corresponding attribute names:

```
>>> extra_attribs = ["rooms_per_hhold", "pop_per_hhold", "bedrooms_per_room"]
>>> cat_one_hot_attribs = list(encoder.classes_)
>>> attributes = num_attribs + extra_attribs + cat_one_hot_attribs
>>> sorted(zip(feature_importances, attributes), reverse=True)
[(0.32649798665134971, 'median_income'),
 (0.15334491760305854, 'INLAND'),
 (0.11305529021187399, 'pop_per_hhold'),
 (0.07793247662544775, 'bedrooms_per_room'),
 (0.071415642259275158, 'longitude'),
 (0.067613918945568688, 'latitude'),
 (0.060436577499703222, 'rooms_per_hhold'),
 (0.04442608939578685, 'housing_median_age'),
 (0.018240254462909437, 'population'),
 (0.01663085833886218, 'total_rooms'),
 (0.016607686091288865, 'total_bedrooms'),
 (0.016345876147580776, 'households'),
 (0.011216644219017424, '<1H OCEAN'),
 (0.0034668118081117387, 'NEAR OCEAN'),
 (0.0026848388432755429, 'NEAR BAY'),
 (8.4130896890070617e-05, 'ISLAND')]
```

With this information, you may want to try dropping some of the less useful features (e.g., apparently only one `ocean_proximity` category is really useful, so you could try dropping the others).

You should also look at the specific errors that your system makes, then try to understand why it makes them and what could fix the problem (adding extra features or, on the contrary, getting rid of uninformative ones, cleaning up outliers, etc.).

Evaluate Your System on the Test Set

After tweaking your models for a while, you eventually have a system that performs sufficiently well. Now is the time to evaluate the final model on the test set. There is nothing special about this process; just get the predictors and the labels from your test set, run your `full_pipeline` to transform the data (call `transform()`, *not* `fit_transform()`!), and evaluate the final model on the test set:

```
final_model = grid_search.best_estimator_

X_test = strat_test_set.drop("median_house_value", axis=1)
y_test = strat_test_set["median_house_value"].copy()

X_test_prepared = full_pipeline.transform(X_test)
```