Grid Search

One way to do that would be to fiddle with the hyperparameters manually, until you find a great combination of hyperparameter values. This would be very tedious work, and you may not have time to explore many combinations.

Instead you should get Scikit-Learn's `GridSearchCV` to search for you. All you need to do is tell it which hyperparameters you want it to experiment with, and what values to try out, and it will evaluate all the possible combinations of hyperparameter values, using cross-validation. For example, the following code searches for the best combination of hyperparameter values for the `RandomForestRegressor`:

```
from sklearn.model_selection import GridSearchCV

param_grid = [
    {'n_estimators': [3, 10, 30], 'max_features': [2, 4, 6, 8]},
    {'bootstrap': [False], 'n_estimators': [3, 10], 'max_features': [2, 3, 4]},
  ]

forest_reg = RandomForestRegressor()

grid_search = GridSearchCV(forest_reg, param_grid, cv=5,
                           scoring='neg_mean_squared_error')

grid_search.fit(housing_prepared, housing_labels)
```

When you have no idea what value a hyperparameter should have, a simple approach is to try out consecutive powers of 10 (or a smaller number if you want a more fine-grained search, as shown in this example with the `n_estimators` hyperparameter).

This `param_grid` tells Scikit-Learn to first evaluate all 3 × 4 = 12 combinations of `n_estimators` and `max_features` hyperparameter values specified in the first `dict` (don't worry about what these hyperparameters mean for now; they will be explained in Chapter 7), then try all 2 × 3 = 6 combinations of hyperparameter values in the second `dict`, but this time with the `bootstrap` hyperparameter set to `False` instead of `True` (which is the default value for this hyperparameter).

All in all, the grid search will explore 12 + 6 = 18 combinations of `RandomForestRegressor` hyperparameter values, and it will train each model five times (since we are using five-fold cross validation). In other words, all in all, there will be 18 × 5 = 90 rounds of training! It may take quite a long time, but when it is done you can get the best combination of parameters like this:

```
>>> grid_search.best_params_
{'max_features': 6, 'n_estimators': 30}
```

Since 30 is the maximum value of `n_estimators` that was evaluated, you should probably evaluate higher values as well, since the score may continue to improve.

You can also get the best estimator directly:

```
>>> grid_search.best_estimator_
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features=6, max_leaf_nodes=None, min_samples_leaf=1,
           min_samples_split=2, min_weight_fraction_leaf=0.0,
           n_estimators=30, n_jobs=1, oob_score=False, random_state=None,
           verbose=0, warm_start=False)
```

If `GridSearchCV` is initialized with `refit=True` (which is the default), then once it finds the best estimator using cross-validation, it retrains it on the whole training set. This is usually a good idea since feeding it more data will likely improve its performance.

And of course the evaluation scores are also available:

```
>>> cvres = grid_search.cv_results_
... for mean_score, params in zip(cvres["mean_test_score"], cvres["params"]):
...     print(np.sqrt(-mean_score), params)
...
64912.0351358 {'max_features': 2, 'n_estimators': 3}
55535.2786524 {'max_features': 2, 'n_estimators': 10}
52940.2696165 {'max_features': 2, 'n_estimators': 30}
60384.0908354 {'max_features': 4, 'n_estimators': 3}
52709.9199934 {'max_features': 4, 'n_estimators': 10}
50503.5985321 {'max_features': 4, 'n_estimators': 30}
59058.1153485 {'max_features': 6, 'n_estimators': 3}
52172.0292957 {'max_features': 6, 'n_estimators': 10}
49958.9555932 {'max_features': 6, 'n_estimators': 30}
59122.260006 {'max_features': 8, 'n_estimators': 3}
52441.5896087 {'max_features': 8, 'n_estimators': 10}
50041.4899416 {'max_features': 8, 'n_estimators': 30}
62371.1221202 {'bootstrap': False, 'max_features': 2, 'n_estimators': 3}
54572.2557534 {'bootstrap': False, 'max_features': 2, 'n_estimators': 10}
59634.0533132 {'bootstrap': False, 'max_features': 3, 'n_estimators': 3}
52456.0883904 {'bootstrap': False, 'max_features': 3, 'n_estimators': 10}
58825.665239 {'bootstrap': False, 'max_features': 4, 'n_estimators': 3}
52012.9945396 {'bootstrap': False, 'max_features': 4, 'n_estimators': 10}
```

In this example, we obtain the best solution by setting the `max_features` hyperparameter to `6`, and the `n_estimators` hyperparameter to `30`. The RMSE score for this combination is 49,959, which is slightly better than the score you got earlier using the 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,
```