It is possible to use Gradient Boosting with other cost functions. This is controlled by the `loss` hyperparameter (see Scikit-Learn's documentation for more details).

Stacking

The last Ensemble method we will discuss in this chapter is called *stacking* (short for *stacked generalization*).[18] It is based on a simple idea: instead of using trivial functions (such as hard voting) to aggregate the predictions of all predictors in an ensemble, why don't we train a model to perform this aggregation? Figure 7-12 shows such an ensemble performing a regression task on a new instance. Each of the bottom three predictors predicts a different value (3.1, 2.7, and 2.9), and then the final predictor (called a *blender*, or a *meta learner*) takes these predictions as inputs and makes the final prediction (3.0).

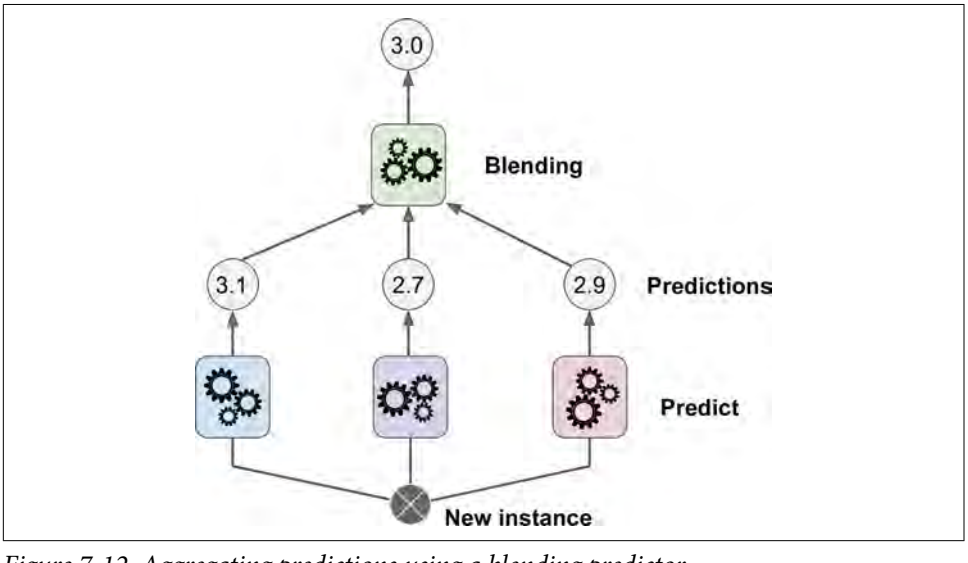

Figure 7-12. Aggregating predictions using a blending predictor

To train the blender, a common approach is to use a hold-out set.[19] Let's see how it works. First, the training set is split in two subsets. The first subset is used to train the predictors in the first layer (see Figure 7-13).

18 "Stacked Generalization," D. Wolpert (1992).

19 Alternatively, it is possible to use out-of-fold predictions. In some contexts this is called *stacking*, while using a hold-out set is called *blending*. However, for many people these terms are synonymous.

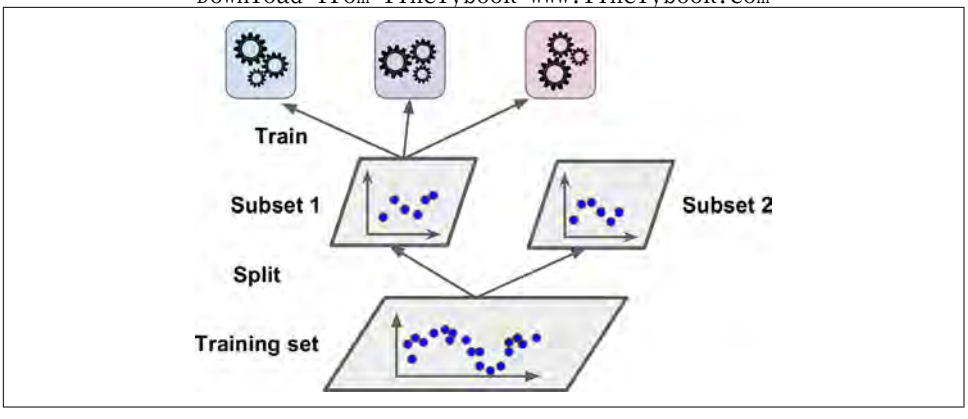

Figure 7-13. Training the first layer

Next, the first layer predictors are used to make predictions on the second (held-out) set (see Figure 7-14). This ensures that the predictions are "clean," since the predictors never saw these instances during training. Now for each instance in the hold-out set there are three predicted values. We can create a new training set using these predicted values as input features (which makes this new training set three-dimensional), and keeping the target values. The blender is trained on this new training set, so it learns to predict the target value given the first layer's predictions.

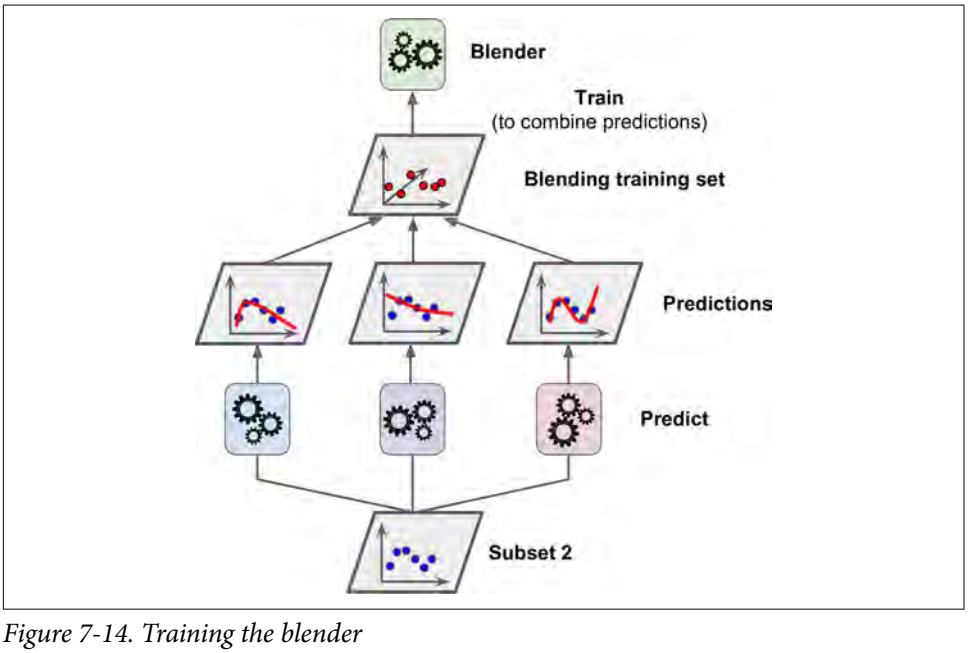

Figure 7-14. Training the blender

It is actually possible to train several different blenders this way (e.g., one using Linear Regression, another using Random Forest Regression, and so on): we get a whole layer of blenders. The trick is to split the training set into three subsets: the first one is used to train the first layer, the second one is used to create the training set used to train the second layer (using predictions made by the predictors of the first layer), and the third one is used to create the training set to train the third layer (using predictions made by the predictors of the second layer). Once this is done, we can make a prediction for a new instance by going through each layer sequentially, as shown in Figure 7-15.

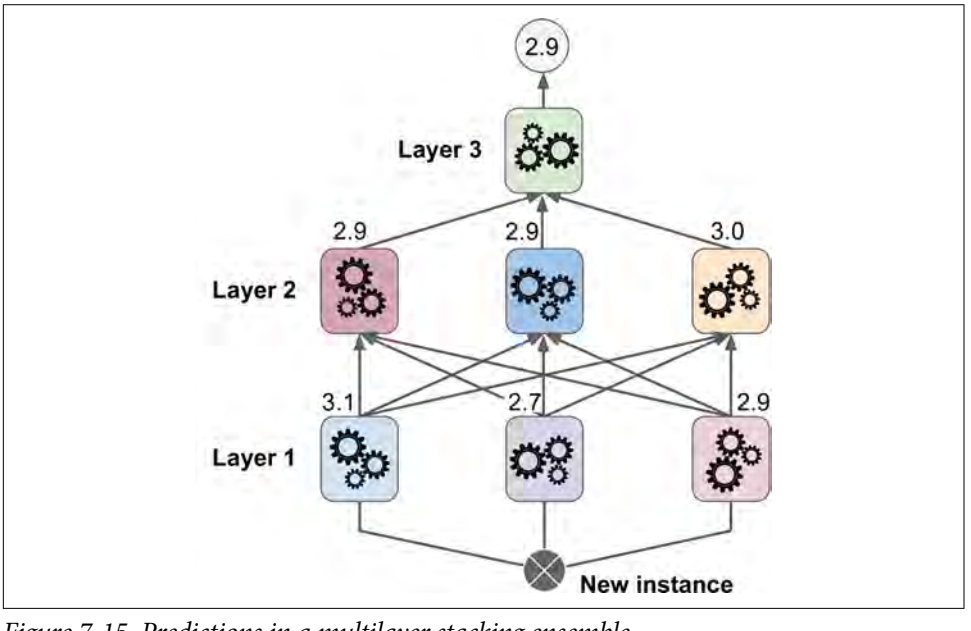

Figure 7-15. Predictions in a multilayer stacking ensemble

Unfortunately, Scikit-Learn does not support stacking directly, but it is not too hard to roll out your own implementation (see the following exercises). Alternatively, you can use an open source implementation such as `brew` (available at *https://github.com/viisar/brew*).

Exercises

1. If you have trained five different models on the exact same training data, and they all achieve 95% precision, is there any chance that you can combine these models to get better results? If so, how? If not, why?
2. What is the difference between hard and soft voting classifiers?