Chapter 7. Ensemble Learning and Random Forests

A group of predictors is called an *ensemble*; thus, this technique is called *Ensemble Learning*, and an Ensemble Learning algorithm is called an *Ensemble method*.

Random Forest: train a group of Decision Tree classifiers, each on a different random subset of the training set. To make predictions, you just obtain the predictions of all individual trees, then predict the class that gets the most votes.

You will often use Ensemble methods near the end of a project, once you have already built a few good predictors, to combine them into an even better predictor.

Voting Classifiers

Hard voting classifier: aggregate the predictions of each classifier and predict the class that gets the most votes.

Somewhat surprisingly, this voting classifier often achieves a higher accuracy than the best classifier in the ensemble. In fact, even if each classifier is a *weak learner* (meaning it does only slightly better than random guessing), the ensemble can still be a *strong learner* (achieving high accuracy), provided there are a sufficient number of weak learners and they are sufficiently diverse.

TIP

> Ensemble methods work best when the predictors are as independent from one another as possible. One way to get diverse classifiers is to train them using very different algorithms. This increases the chance that they will make very different types of errors, improving the ensemble's accuracy.

Soft voting: If all classifiers are able to estimate class probabilities, then predict the class with the highest class probability, averaged over all the individual classifiers. (It often achieves higher performance than hard voting because it gives more weight to highly confident votes.)

Bagging and Pasting

Another approach is to use the same training algorithm for every predictor, but to train them on different random subsets of the training set. When sampling is performed with replacement, this method is called *bagging* (short for bootstrap aggregating). When sampling is performed without replacement, it is called *pasting*.

Once all predictors are trained, the ensemble can make a prediction for a new instance by simply aggregating the predictions of all predictors. The aggregation function is typically the *statistical mode* (i.e., the most frequent prediction, just like a hard voting classifier) for classification, or the average for regression. Each individual predictor has a higher bias than if it were trained on the original training set, but aggregation reduces both bias and variance. Generally, the net result is that the ensemble has a similar bias but a lower variance than a single predictor trained on the original training set.

Predictors can all be trained in parallel, via different CPU cores or even different servers. Similarly, predictions can be made in parallel. This is one of the reasons why bagging and pasting are such popular methods: they scale very well.

Bootstrapping introduces a bit more diversity in the subsets that each predictor is trained on, so bagging ends up with a slightly higher bias than pasting, but this also means that predictors end up being less correlated so the ensemble's variance is reduced. Overall, bagging often results in better models, which explains why it is generally preferred.

Out-of-Bag Evaluation

About 37% of the training instances on average for each predictor that are not sampled are called *out-of-bag* (oob) instances.

Since a predictor never sees the oob instances during training, it can be evaluated on these instances, without the need for a separate validation set or cross-validation. You can evaluate the ensemble itself by averaging out the oob evaluations of each predictor.

Random Patches and Random subspaces

Sampling features is particularly useful when dealing with high-dimensional inputs (e.g., images).

Random Patches method: sampling both training instances and features.

Random subspace method: keeping all training instances but sampling features.

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

Random Forests

Random Forest 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.

The Random Forest algorithm introduces extra randomness when growing trees; instead of searching for the very best feature when splitting a node, it searches for the best feature among a random subset of features. This results in a greater tree diversity, which trades a higher bias for

a lower variance, generally yielding an overall better model.

Extra-Trees

Extremely Randomized Trees ensemble (Extra-Trees): use random thresholds for each feature rather than searching for the best possible thresholds.

More bias, lower variance.

Much faster to train than regular Random Forests since finding the best possible threshold for each feature at every node is one of the most time-consuming tasks of growing a tree.

Feature Importance

Estimate a feature's importance by computing the average depth at which it appears across all trees in the forest.

Random Forests are very handy for feature selection.

Boosting

Boosting (*hypothesis boosting*): any Ensemble method that can combine several weak learners into a strong learner.

General idea: train predictors sequentially, each trying to correct its predecessor.

Most popular boosting methods:

- *AdaBoost* (*Adaptive Boosting*)
- _Gradient Boosting

AdaBoost

One way for a new predictor to correct its predecessor is to pay a bit more attention to the training instances that the predecessor underfitted. This results in new predictors focusing more and more on the hard cases.

Sequential learning technique has some similarities with Gradient Descent, except that instead of tweaking a single predictor's parameters to minimize a cost function, AdaBoost adds predictors to the ensemble, gradually making it better.

Once all predictors are trained, the ensemble makes predictions very much like bagging or pasting, except that predictors have different weights depending on their overall accuracy on the weighted training set.

WARNING

> There is one important drawback to this sequential learning technique: it cannot be parallelized (or only partially), since each predictor can only be trained after the previous predictor has been trained and evaluated. As a result, it does not scale as well as bagging or pasting.

Weighted erroer rate of the j^{th} predictor

$$r_j = \sum_{\substack{i=1 \\ \hat{y}_j^{(i)} \neq y^{(i)}}}^{m} w^{(i)} \; / \; \sum_{i=1}^{m} w^{(i)}$$

Predictor weight

$$\alpha_j = \eta \; log \frac{1 - r_j}{r_j}$$

Weight update rule

for $i = 1, 2, \cdots, m$

$$w^{(i)} \leftarrow \begin{cases} w^{(i)} & if \;\; \hat{y}_j^{(i)} = y^{(i)} \\ w^{(i)} exp(\alpha_j) & if \;\; \hat{y}_j^{(i)} \neq y^{(i)} \end{cases}$$

To make predictions, AdaBoost simply computes the predictions of all the predictors and weighs them using the predictor weights α_j. The predicted class is the one that receives the majority of weighted votes.

AdaBoost predictions

$$\hat{y}(\mathbf{x}) = \underset{k}{argmax} \sum_{\substack{j=1 \\ \hat{y}_j(\mathbf{x})=k}}^{N} \alpha_j$$

where N is the number of predictors.

TIP

> If your AdaBoost ensemble is overfitting the training set, you can try reducing the number of estimators or more strongly regularizing the base estimator.

Gradient Boosting

Tries to fit the new predictor to the residual errors made by the previous predictor.

Grident Tree Boosting, or *Gradient Boosted Regression Trees* (*GBRT*): regression using Decision Trees as the base predictors.

The learning_rate hyperparameter scales the contribution of each tree. If you set it to a low value, such as 0.1 , you will need more trees in the ensemble to fit the training set, but the predictions will usually generalize better. This is a regularization technique called shrinkage.

Shrinkage: $learning_rate$ hyperparameter scales the contribution of each tree. If you set it to a low value, such as 0.1 , you will need more trees in the ensemble to fit the training set, but the predictions will usually generalize better. Use early stopping to find the optimal number of trees.

Stochastic Gradient Boosting: train each tree with fraction of training instances.

Stacking

Stacking, *Stacked generalization*: train a model to perform aggregation of all predictors in an ensemble.

Final predictor is called _blender, or *meta learner*.

To train the blender, a common approach is to use a hold-out set.

- 1. Split training set into two subset.
- 2. The first subset is used to train the predictors in the first layer.
- 3. Next, the first layer predictors are used to make predictions on the second (held-out) set.
- 4. 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.

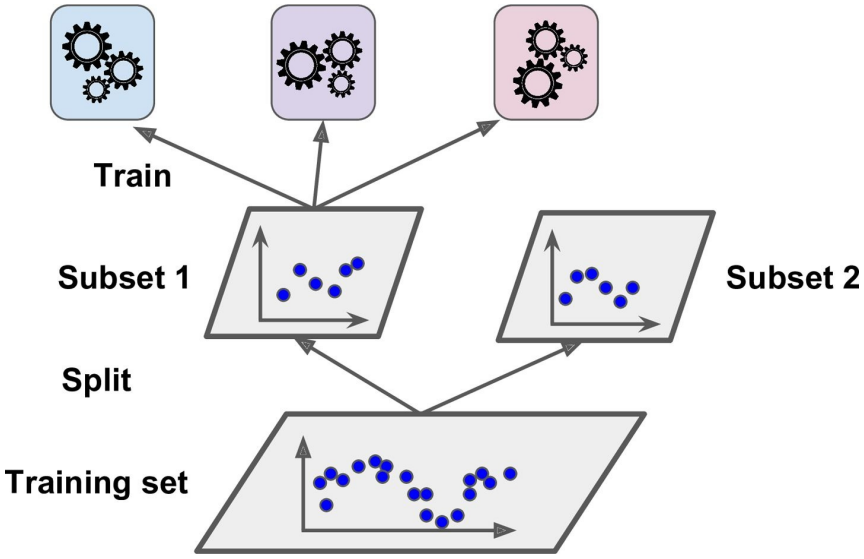

Figure 7-13. Training the first layer

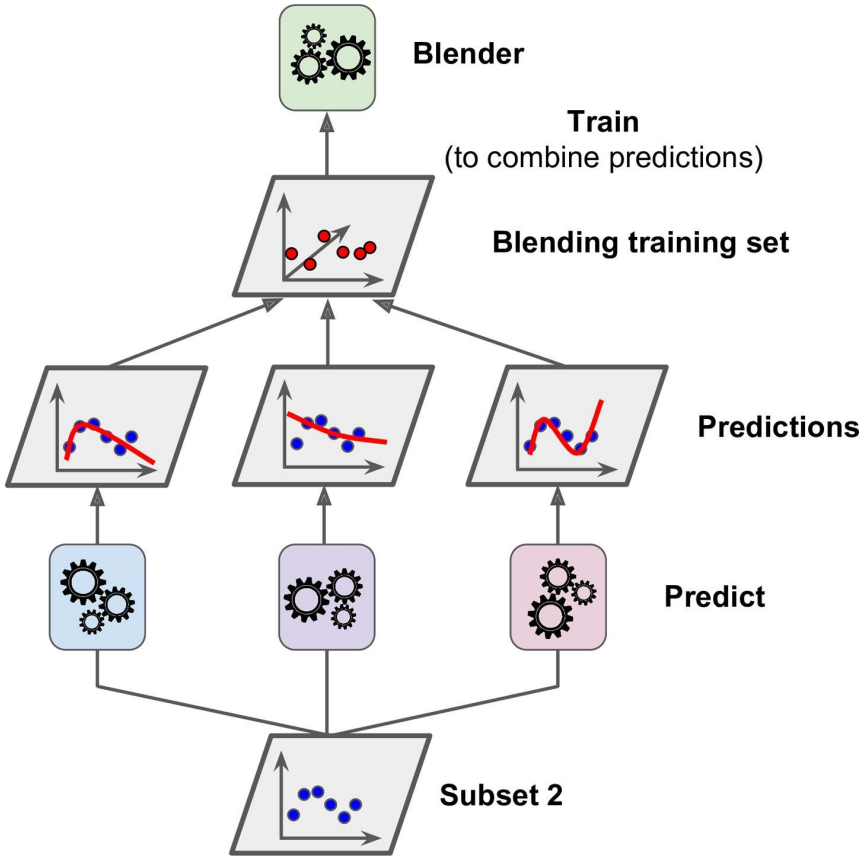

Figure 7-14. Training the blender

Train several different blenders: split the training set into three subsets.

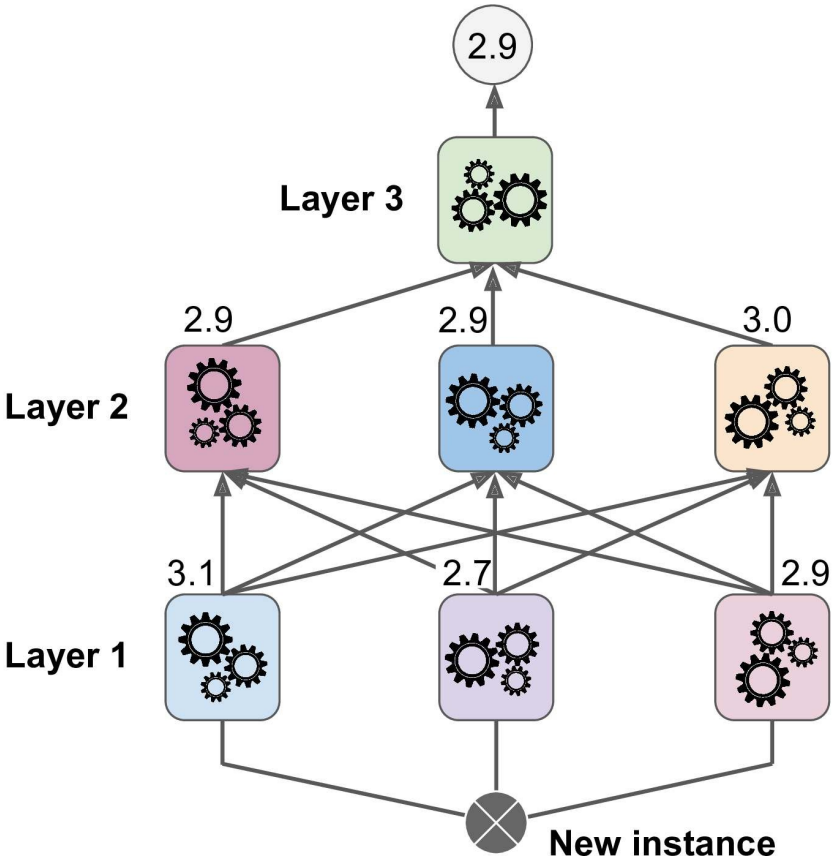

Figure 7-15. Predictions in a multilayer stacking ensemble