Module: II: Ensemble Learning

[8 Sessions] [Apply]

Ensemble Learning – using subset of instances – Bagging, Pasting, using subset of features –random patches and random subspaces method; Voting Classifier, Random Forest; Boosting – AdaBoost, Gradient Boosting, Extremely Randomized Trees, Stacking.

Why Ensemble Learning ?

Suppose you ask a complex question to thousands of random people, then aggregate their answers. In many cases you will find that this aggregated answer is better than an expert's answer

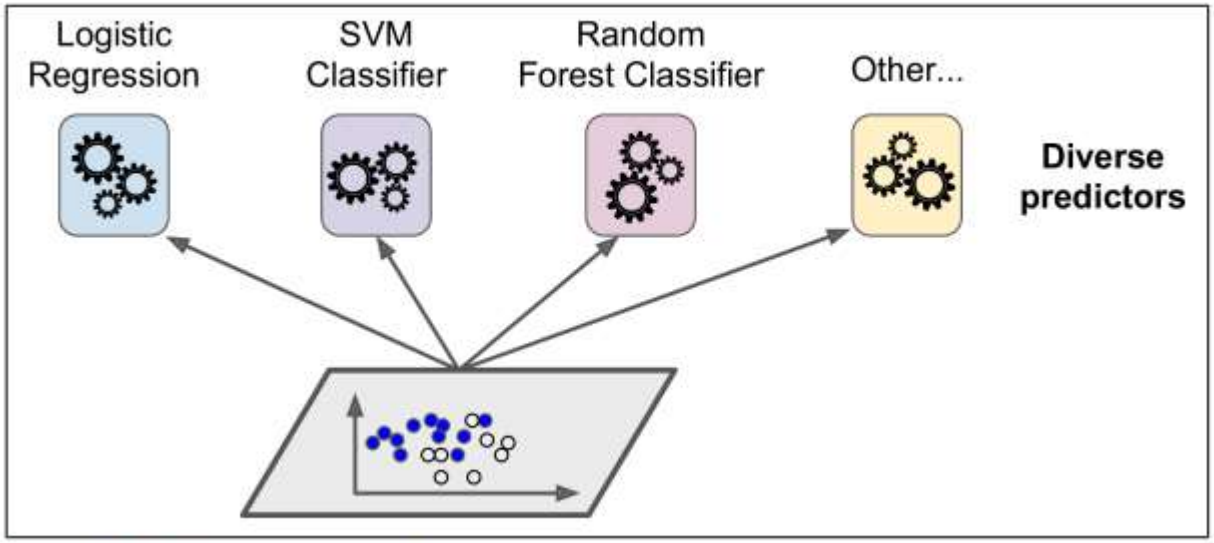
Similarly, if you aggregate the predictions of a group of predictors (such as classifiers or regressors), you will often get better predictions than with the best individual predictor.

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.

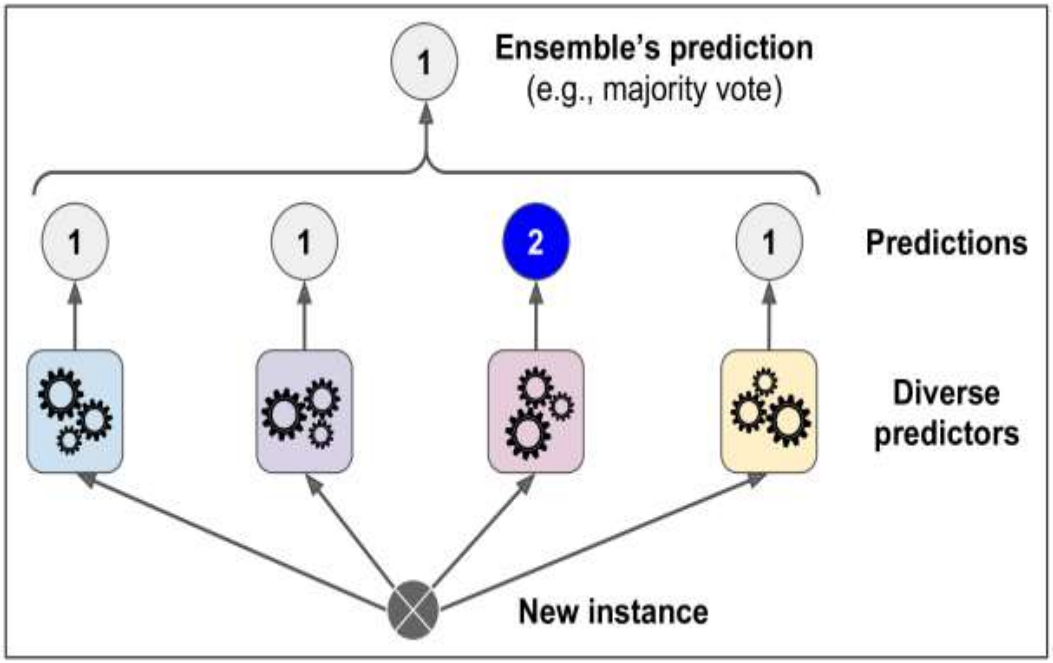
Voting Classifiers

Suppose you have trained a few classifiers, each one achieving about 80% accuracy.
You may have :

- Logistic Regression classifier
- SVM classifier
- Random Forest classifier
- K-Nearest Neighbors classifier



A very simple way to create an even better classifier is to aggregate the predictions of each classifier and predict the class that gets the most votes. This majority-vote classifier is called a hard voting classifier



Hard Voting Classifier Predictions

Surprisingly, this voting classifier often achieves a higher accuracy than the best classifier in the ensemble.

The following code creates and trains a voting classifier in Scikit-Learn, composed of three diverse classifiers:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC

log_clf = LogisticRegression()
rnd_clf = RandomForestClassifier()
svm_clf = SVC()

voting_clf = VotingClassifier(
    estimators=[('lr', log_clf), ('rf', rnd_clf), ('svc', svm_clf)],
    voting='hard')
voting_clf.fit(X_train, y_train)
```

Let's look at each classifier's accuracy on the test set:

```
>>> from sklearn.metrics import accuracy_score
>>> for clf in (log_clf, rnd_clf, svm_clf, voting_clf):
...     clf.fit(X_train, y_train)
...     y_pred = clf.predict(X_test)
...     print(clf.__class__.__name__, accuracy_score(y_test, y_pred))
...
LogisticRegression 0.864
RandomForestClassifier 0.896
SVC 0.888
VotingClassifier 0.904
```

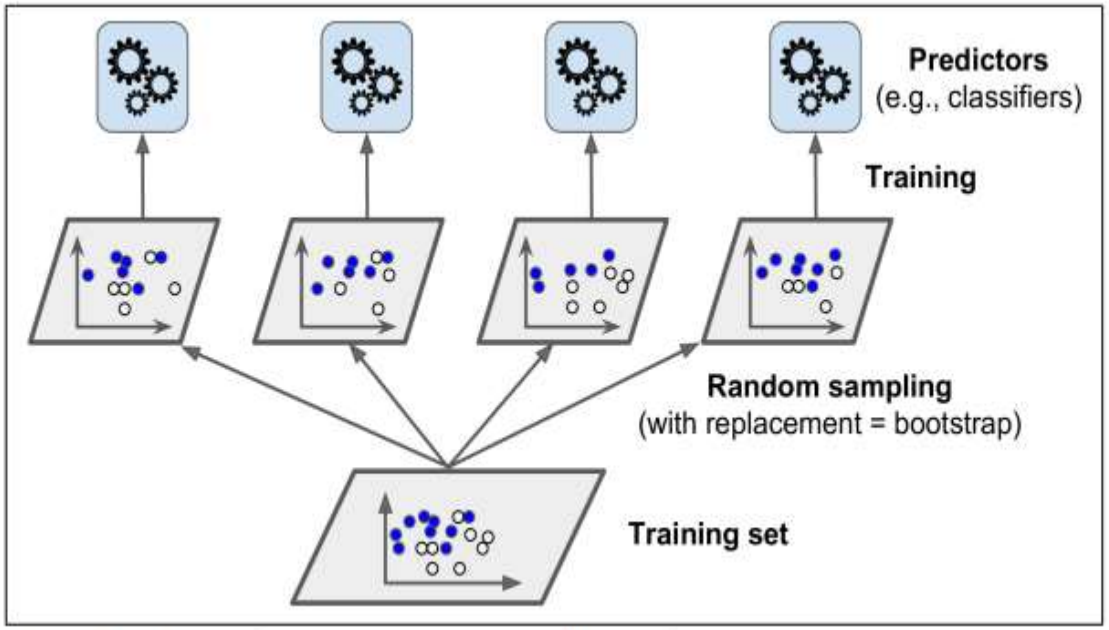
The Voting Classifier outperforms the individual classifier with Accuracy of 90.4%

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.***

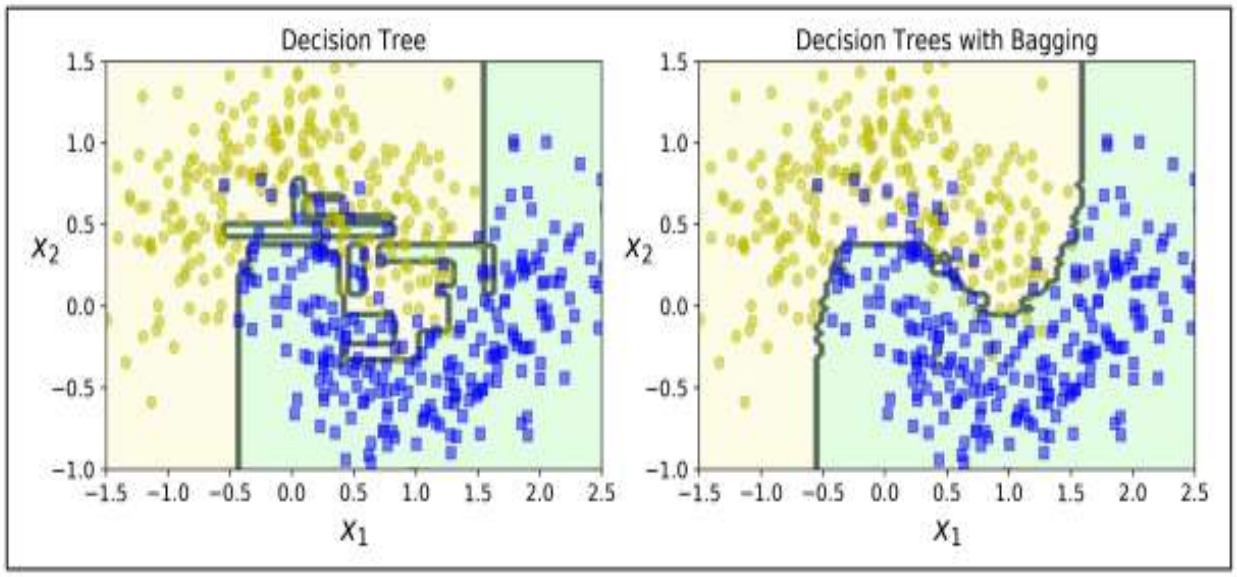


Pasting/Bagging Training Set sampling and Training

The following code trains an ensemble of ***500 Decision Tree classifiers***,5 each trained on ***100 training instances*** randomly sampled from the training set with replacement (***this is an example of bagging, but if you want to use pasting instead, just set bootstrap=False***)

```
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier

bag_clf = BaggingClassifier(
    DecisionTreeClassifier(), n_estimators=500,
    max_samples=100, bootstrap=True, n_jobs=-1)
bag_clf.fit(X_train, y_train)
y_pred = bag_clf.predict(X_test)
```



A single Decision Tree versus a bagging ensemble of 500 trees

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

Sampling both training instances and features is called the *Random Patches method*.
Keeping all training instances but sampling features (i.e., bootstrap_features=True and/or max_fea tures smaller than 1.0) is called the *Random Subspaces method.*

This is particularly useful when you are dealing with high-dimensional inputs (such as images).

Random Forests

Random Forest is an ensemble of Decision Trees, generally trained via the bagging method (or sometimes pasting).

Instead of building a Bagging Classifier and passing it a Decision Tree Classifier, you can instead use the Random Forest Classifier class, which is more convenient and optimized for Decision Trees.

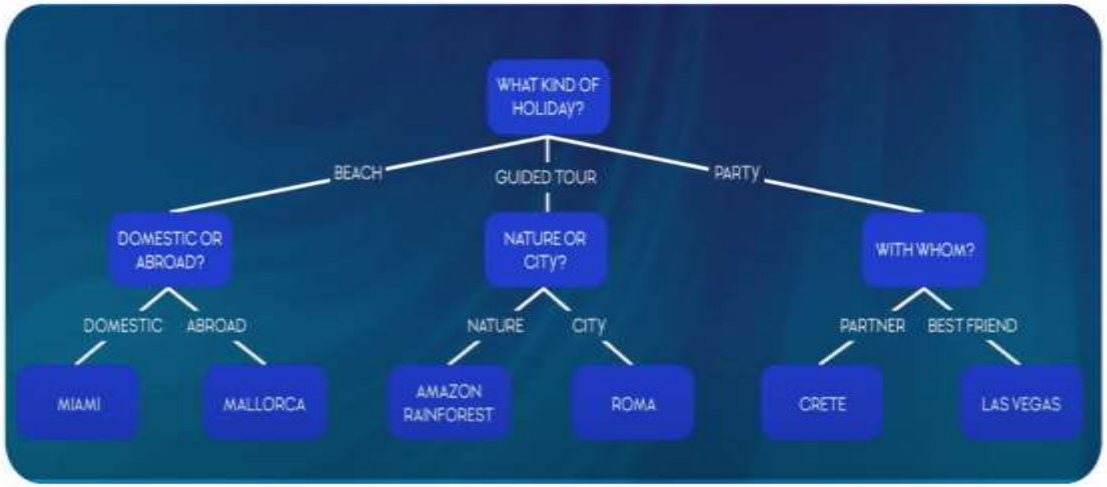
Random forest tends to combine hundreds of decision trees and then trains each decision tree on a different sample of the observations.

How is the decision tree concept used in ML?

Decision tree algorithm **is a series of if-else statements** that can be used to **predict a result based on data**.

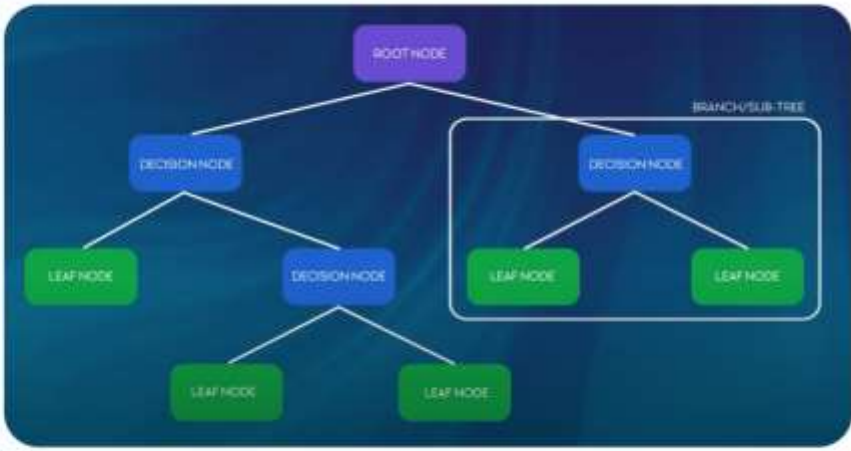
Simply put, it...

- ...randomly asks questions,
- ...checks if the dataset is divided correctly,
- ...changes the questions accordingly.



Decision Tree is a series of Nodes

- **Root or top node -** the starting point of the tree
- Then we have **a set of decision nodes** until we reach **the leaf node**
- **Leaf node -** the final category or prediction/result



Pruning-To reduce the complexity of Decision Tree Algorithm

What is pruning?

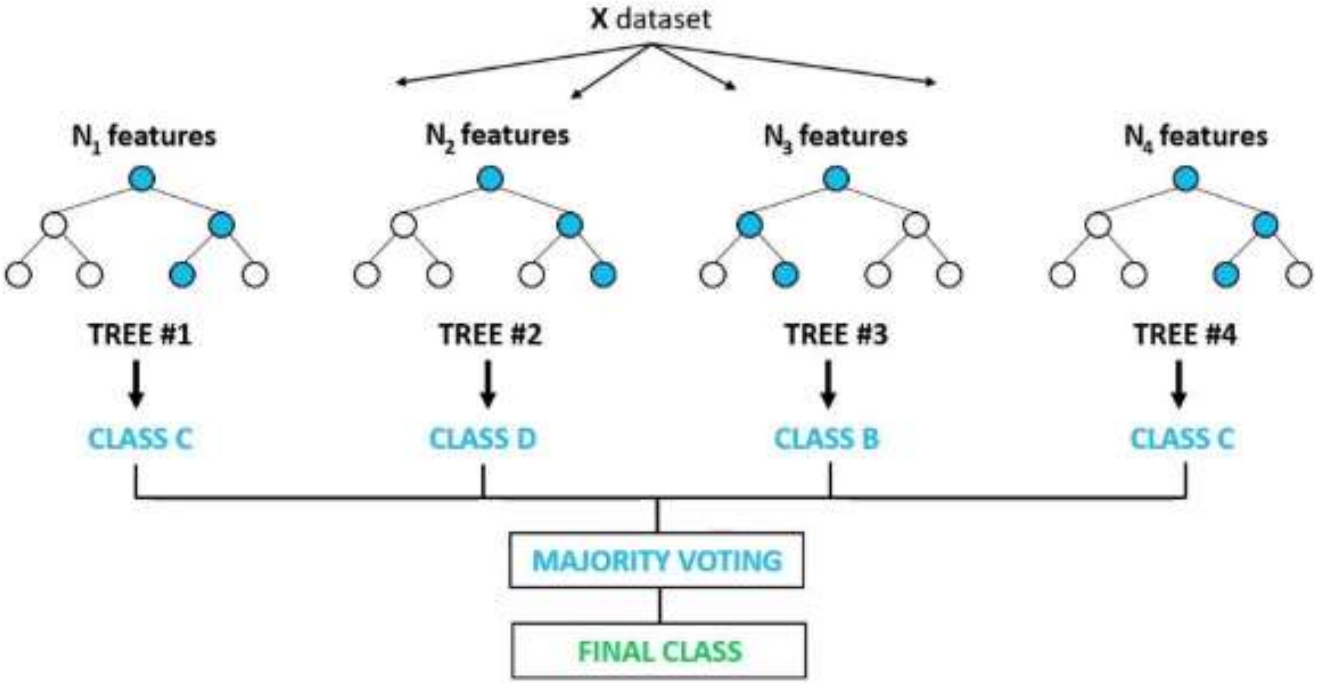
Pruning is removing some unnecessary features based on the information gain in order to get less complex decision tree. Pruning helps to **prevent overfitting** and **high variance** so that the model **generalizes well to unseen data**.

Random Forest-An Improved version of Decision Tree

How does random forest algorithm work?

The random forest algorithm uses many trees instead of using only one. It runs a test dataset on each tree, gets predictions from each and based on the majority votes, that means the category that appears the most, we get a final output.

Random Forest Classifier



Extremely Randomized Trees ensemble

When you are growing a tree in a Random Forest, at each node only a random subset of the features is considered for splitting.
Trees can be made even more random by also using random thresholds for each feature rather than searching for the best possible thresholds.
A forest of such extremely random trees is simply called an ***Extremely Randomized Trees ensemble***

Boosting

Boosting (originally called hypothesis boosting) refers to any Ensemble method that can combine several weak learners into a strong learner.

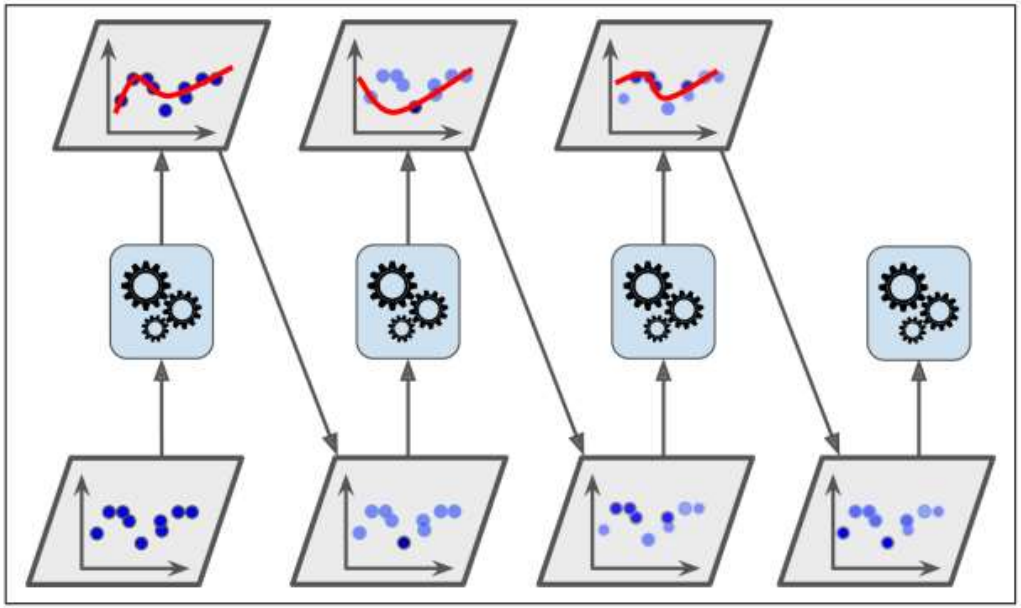
The general idea of most boosting methods is to train predictors sequentially, each trying to correct its predecessor.

Most popular Boosting methods are:

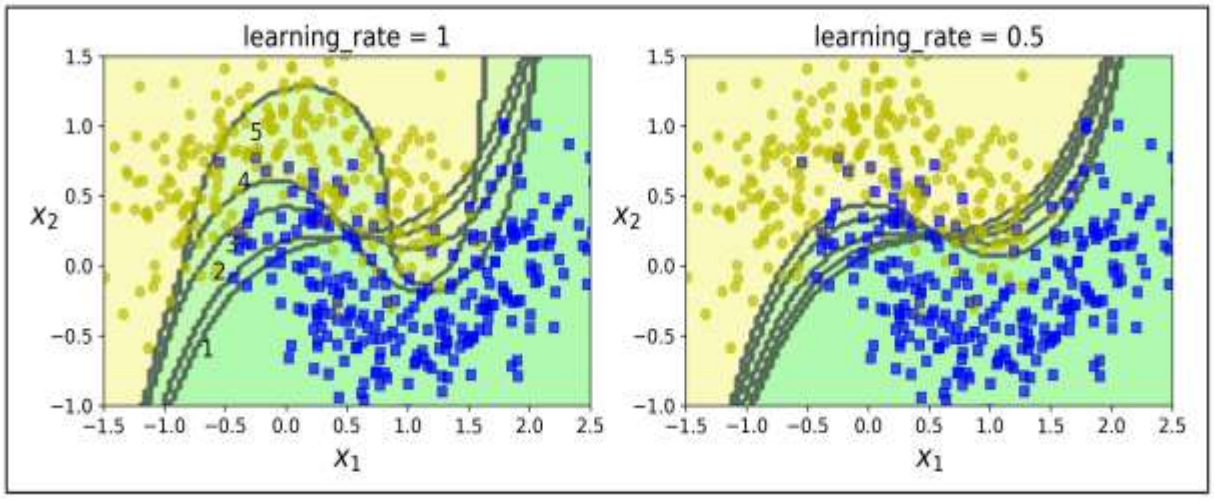
- AdaBoost(Adaptive Boosting)
- Gradient Boosting

AdaBoost

A first base classifier (such as a Decision Tree) is trained and used to make predictions on the training set. The relative weight of misclassified training instances is then increased. A second classifier is trained using the updated weights and again it makes predictions on the training set, weights are updated, and so on



AdaBoost sequential training with instance weight updates



The first classifier gets many instances wrong, so their weights get boosted.

The second classifier therefore does a better job on these instances, and so on.

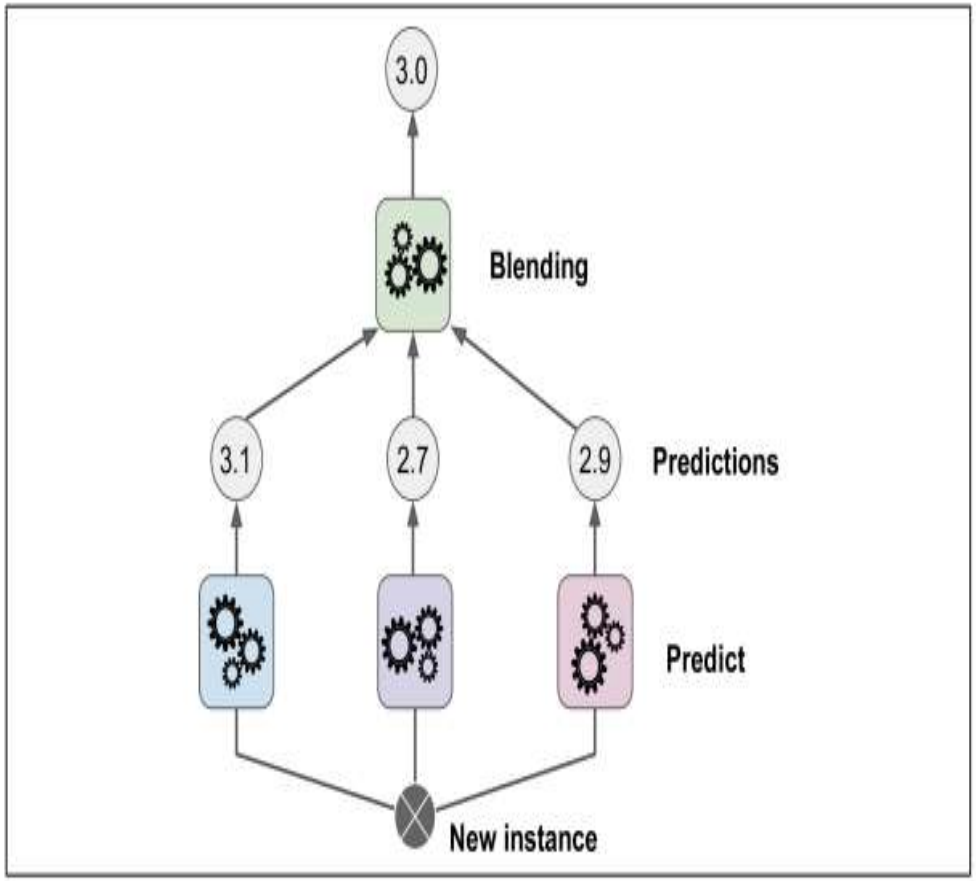
The plot on the right represents the same sequence of predictors except that the learning rate is halved (i.e., the misclassified instance weights are boosted half as much at every iteration).

Gradient Boosting

- Gradient Boosting works by sequentially adding predictors to an ensemble, each one correcting its predecessor. However, instead of tweaking the instance weights at every iteration like AdaBoost does, this method tries to fit the new predictor to the residual errors made by the previous predictor.

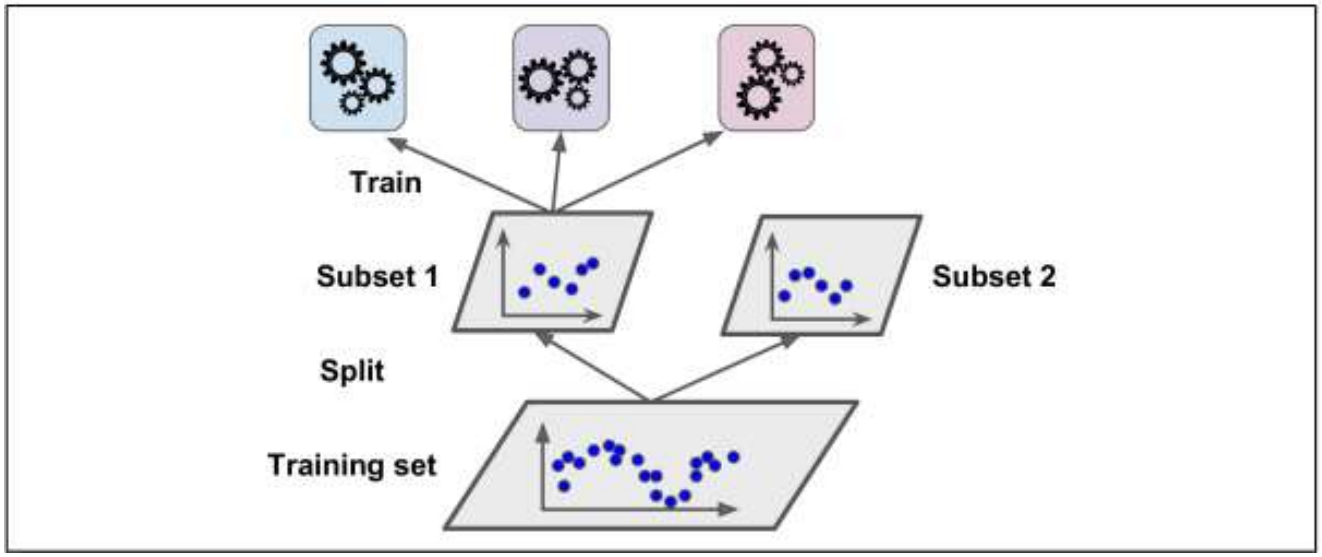
Stacking

Instead of using trivial functions (such as hard voting) to aggregate the predictions of all predictors in an ensemble, we train a model to perform this aggregation.



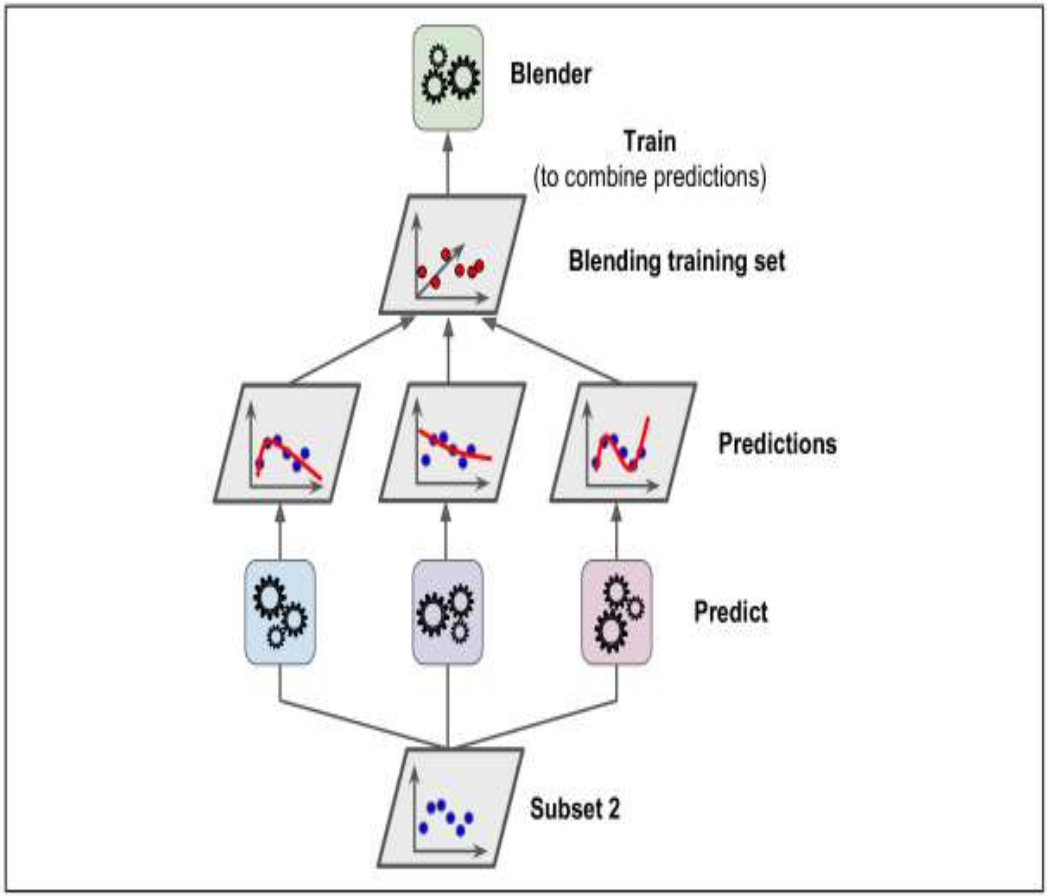
Each of the bottom three predictors predicts a different value (3.1, 2.7, and 2.9), and then the final predictor (called a blender, or a meta learner) takes these predictions as inputs and makes the final prediction (3.0).

Training the First layer



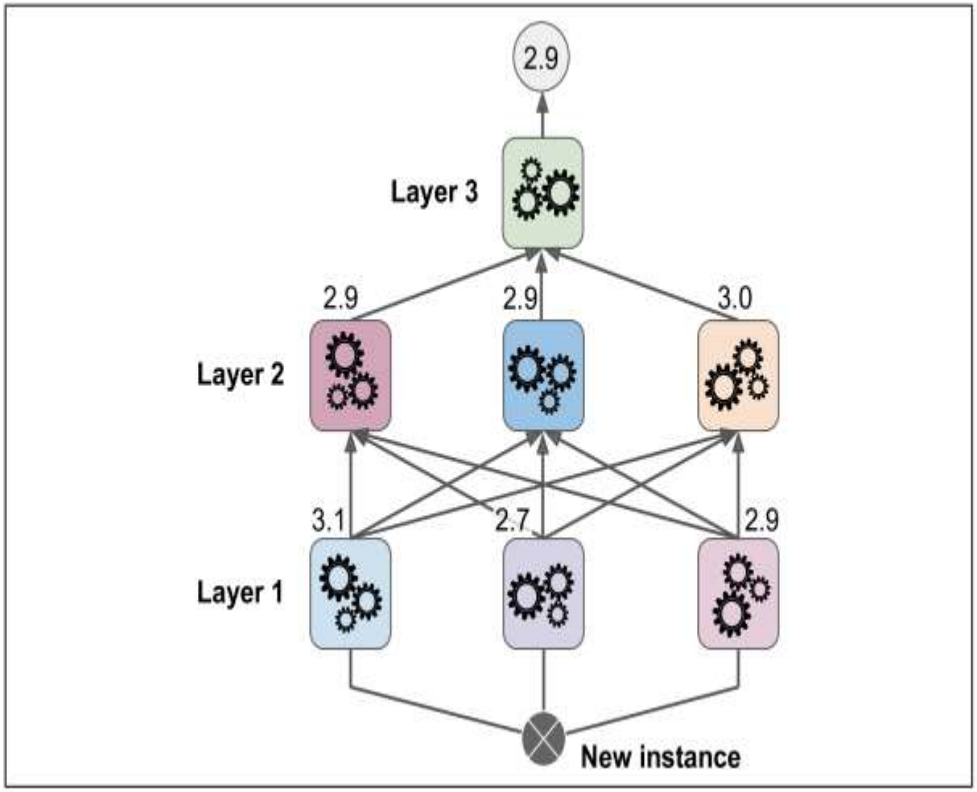
The training set is split in two subsets. The first subset is used to train the predictors in the first layer.
Next, the first layer predictors are used to make predictions on the second (held-out) set .

Training the Blender



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.

Predictions in a multilayer stacking ensemble



- The first training subset is used to train the first layer, the second one is used to create the training set used to train the second layer and the third one is used to create the training set to train the third layer.

- Once this is done, we can make a prediction for a new instance by going through each layer sequentially.


Thank you!