Chapter. 7

Ensemble method

- If you aggregate the predictions of a group of predictors,

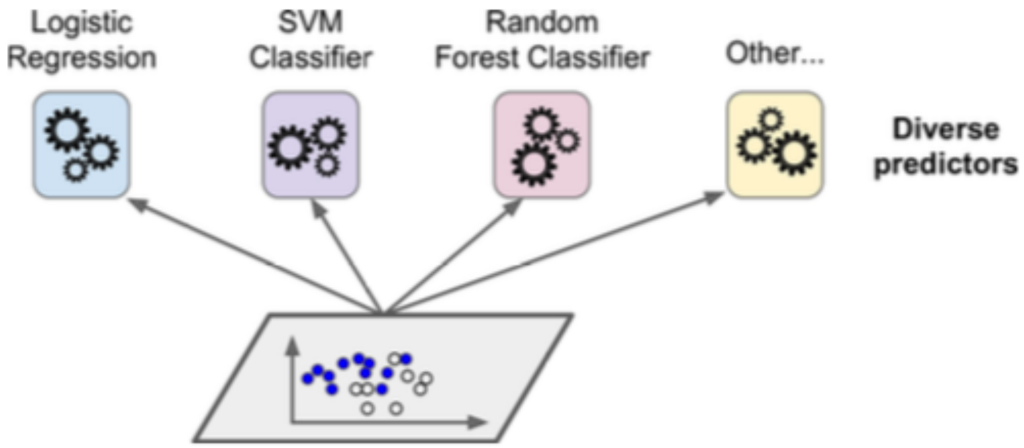
you will get predictions than with the best individual predictors

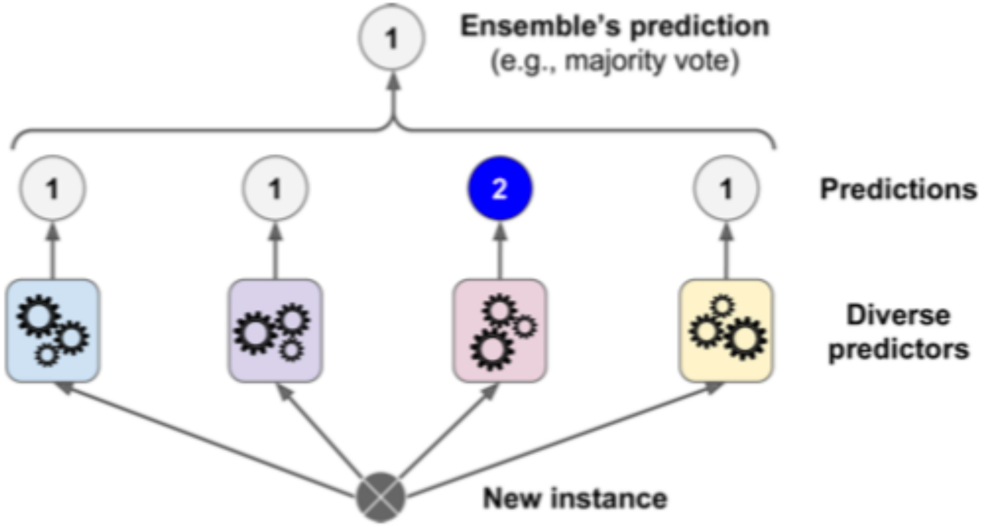
Random Forest

- An ensemble of Decision Tree

Voting Classifiers: hard voting

- Aggregate the predictions of each classifier and predict the class that gets the most votes





Voting Classifiers: soft voting

- If all classifiers are able to estimate class probabilities,

Then, you call predict the class with the highest class probability

- Higher performance than hard voting because it gives more weight to

highly confident votes

Bagging and Pasting

- Use the same training algorithm for every predictor and train them on different random subsets of the training set

Bagging

- Performed with replacement

Pasting

- Performed without replacement

OOB Evaluation

- With bagging, some instance may be sampled several times for any given predictor, while other may not be sampled at all

As m grows, the ratio approach $1 - (1-1/m)^m = 1-\exp(-1) = 63\%$

Random patches method / Random subspaces method

- Random patches method: sampling both training instances and features
- Random subspace method: all training instance but sampling features

Random Forest

- An ensemble of Decision Trees

```
>>> bag_clf = BaggingClassifier(
...     DecisionTreeClassifier(), n_estimators=500,
...     bootstrap=True, n_jobs=-1, oob_score=True)
...
>>> bag_clf.fit(X_train, y_train)
>>> bag_clf.oob_score_
```

```
from sklearn.ensemble import RandomForestClassifier

rnd_clf = RandomForestClassifier(n_estimators=500, max_leaf_nodes=16, n_jobs=-1)
rnd_clf.fit(X_train, y_train)

y_pred_rf = rnd_clf.predict(X_test)
```

Extra Trees (Extremely randomized trees)

- Using random thresholds for each feature rather than searching for the best possible thresholds

Feature Importance

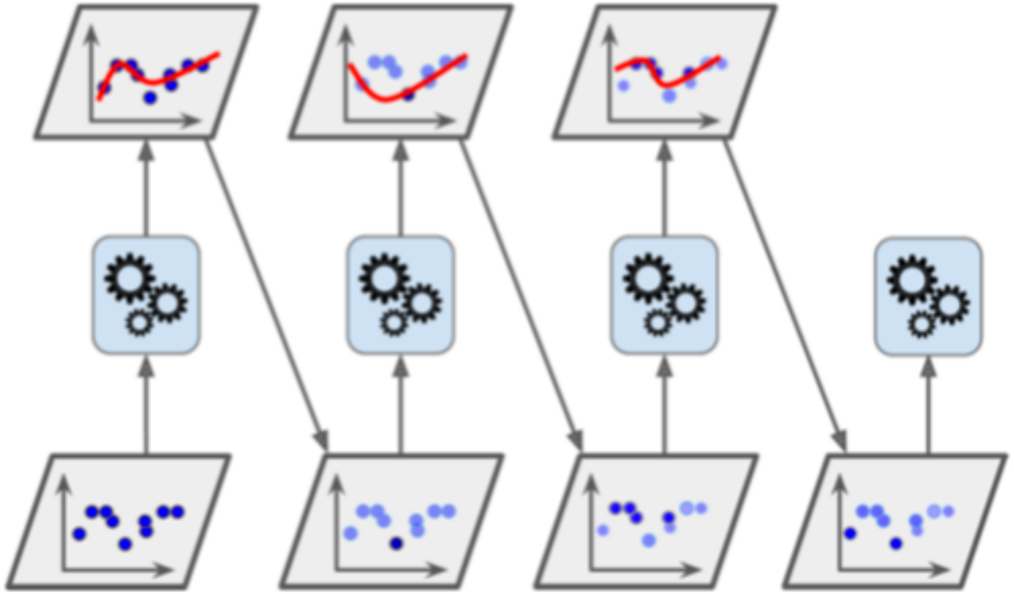
- It is easy to measure the relative importance of each feature
- How much the tree nodes that uses feature reduce impurity on average

Boosting

- Combine several weak learners into a strong learner

Adaboosting

- Pay a bit more attention to the training instances that the predecessor underfited



Adaboosting

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

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

$$\text{for } i = 1, 2, \cdots, m$$
$$w^{(i)} \leftarrow \begin{cases} w^{(i)} & \text{if } \widehat{y_j}^{(i)} = y^{(i)} \\ w^{(i)} \exp\left(\alpha_j\right) & \text{if } \widehat{y_j}^{(i)} \neq y^{(i)} \end{cases}$$

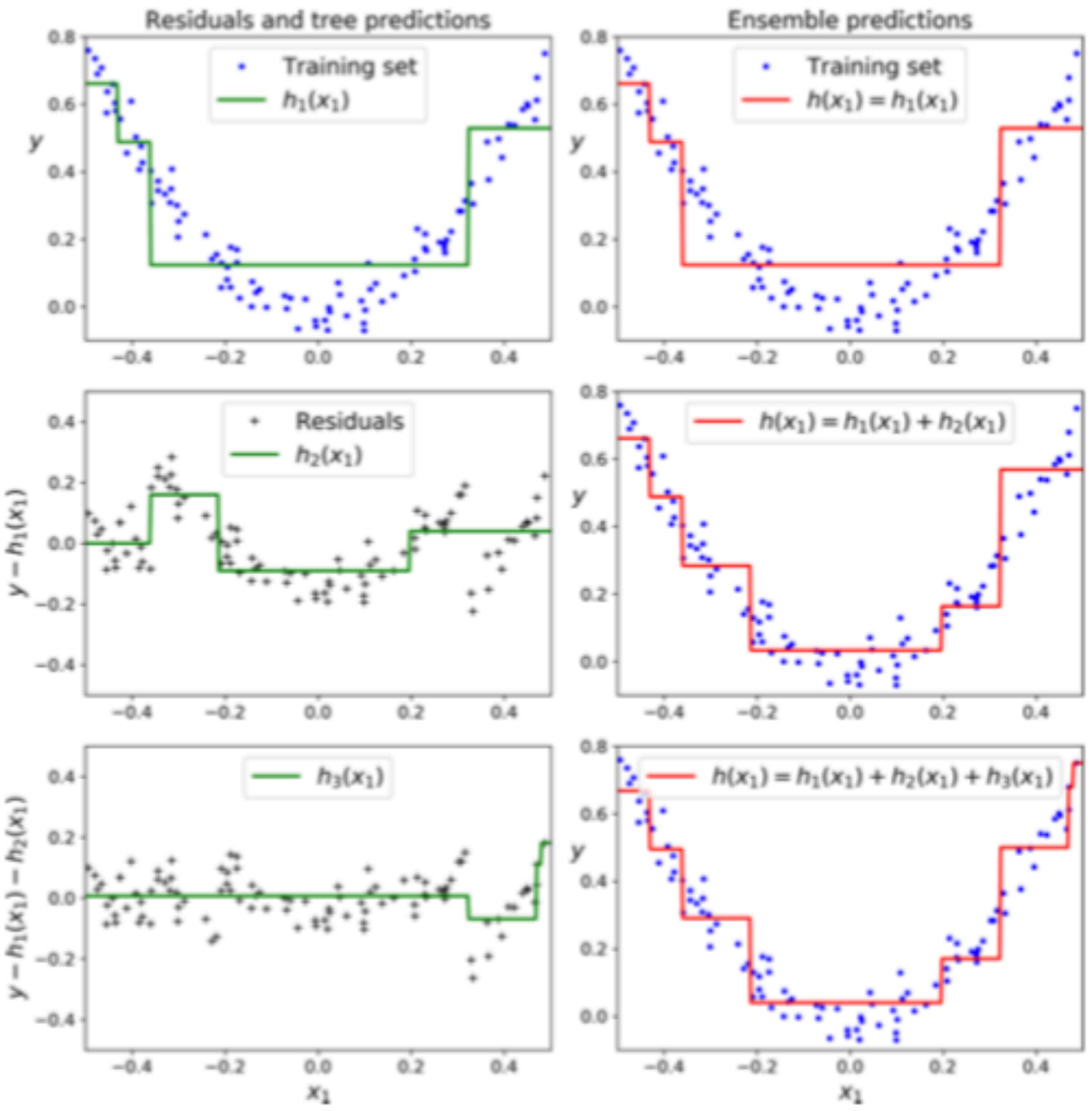
$$\hat{y}(\mathbf{x}) = \underset{k}{\operatorname{argmax}} \sum_{\substack{j=1 \\ \hat{y}_j(\mathbf{x}) = k}}^{N} \alpha_j$$

Gradient boosting

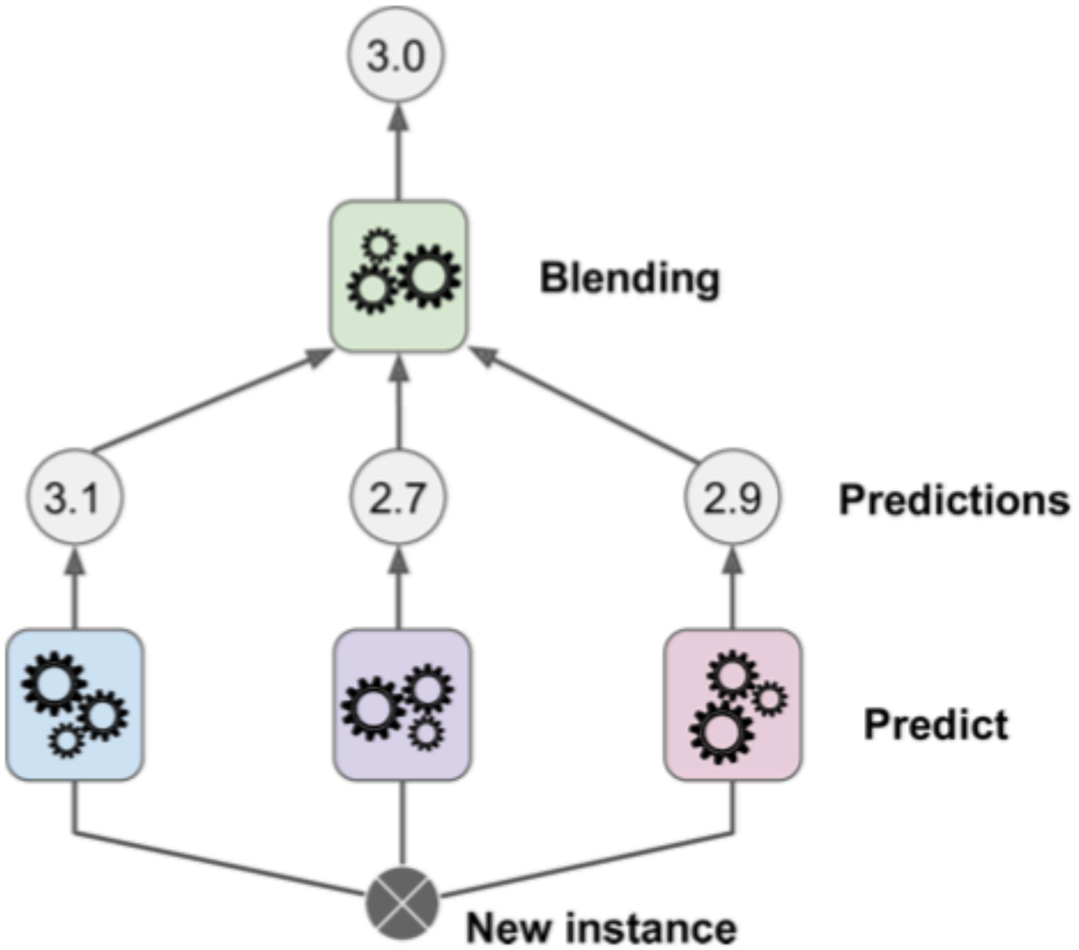
- Work by sequentially adding predictors to an ensemble, each one correcting its predecessor.
- Fit the new predictor to the residual errors (잔여 오차)

```
from sklearn.tree import DecisionTreeRegressor

tree_reg1 = DecisionTreeRegressor(max_depth=2)
tree_reg1.fit(X, y)

y2 = y - tree_reg1.predict(X)
tree_reg2 = DecisionTreeRegressor(max_depth=2)
tree_reg2.fit(X, y2)

y3 = y2 - tree_reg2.predict(X)
tree_reg3 = DecisionTreeRegressor(max_depth=2)
tree_reg3.fit(X, y3)

y_pred = sum(tree.predict(X_new) for tree in (tree_reg1, tree_reg2, tree_reg3))
```

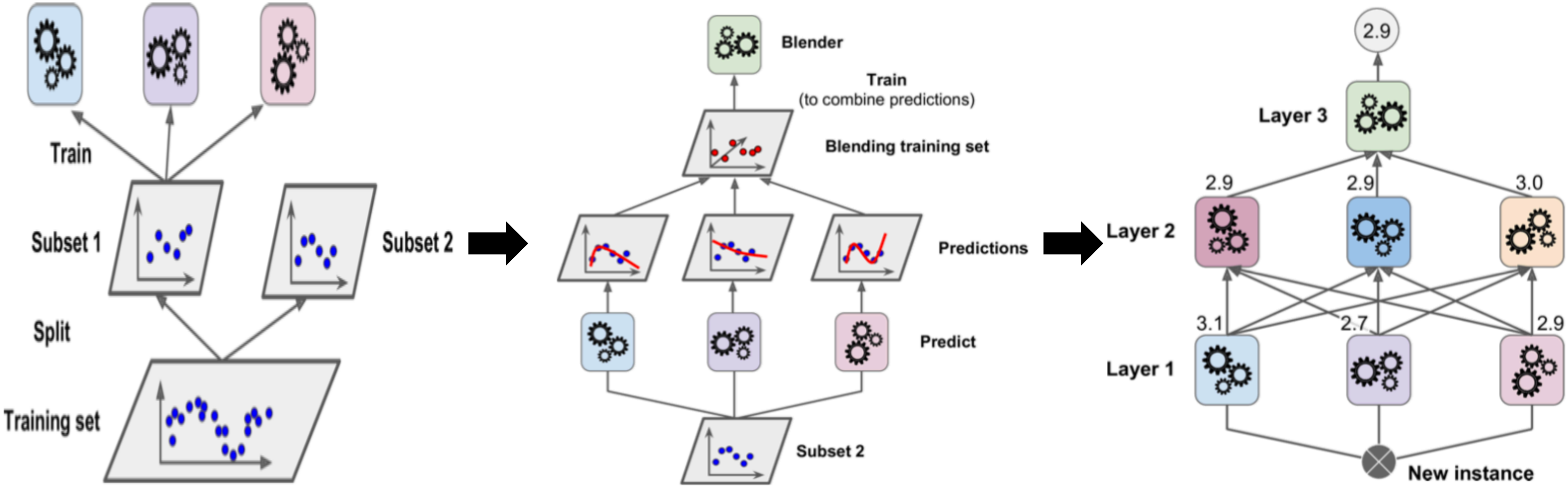
Gradient boosting



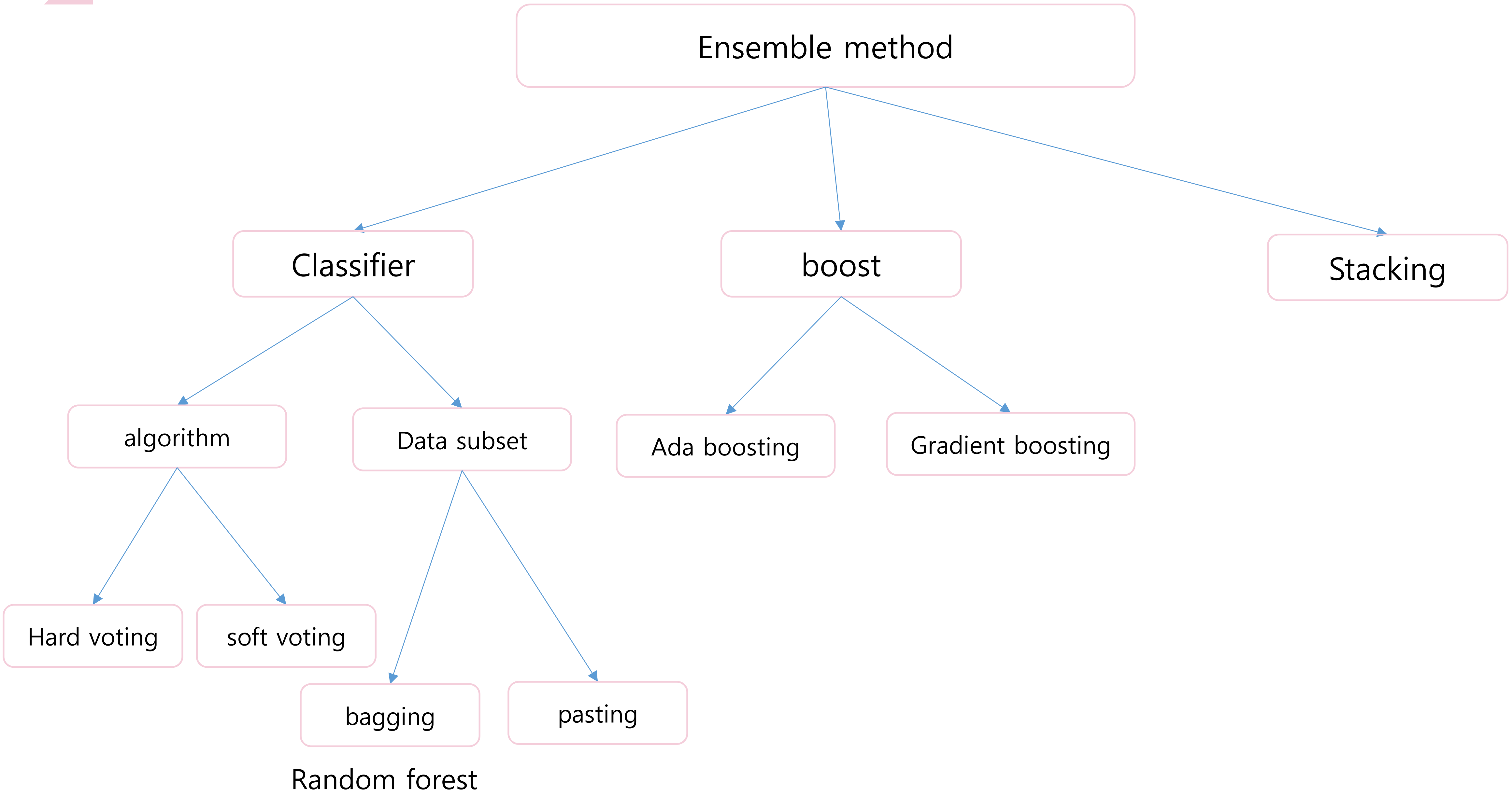
Stacking



Stacking



Train
Subset 1
Subset 2
Split
Training set
Blender
Train
(to combine predictions)
Blending training set
Predictions
Predict
Subset 2
2.9
Layer 3
2.9
2.9
3.0
Layer 2
3.1
2.7
2.9
Layer 1
New instance


Ensemble method
Classifier
boost
Stacking
algorithm
Data subset
Ada boosting
Gradient boosting
Hard voting
soft voting
bagging
pasting
Random forest