Chapter 7: Ensemble Learning and Random Forest

Dr. Xudong Liu
Assistant Professor
School of Computing
University of North Florida

Monday, 9/23/2019

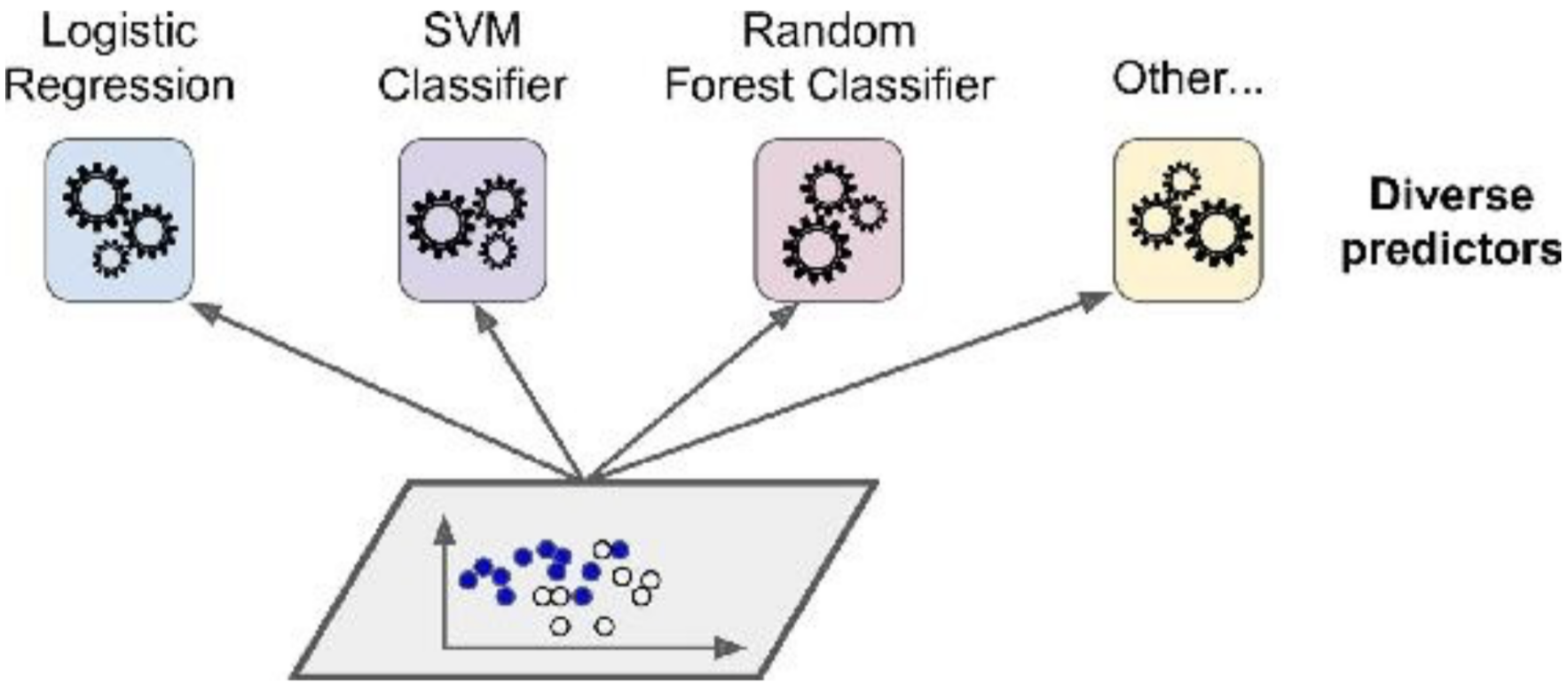
Notations

1. Voting classifiers: hard and soft
2. Bagging and pasting
   - Random Forests
3. Boosting: AdaBoost, GradientBoost, Stacking

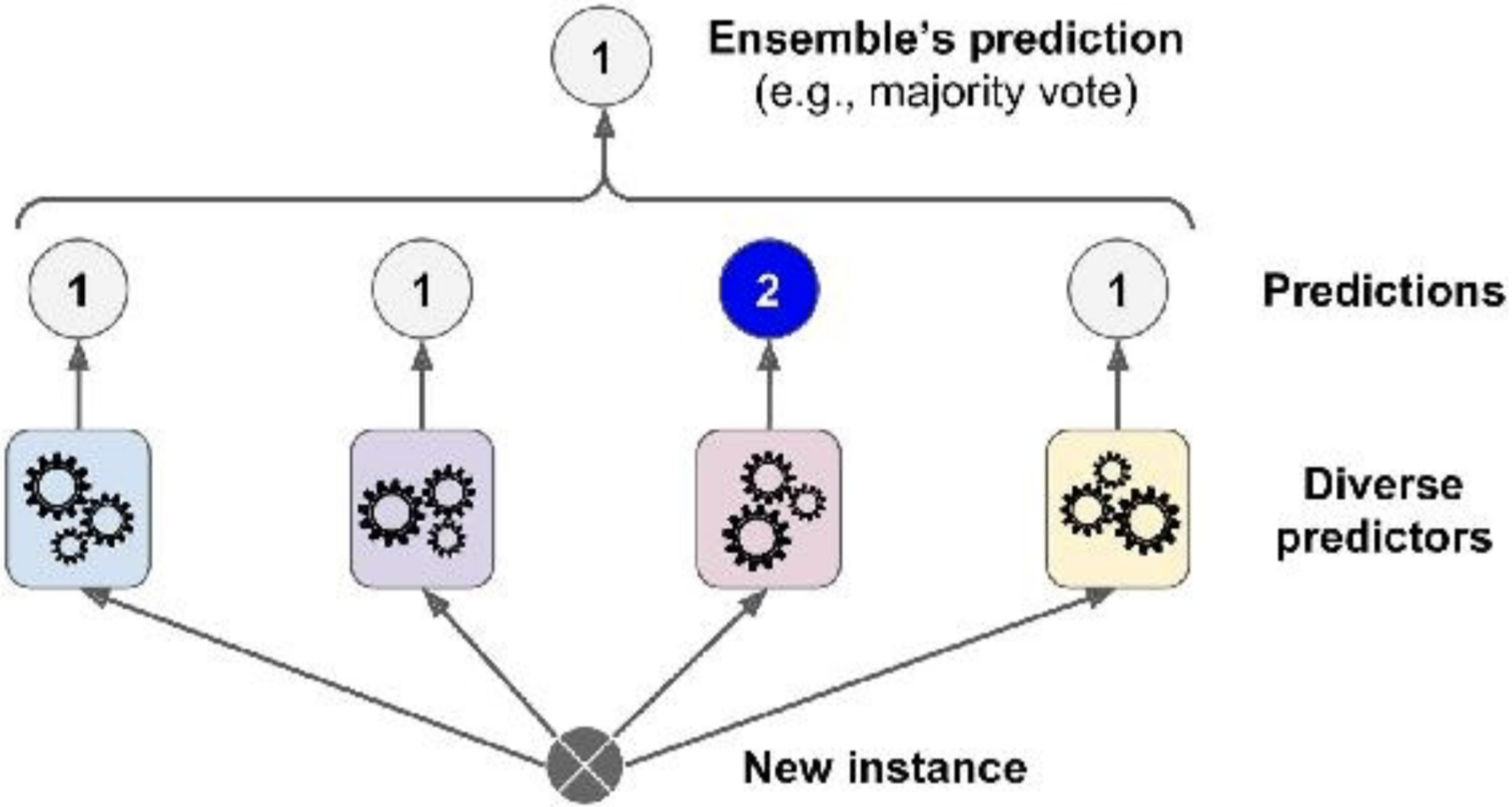
Hard Voting Classifiers

- In the setting of binary classification, *hard voting* is a simple way for an ensemble of classifiers to make predictions, that is, to output the *majority* winner between the two classes.
  - If multi-classes, output the *Plurality* winner instead, or the winner according another voting rule.
- Even if each classifier is a weak learner, the ensemble can be a strong learner under hard voting, provided sufficiently many weak yet diverse learners.

Training Diverse Classifiers

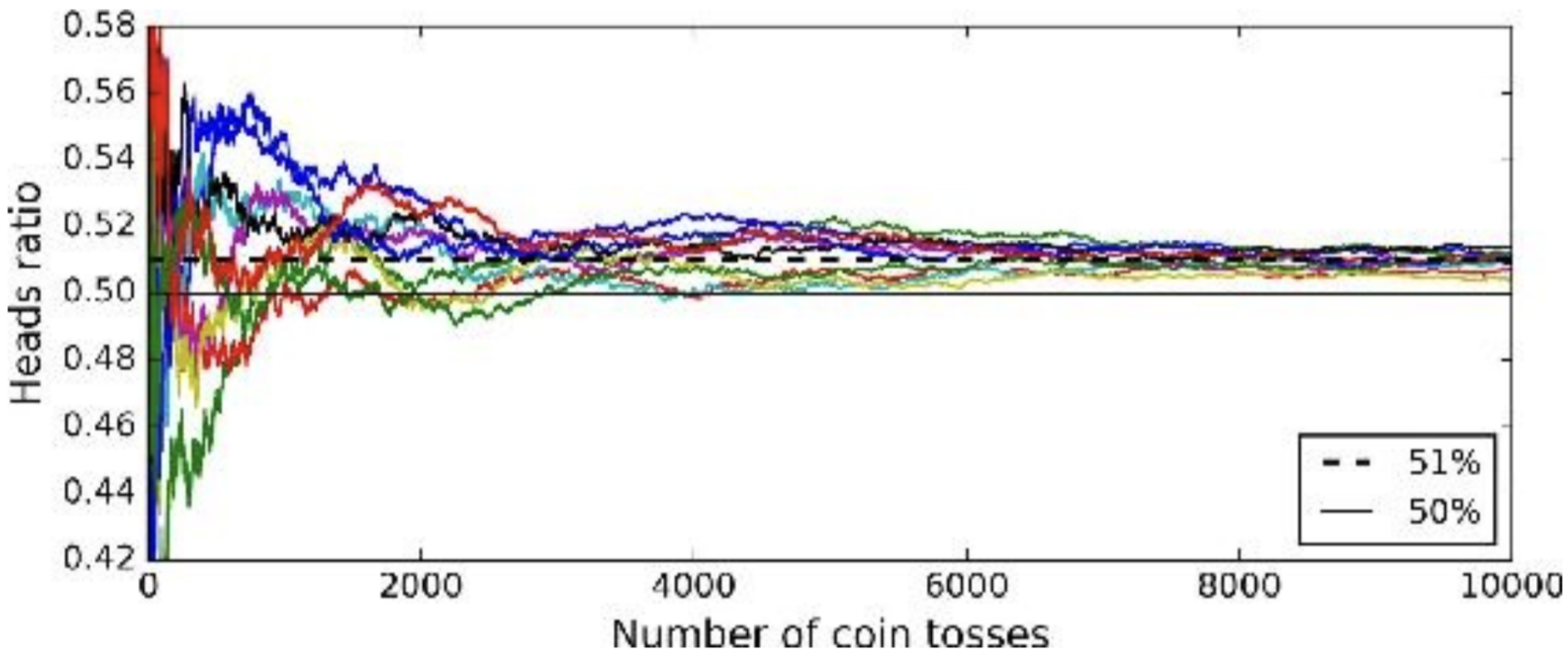


Hard Voting Predictions



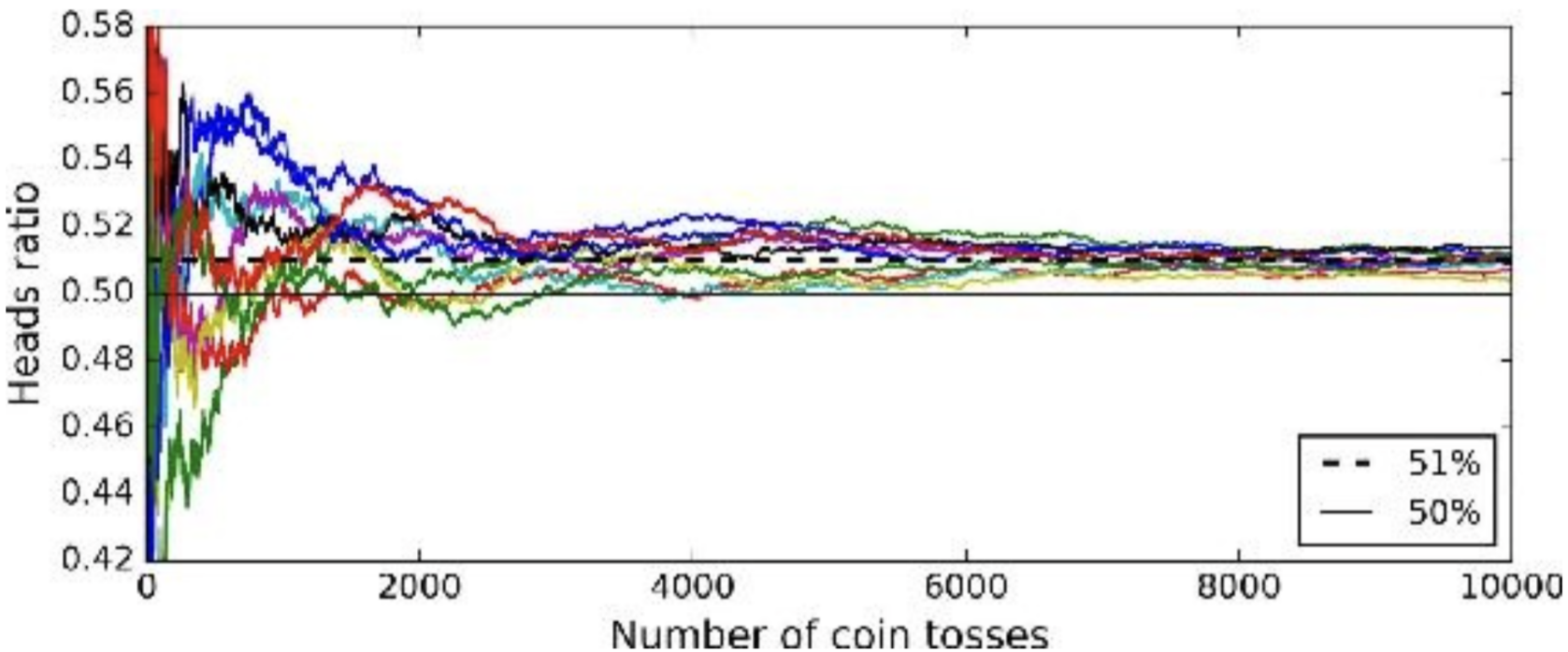
Ensemble of Weak is Strong?

- Think of a slightly biased coin with 51% chance of heads and 49% of tails.
- Law of large numbers: as you keep tossing the coin, assuming every toss is *independent* of others, the ratio of heads gets closer and closer to the probability of heads 51%.



Ensemble of Weak is Strong?

- Eventually, all 10 series end up consistently above 50%.
- As a result, the 10,000 tosses as a whole will output heads with close to 100% chance!
- Even for an ensemble of 1000 classifiers, each correct 51% of the time, using hard voting it can be of up to 75% accuracy.
- Scikit-Learn: *from sklearn.ensemble import VotingClassifier*



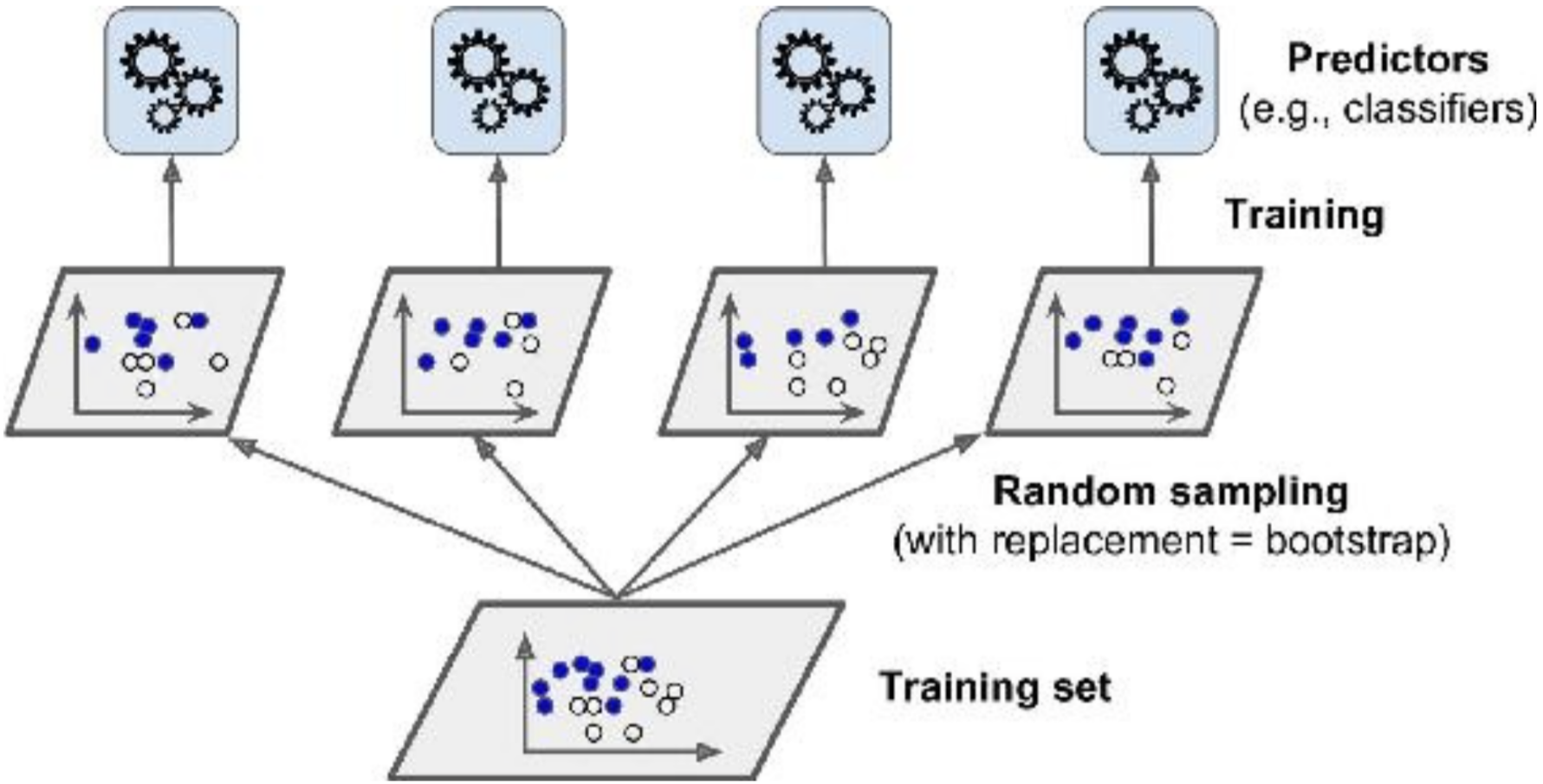
Soft Voting Predictions

- If the classifiers in the ensemble have *class probabilities*, we may use *soft voting* to aggregate.
- Soft voting: the ensemble will predict the class with the *highest* class probability, averaged over all the individual classifiers.
- Often better than hard voting.
- Scikit-Learn: set *voting*=*"soft"*
- But how do we train the individual classifiers that are *diverse*?

Bagging and Pasting

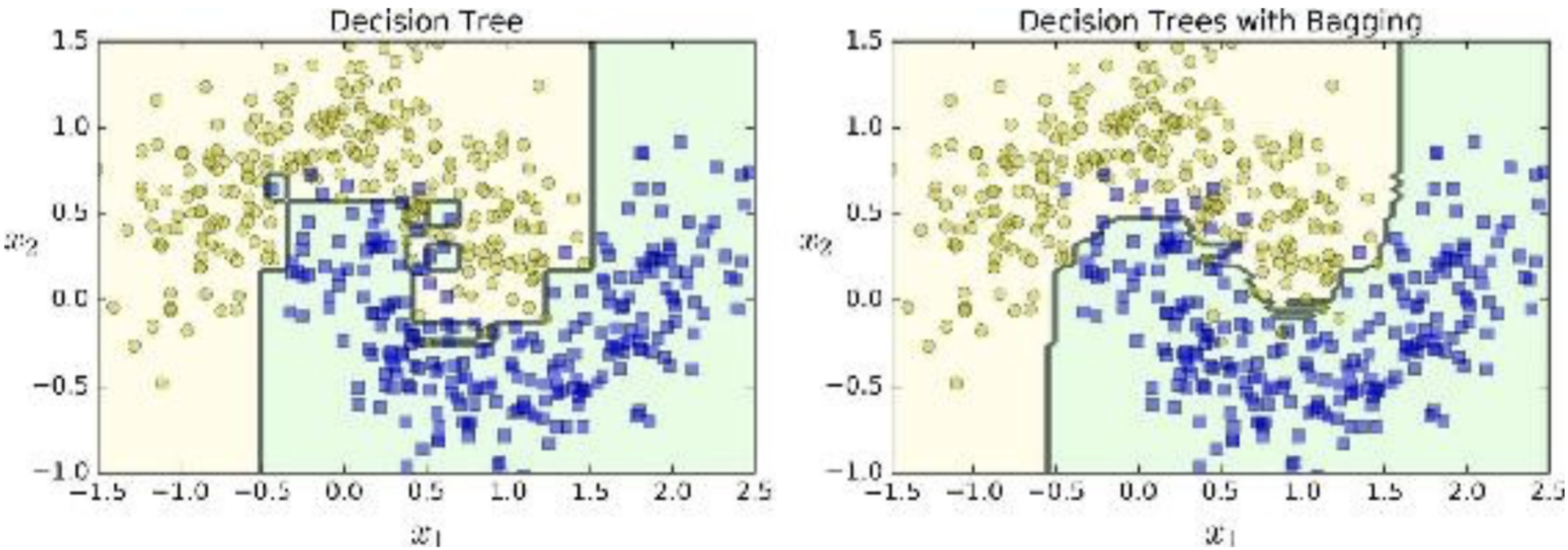
- Both bagging and pasting are to use the same training algorithm for every predictor, but to train them on different random subsets.
- Bagging: sampling is performed *with* replacement.
- Pasting: sampling is performed *without* replacement.
- Scikit-Learn: *BaggingClassifier* and *BaggingRegressor*. Set *bootstrap=False* if you want pasting.

Bagging and Pasting



Random Forests

- A random forest is an ensemble of decision trees trained using bagging.
- To ensure diversity, when splitting in a member decision tree, the algorithm searches for the best feature among a *random* subset of attributes.



Boosting

- *Boosting* is a learning method for ensemble models, where individual predictors are trained *sequentially*, each trying to correct its predecessor.
- We will talk about AdaBoost (Adaptive Boosting) [1] and GradientBoost [2].

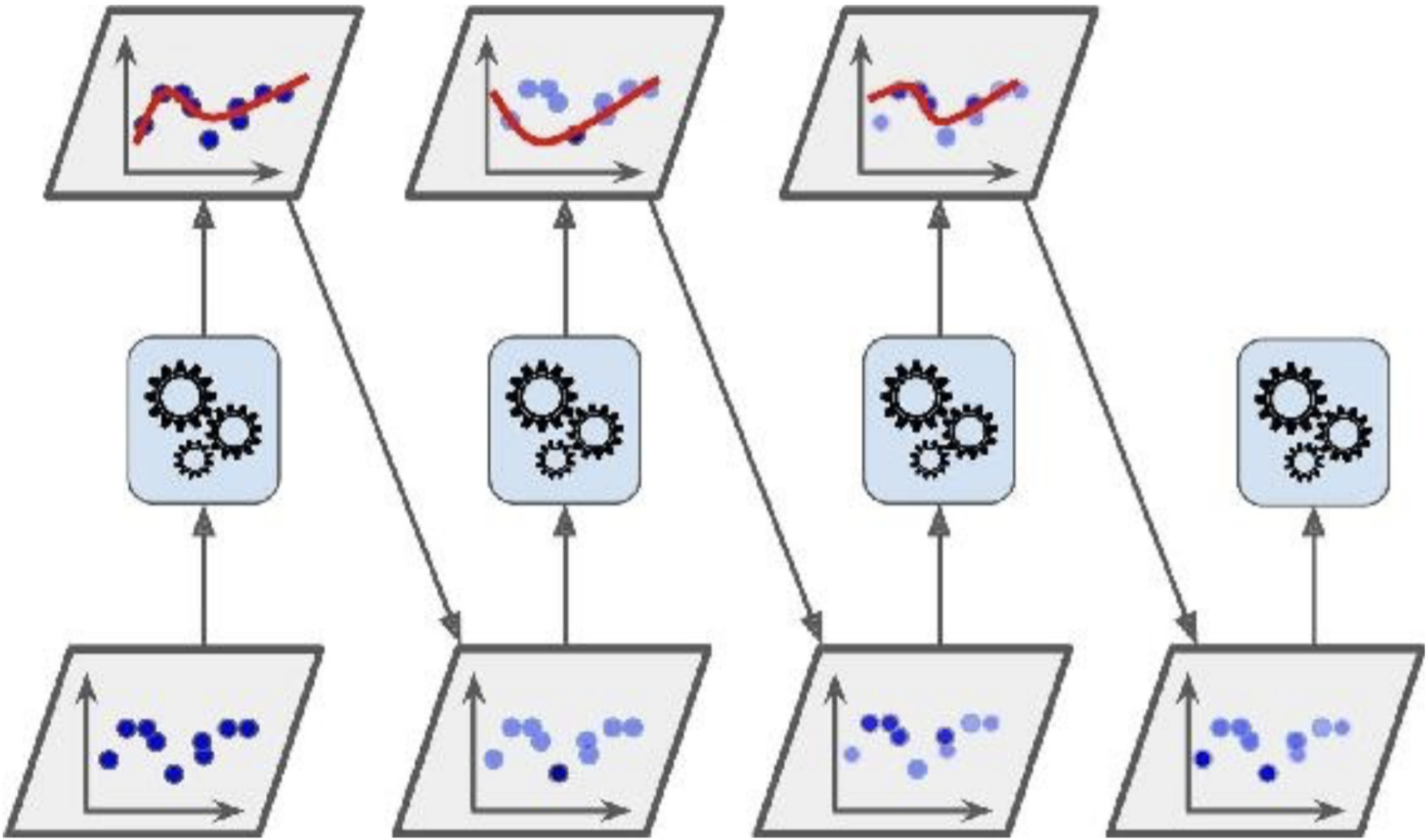
[1] *A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting*, Freund and Schapire, 1997.

[2] *Arcing the Edge*, Breiman, 1997.

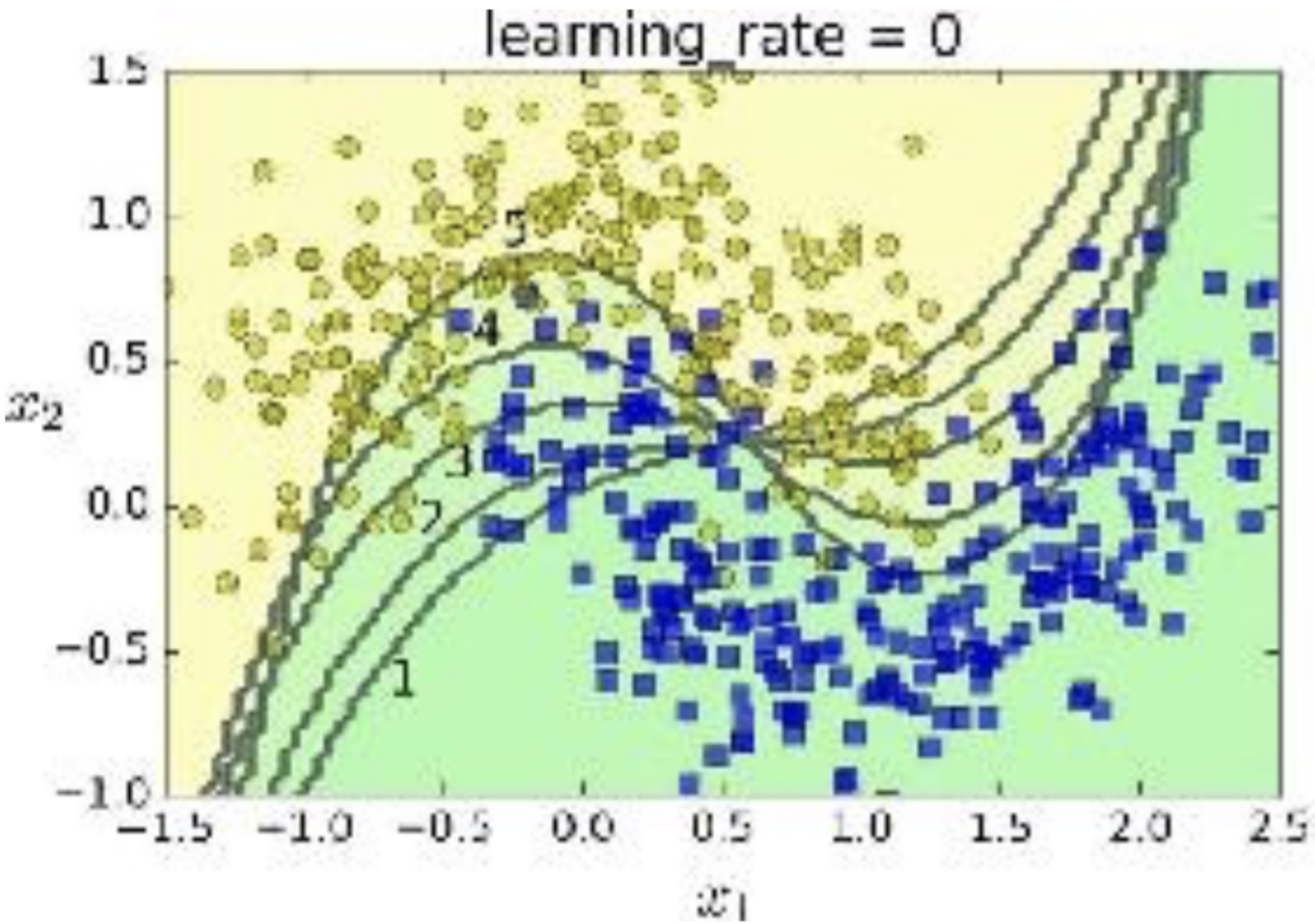
AdaBoost

- First, a base classifier is trained and used to predict on the training set.
- The weights of those **misclassified** training examples are increased.
- A second classifier is trained using the the same training set but with updated weightes.
- Again, weights of misclassified examples are increased. And the algorithm repeats, until either the desired number of predictors is reached, or when a perfect predictor is found.
- Scikit-Learn: *AdaBoostClassifier*

AdaBoost



AdaBoost


learning_rate = 0
1.5
1.0
0.5
0.0
−0.5
−1.0
x_2
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
x_1
1
3
4
5

AdaBoost Algorithm

- Each example weight $w^{(i)}$ in the training set is initialized to $\frac{1}{m}$, where m is the number of examples in training set.
- Set $j = 1$, we train the j-th predictor and compute its weighted error r_j and its weight α_j:

$$r_j = \frac{\sum_{i=1,\, \hat{y}_j^{(i)} \neq y^{(i)}}^{m} w^{(i)}}{\sum_{i=1}^{m} w^{(i)}}, \; \alpha_j = \eta \cdot \log \frac{1 - r_j}{r_j}.$$

- Next, we increase weights of misclassified examples. For $i \leftarrow 1$ to m, we update

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

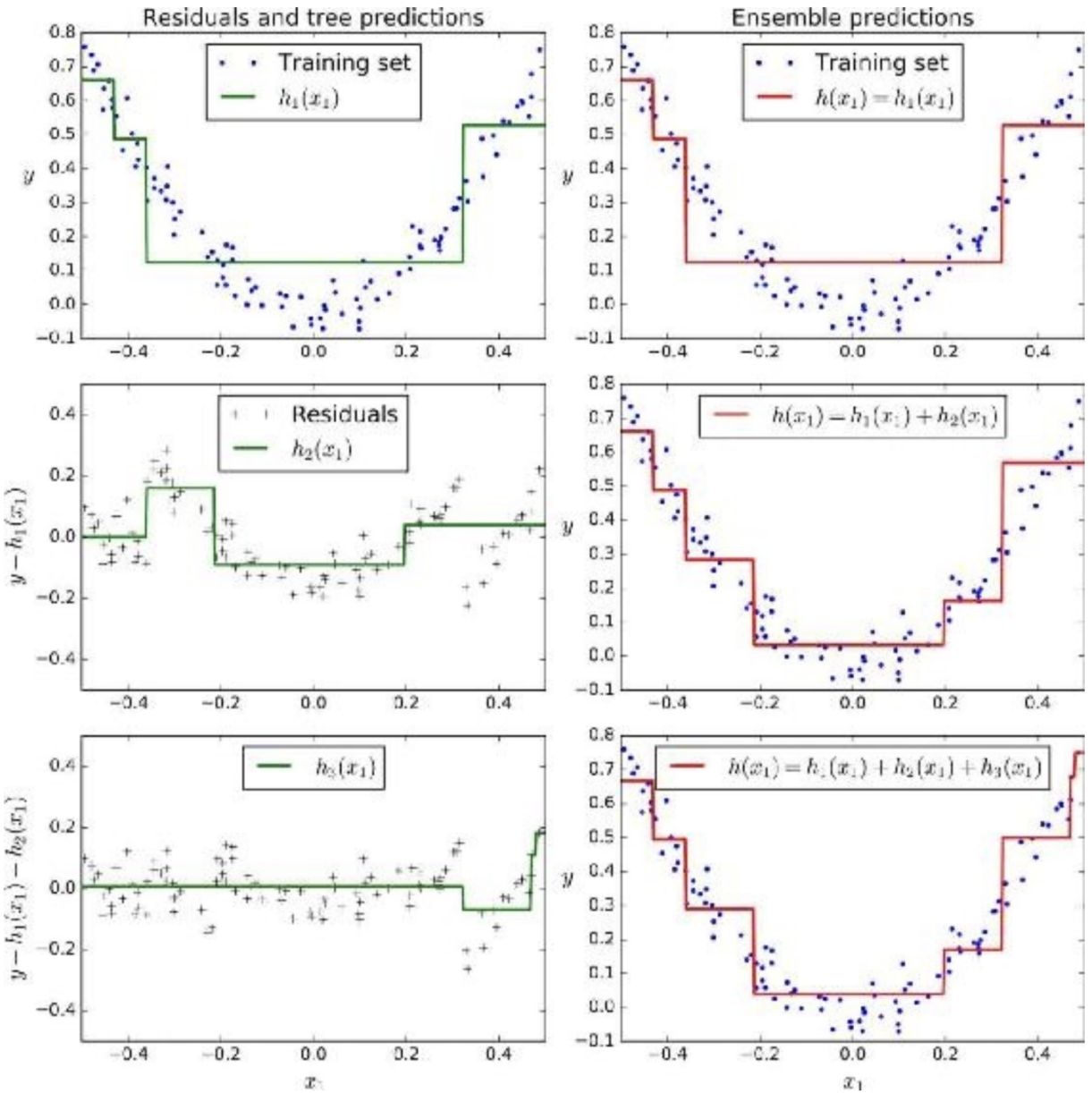
- Finally, to make predictions, assuming N predictors, we have

$$\hat{y}_j^{(i)} = \arg\max_k \sum_{j=1,\, \hat{y}_j(\mathbf{x})=k}^{N} \alpha_j$$

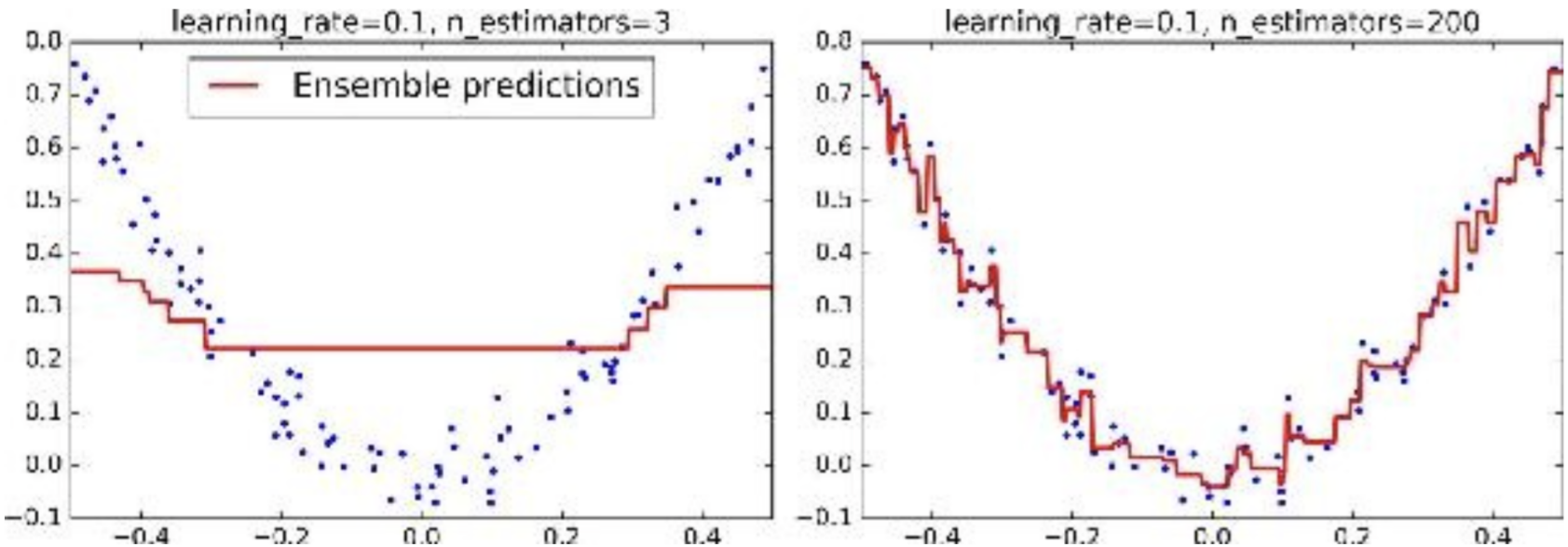
Gradient Boosting

- Like AdaBoost, GradientBoost sequentially adds predictors to the ensemble, each one correcting its predecessor.
- Unlike AdaBoost, GradientBoost tries to fit the new predictor to the *residual errors* made by the previous predictor.
- Scikit-Learn: *GradientBoostingClassifier* and *GradientBoostingRegressor*

Gradient Boosting

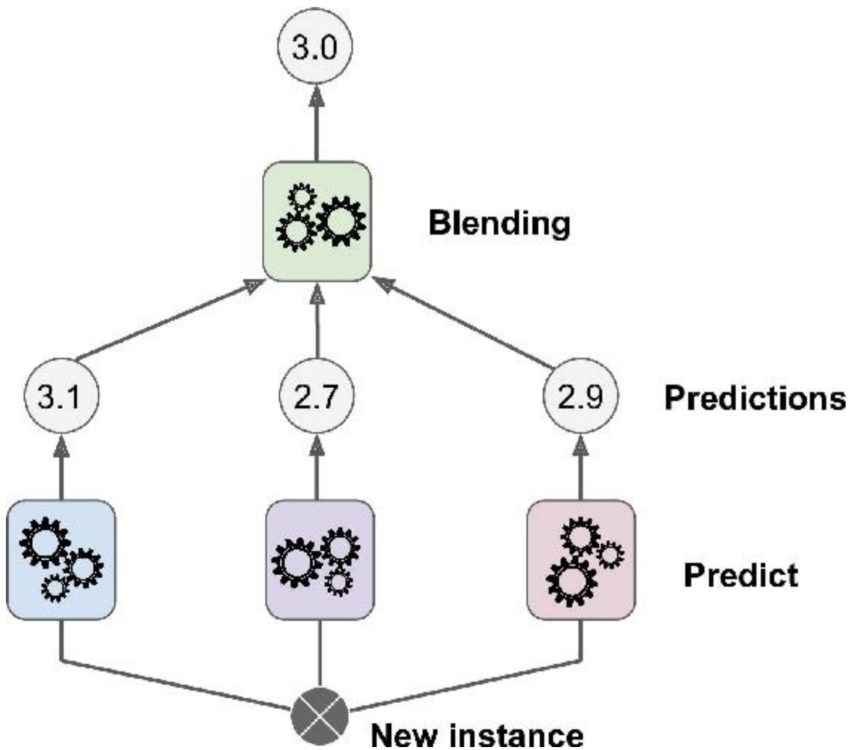


Gradient Boosting



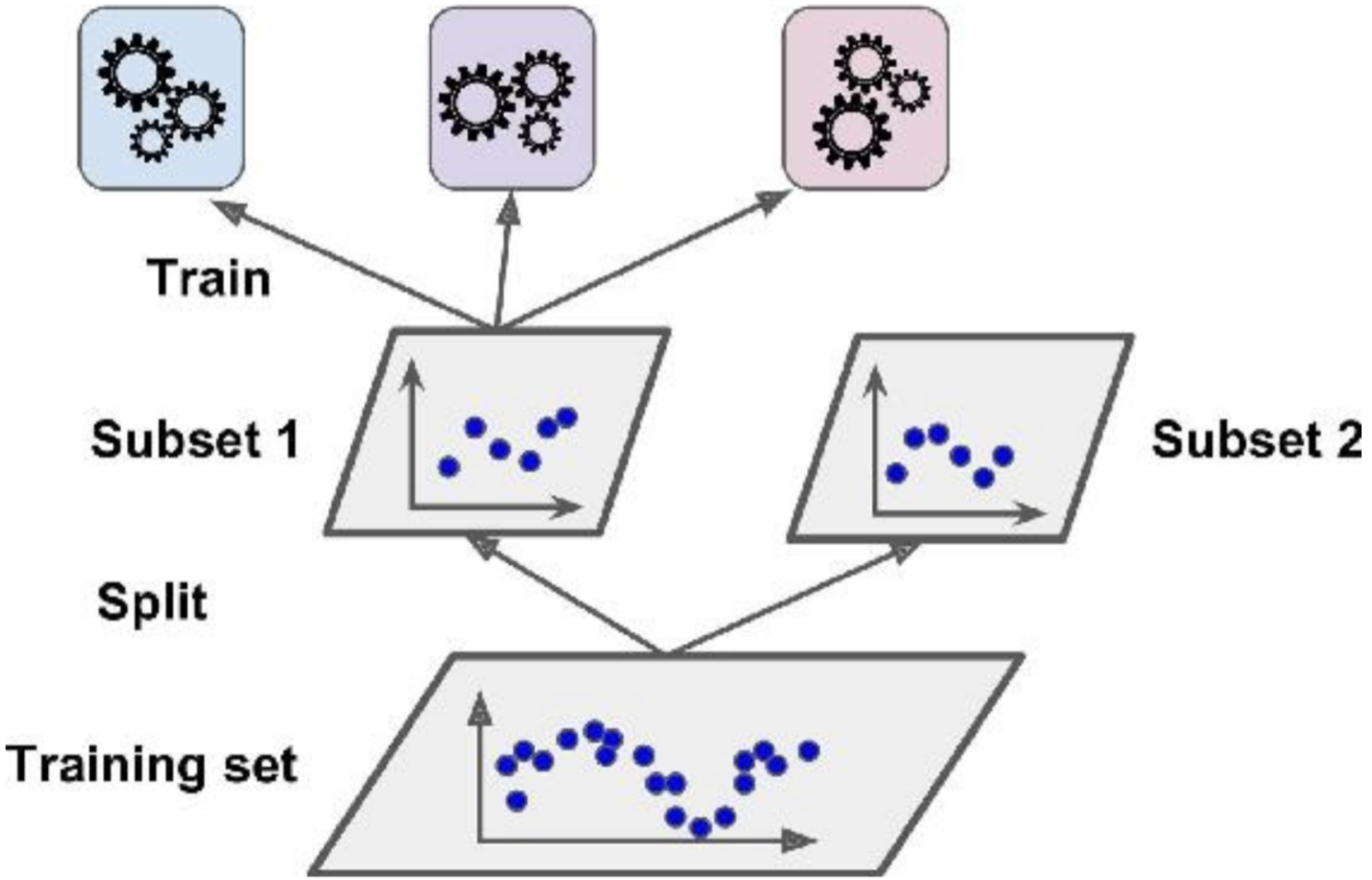
Stacking

- In the *Stacking* [3] method, we *train a model* to perform the aggregation of the predictions from the member predictors in the ensemble, instead of using trivial aggregating functions such as hard and soft voting.

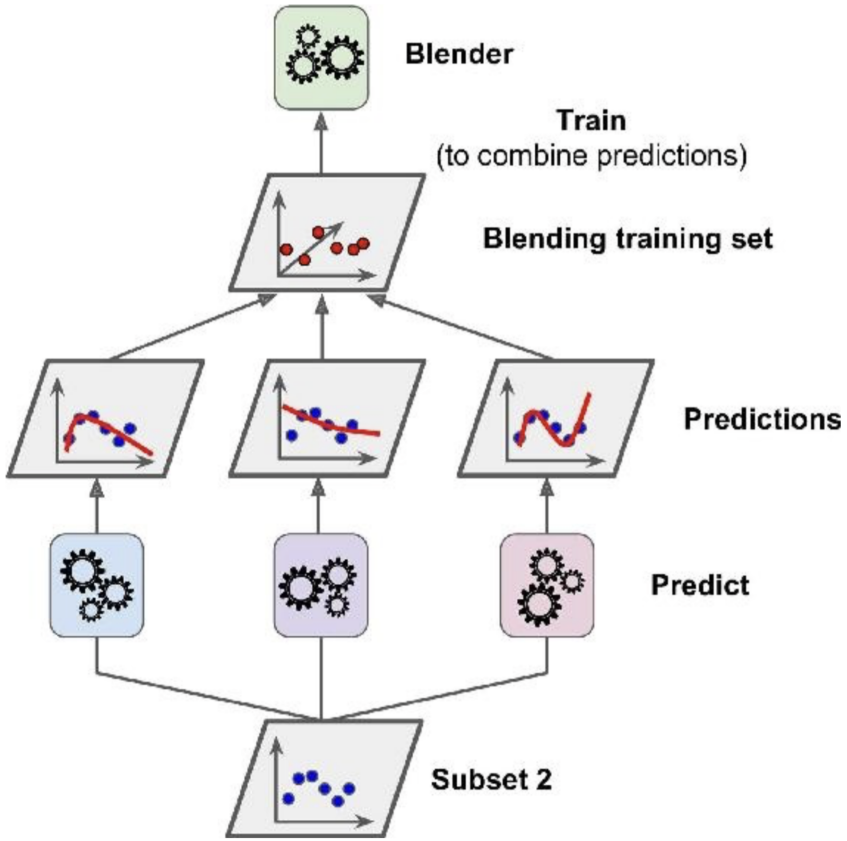


[3] *Stacked Generalization*, Wolpert, 1992.

Stacking



Stacking


Blender
Train
(to combine predictions)
Blending training set
Predictions
Predict
Subset 2

Stacking