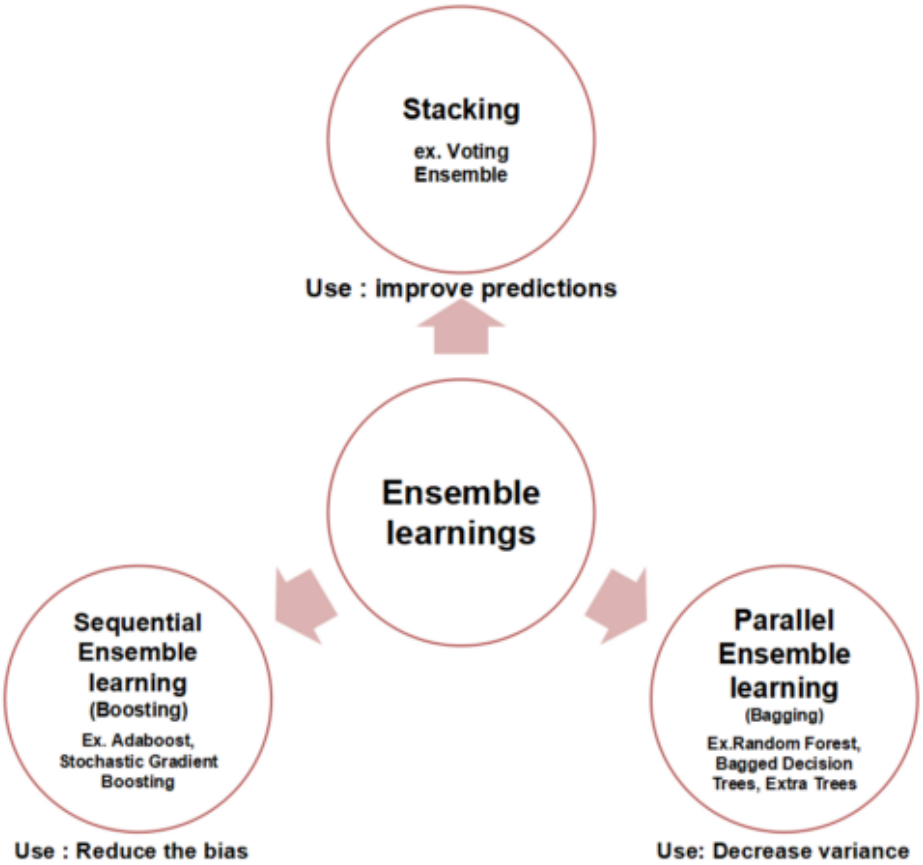
Chapter 4: ML Models for Tabular Datasets

Amir Goudarzi, Fall 2022

CE Department
Sharif University of Technology

Ensemble methods

- **Ensemble methods** is a machine learning technique that combines several base models in order to produce one optimal predictive model.

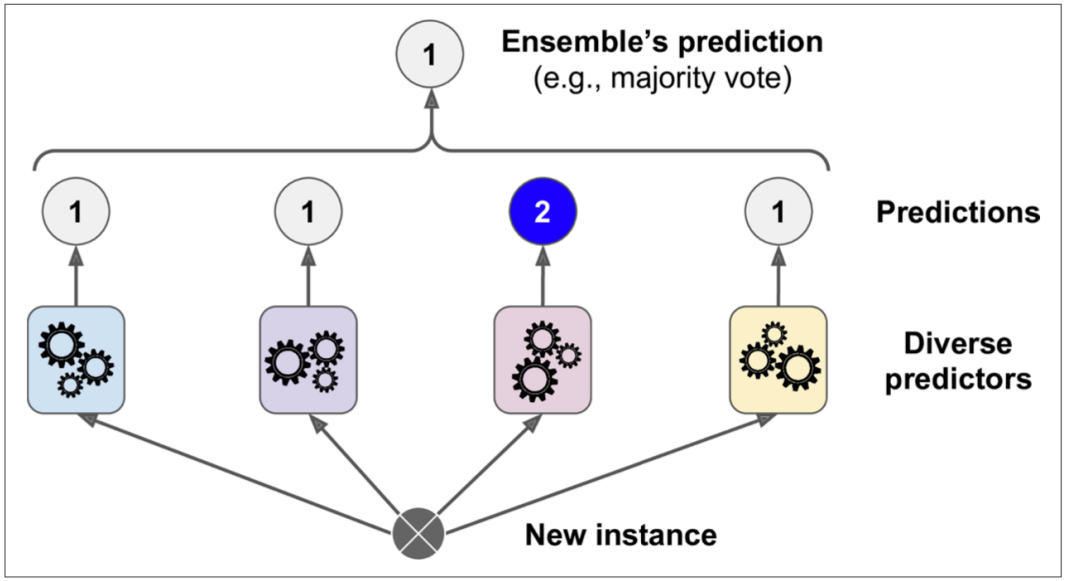


When to use ensemble learning

- Ensemble learning works best when the base models are not correlated. The idea behind using uncorrelated models is that each may be solving a weakness of the other.
- It also helps with reducing variance and making more robust models.
- 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.

Hard Voting

- **Hard Voting** →Majority Vote



$$\hat{y} = arg\max_{j} \sum_{i=1}^{n} h_i^j(x)$$

$$H(x) = \{h_1, h_2, \ldots, h_n\}, h_i^j(x) \in [0, 1]$$

Where $h_i^j(x)$ is the predicted class membership probability ofthe ith classifier for class label j.

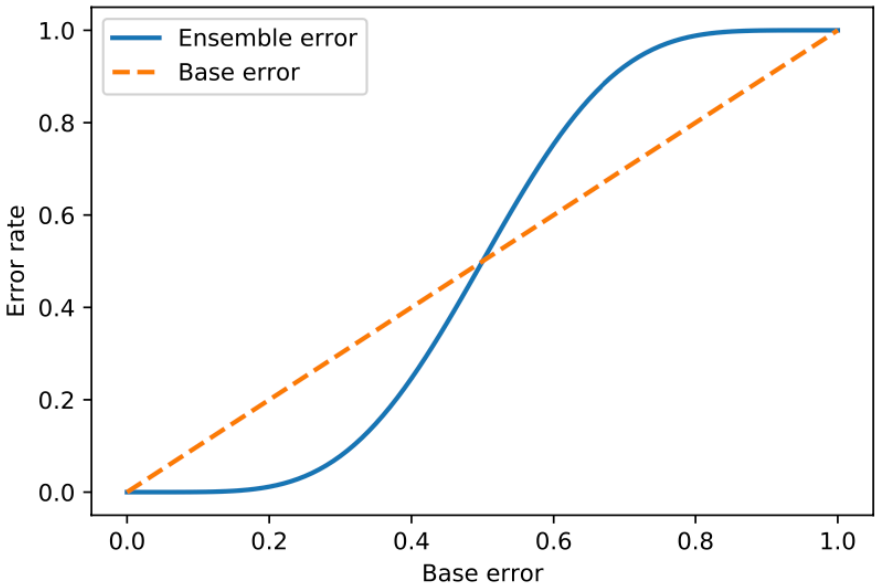
Why Majority Voting?

- Assume n independent(errors are uncorrelated) binary classifiers with a base error rate ϵ (better than random guessing)
- Probability of making a wrong prediction via the ensemble if k classifiers predict same class label:

$$P(k) = \binom{n}{k} \epsilon^k (1-\epsilon)^{n-k}, k > \lceil n/2 \rceil$$

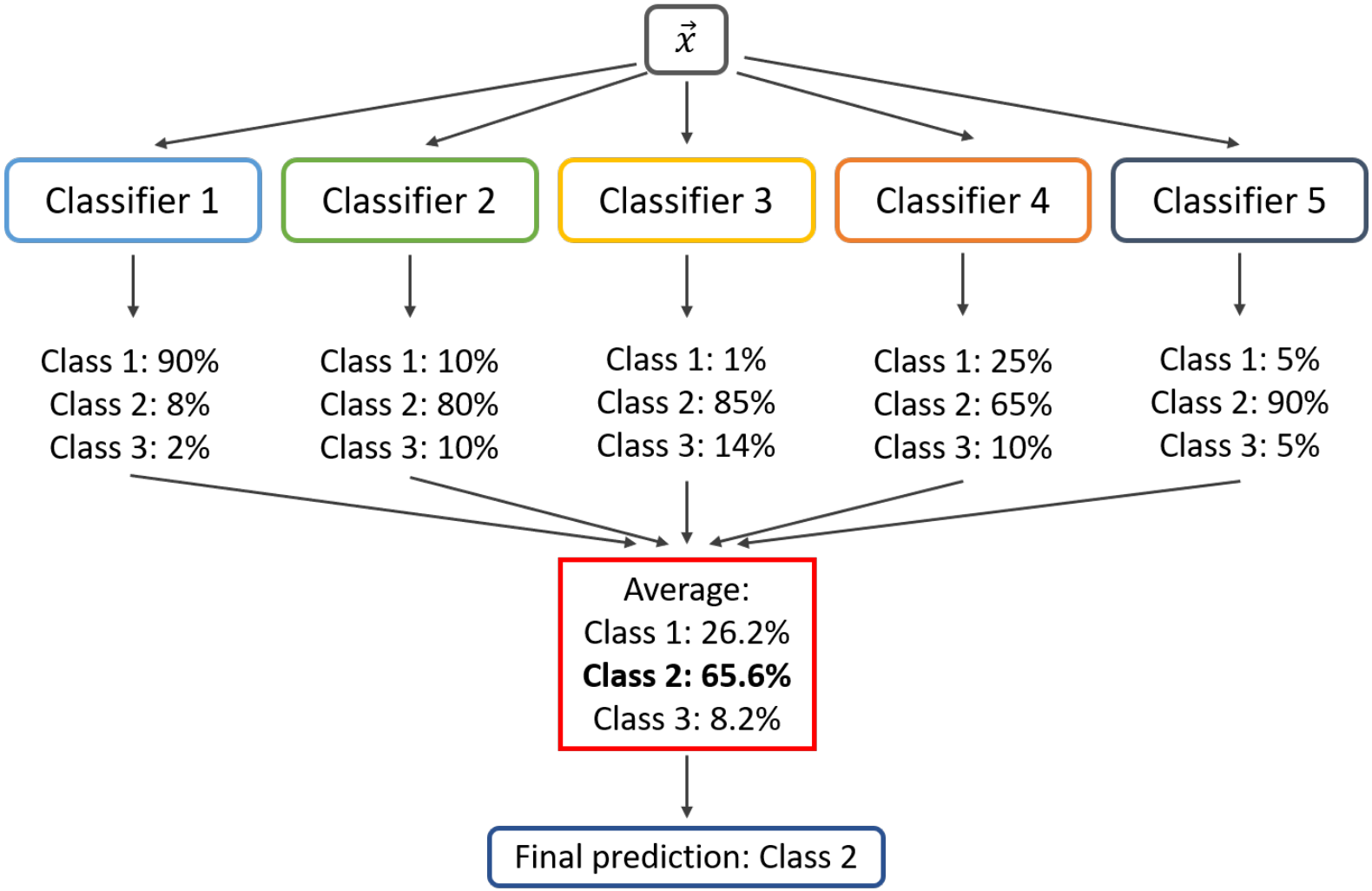
- Ensemble error:

$$\epsilon_{ens} = \sum_{k}^{n} \epsilon^k (1-\epsilon)^{n-k}, k > \lceil n/2 \rceil$$



Soft Voting

- **Soft Voting** →Highest class probability(When **all** classifiers are able to estimate class probabilities)



Soft Voting

$$\hat{y} = arg\max_j \sum_{i=1}^{n} w_i h_i^j(x), w_i > 0, \sum_{i=1}^{n} w_i = 1$$

$$H(x) = \{h_1, h_2, \ldots, h_n\}, h_i^j(x) \in [0, 1]$$

- Where $h_i^j(x)$ is the predicted class membership probability ofthe ith classifier for class label j, w_i is the optional weighting parameter.
- Example:

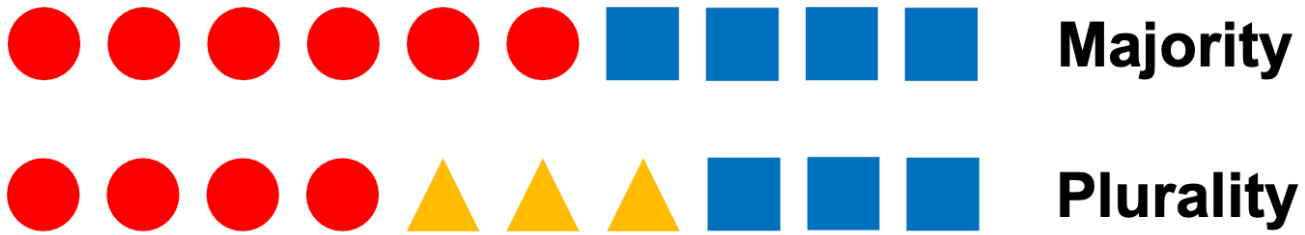
$H(x)$	$j = 0$	$j = 1$	w_i
h_1	0.9	0.1	0.2
h_2	0.8	0.2	0.3
h_3	0.3	0.7	0.5

$$p(j = 0|x) = 0.2 * 0.9 + 0.3 * 0.8 + 0.3 * 0.5 = 0.57$$
$$p(j = 1|x) = 0.2 * 0.1 + 0.3 * 0.2 + 0.3 * 0.7 = 0.28$$

$$\hat{y} = arg\max_j p(j = 0|x), p(j = 1|x) = 0$$

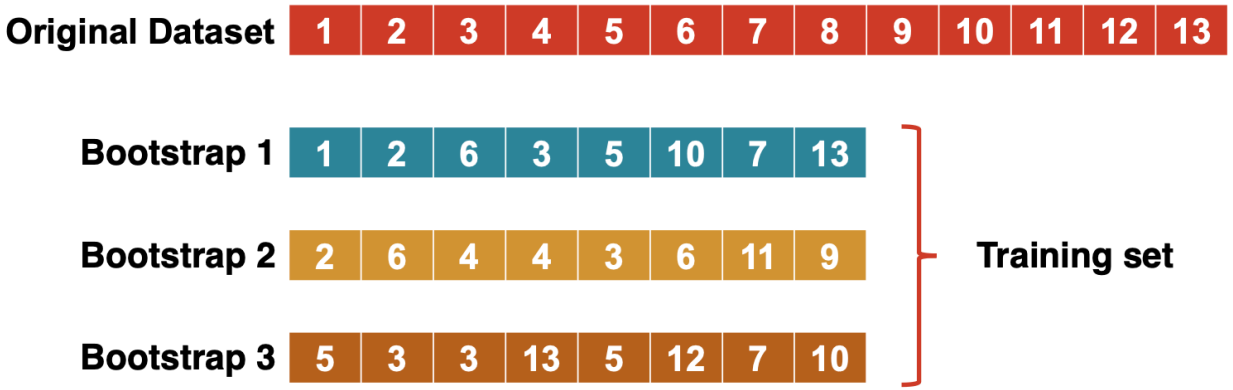
Plurality Voting

- **Plurality Voting(Unstable)** →Most Voted (When none of the predictions get more than half of the votes)



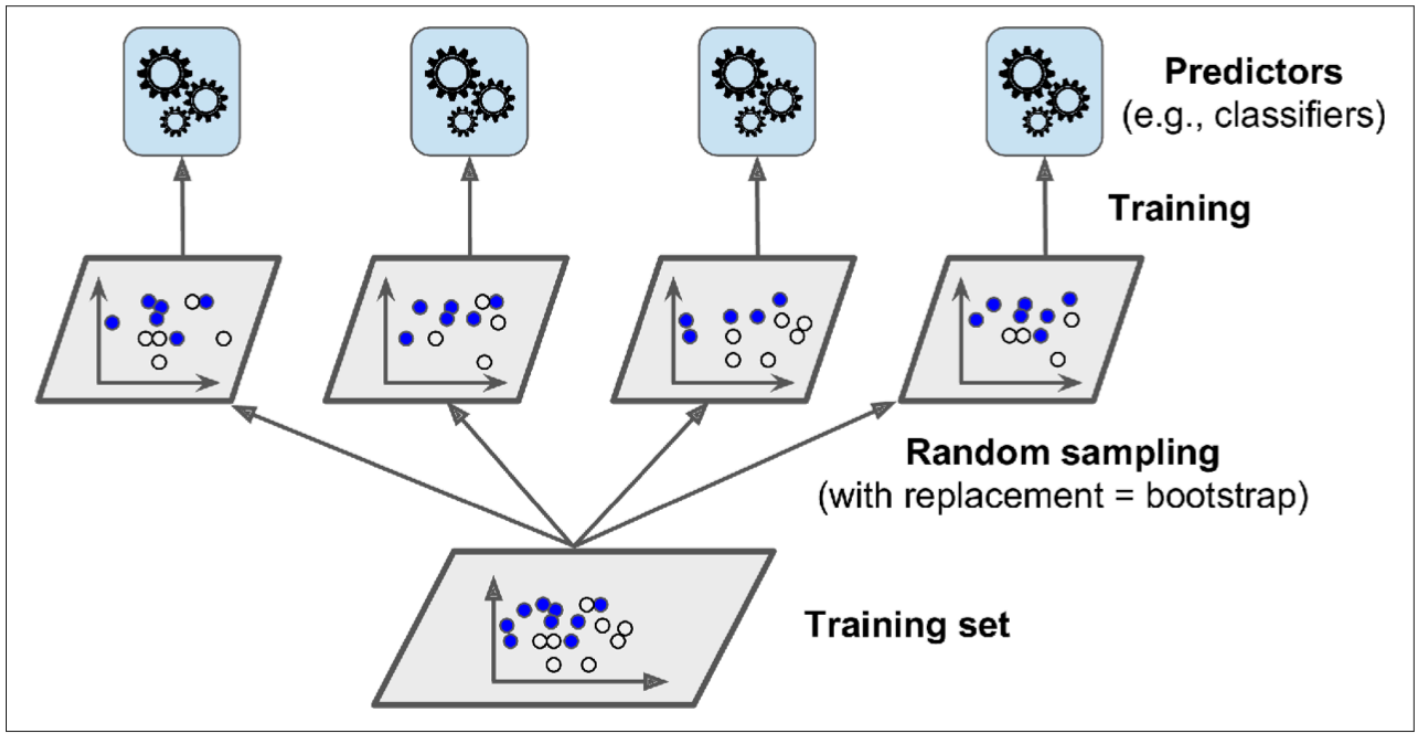
Bootstrapping

- A resample method that consists of repeatedly drawn, with replacement, samples from data to form other smaller datasets.(The observations **can** appear more than one time)



Bagging

- Another approach is to use the same training algorithm for every predictor and train them on different random subsets of the training set.
  - Sampling with replacement is called **Bagging**(short for bootstrap aggregating).
  - Sampling without replacement is called **Pasting**.



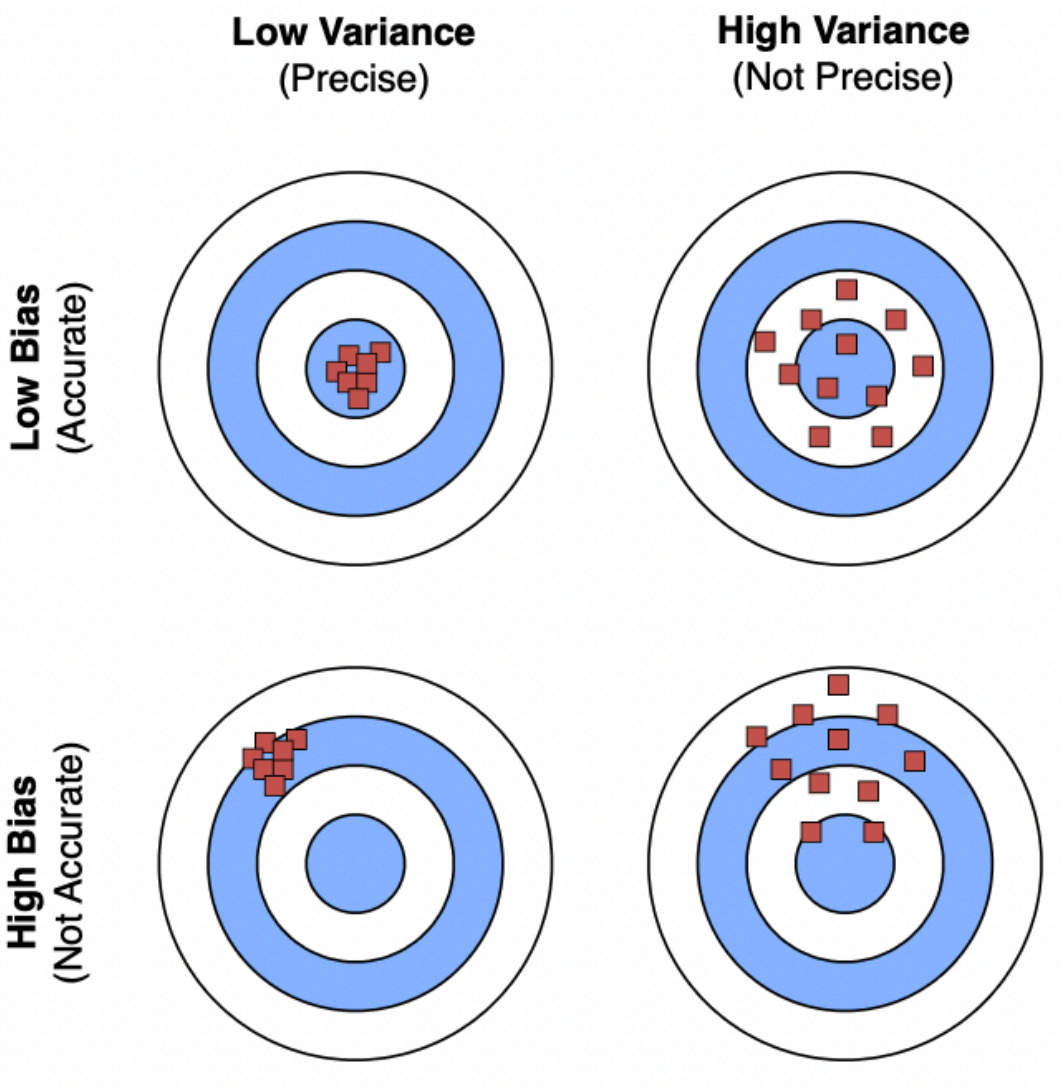
Bagging

- In this approach we generate B different bootstrapped training data sets.

$$\hat{f}_{bag}(x) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}^{*b}(x)$$

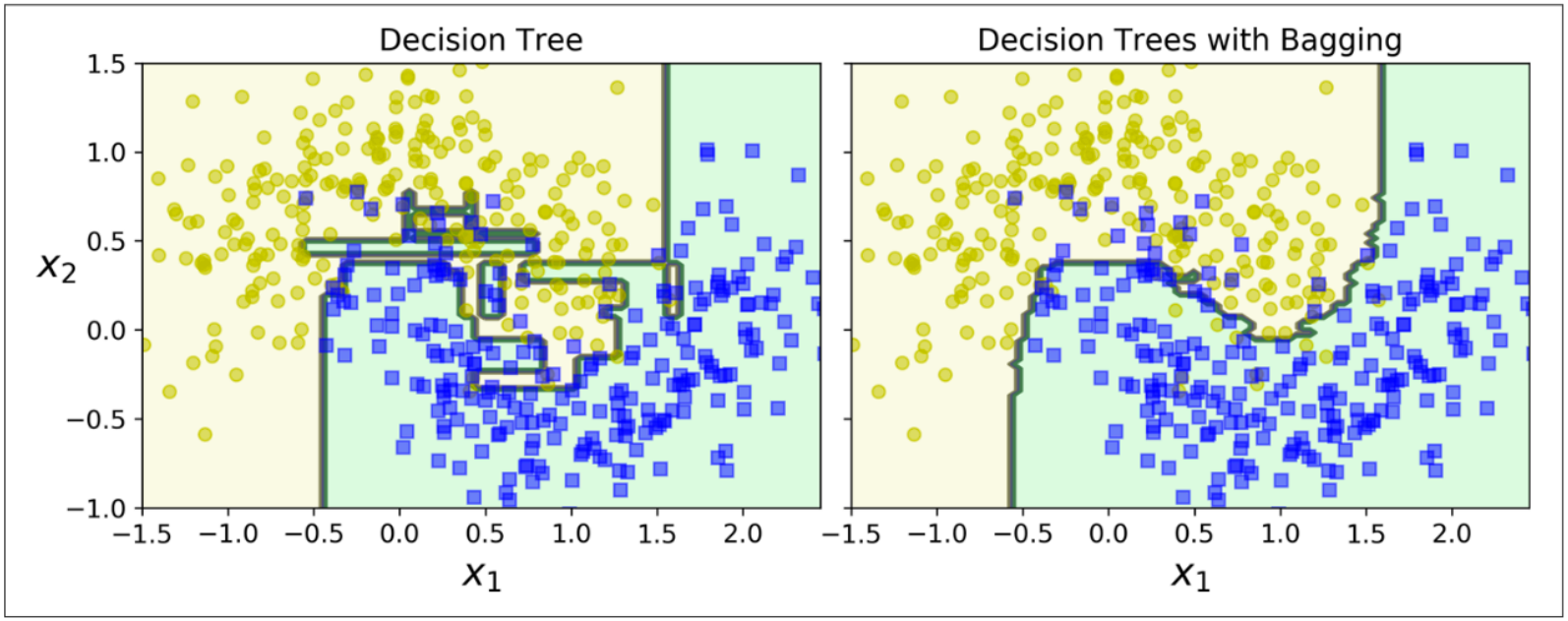
- The aggregation function is typically:
  - Mode(Most frequent) for Classification
  - Average for Regression
- Predictors can all be trained in parallel, via different CPU cores or even different servers. Similarly, predictions can be made in parallel. This is one of the reasons bagging and pasting are such popular methods: they scale very well.

Bias and Variance



Bagging effect on Bias and Variance

- Each individual predictor has a higher bias than if it were trained on the original training set, but aggregation reduces both bias and variance.



- 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.

Out-of-Bag Evaluation

- With bagging, some instances may be sampled several times for any given predictor, while others may not be sampled at all.
- The remaining of the training instances that are not sampled are called **out-of-bag (OOB)** instances.(Note that they are not the same for all predictors.)

$$P(NotChosen) = (1 - \frac{1}{n})^n$$

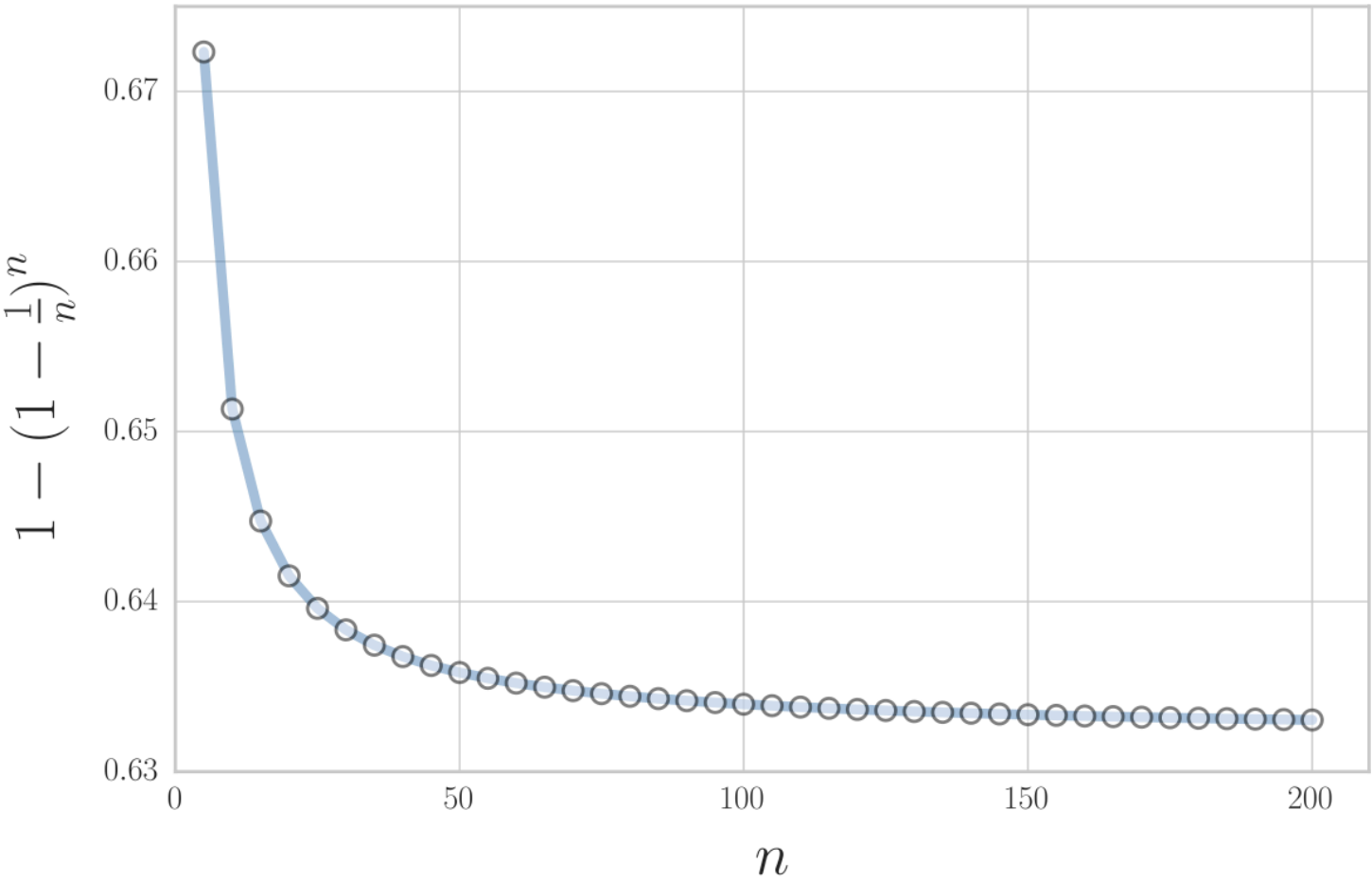
$$\frac{1}{e} \approx 0.368, n \to \infty$$

$$P(Chosen) = 1 - (1 - \frac{1}{n})^n \approx 0.632$$

- This means that when the dataset is big enough, **37%** of its samples are never selected and we could use it to test our model(Since a predictor never sees them).

Out-of-Bag Evaluation

- This means that when the dataset is big enough, **37%** of its samples are never selected and we could use it to test our model(Since a predictor never sees them).

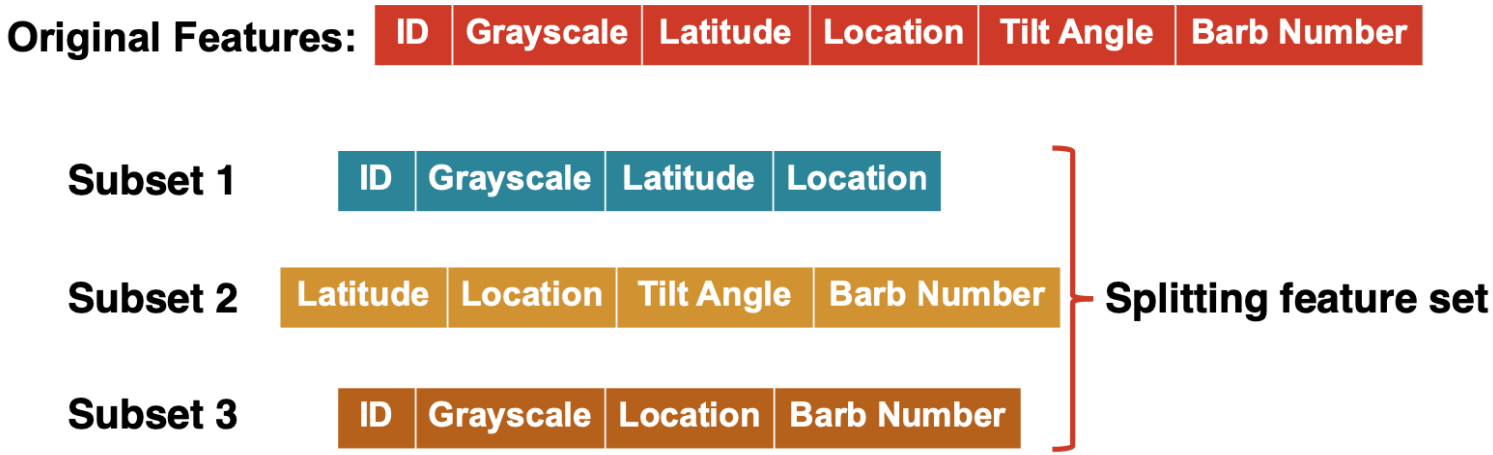


Random Patches and Random Subspaces

- Each predictor will be trained on a random subset of the input **features** instead of instances.
- Sampling both training instances and features is called the **Random Patches** method.
- Keeping all training instances, but sampling features is called the **Random Subspaces** method.
- They are useful when you are dealing with high-dimensional inputs (such as images).
- Sampling features leads to trading a bit more bias for a lower variance.

Random Forests

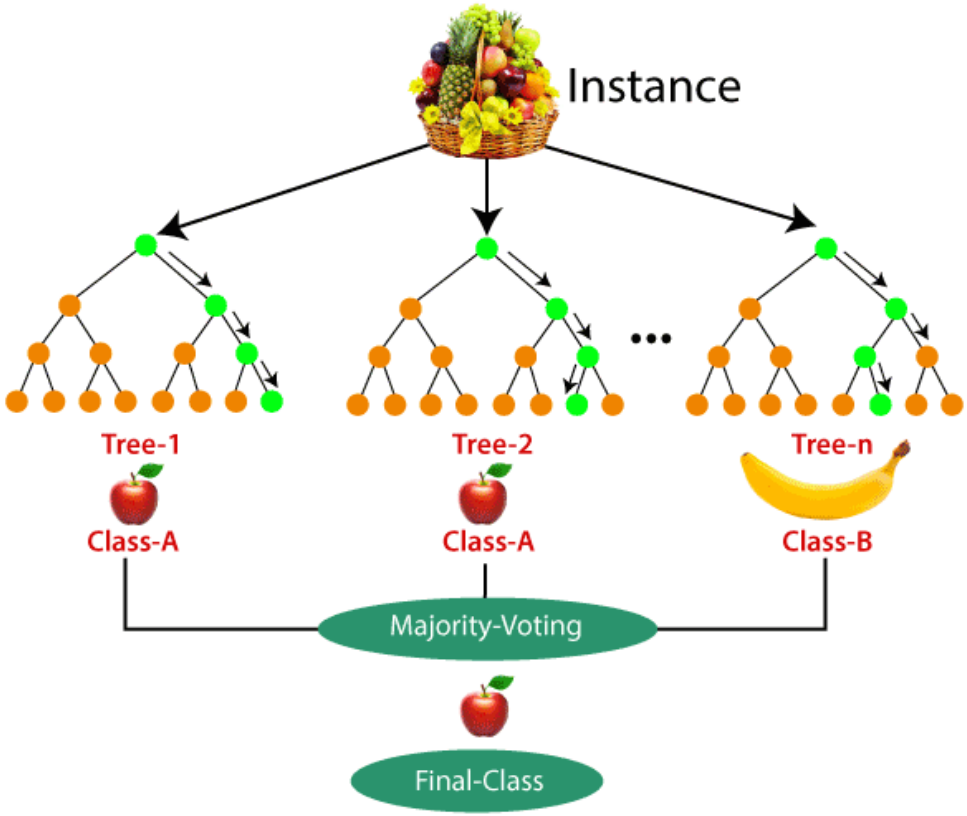
- **Ensemble of Decision Trees**. Generally uses bagging and feature randomness to try to create an uncorrelated forest of trees.
- Instead of searching for the very best feature when splitting a node, it searches for the best feature among a **random subset of features**.



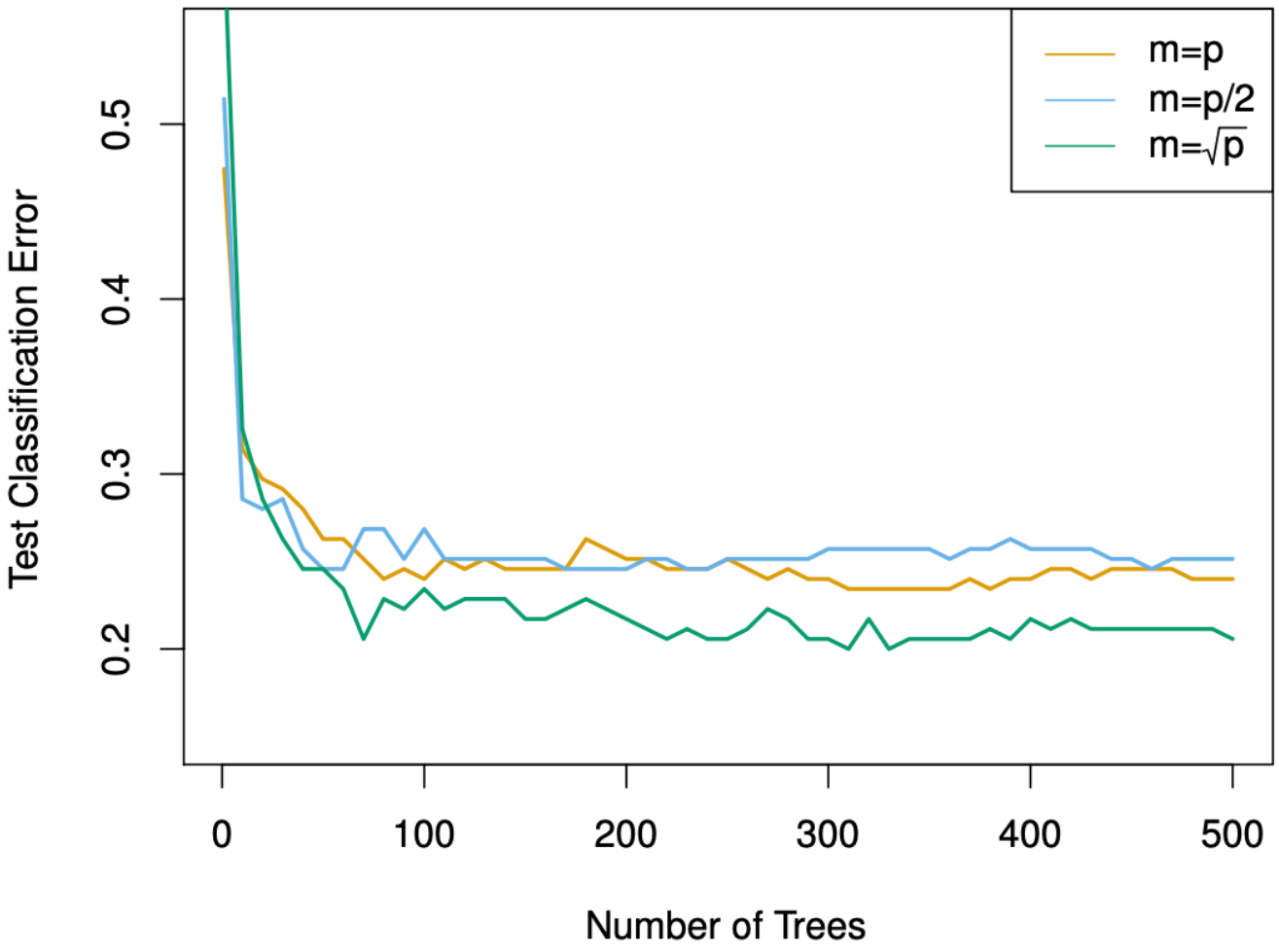
- While the size of the feature subset to consider at each node is a hyperparameter that we can tune, a "rule-of-thumb" suggestion is to use NumFeatures = $log_2\, m + 1$.

Steps of Random Forest

- **Step 1**: In Random forest m number of random records are taken from the data set having p number of records.$(n \approx \sqrt{k})$
- **Step 2**: Individual decision trees are constructed for each sample.
- **Step 3**: Each decision tree will generate an output.
- **Step 4**: Final output is considered based on Majority Voting or Averaging for Classification and regression respectively.



Why $m \approx \sqrt{p}$?



Features of Random Forest

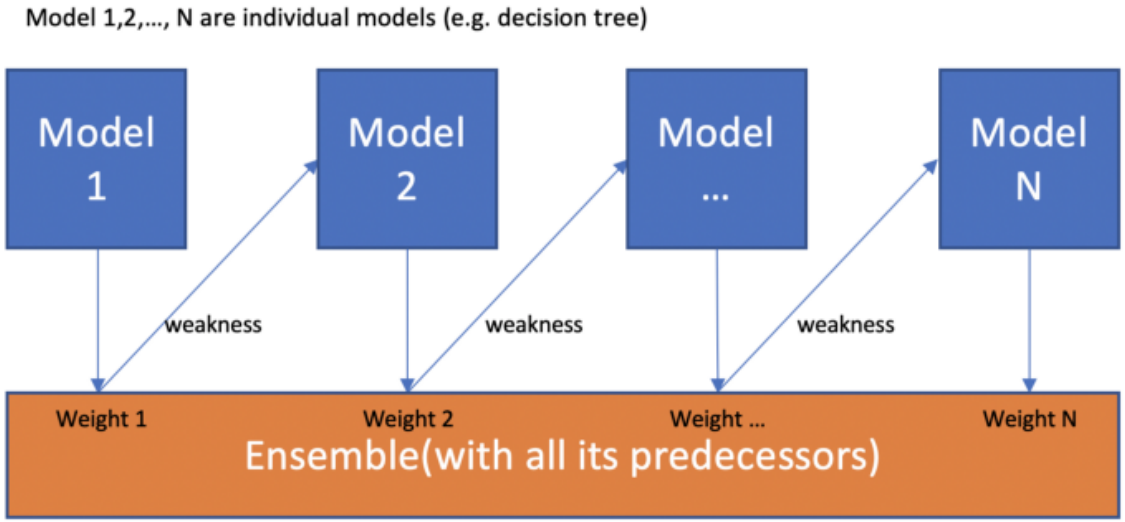
- Pros:
  - Often close to the best performance of any method on the first run
  - Predictions are very fast
  - Feature importance: Measures a feature's importance by looking at how much the tree nodes that use that feature reduce impurity on average (across all trees in the forest).
- Cons:
  - Many parameters: depth of tree, number of trees, type of node tests, random sampling
  - Requires a lot of training data and large memory footprint

Extra-Trees

- It is possible to make trees even more random by also using random thresholds for each feature rather than searching for the best possible thresholds. It is called an **Extremely Randomized Trees ensemble** (or **Extra-Trees** for short).
- This technique trades more bias for a lower variance.
- It also makes Extra-Trees much faster to train than regular Random Forests, since finding the best possible threshold is time-consuming.
- The only way to know which one perform better between Extra-Trees and Random Forest is to try both and compare them using cross-validation (tuning the hyperparameters using grid search).

Boosting

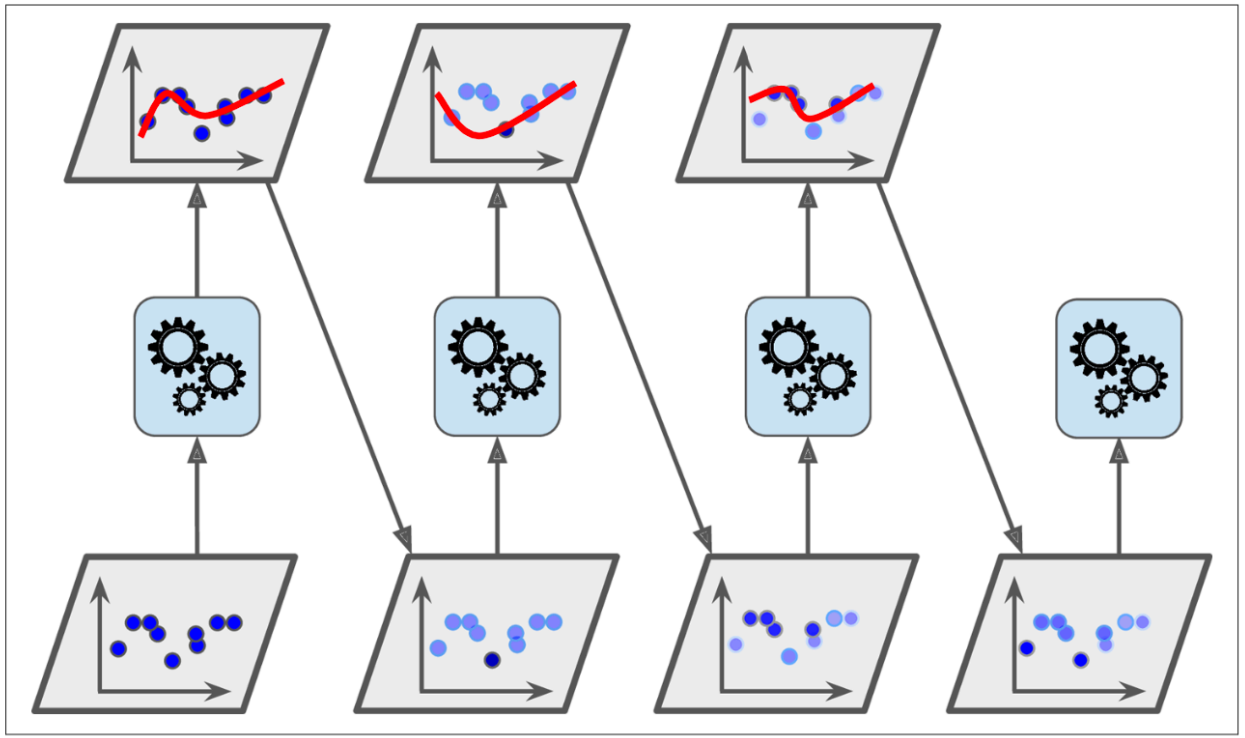
- **Boosting** (originally called hypothesis boosting) refers to any Ensemble method that can combine several weak learners into a strong learner.
- The model is weak if it has a substantial error rate, but the performance is not random.
- The general idea of most boosting methods is to train predictors sequentially, each trying to correct its predecessor.



- Most popular boosting methods(Differ in terms of how weights are updated & classifiers are combined):
  - Adaptive Boosting(AdaBoost)
  - Gradient Boosting(LightGBM, XGBoost)

AdaBoost

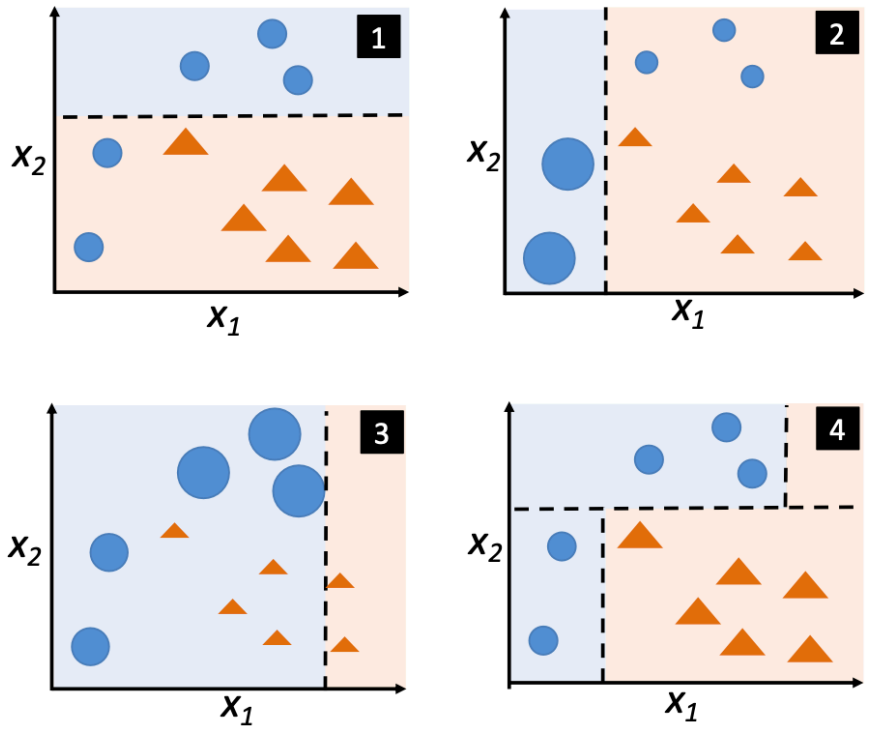
- One way for a new predictor to correct its predecessor is to pay a bit more attention to the training instances that the predecessor underfitted.
- This results in new predictors focusing more and more on the hard cases.



- **Drawback of AdaBoost**: It doesn't scale as well as bagging(or pasting).

AdaBoost Steps

- First trains a base classifier and uses it to make predictions
- Then increases the relative weight of misclassified instances
- Then it trains a second classifier, using the updated weights, and again makes predictions and so on...



- Once all predictors are trained, the ensemble makes predictions very much like bagging, except that predictors have different weights.

AdaBoost Algorithm

- Each instance weight $w^{(i)}$ is initially set to $1/m$ and first predictor's weighted error rate is r_1.
- Weighted error rate of the j^{th} predictor:

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

where $\hat{y}_j^{(i)}$ is the j^{th} predictor's prediction for the i^{th} instance.

- The predictor's weight:

$$\alpha_j = \eta \, Log \, \frac{1 - r_j}{r_j}$$

where η is the learning rate.

  - The more accurate the predictor is, the higher its weight will be.
  - If it is just guessing randomly, then its weight will be close to zero.
  - However, if it is most often wrong, then its weight will be negative.

AdaBoost Algorithm

- Next, the AdaBoost algorithm update the instance weights, which boosts the weights of the misclassified instance:

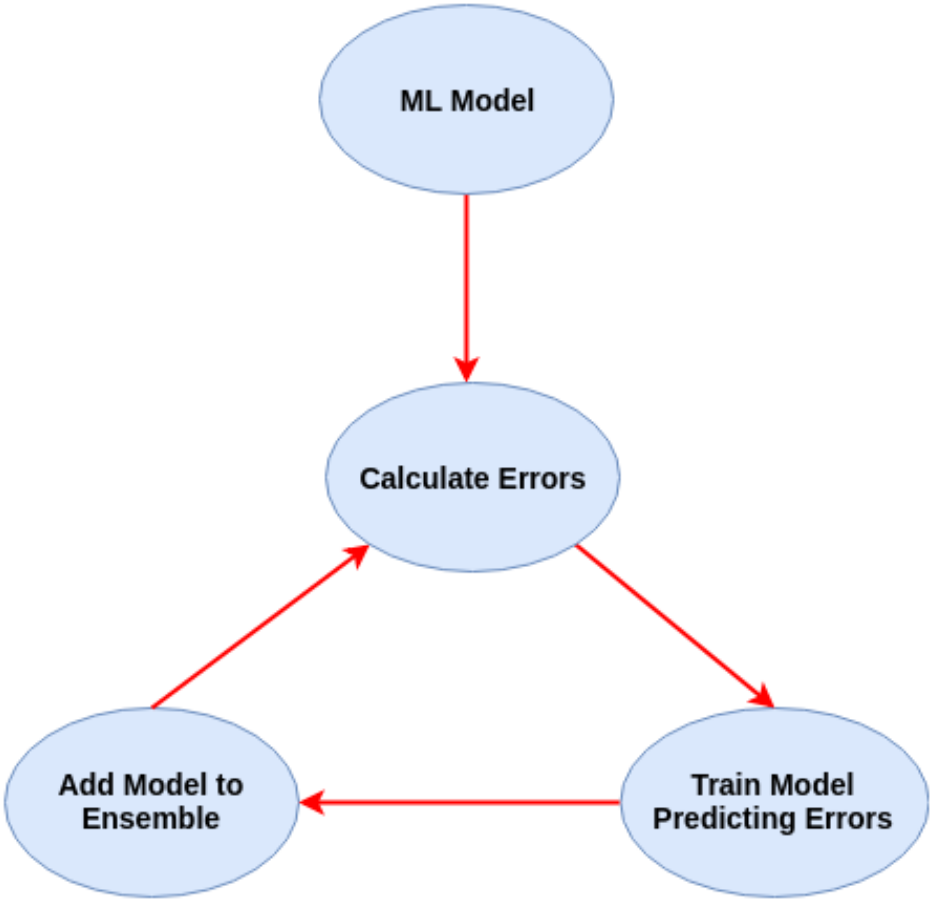
$$w^{(i)} \leftarrow \begin{cases} w^{(i)} & \text{if } \hat{y}_j^{(i)} = y^{(i)} \\ w^{(i)}\, exp(\alpha_j) & \text{if } \hat{y}_j^{(i)} \neq y^{(i)} \end{cases}$$

  Then all the instance weights gets normalized by dividing by $\sum_{i=1}^{m} w^{(i)}$.
- Finally, a new predictor is trained using the updated weights, and the whole process is repeated.
- The algorithm stops when the desired number of predictors is reached, or when a perfect predictor is found.
- AdaBoost preidctions:

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

Gradient Boosting

- In this approach, 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.



Gradient Boosting Steps

X_0= ID	X_1= Grayscale	X_2= Length	X_3= Barb Number	Y= Latitude
1	200	500	10	60
2	185	450	8	70
3	145	620	12	65
4	195	150	2	30

- Construct base tree(just the root node):

$$y_1^* = \frac{1}{n}\sum_{i=1}^{n} y^i = 56.25$$

Gradient Boosting Steps

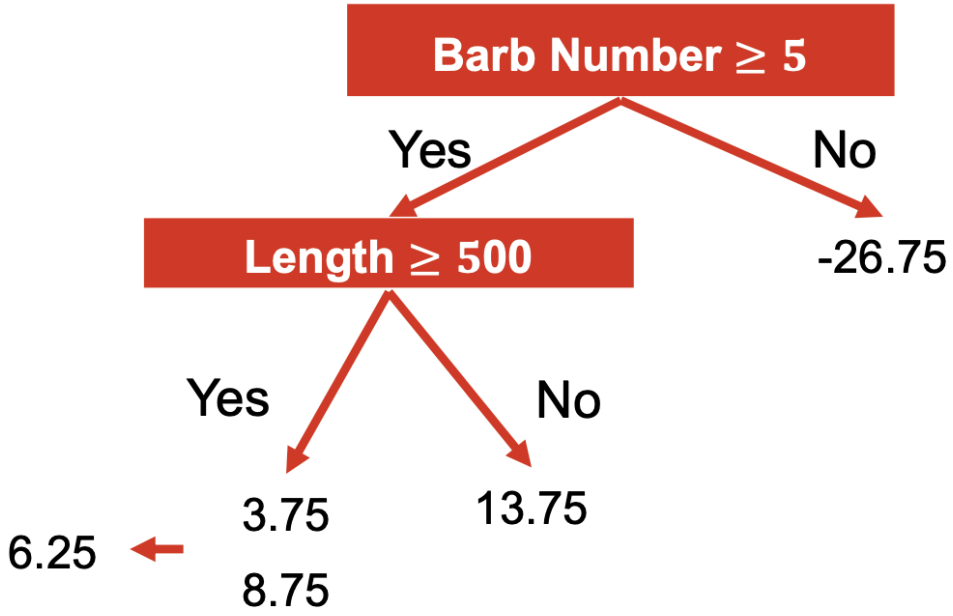
- Build next tree based on errors of the previous tree:

$$r_1 = y_1 - y_1^*$$

X_0= ID	X_1= Grayscale	X_2= Length	X_3= Barb Number	Y= Latitude	r_1= residual
1	200	500	10	60	60-56.25=3.75
2	185	450	8	70	70-56.25=13.75
3	145	620	12	65	65-56.25=8.75
4	195	150	2	30	30-56.75=-26.75

Gradient Boosting Steps

- Then, create a tree based on $x_1, x_2, \ldots, x_m$ to fit the residuals:



- Combine tree from step 1 with tree:

$$\text{Predict ID=3:} 56.25 + \alpha * 6.25$$

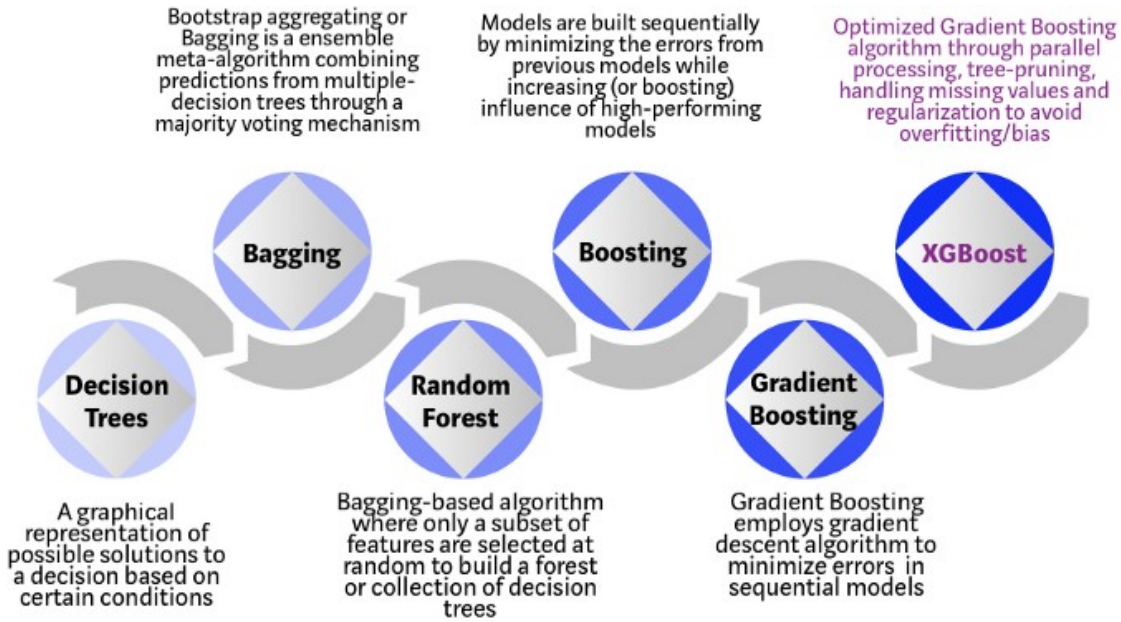
Where α learning rate between 0 and 1(if $\alpha = 1$, low bias but high variance)

Gradient Boosting Algorithm

Gradient Boosting Algorithm

XGBoost

- **XGBoost**, which stands for Extreme Gradient Boosting, is a scalable, distributed gradient-boosted decision tree (GBDT) machine learning library.
- It provides parallel tree boosting and is the leading machine learning library for regression, classification, and ranking problems.
- Evolution of tree-based algorithms:



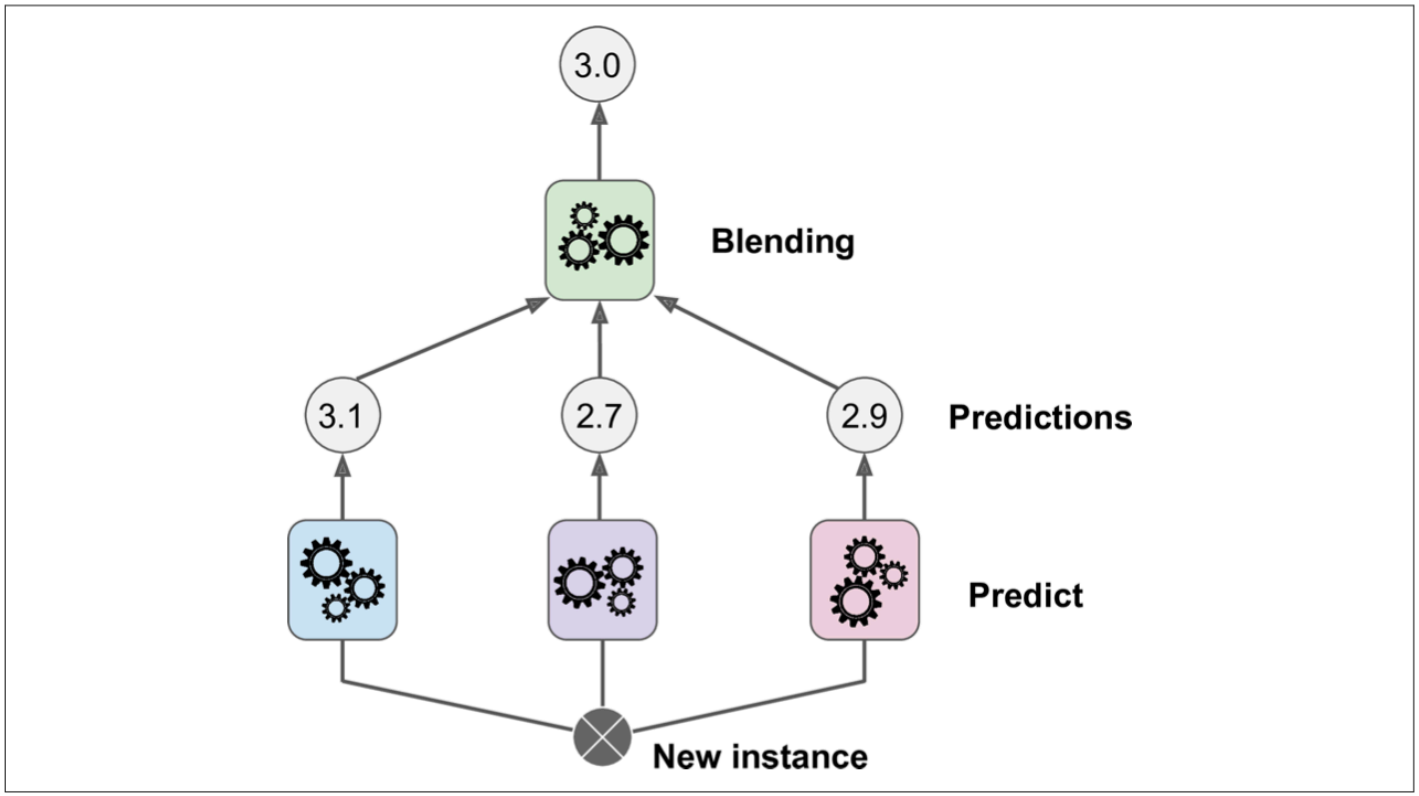
XGBoost

- XGBoost uses pre-sorted algorithm & Histogram-based algorithm for computing the best split.
- **Pre-Sorting splitting**
  - For each node, enumerate over all features
  - For each feature, sort the instances by feature value
  - Use a linear scan to decide the best split along that feature basis information gain
  - Take the best split solution along all the features
- In simple terms, Histogram-based algorithm splits all the data points for a feature into discrete bins and uses these bins to find the split value of histogram.

CatBoost

Stacking

- **Blender** or **Meta-learner** or **Meta-classifier** takes predictions of each predictor as inputs and makes the final prediction. Basically we train a model to perform aggregation.

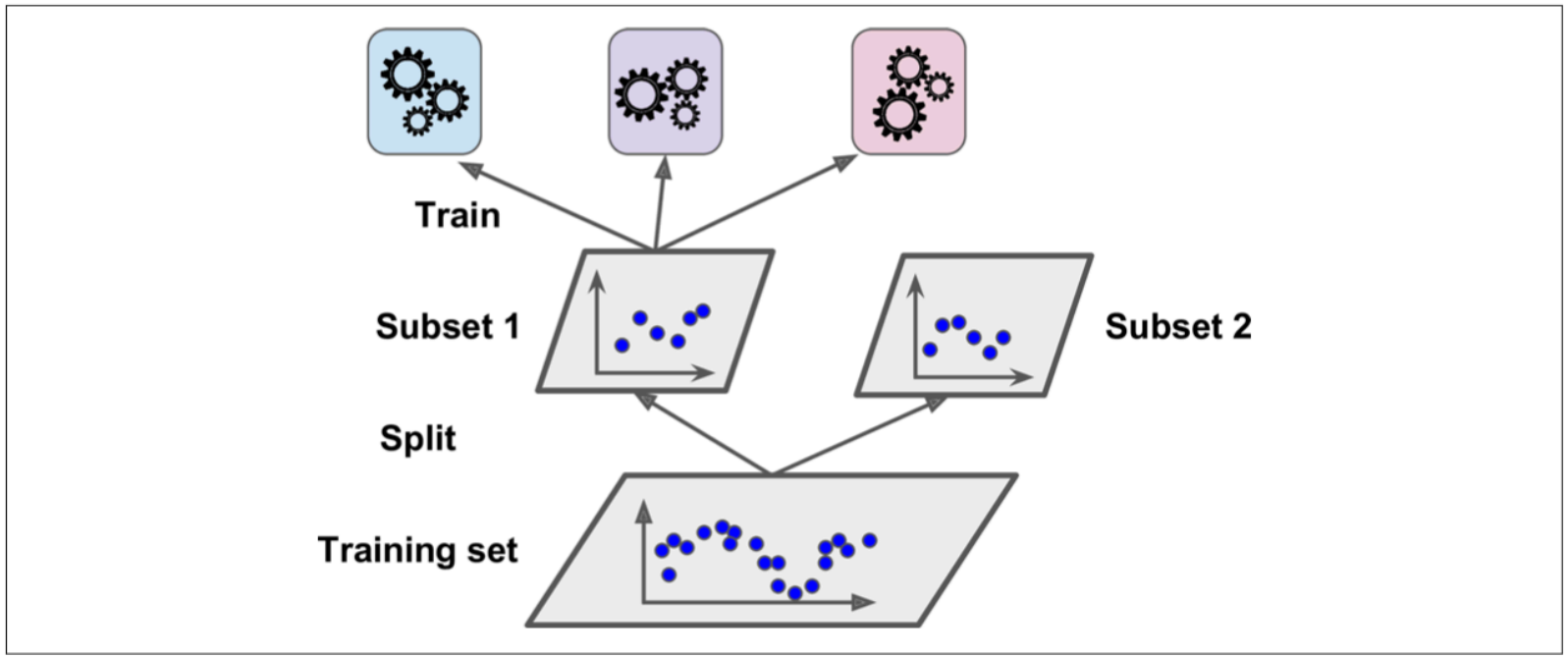


Features of Stacking

- Usually has weak-learners of different types.
- Meta-learner tries to learn which weak-learner is more important.
- The weak-learners are trained in **parallel**, but the meta learner is trained **sequentially**.
- Once the weak-learners are trained, their weights are kept static to train the meta-learner.
- Usually, the meta-learner is trained on a different subset than what was used to train the weak-learners.
- It is common to use a linear model for aggregation to avoid complexity.

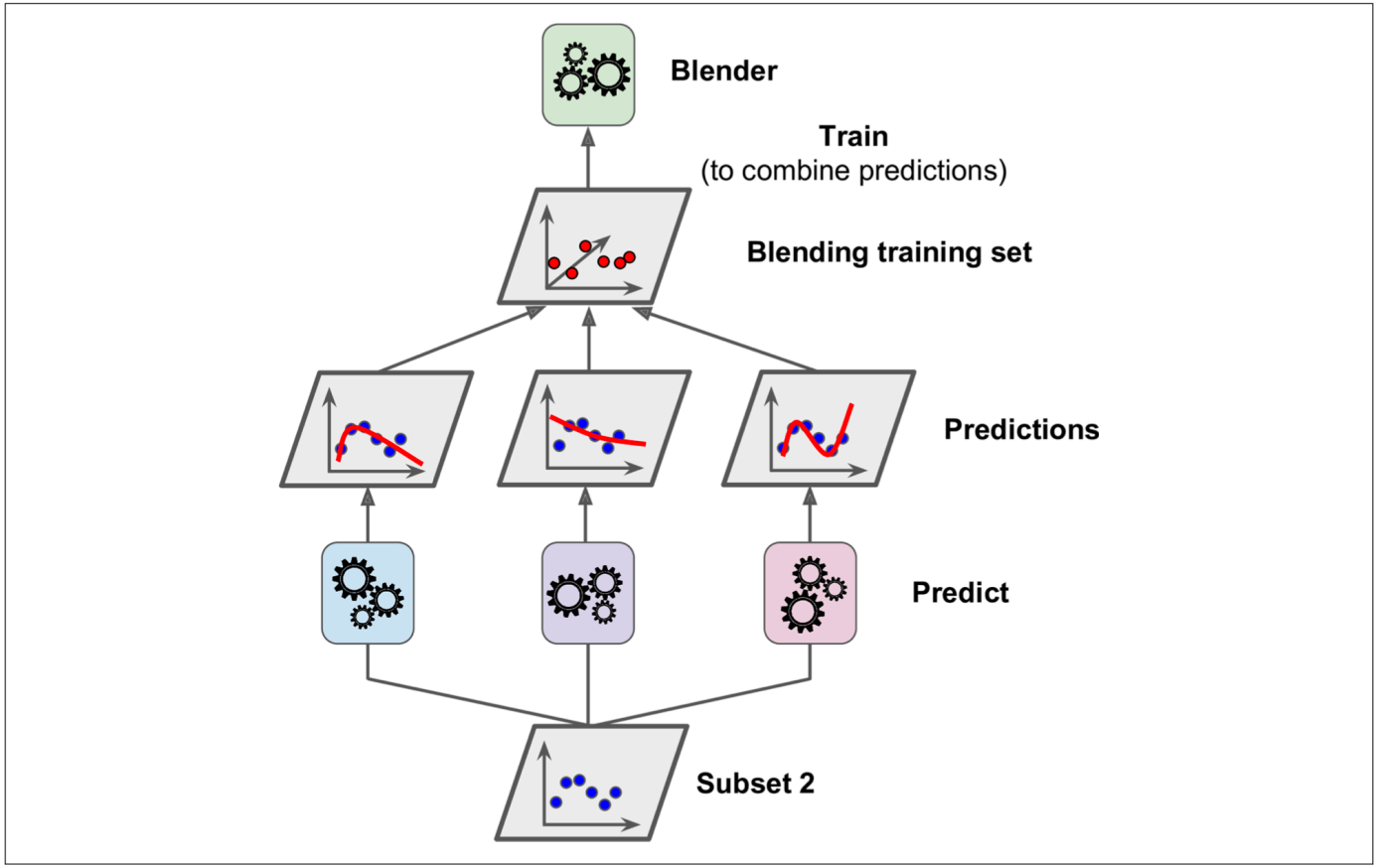
Training Blender

- To train the blender, a common approach is to use a **hold-out set**.
- First, the training set is split into two subsets. The first subset is used to train the predictors in the first layer.



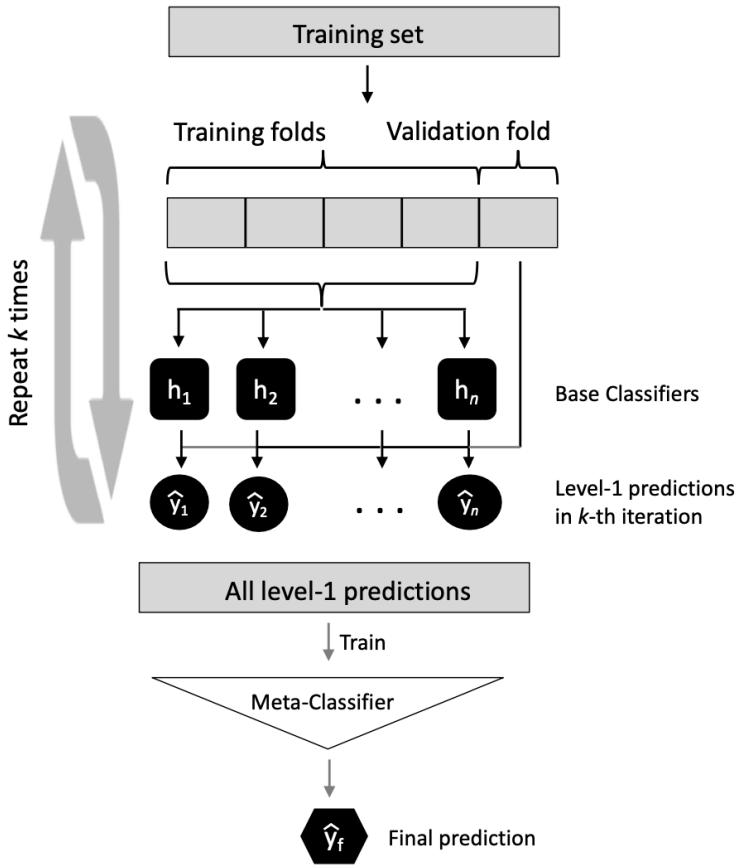
Training Blender

- Next, the first layer's predictors are used to make predictions on the second (held-out) set.
- Then we train the blender on this new training set, so it learns to predict the target value, given the first layer's predictions.



Stacking with Cross-Validation

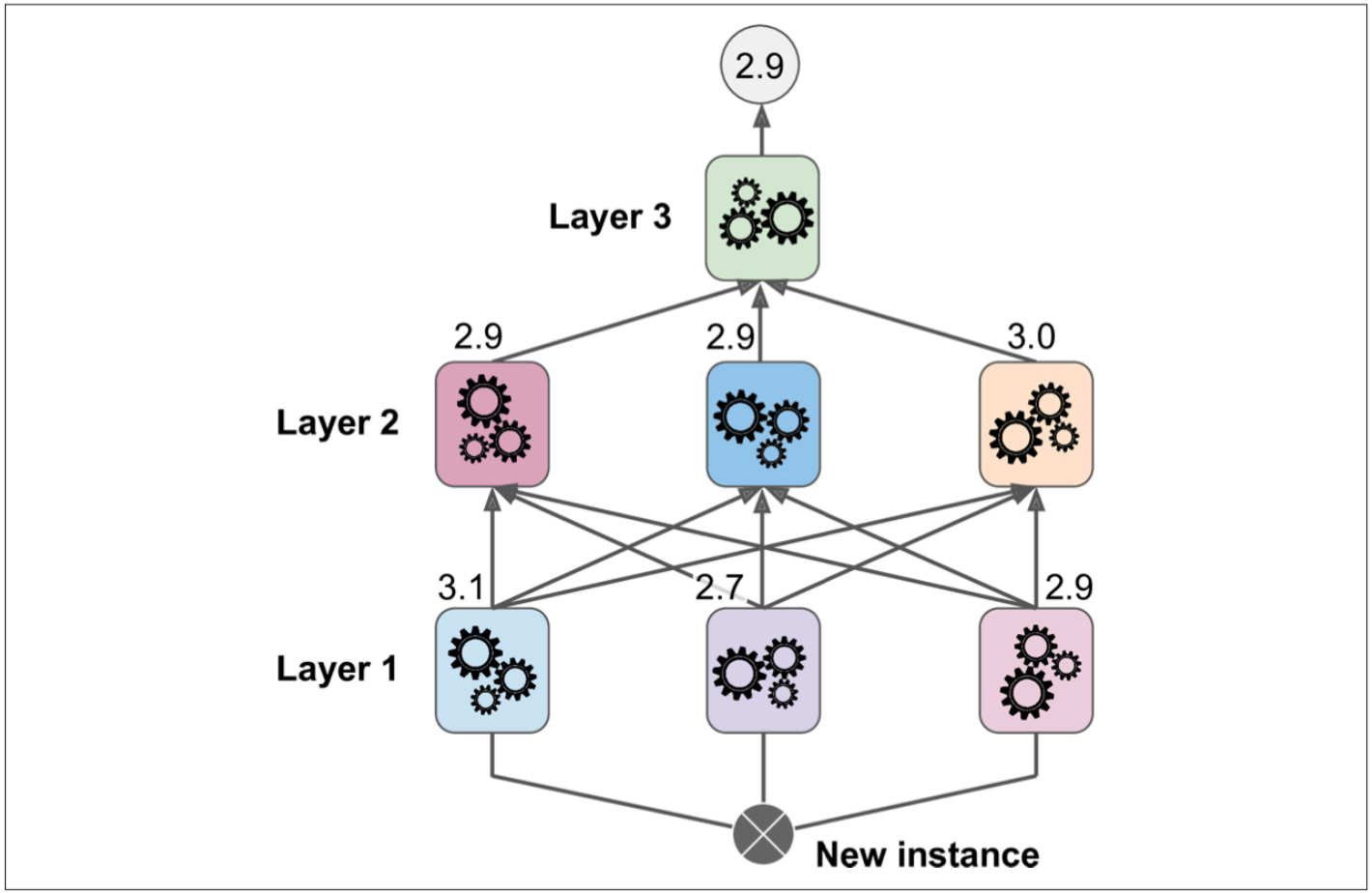
- The problem with this method is that it has a high tendency to suffer from extensive overfitting. A better alternative would be to use stacking with k-fold cross-validation or leave-one-out cross-validation.



Multilayer Stacking

- It is possible to train several different blenders, to get a whole layer of blenders.
  - 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)
  - 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.

Multilayer Stacking



Key Elements

- Bagging
  - Bootstrap samples of the training dataset
  - Unpruned decision trees fit on each sample
  - Simple voting or averaging of predictions
- Stacking
  - Unchanged training dataset
  - Different machine learning algorithms for each ensemble member
  - Machine learning model to learn how to best combine predictions
- Boosting
  - Bias training data toward those examples that are hard to predict
  - Iteratively add ensemble members to correct predictions of prior models
  - Combine predictions using a weighted average of models

Parameter	Bagging	Boosting	Stacking
Focuses on	Reducing variance	Reducing bias	Improving accuracy
Nature of weak learners is	Homogenous	Homogenous	Heterogenous
Weak learners are aggregated by	Simple voting	Weighted voting	Learned voting (meta-learner)

Thank You!

Any Question?