Machine Learning
DSECL ZG565

Ensemble Learning

- Dr. Monali Mavani

BITS Pilani
Pilani Campus

These slides are prepared by the instructor, with grateful acknowledgement of Tom Mitchell, Andrew Ng< Prof. Chetana, Prof. Vimal and many others who made their course materials freely available online.

Contents

- Combining classifiers
- Bagging
- Boosting
- Random Forest Algorithm
- AdaBoost Algorithm
- Gradient Boosting

The Single Model Philosophy

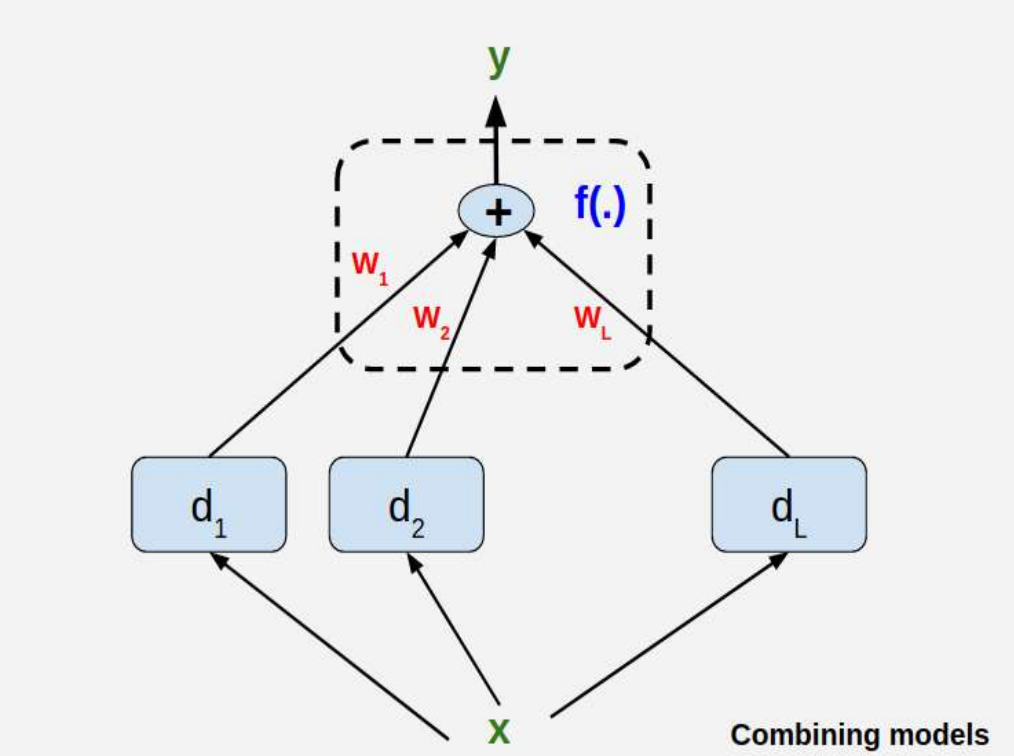
- Motivation: Occam's Razor
  - "one should not increase, beyond what is necessary, the number of entities required to explain anything"
- Infinitely many models can explain any given dataset
  - Might as well pick the smallest one…

Ensemble Philosophy

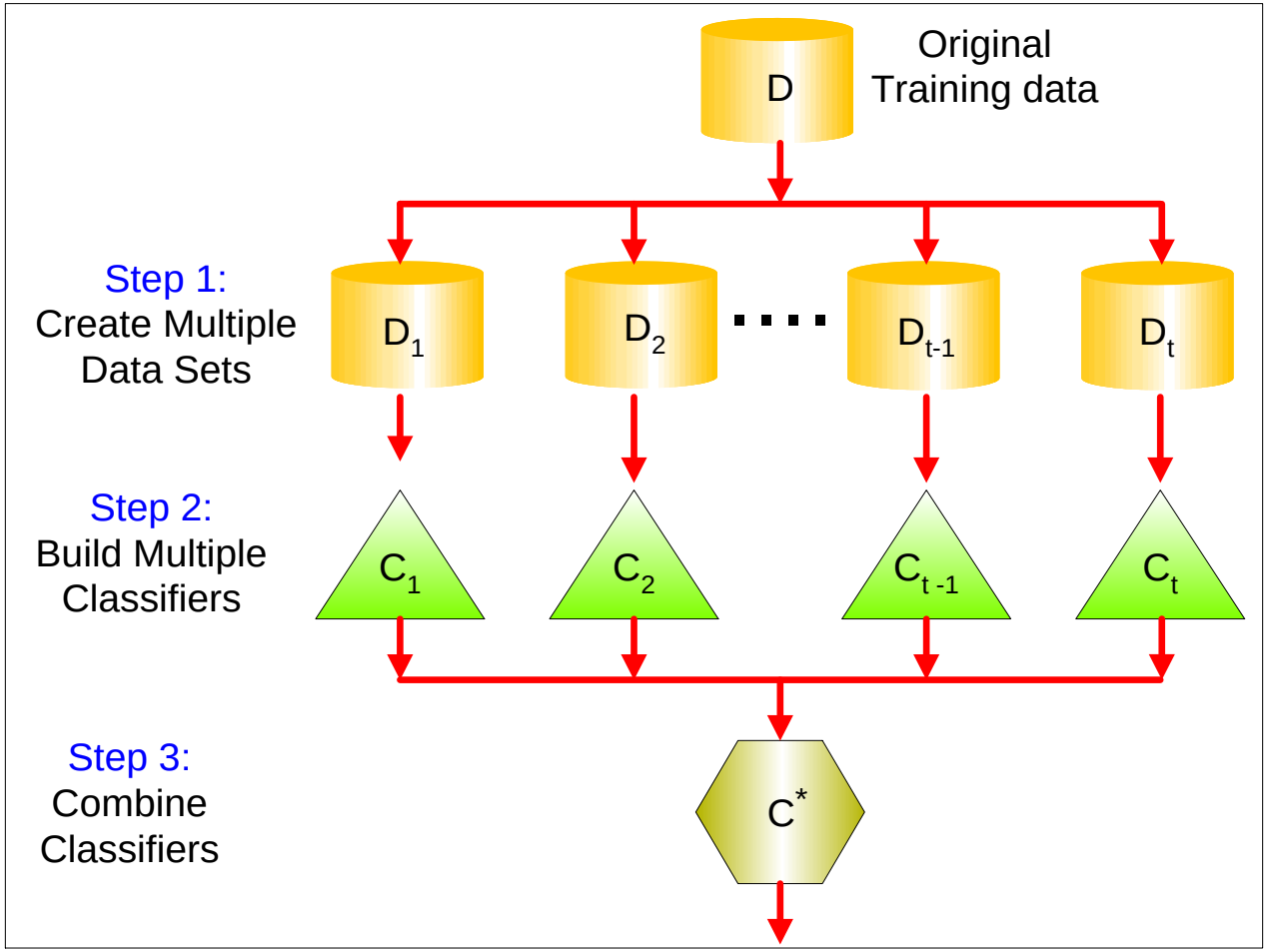
- No Free Lunch Theorem: There is no algorithm that is always the most accurate
- Each learning algorithm dictates a certain model that comes with a set of assumptions
  - Each algorithm converges to a different solution and fails under different circumstances
    - The best tuned learners could miss some examples and there could be other learners which works better on (may be only) those !
  - In the absence of a single expert (*a superior model*) , a committee (*combinations of models*) can do better !
    - A committee can work in many ways ...

Committee of Models

- Committee Members are base learners !
- Major challenges dealing with this committee
  - Expertise of each of the members (Does it help / not?)
  - Combining the results from the members for better performance

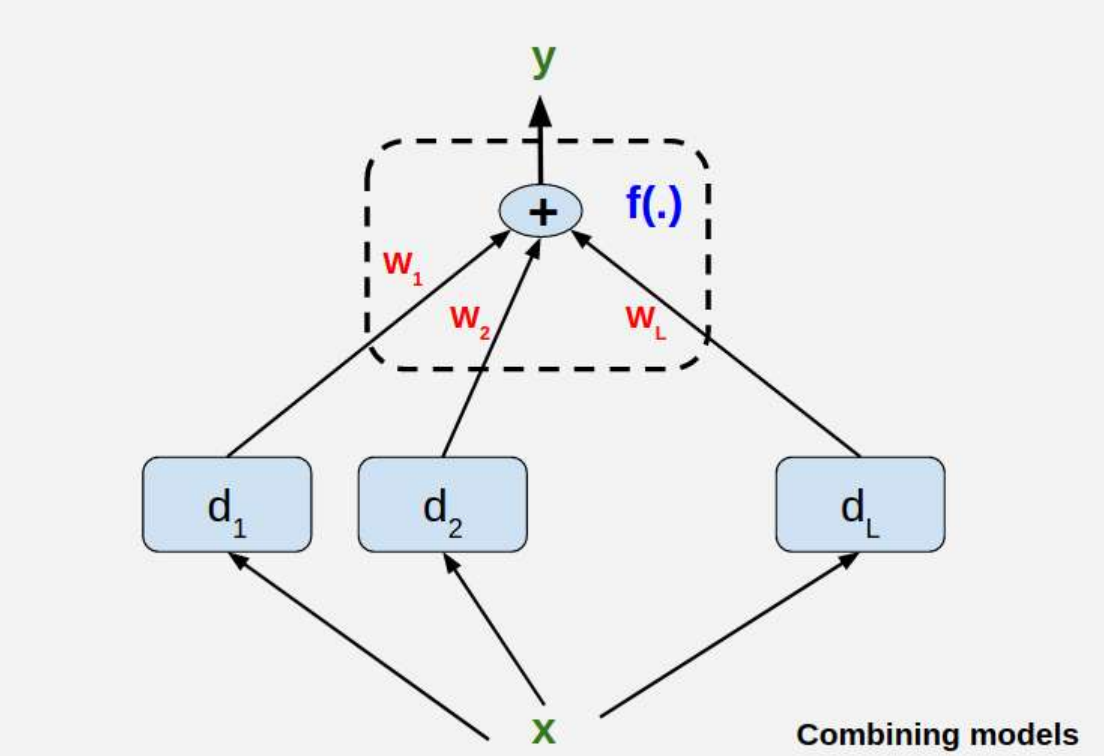


General Approach



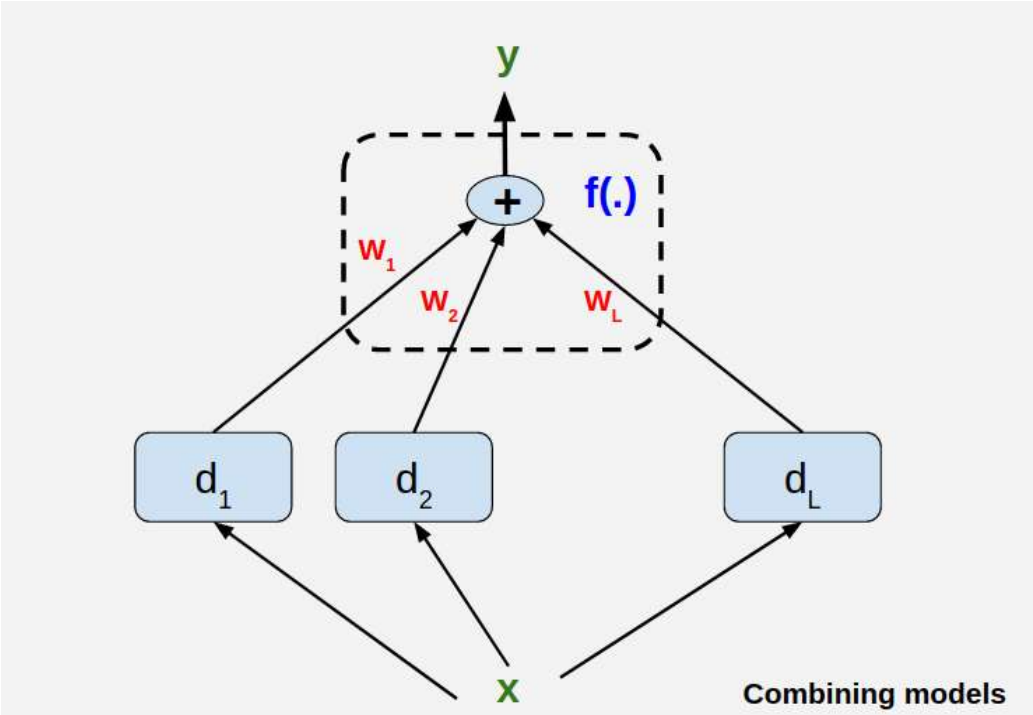
Issue 1 : On the members (Base Learners)

- It does not help if all learners are good/bad at roughly same thing
  - Need Diverse Learners



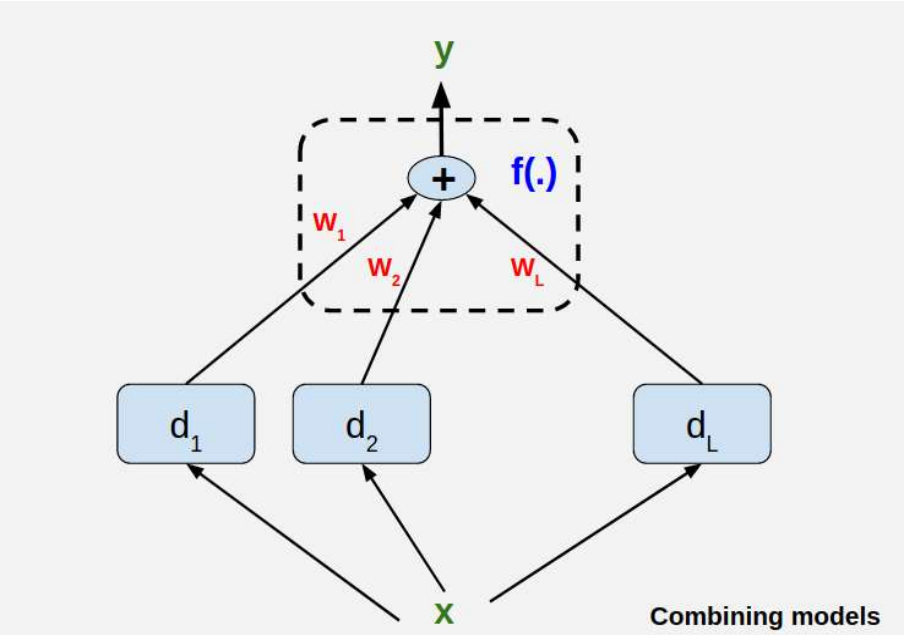
Issue 1 : On the members (Base Learners)

- Use Different Algorithms
  - Different algorithms make different assumptions
- Use Different Hyperparameters,
  - E.g vary the structure of neural nets

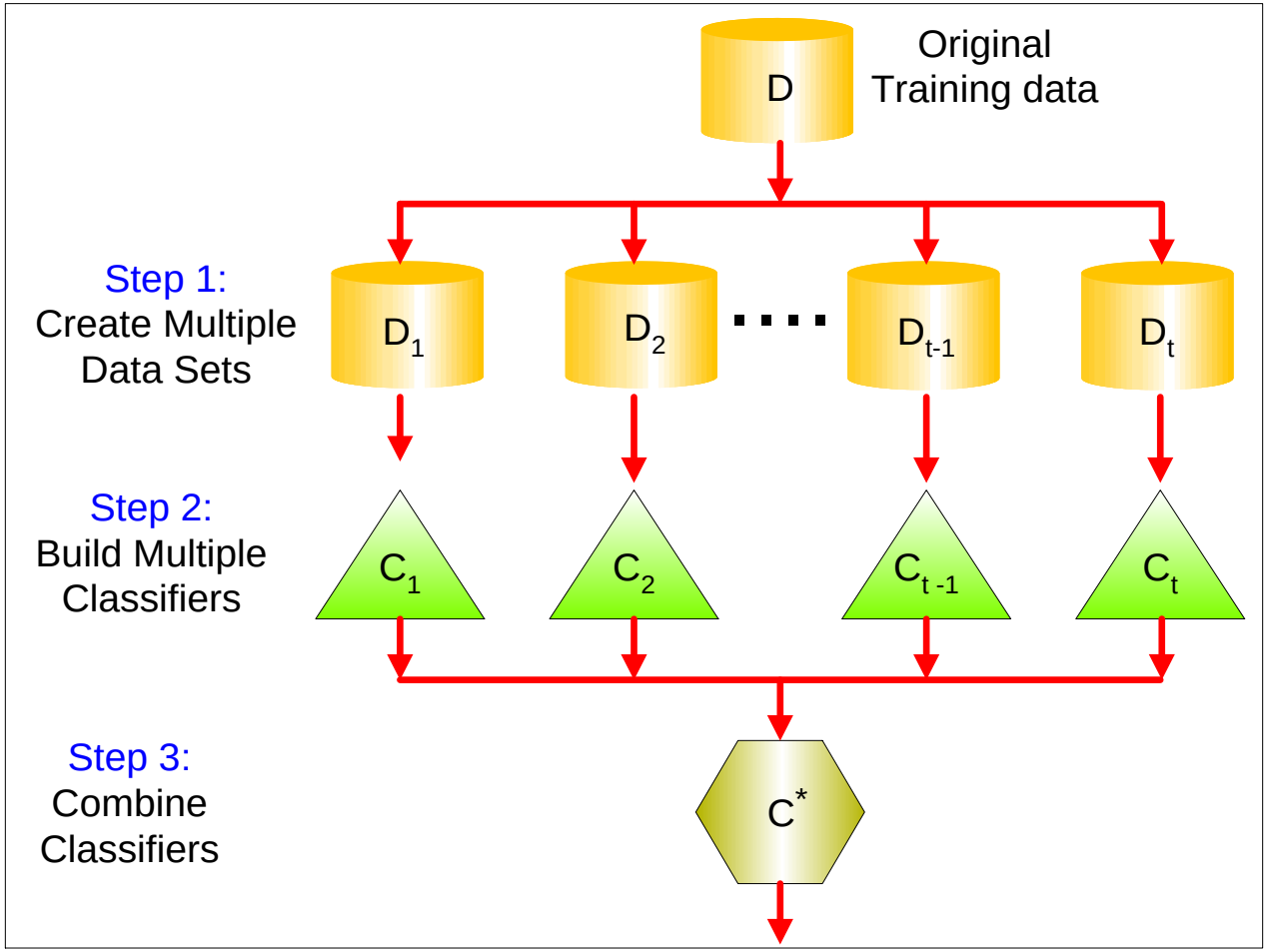


Issue 1 : On the members (Base Learners)

- Different input representations
  - Uttered words + video information of speakers clips
  - image + text annotations
- Different training sets
  - Draw different random samples of data
  - Partition data in the input space and have learners specialized in those spaces (mixture of experts)



General Approach



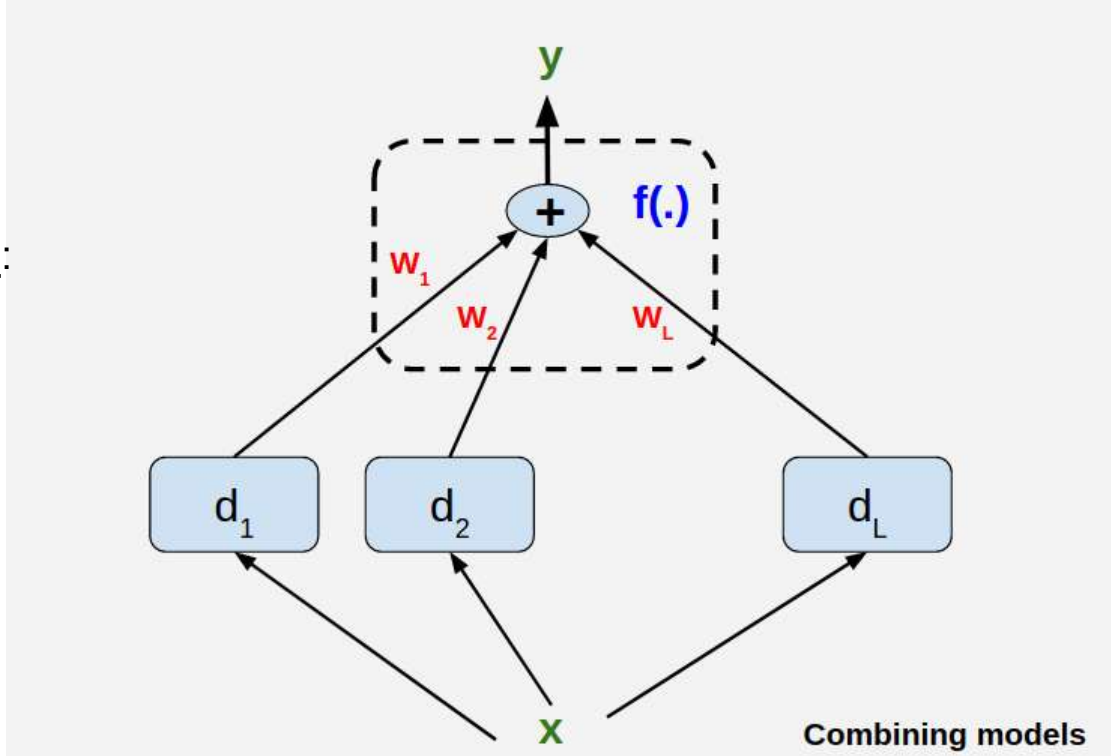
Issue -2 : Combining Results

$$y = f(d_1, d_2, \ldots, d_L | \Phi)$$

A Simple Combination Scheme:

$$y = \sum_{j=1}^{L} w_j d_j$$

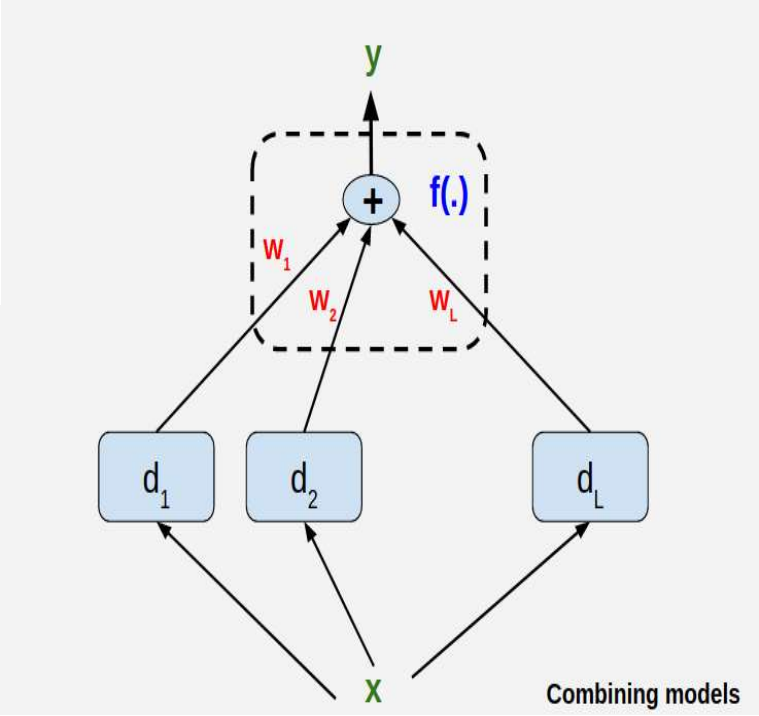
$$w_j \geq 0 \text{ and } \sum_{j=1}^{L} w_j = 1$$



Issue -2 : Combining Results

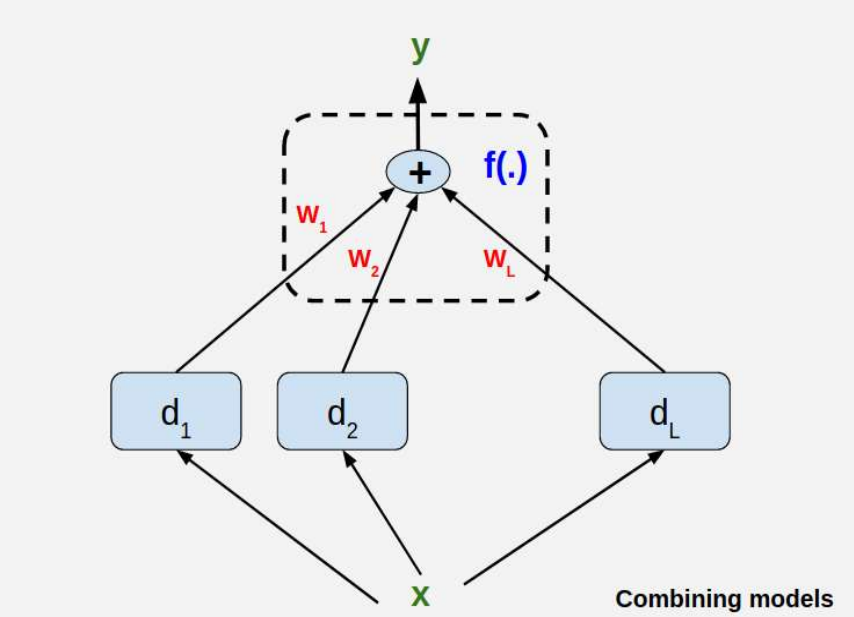
$$y = f(d_1, d_2, \ldots, d_L | \Phi)$$

Rule	Fusion function $f(\cdot)$
Sum	$y_i = \frac{1}{L}\sum_{j=1}^{L} d_{ji}$
Weighted sum	$y_i = \sum_j w_j d_{ji}, w_j \geq 0, \sum_j w_j = 1$
Median	$y_i = \text{median}_j d_{ji}$
Minimum	$y_i = \min_j d_{ji}$
Maximum	$y_i = \max_j d_{ji}$
Product	$y_i = \prod_j d_{ji}$



Issue -2 : Combining Results

	C_1	C_2	C_3
d_1	0.2	0.5	0.3
d_2	0.0	0.6	0.4
d_3	0.4	0.4	0.2
Sum	0.2	**0.5**	0.3
Median	0.2	**0.5**	0.4
Minimum	0.0	**0.4**	0.2
Maximum	0.4	**0.6**	0.4
Product	0.0	**0.12**	0.032

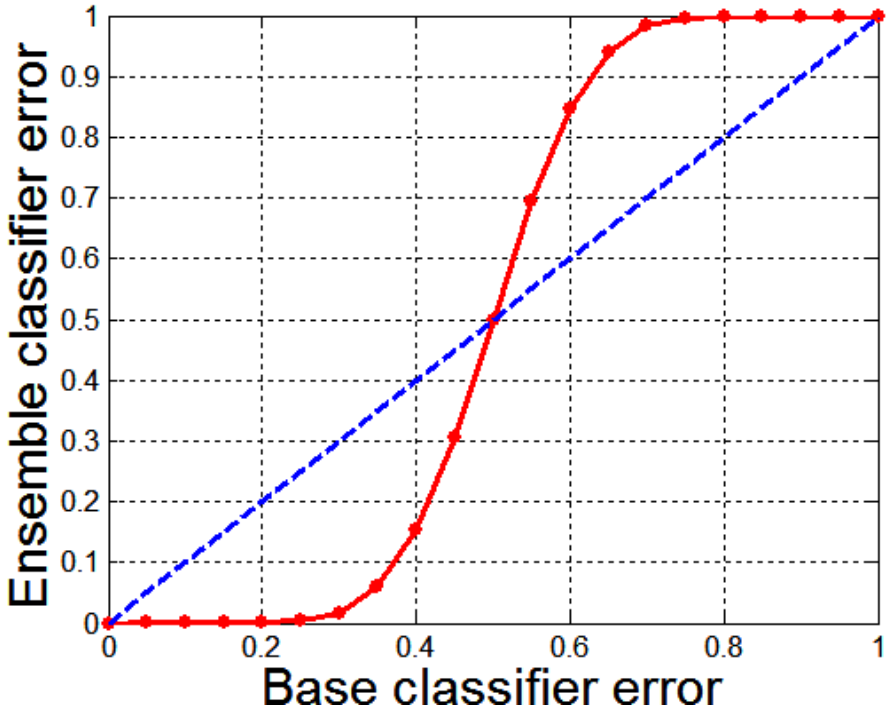


When does Ensemble work?

- Ensemble classifier performs better than the base classifiers when e is smaller than 0.5
- Necessary conditions for an ensemble classifier to perform better than a single classifier:
  - Base classifiers should be independent of each other
  - Base classifiers should do better than a classifier that performs random guessing

Why Ensemble Methods work?

- 25 base classifiers
- Each classifier has error rate, $\varepsilon = 0.35$
- If base classifiers are identical, then the ensemble will misclassify the same examples predicted incorrectly by the base classifiers depicted by dotted line
- Assume errors made by classifiers are uncorrelated
- Ensemble makes a wrong prediction only if base classifiers error is more than 0.5



Types of Ensemble Methods

Manipulate data distribution

- Example: bagging, boosting

Manipulate input features

- Example: random forests

Bagging (Bootstrap Aggregating)

Main Assumption:

- Combining many unstable predictors to produce a ensemble (stable) predictor.
- Unstable Predictor: small changes in training data produce large changes in the model.
  - e.g. Neural Nets, trees
  - Stable: SVM (sometimes), Nearest Neighbor.

Hypothesis Space

- Variable size (nonparametric):
  - Can model any function if you use an appropriate predictor (e.g. trees)

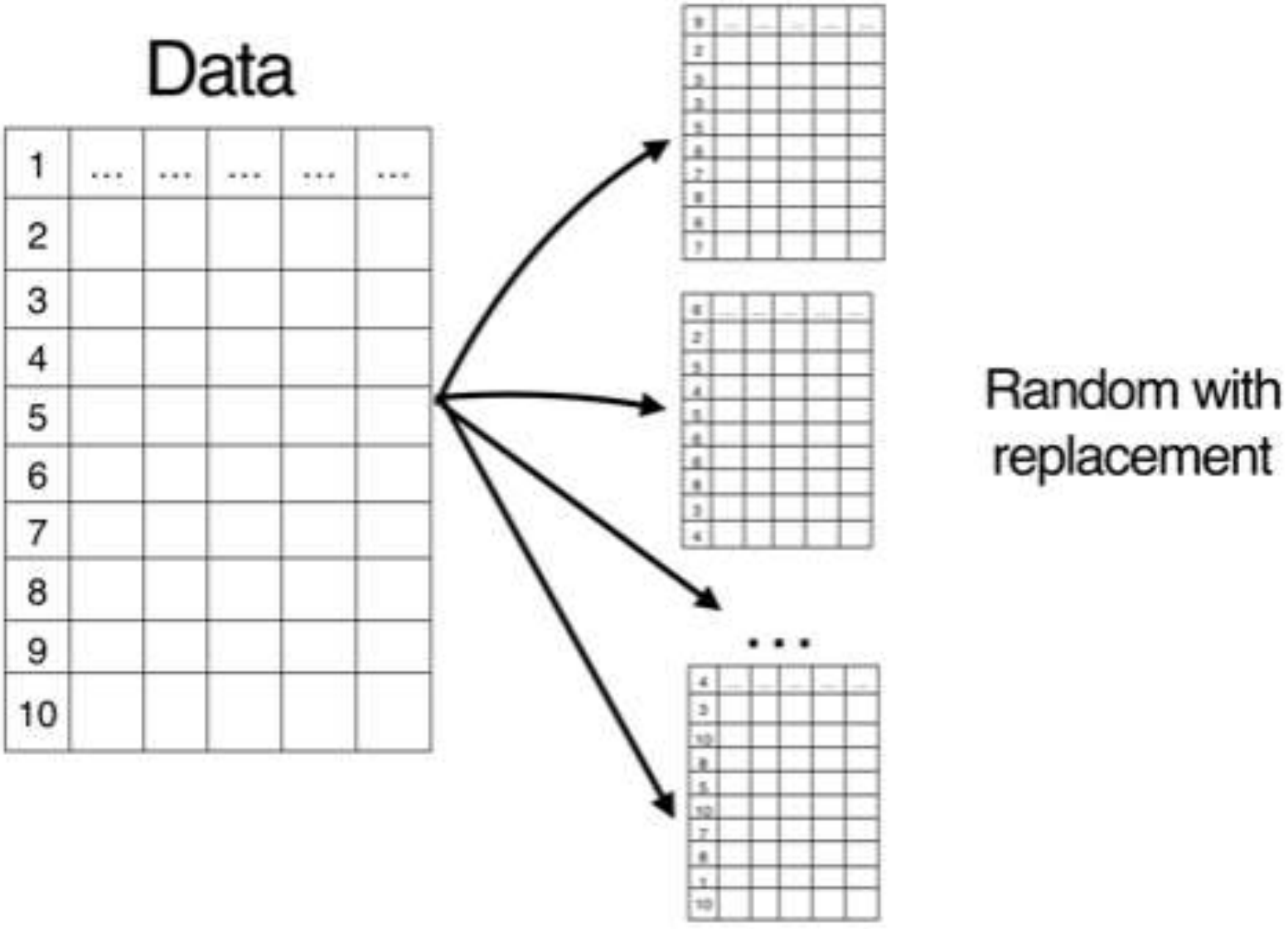
Bootstrap Sampling

- In Bootstrapping sampling technique, multiple subsets are created from the original dataset, selecting observations with replacement.
- Given data: $D = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_N)\}$
- For m=1:M (use validation data to pick M)
- Obtain bootstrap sample D_m from the training data D
- Build a model $G_m(x)$ from bootstrap data D_m

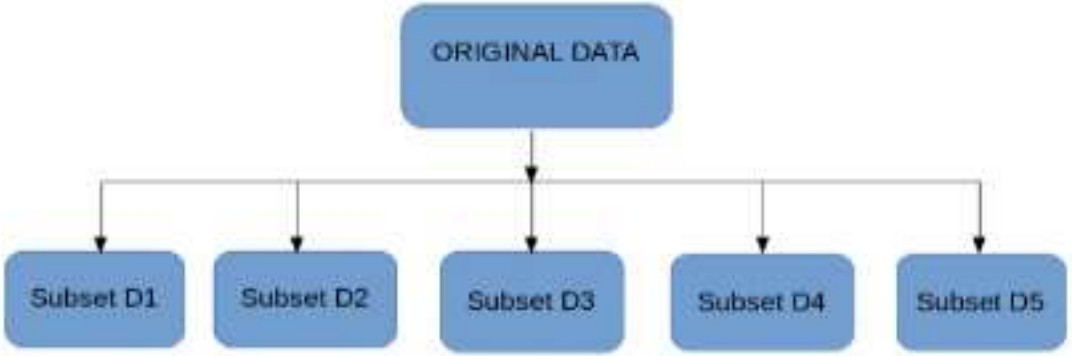
- Bootstrap sample of N instances is obtained by drawing N examples at random, with replacement.

- On average each bootstrap sample has 63% of instances

Bootstrap Sampling



Bagging (Bootstrap Aggregating)



- The size of subsets created for bagging may be less than the original set.
- A base model (weak model) is created on each of these subsets.
- The models run in parallel and are independent of each other.
- The final predictions are determined by combining the predictions from all the models.

The Bagging Model

- Regression

$$\hat{y} = \frac{1}{M}\sum_{m=1}^{M} G_m(\mathbf{x})$$

- Classification:
  - Vote over classifier outputs $G_1(\mathbf{x}), \ldots, G_M(\mathbf{x})$

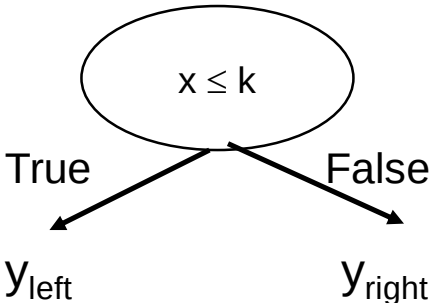
Bagging Example

- Consider 1-dimensional data set:

Original Data:

x	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
y	1	1	1	-1	-1	-1	-1	1	1	1

- Classifier is a decision stump
  - Decision rule: $x \leq k$ versus $x > k$
  - $x < 0.35$ or $X >= 0.75$.
  - Split point k is chosen based on entropy



Bagging Example

Bagging Round 1:

x	0.1	0.2	0.2	0.3	0.4	0.4	0.5	0.6	0.9	0.9
y	1	1	1	1	-1	-1	-1	-1	1	1

x <= 0.35 ➔ y = 1
x > 0.35 ➔ y = -1

Bagging Round 2:

x	0.1	0.2	0.3	0.4	0.5	0.5	0.9	1	1	1
y	1	1	1	-1	-1	-1	1	1	1	1

X < = 0.01 -> y= -1
X > 0.01 -> y= 1

Bagging Round 3:

x	0.1	0.2	0.3	0.4	0.4	0.5	0.7	0.7	0.8	0.9
y	1	1	1	-1	-1	-1	-1	-1	1	1

x <= 0.35 ➔ y = 1
x > 0.35 ➔ y = -1

Bagging Round 4:

x	0.1	0.1	0.2	0.4	0.4	0.5	0.5	0.7	0.8	0.9
y	1	1	1	-1	-1	-1	-1	-1	1	1

x <= 0.3 ➔ y = 1
x > 0.3 ➔ y = -1

Bagging Round 5:

x	0.1	0.1	0.2	0.5	0.6	0.6	0.6	1	1	1
y	1	1	1	-1	-1	-1	-1	1	1	1

x <= 0.35 ➔ y = 1
x > 0.35 ➔ y = -1

Bagging Example

Bagging Round 6:

x	0.2	0.4	0.5	0.6	0.7	0.7	0.7	0.8	0.9	1
y	1	-1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ➔ y = -1
x > 0.75 ➔ y = 1

Bagging Round 7:

x	0.1	0.4	0.4	0.6	0.7	0.8	0.9	0.9	0.9	1
y	1	-1	-1	-1	-1	1	1	1	1	1

x <= 0.75 ➔ y = -1
x > 0.75 ➔ y = 1

Bagging Round 8:

x	0.1	0.2	0.5	0.5	0.5	0.7	0.7	0.8	0.9	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ➔ y = -1
x > 0.75 ➔ y = 1

Bagging Round 9:

x	0.1	0.3	0.4	0.4	0.6	0.7	0.7	0.8	1	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ➔ y = -1
x > 0.75 ➔ y = 1

Bagging Round 10:

x	0.1	0.1	0.1	0.1	0.3	0.3	0.8	0.8	0.9	0.9
y	1	1	1	1	1	1	1	1	1	1

x <= 0.05 ➔ y = 1
x > 0.05 ➔ y = 1

Bagging Example

Summary of Training sets:

Round	Split Point	Left Class	Right Class
1	0.35	1	-1
2	0.7	1	1
3	0.35	1	-1
4	0.3	1	-1
5	0.35	1	-1
6	0.75	-1	1
7	0.75	-1	1
8	0.75	-1	1
9	0.75	-1	1
10	0.05	1	1

Bagging Example

- Assume test set is the same as the original data
- Use majority vote to determine class of ensemble classifi

Original Data:

x	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
y	1	1	1	-1	-1	-1	-1	1	1	1

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	1	1	1	-1	-1	-1	-1	-1	-1	-1
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	-1	-1	-1	-1	-1	-1	-1
4	1	1	1	-1	-1	-1	-1	-1	-1	-1
5	1	1	1	-1	-1	-1	-1	-1	-1	-1
6	-1	-1	-1	-1	-1	-1	-1	1	1	1
7	-1	-1	-1	-1	-1	-1	-1	1	1	1
8	-1	-1	-1	-1	-1	-1	-1	1	1	1
9	-1	-1	-1	-1	-1	-1	-1	1	1	1
10	1	1	1	1	1	1	1	1	1	1
Sum	2	2	2	-6	-6	-6	-6	2	2	2
Sign	1	1	1	-1	-1	-1	-1	1	1	1

Predicted Class

Bagging Algorithm

Algorithm 5.6 Bagging Algorithm

1: Let k be the number of bootstrap samples.
2: for $i = 1$ to k do
3: Create a bootstrap sample of size n, D_i.
4: Train a base classifier C_i on the bootstrap sample D_i.
5: end for
6: $C^*(x) = \arg\max_y \sum_i \delta\big(C_i(x) = y\big)$, $\{\delta(\cdot) = 1$ if its argument is true, and 0 otherwise.$\}$

Bagging – Effect on Variance

- Improves generalization error by reducing the variance of the base classifiers.
- If a base classifier is unstable, bagging helps to reduce the errors associated with random fluctuations in the training data.
- Since every sample has an equal probability of being selected, bagging does not focus on any particular instance of the training data.
  - It is therefore less susceptible to model overfitting when applied to noisy data.

Bagging – Effect on Bias

- If a base classifier is stable, i.e., robust to minor perturbations in the training set, then the error of the ensemble is primarily caused by bias in the base classifier.
- In this situation, bagging may not be able to improve the performance of the base classifiers significantly.
- It may even degrade the classifier's performance because the effective size of each training set is about 37% smaller than the original data.

Random Forest

- Random Forest is ensemble machine learning algorithm that follows the bagging technique.
- The random forest consists of many decisions trees.
- It uses bagging and feature randomness when building each individual tree to try to create an uncorrelated forest of trees whose prediction by committee is more accurate than that of any individual tree.

Feature Randomness

Decision tree : to split a node, we consider every possible feature and pick the one that produces the most separation between the observations in the left node vs. those in the right node.

Random forest : each tree can pick only from a random subset of features. This forces even more variation amongst the trees in the model and ultimately results in lower correlation across trees

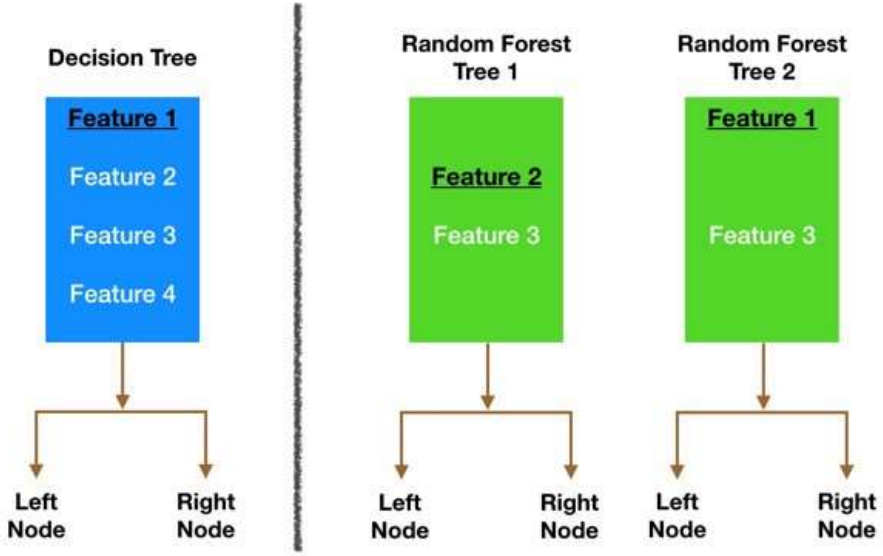
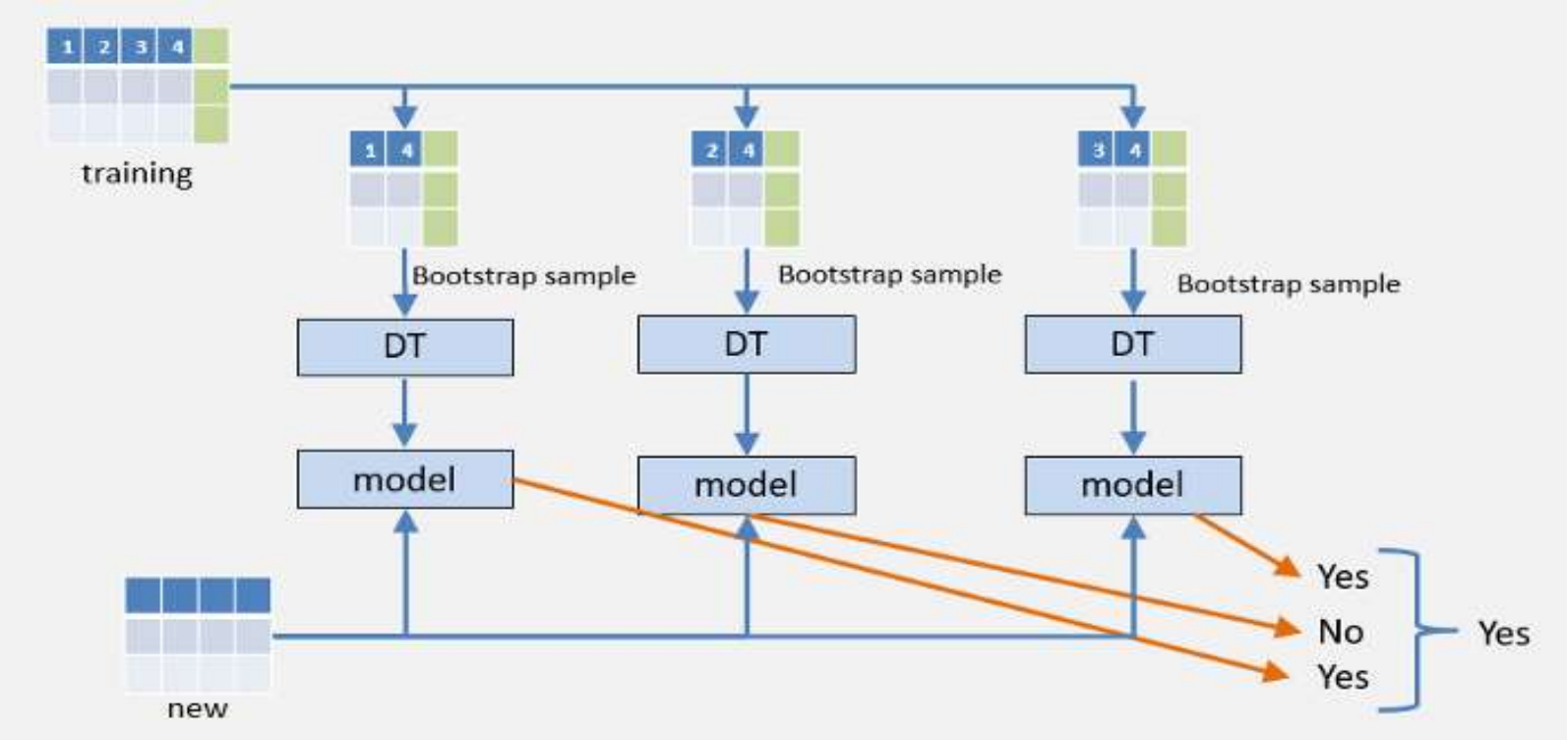


Image credit: https://medium.com

Random Forest



- **Trees that are trained on different sets of data (bagging)**
- **but also use different features to make decisions.**

Image credit: https://medium.com

Random Forest

- Random Forest need features that have at least some predictive power.
- The trees of the forest and more importantly their predictions need to be uncorrelated (or at least have low correlations with each other).
- May over-fit data sets that are particularly noisy.

Advantages of Random Forest

- Algorithm can solve both type of problems i.e. classification and regression
- Power to handle large data set with higher dimensionality.
- It can handle thousands of input variables and identify most significant variables so it is considered as one of the dimensionality reduction methods.
- Model outputs **Importance of variable,** which can be a very handy feature (on some random data set).

Boosting

- Main Assumption:
  - Combining many weak predictors (e.g. tree stumps) to produce an ensemble predictor
  - The weak predictors or classifiers need to be **stable**
- Hypothesis Space
  - Variable size (nonparametric):
    - Can model any function if you use an appropriate predictor (e.g. trees)

Boosting

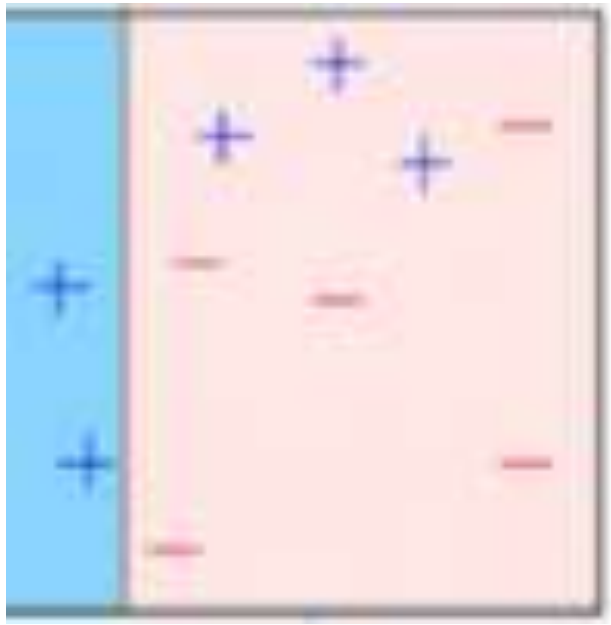
- What if a data point is incorrectly predicted by the first model, and then the next (probably all models), will combining the predictions provide better results? Such situations are taken care of by boosting.
- Boosting is a sequential process, where each subsequent model attempts to correct the errors of the previous model.
- The succeeding models are dependent on the previous model.

Boosting

- An iterative procedure to adaptively change distribution of training data by focusing more on previously misclassified records
  - Initially, all N records are assigned equal weights
  - Unlike bagging, weights may change at the end of each boosting round

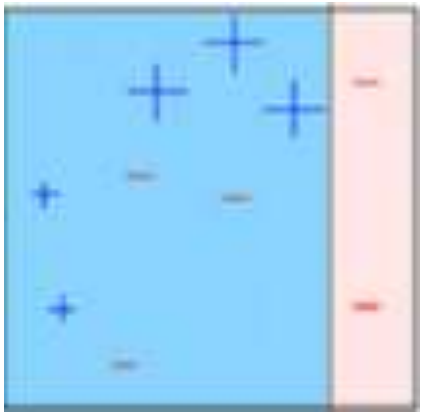
Boosting

- A subset is created from the original dataset.
- Initially, all data points are given equal weights.
- A base model is created on this subset.
- This model is used to make predictions on the whole dataset.



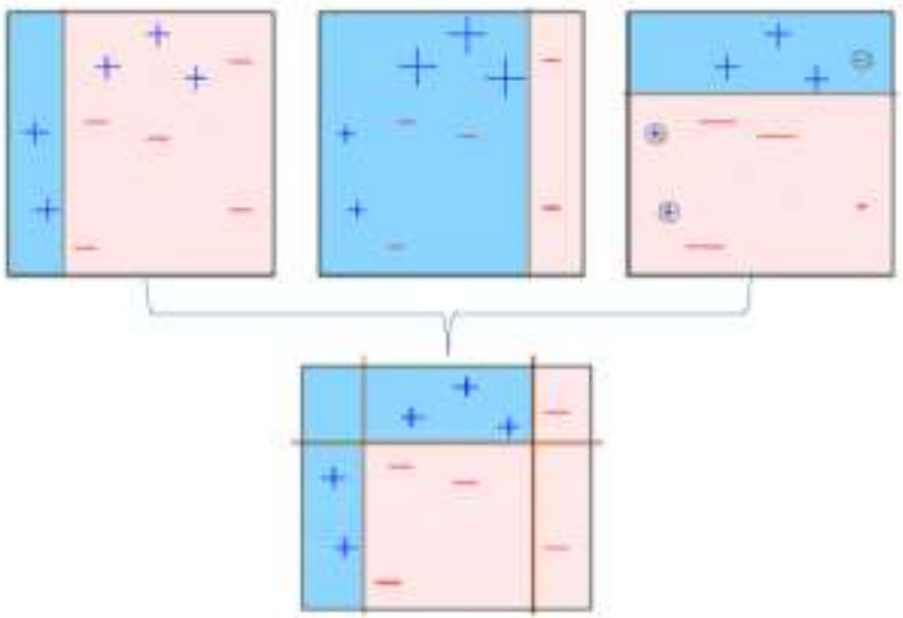
Boosting

- Errors are calculated using the actual values and predicted values.
- The observations which are incorrectly predicted, are given higher weights. (Here, the three misclassified blue-plus points will be given higher weights)
- Another model is created and predictions are made on the dataset. (This model tries to correct the errors from the previous model)



Boosting

- Similarly, multiple models are created, each correcting the errors of the previous model.
- The final model (strong learner) is the weighted mean of all the models (weak learners).
- Individual models would not perform well on the entire dataset, but they work well for some part of the dataset. Thus, each model actually boosts the performance of the ensemble.



AdaBoost

- Adaptive boosting or AdaBoost is one of the simplest boosting algorithms. Usually, decision trees are used for modelling. Multiple sequential models are created, each correcting the errors from the last model.
- AdaBoost assigns weights to the observations which are incorrectly predicted and the subsequent model works to predict these values correctly.

AdaBoost Algorithm

- Initially, all observations (n) in the dataset are given equal weights (1/n).
- A model is built on a subset of data.
- Using this model, predictions are made on the whole dataset.
- Errors are calculated by comparing the predictions and actual values.
- While creating the next model, higher weights are given to the data points which were predicted incorrectly.

Adaboost Algorithm

- Weights can be determined using the error value. For instance, higher the error more is the weight assigned to the observation.
- This process is repeated until the error function does not change, or the maximum limit of the number of estimators is reached.

AdaBoost

- Base classifiers C_i: C_1, C_2, …, C_T
- Error rate: N input samples, (Averaging over weights of training examples of misclassified points)

$$\varepsilon_i = \frac{1}{N}\sum_{j=1}^{N} w_j \delta\left(C_i(x_j) \neq y_j\right)$$

- Other two variants for calculating error rate
  - Taking sum of weights of training examples of misclassified points
  - Taking weighted average over weights of training examples of misclassified points
- **Notice that the error is measured with respect to the same distribution D_t on which the base classifier was trained.**
- Importance of a classifier:

$$\alpha_i = \frac{1}{2}\ln\left(\frac{1-\varepsilon_i}{\varepsilon_i}\right)$$

https://en.wikipedia.org/wiki/AdaBoost#Choosing_at

AdaBoost: Weight Update

Weight Update:

$$w_i^{(j+1)} = \frac{w_i^{(j)}}{Z_j} \begin{cases} \exp^{-\alpha_j} & \text{if } C_j(x_i) = y_i \\ \exp^{\alpha_j} & \text{if } C_j(x_i) \neq y_i \end{cases}$$

<- Eqn:5.88

where Z_j is the normalizat ion factor

- Reduce weight if correctly classified else increase
- If any intermediate rounds produce error rate higher than 50%, the weights are reverted back to 1/n and the resampling procedure is repeated
- Prediction:

$$C^*(x) = \arg\max_y \sum_{j=1}^{T} \alpha_j \delta\left(C_j(x) = y\right)$$

AdaBoost Algorithm- Version 1 (with bootstrapping + average for error term)

Algorithm 5.7 AdaBoost Algorithm

1: $\mathbf{w} = \{w_j = 1/n \mid j = 1, 2, \cdots, n\}$. {Initialize the weights for all n instances.}
2: Let k be the number of boosting rounds.
3: **for** $i = 1$ to k **do**
4: Create training set D_i by sampling (with replacement) from D according to $\mathbf{w}$.
5: Train a base classifier C_i on D_i.
6: Apply C_i to all instances in the original training set, D.
7: $\epsilon_i = \frac{1}{n}\left[\sum_j w_j \, \delta\left(C_i(x_j) \neq y_j\right)\right]$ {Calculate the weighted error}
8: **if** $\epsilon_i > 0.5$ **then**
9: $\mathbf{w} = \{w_j = 1/n \mid j = 1, 2, \cdots, n\}$. {Reset the weights for all n instances.}
10: Go back to Step 4.
11: **end if**
12: $\alpha_i = \frac{1}{2} \ln \frac{1-\epsilon_i}{\epsilon_i}$.
13: Update the weight of each instance according to equation (5.88).
14: **end for**
15: $C^*(\mathbf{x}) = \arg\max_y \sum_{j=1}^{T} \alpha_j \delta(C_j(\mathbf{x}) = y))$.

AdaBoost Algorithm Version 2 (without bootstrapping + sum for error term)

INPUT: training data $X, y = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$,
the number of iterations T

1: Initialize a vector of n uniform weights $\mathbf{w}_1 = \left[\frac{1}{n}, \ldots, \frac{1}{n}\right]$

2: **for** $t = 1, \ldots, T$

3: Train model h_t on X, y with instance weights $\mathbf{w}_t$

4: Compute the weighted training error rate of h_t:

$$\epsilon_t = \sum_{i:y_i \neq h_t(\mathbf{x}_i)} w_{t,i}$$

5: Choose $\beta_t = \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$

6: Update all instance weights:

$$w_{t+1,i} = w_{t,i} \exp\left(-\beta_t y_i h_t(\mathbf{x}_i)\right) \quad \forall i = 1, \ldots, n$$

7: Normalize $\mathbf{w}_{t+1}$ to be a distribution:

$$w_{t+1,i} = \frac{w_{t+1,i}}{\sum_{j=1}^{n} w_{t+1,j}} \quad \forall i = 1, \ldots, n$$

8: **end for**

9: **Return** the hypothesis

$$H(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \beta_t h_t(\mathbf{x})\right)$$

Member classifier with less error are given more weight in final ensemble hypothesis. Final prediction is a weighted combination of each members prediction

AdaBoost Example (version 1)

Original Data:

x	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
y	1	1	1	-1	-1	-1	-1	1	1	1

Training sets for the first 3 boosting rounds:

AdaBoost Example

Boosting Round 1:

x	0.1	0.4	0.5	0.6	0.6	0.7	0.7	0.7	0.8	1
y	1	-1	-1	-1	-1	-1	-1	-1	1	1

Boosting Round 2:

x	0.1	0.1	0.2	0.2	0.2	0.2	0.3	0.3	0.3	0.3
y	1	1	1	1	1	1	1	1	1	1

Boosting Round 3:

x	0.2	0.2	0.4	0.4	0.4	0.4	0.5	0.6	0.6	0.7
y	1	1	-1	-1	-1	-1	-1	-1	-1	-1

- Summary:

Round	Split Point	Left Class	Right Class	alpha
1	0.75	-1	1	1.738
2	0.05	1	1	2.7784
3	0.3	1	-1	4.1195

Boosting Round 1:

x	0.1	0.4	0.5	0.6	0.6	0.7	0.7	0.7	0.8	1
y	1	-1	-1	-1	-1	-1	-1	-1	1	1

Round - 1				
x	y	W1	W2	norm w2
0.1	1	0.1	0.5686	0.311
0.2	1	0.1	0.5686	0.311
0.3	1	0.1	0.5686	0.311
0.4	-1	0.1	0.0176	0.010
0.5	-1	0.1	0.0176	0.010
0.6	-1	0.1	0.0176	0.010
0.7	-1	0.1	0.0176	0.010
0.8	1	0.1	0.0176	0.010
0.9	1	0.1	0.0176	0.010
1	1	0.1	0.0176	0.010
	ε1=	0.0300		
	α1=	1.7380		
		sum(w1-w10)	1.83	

$$\varepsilon_i = \frac{1}{N}\sum_{j=1}^{N} w_j \delta\left(C_i(x_j) \neq y_j\right)$$

$$\alpha_i = \frac{1}{2}\ln\left(\frac{1-\varepsilon_i}{\varepsilon_i}\right)$$

$$w_i^{(j+1)} = \frac{w_i^{(j)}}{Z_j}\begin{cases}\exp^{-\alpha_j} & \text{if } C_j(x_i) = y_i \\ \exp^{\alpha_j} & \text{if } C_j(x_i) \neq y_i\end{cases}$$

where Z_j is the normalizat ion factor

Boosting Round 2:

x	0.1	0.1	0.2	0.2	0.2	0.2	0.3	0.3	0.3	0.3
y	1	1	1	1	1	1	1	1	1	1

		Round - 2		
norm w2	x	y	w3	norm w3
0.311	0.1	1	0.019	0.028
0.311	0.2	1	0.019	0.028
0.311	0.3	1	0.019	0.028
0.010	0.4	-1	0.155	0.228
0.010	0.5	-1	0.155	0.228
0.010	0.6	-1	0.155	0.228
0.010	0.7	-1	0.155	0.228
0.010	0.8	1	0.001	0.001
0.010	0.9	1	0.001	0.001
0.010	1	1	0.001	0.001
0.004	=ε2			
2.778	=α2			
		sum(w1-w10)	0.679	

$$\varepsilon_i = \frac{1}{N}\sum_{j=1}^{N} w_j \delta\left(C_i(x_j) \neq y_j\right)$$

$$\alpha_i = \frac{1}{2}\ln\left(\frac{1-\varepsilon_i}{\varepsilon_i}\right)$$

$$w_i^{(j+1)} = \frac{w_i^{(j)}}{Z_j}\begin{cases}\exp^{-\alpha_j} & \text{if } C_j(x_i) = y_i \\ \exp^{\alpha_j} & \text{if } C_j(x_i) \neq y_i\end{cases}$$

where Z_j is the normalizat ion factor

Boosting Round 3:

x	0.2	0.2	0.4	0.4	0.4	0.4	0.5	0.6	0.6	0.7
y	1	1	-1	-1	-1	-1	-1	-1	-1	-1

$$\varepsilon_i = \frac{1}{N}\sum_{j=1}^{N} w_j \delta\left(C_i(x_j) \neq y_j\right)$$

$$\alpha_i = \frac{1}{2}\ln\left(\frac{1-\varepsilon_i}{\varepsilon_i}\right)$$

$$w_i^{(j+1)} = \frac{w_i^{(j)}}{Z_j}\begin{cases}\exp^{-\alpha_j} & \text{if } C_j(x_i) = y_i \\ \exp^{\alpha_j} & \text{if } C_j(x_i) \neq y_i\end{cases}$$

where Z_j is the normalizat ion factor

	Round 3			
norm w3	x	y	w4	norm w4
0.028	0.1	1	0.000463	0.002588738
0.028	0.2	1	0.000463	0.002588738
0.028	0.3	1	0.000463	0.002588738
0.228	0.4	-1	0.003706	0.020736589
0.228	0.5	-1	0.003706	0.020736589
0.228	0.6	-1	0.003706	0.020736589
0.228	0.7	-1	0.003706	0.020736589
0.001	0.8	1	0.054162	0.30309581
0.001	0.9	1	0.054162	0.30309581
0.001	1	1	0.054162	0.30309581
0.000264	=ε3			
4.119	=α3			
		sum(w1-w10)	0.178695	

final prediction						Predicted class
x	y	R1	R2	R3	sum	sign(sum)
0.1	1	-1	1	1	5.159853	1
0.2	1	-1	1	1	5.159853	1
0.3	1	-1	1	1	5.159853	1
0.4	-1	-1	1	-1	-3.07912	-1
0.5	-1	-1	1	-1	-3.07912	-1
0.6	-1	-1	1	-1	-3.07912	-1
0.7	-1	-1	1	-1	-3.07912	-1
0.8	1	1	1	-1	0.396975	1
0.9	1	1	1	-1	0.396975	1
1	1	1	1	-1	0.396975	1

α_1	1.7380
α_2	2.778
α_3	4.119

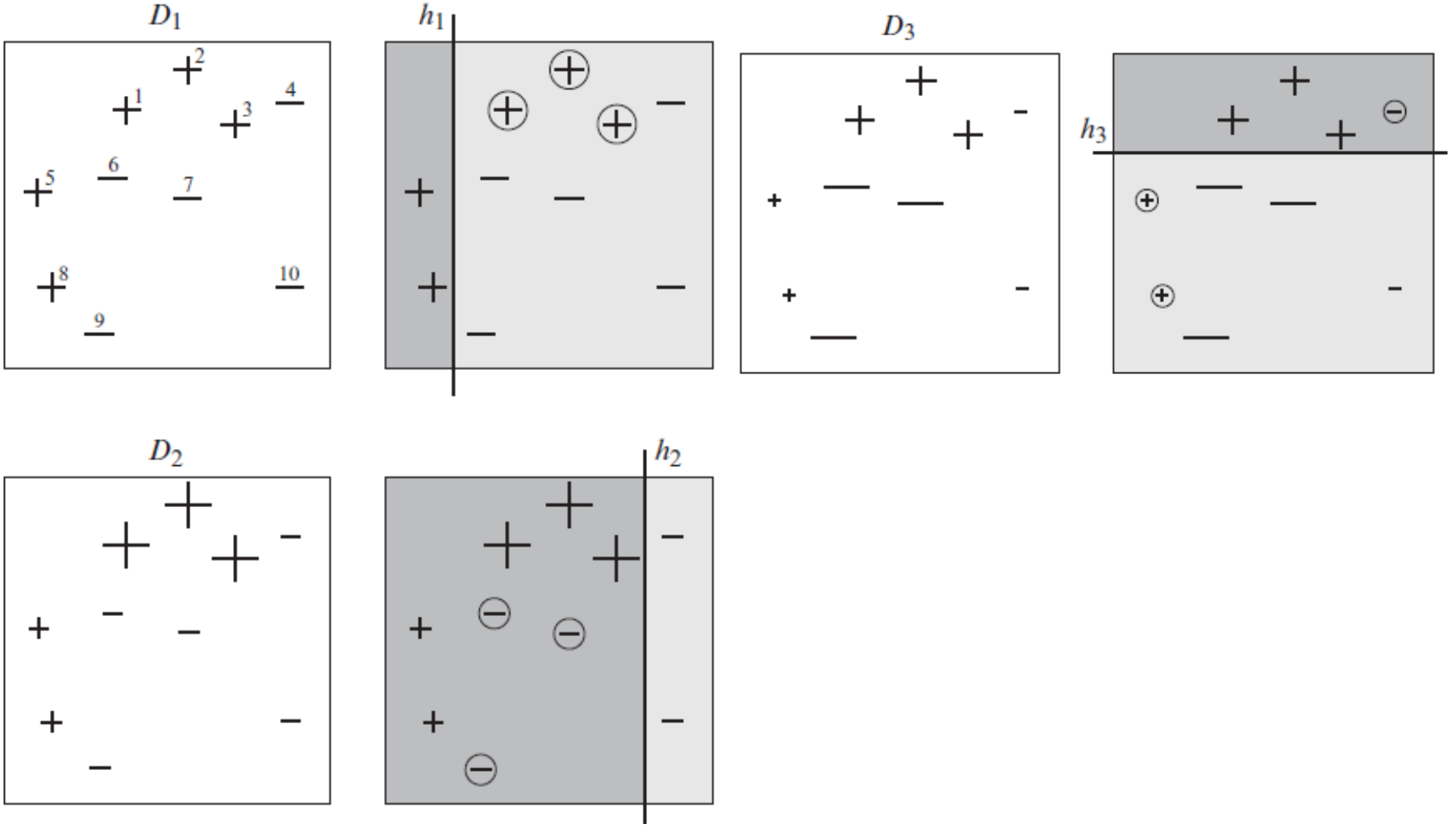
$$C^*(x) = \arg\max_y \sum_{j=1}^{T} \alpha_j \delta\left(C_j(x) = y\right)$$

OR

$$H(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \beta_t h_t(\mathbf{x})\right)$$

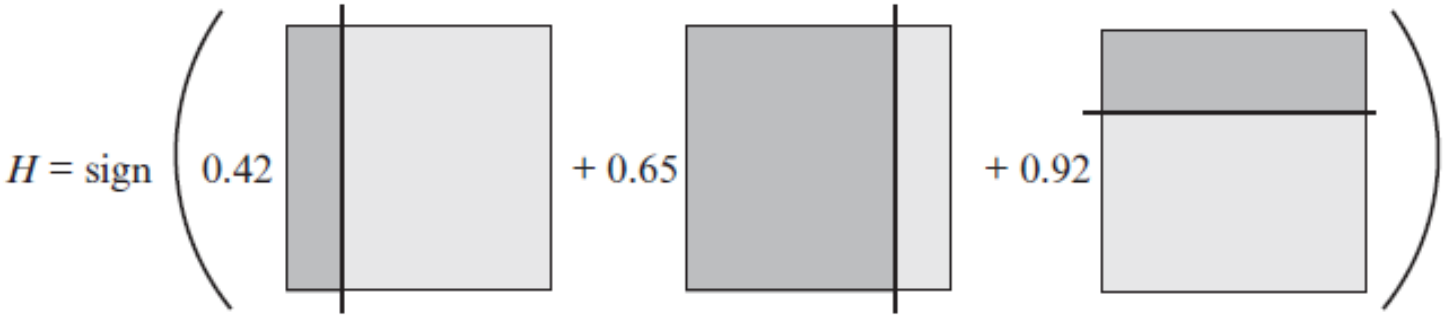
AdaBoost error function takes into account the fact that only the sign of the final result is used, thus sum can be far larger than 1 without increasing error

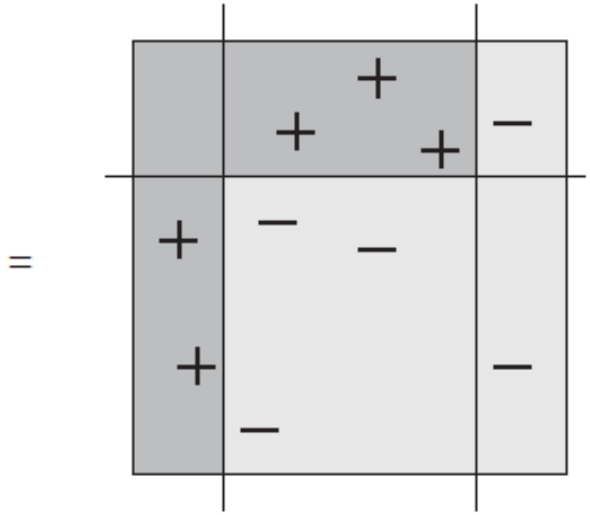
AdaBoost Example (version 2)



Example

How do we combine the results now?





Schapire and Freund, 2012

Example

Table 1.1
The numerical calculations corresponding to the toy example in figure 1.1

	1	2	3	4	5	6	7	8	9	10	
$D_1(i)$	<u>0.10</u>	<u>0.10</u>	<u>0.10</u>	0.10	0.10	0.10	0.10	0.10	0.10	0.10	$\epsilon_1 = 0.30, \alpha_1 \approx 0.42$
$e^{-\alpha_1 y_i h_1(x_i)}$	1.53	1.53	1.53	0.65	0.65	0.65	0.65	0.65	0.65	0.65	
$D_1(i)\, e^{-\alpha_1 y_i h_1(x_i)}$	0.15	0.15	0.15	0.07	0.07	0.07	0.07	0.07	0.07	0.07	$Z_1 \approx 0.92$
$D_2(i)$	0.17	0.17	0.17	0.07	0.07	<u>0.07</u>	<u>0.07</u>	0.07	<u>0.07</u>	0.07	$\epsilon_2 \approx 0.21, \alpha_2 \approx 0.65$
$e^{-\alpha_2 y_i h_2(x_i)}$	0.52	0.52	0.52	0.52	0.52	1.91	1.91	0.52	1.91	0.52	
$D_2(i)\, e^{-\alpha_2 y_i h_2(x_i)}$	0.09	0.09	0.09	0.04	0.04	0.14	0.14	0.04	0.14	0.04	$Z_2 \approx 0.82$
$D_3(i)$	0.11	0.11	0.11	<u>0.05</u>	<u>0.05</u>	0.17	0.17	<u>0.05</u>	0.17	0.05	$\epsilon_3 \approx 0.14, \alpha_3 \approx 0.92$
$e^{-\alpha_3 y_i h_3(x_i)}$	0.40	0.40	0.40	2.52	2.52	0.40	0.40	2.52	0.40	0.40	
$D_3(i)\, e^{-\alpha_3 y_i h_3(x_i)}$	0.04	0.04	0.04	0.11	0.11	0.07	0.07	0.11	0.07	0.02	$Z_3 \approx 0.69$

Calculations are shown for the ten examples as numbered in the figure. Examples on which hypothesis h_t makes a mistake are indicated by underlined figures in the rows marked D_t.

Schapire and Freund, 2012

AdaBoost base learners

- AdaBoost works best with "weak" learners
  - Should not be complex
  - Typically high bias classifiers
  - Works even when weak learner has an error rate just slightly under 0.5 (i.e., just slightly better than random)
    - Can prove training error goes to 0 in O(log n) iterations

- Examples:
  - Decision stumps (1 level decision trees)
  - Depth-limited decision trees
  - Linear classifiers

AdaBoost in practice

Strengths:

- Fast and simple to program
- No parameters to tune (besides T)
- No assumptions on weak learner

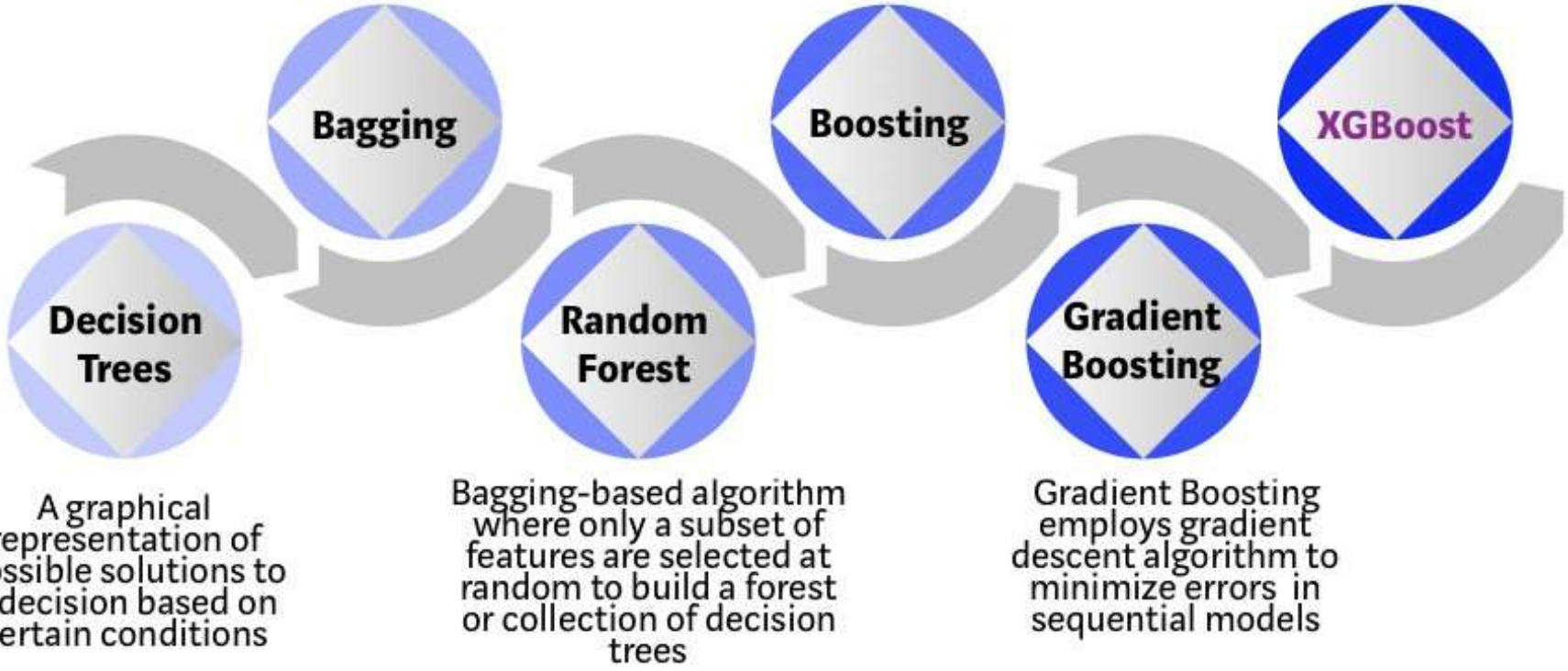
When boosting can fail:

- Given insufficient data
- Overly complex weak hypotheses
- Can be susceptible to noise
- When there are a large number of outliers

- The model cannot be parallelized since each predictor can only be trained after the previous one has been trained and evaluated.

Fine Tuning Ensembles

- Model combination does not always guaranteed to decrease error, unless
  - base-learners are diverse and accurate
- Ignore poor base learners
  - Use accuracy as a cut-off
  - Introduce some pruning with which at each iteration remove poor learners / learners whose absence lead to improvement (if any)
    - Modify iterations to allow both additions / deletions of learners
  - Discarding appropriately leads to better performance



https://towardsdatascience.com/

Gradient Boosting

- The idea of gradient boosting originated in the observation by Leo Breiman that boosting can be interpreted as an optimization algorithm on a suitable cost function
- Explicit regression gradient boosting algorithms were subsequently developed
- That is, algorithms that optimize a cost function over function space by iteratively choosing a function (weak hypothesis) that points in the negative gradient direction.
- predictor can be any machine learning algorithm like SVM, Logistic regression, KNN , Decision tree etc. But Decision tree version of gradient boosting is much popular

Gradient Boosting

- Gradient Boosting for Different Problems
- Difficulty: regression ===> classification ===> ranking
- In Gradient Boosting, "shortcomings" are identified by gradients.
- Recall that, in Adaboost, "shortcomings" are identified by high-weight data points.
- Both high-weight data points and gradients tell us how to improve our model.

Gradient Boosting

- You are given (x1, y1),(x2, y2), ...,(xn, yn), and the task is to fit a model F(x) to minimize square loss
- There are some mistakes: F(x1) = 0.8, while y1 = 0.9, and F(x2) = 1.4 while y2 = 1.3... How can you improve this model?
- Rules:
  - You are not allowed to remove anything from F or change any parameter in F.
  - You can add an additional model (regression tree) h to F, so the new prediction will be F(x) + h(x).

Gradient Boosting

- You wish to improve the model such that
  - $F(x1) + h(x1) = y1$
  - $F(x2) + h(x2) = y2$ ...
  - $F(xn) + h(xn) = yn$

Or, equivalently, you wish

$h(x1) = y1 - F(x1)$

$h(x2) = y2 - F(x2)$ ...

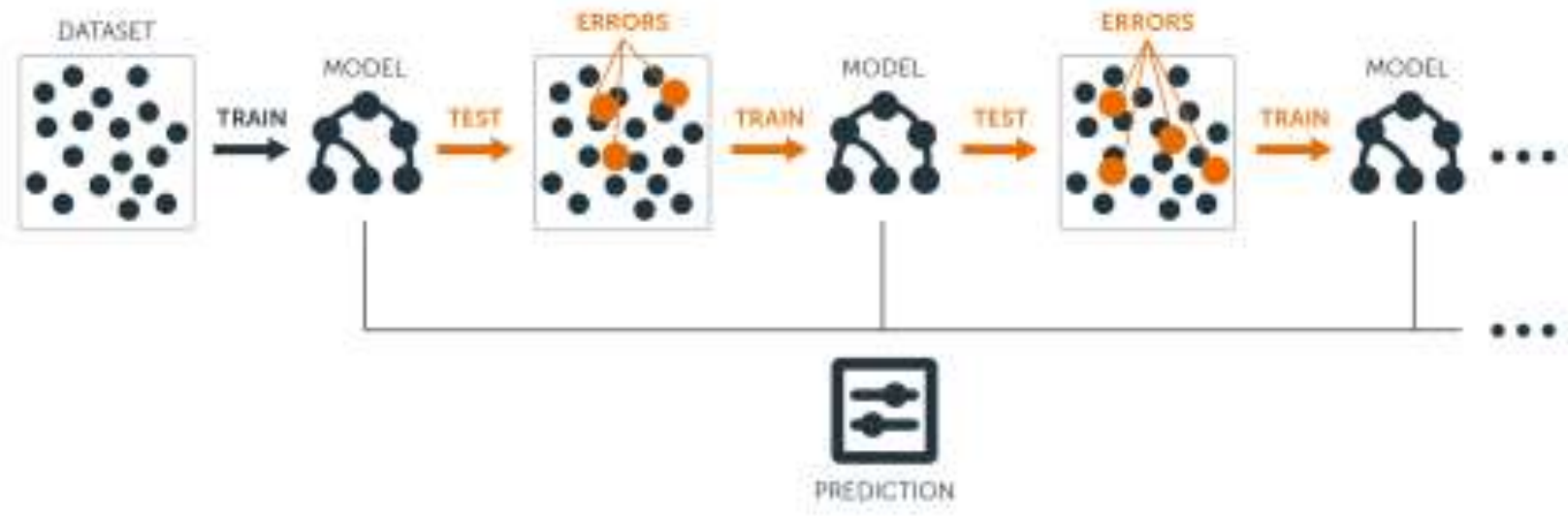
$h(xn) = yn - F(xn)$

Fit a regression tree h to data

$(x1, y1 - F(x1)),(x2, y2 - F(x2)), ...,(xn, yn - F(xn))$

Gradient Boosting

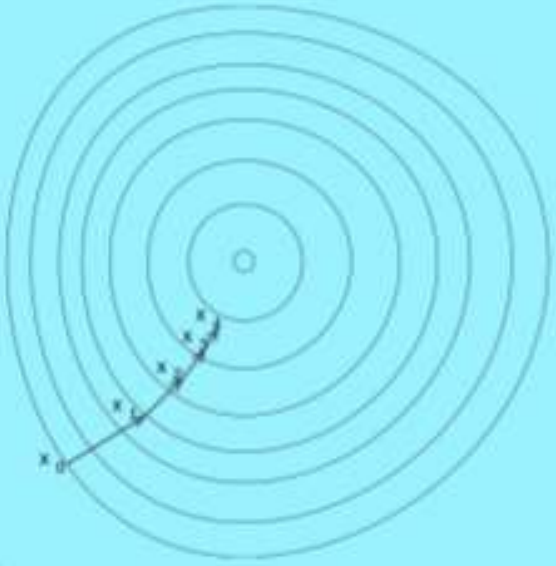
- Simple solution: yi − F(xi) are called residuals. These are the parts that existing model F cannot do well.
- The role of h is to compensate the shortcoming of existing model F.
- If the new model F + h is still not satisfactory, we can add another regression tree...
- We are improving the predictions of training data, is the procedure also useful for test data?



Gradient Descent

Minimize a function by moving in the opposite direction of the gradient.

$$\theta_i := \theta_i - \rho \frac{\partial J}{\partial \theta_i}$$



Gradient Boosting for regression

Loss function $L(y, F(x)) = (y - F(x))^2/2$
We want to minimize $J = \sum_i L(y_i, F(x_i))$ by adjusting $F(x_1), F(x_2), ..., F(x_n)$.
Notice that $F(x_1), F(x_2), ..., F(x_n)$ are just some numbers. We can treat $F(x_i)$ as parameters and take derivatives

$$\frac{\partial J}{\partial F(x_i)} = \frac{\partial \sum_i L(y_i, F(x_i))}{\partial F(x_i)} = \frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} = F(x_i) - y_i$$

So we can interpret residuals as negative gradients.

$$y_i - F(x_i) = -\frac{\partial J}{\partial F(x_i)}$$

$$F(x_i) := F(x_i) + h(x_i)$$
$$F(x_i) := F(x_i) + y_i - F(x_i)$$
$$F(x_i) := F(x_i) - 1\frac{\partial J}{\partial F(x_i)}$$
$$\theta_i := \theta_i - \rho\frac{\partial J}{\partial \theta_i}$$

For regression with **square loss**,

residual $\Leftrightarrow$ *negative gradient*

fit h to residual $\Leftrightarrow$ *fit h to negative gradient*

update F based on residual $\Leftrightarrow$ *update F based on negative gradient*

So we are actually updating our model using **gradient descent**!

Gradient boosting algorithm

let F_0 be a “dummy” constant model

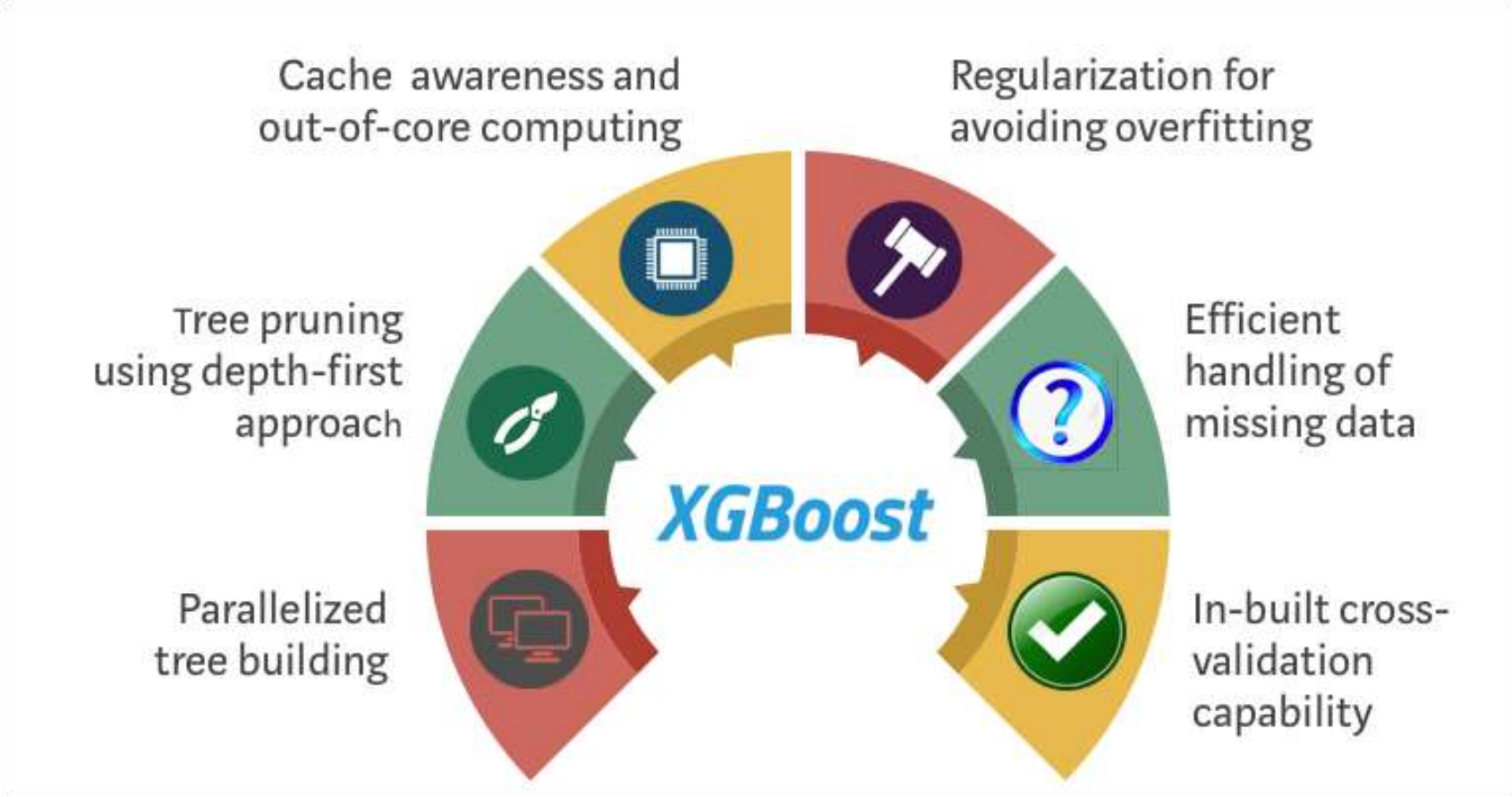
for m = 1, . . . ,M

- **for** each pair $(\mathbf{x}_i, y_i)$ in the training set
- compute the pseudo-residual $R(y_i, F_{m-1}(\mathbf{x}_i))$ = negative gradient of the loss
- train a regression sub-model h_m on the pseudo-residuals
- add h_m to the ensemble: $F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \rho . h_m(\mathbf{x})$
- **return** the ensemble F_M

Why XGBoost so popular?

- **Speed** : faster than other ensemble classifiers.
- **Core algorithm is parallelizable**: harness the power of multi-core computers and networks of computers enabling to train on very large datasets **Consistently outperforms other algorithm methods** : It has shown better performance on a variety of machine learning benchmark datasets.
- **Wide variety of tuning parameters** : cross-validation, regularization, missing values, tree parameters, etc
- XGBoost (Extreme Gradient Boosting) uses the gradient boosting (GBM) framework at its core.

XGBoost


Cache awareness and out-of-core computing
Regularization for avoiding overfitting
Tree pruning using depth-first approach
Efficient handling of missing data
XGBoost
Parallelized tree building
In-built cross-validation capability

References

- Introduction to Data Mining, by Pang-Ning Tan, Michael Steinbach , Vipin Kumar
- Bishop - Pattern Recognition And Machine Learning - Springer 2006

Thank You!