# Tree Ensemble Models

Data Mining

Prof. Sujee Lee

Department of Systems Management Engineering

Sungkyunkwan University

## Decision Tree

- **The main hyperparameters of decision trees**
  - Picking one of the pre-pruning strategies (max_depth, max_leaf_nodes, or min_samples_leaf) is sufficient to prevent overfitting.

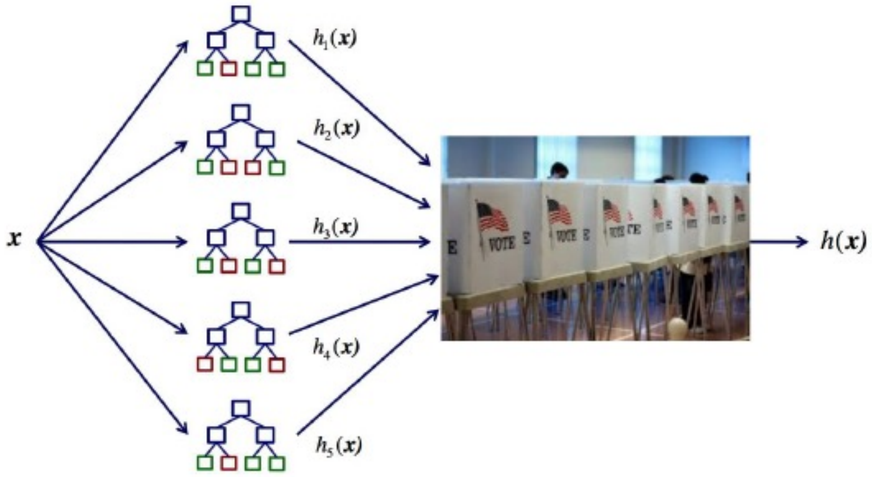
  * Typically chosen to have the highest performance in validation data.

- **Strengths**
  - Decision trees work well when you have a mix of continuous and categorical features.
  - The algorithms are completely invariant to scaling of the data. (no data scaling is needed)
  - Feature selection & reduction is automatic.
  - It is robust to noise.
  - The resulting model can easily be visualized and understood.

- **Weaknesses**
  - Even with the use of pre-pruning, they tend to overfit and provide poor generalization performance.
    - Thus, the ensemble methods are usually used in place of a single decision tree.

# Tree Ensembles

- **Tree Ensembles**
  - Classification trees get overfitted easily (low bias – high variance) and are seldom accurate
  - Thus, we will build multiple trees and average their results!
  - **Ensemble** is a way of averaging multiple deep decision trees, trained on different parts of the same training set, to reduce the variance
    - combine the outputs of many "weak" classifiers to produce a powerful "committee"



regression :

$$f(x) = \frac{1}{\mathrm{B}} \sum_{b=1}^{B} f_b(x)$$

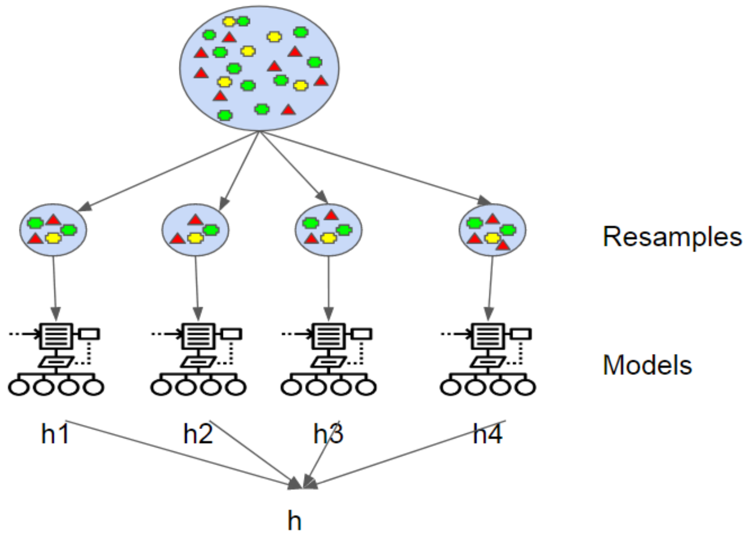
classification :

$$C(x) = majority\ vote\ \{C_b(x)\}_1^B$$

- **ing, Boosting, Random Forest**

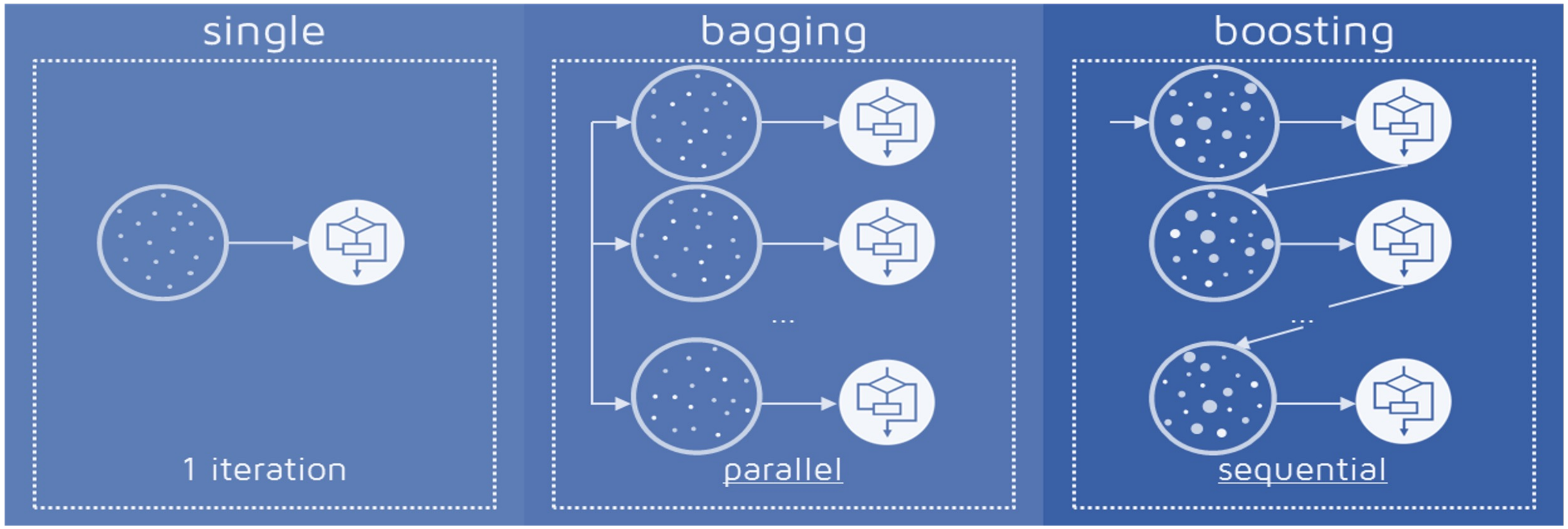
# Bagging

- **Bagging (Bootstrap Aggregating)**
  - Each tree might do a relatively good job of predicting, but will overfit on part of the data in different ways.
  - If we build many trees, we can reduce the amount of overfitting (reduce the variance of the models) by averaging their results while retaining the predictive power of the trees.
  - In Bagging, it simply
    - repeat [choose different subsamples (bootstrap samples) + fit a model]
    - take the majority vote
  - Not only for trees!
    - works well for high-variance, low-bias models



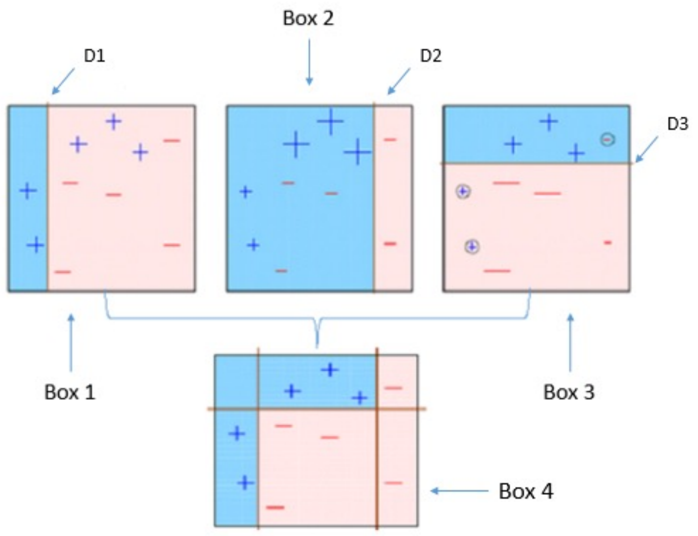
# Boosting

- **Boosting**
  - Unlike Bagging, the committee of weak learners evolves over time (sequentially learning trees), and the members cast a weighted vote.
  - Boosting appears to dominate bagging on most problems and preferred.



# Boosting

- **AdaBoost (Adaptive Boosting)**
  - Combine multiple weak learners (decision trees with a single split)
  - In each step, give more weights on observations misclassified in the previous step

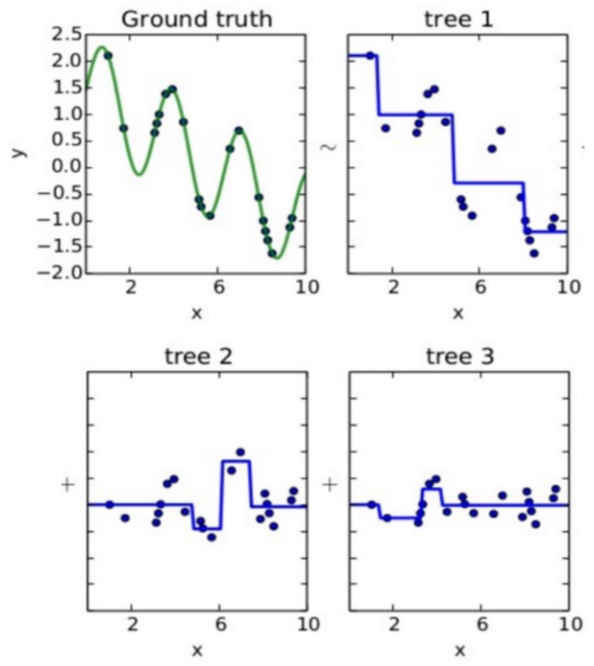


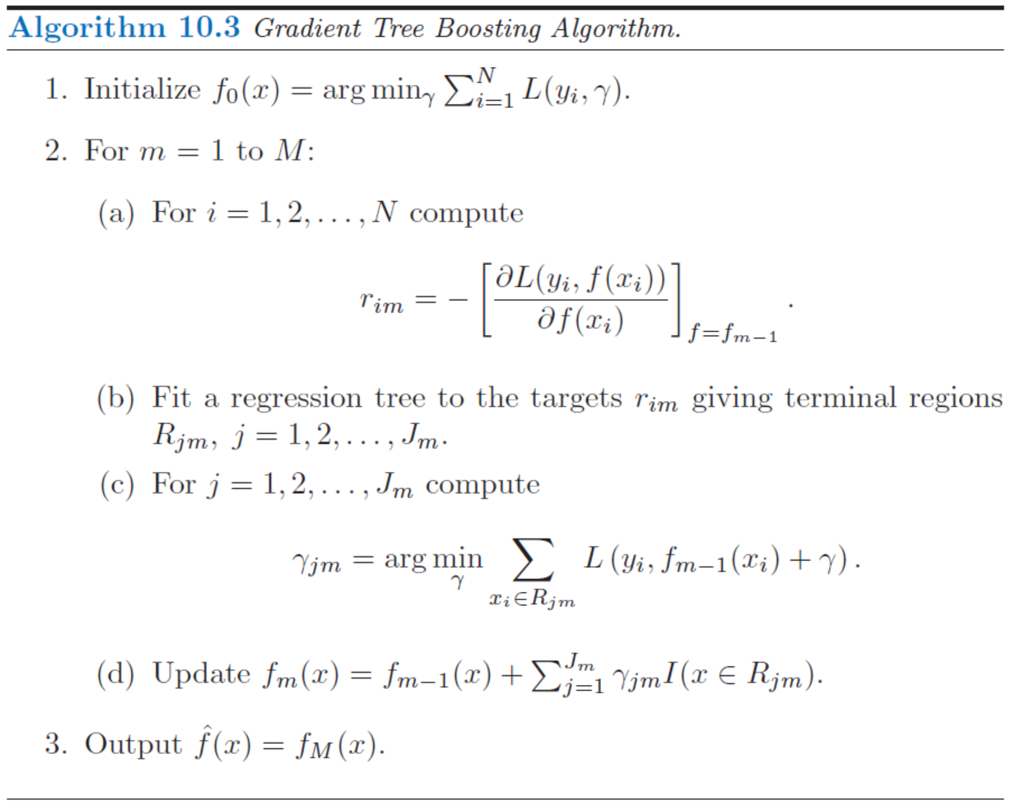
**Algorithm 10.1** *AdaBoost.M1.*

```
1. Initialize the observation weights $w_i = 1/N,\ i = 1, 2, \ldots, N$.
2. For $m = 1$ to $M$:
    (a) Fit a classifier $G_m(x)$ to the training data using weights $w_i$.
    (b) Compute
        $$\mathrm{err}_m = \frac{\sum_{i=1}^N w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^N w_i}.$$
    (c) Compute $\alpha_m = \log((1 - \mathrm{err}_m)/\mathrm{err}_m)$.
    (d) Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))],\ i = 1, 2, \ldots, N$.
3. Output $G(x) = \mathrm{sign}\left[\sum_{m=1}^M \alpha_m G_m(x)\right]$.
```

# Boosting

- **Gradient Boosting**
  - Try to fit the new classifier to the residual errors made by the previous classifier





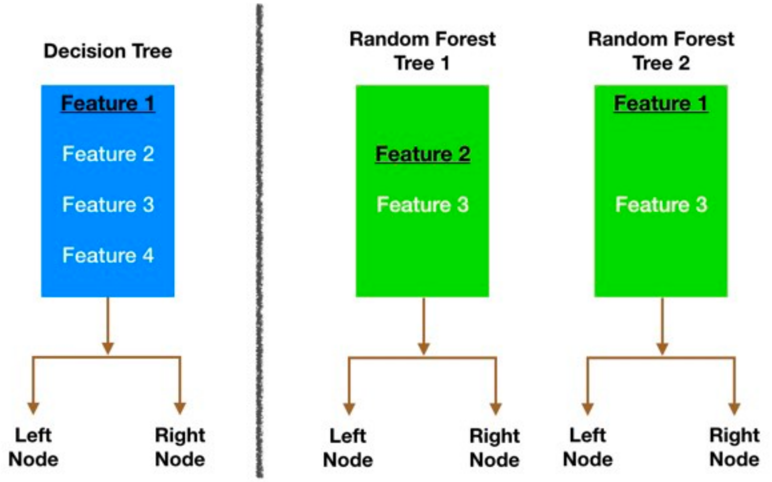
**Algorithm 10.3** *Gradient Tree Boosting Algorithm.*

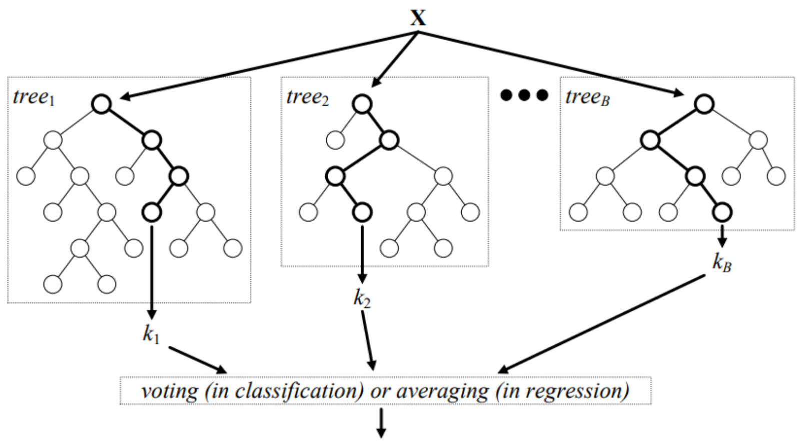
```
1. Initialize $f_0(x) = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$.
2. For $m = 1$ to $M$:
    (a) For $i = 1, 2, \ldots, N$ compute
            $r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}$.
    (b) Fit a regression tree to the targets $r_{im}$ giving terminal regions
        $R_{jm},\ j = 1, 2, \ldots, J_m$.
    (c) For $j = 1, 2, \ldots, J_m$ compute
            $\gamma_{jm} = \arg\min_{\gamma} \sum_{x_i \in R_{jm}} L\left(y_i, f_{m-1}(x_i) + \gamma\right)$.
    (d) Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$.
3. Output $\hat{f}(x) = f_M(x)$.
```

  - **XGBoost** (eXtreme Gradient Boosting) : Faster and efficient implementation of Gradient Boosting

# Random Forest

- **Random Forest**
  - Similar to Bagging, random forest consists of multiple trees (*forest*).
  - Unlike bagging, it builds a large collection of *de-correlated* trees, and averages them.
  - Generate different tress by using
    - Random *samples of training data points* when building trees
    - Random *subsets of features* when splitting nodes
  - Random forests get their name from injecting randomness into the tree building to ensure each tree is different.





## Random Forest

- Two ways in which the trees in a random forest are randomized
  - **by selecting the data points used to build a tree**
    - **bootstrap**: It leads to each decision tree in the random forest being built on a slightly different dataset.
      - From a list ['a', 'b', 'c', 'd'], possible examples of bootstrap samples are ['b', 'd', 'd', 'c'] and ['d', 'a', 'd', 'a'].
  - **by selecting the features in each split test.**
    - **max_features**: in each node, the algorithm randomly selects a subset of the features, and it looks for the best possible test involving one of these features
      - each node in a tree can make a decision using a different subset of the features.
      - A high **max_features** means that the trees in the random forest will be quite similar, and they will be able to fit the data easily, using the most distinctive features.
      - A low **max_features** means that the trees in the random forest will be quite different, and that each tree might need to be very deep in order to fit the data well.

  * Typically, for a classification problem with p features, √p(rounded down) features are used in each split. For regression problems the inventors recommend p/3(rounded down) as the default. In practice the best value will depend on the problem.

# Feature Importance

- **Feature Importance in Decision Tree**
  - Feature importance summarizes the workings of a tree by rating how important each feature is for the decision the tree makes.
  - The importance of a feature is computed as the total reduction of the criterion brought by that feature. (Thus, it is called as impurity-based feature importance)
  - It is a number between 0 and 1 for each feature, where 0 means “not used at all” and 1 means “perfectly predicts the target.”
- **Feature Importance in Ensemble models**
  - Similar to the decision tree, the ensemble models provide feature importances.
  - Computed by aggregating the feature importances over the trees in the forest.
  - Typically, the feature importances provided by the random forest are more reliable than the ones provided by a single tree.
  - cf. https://scikit-learn.org/stable/auto_examples/inspection/plot_permutation_importance.html