COMP5212: Machine Learning

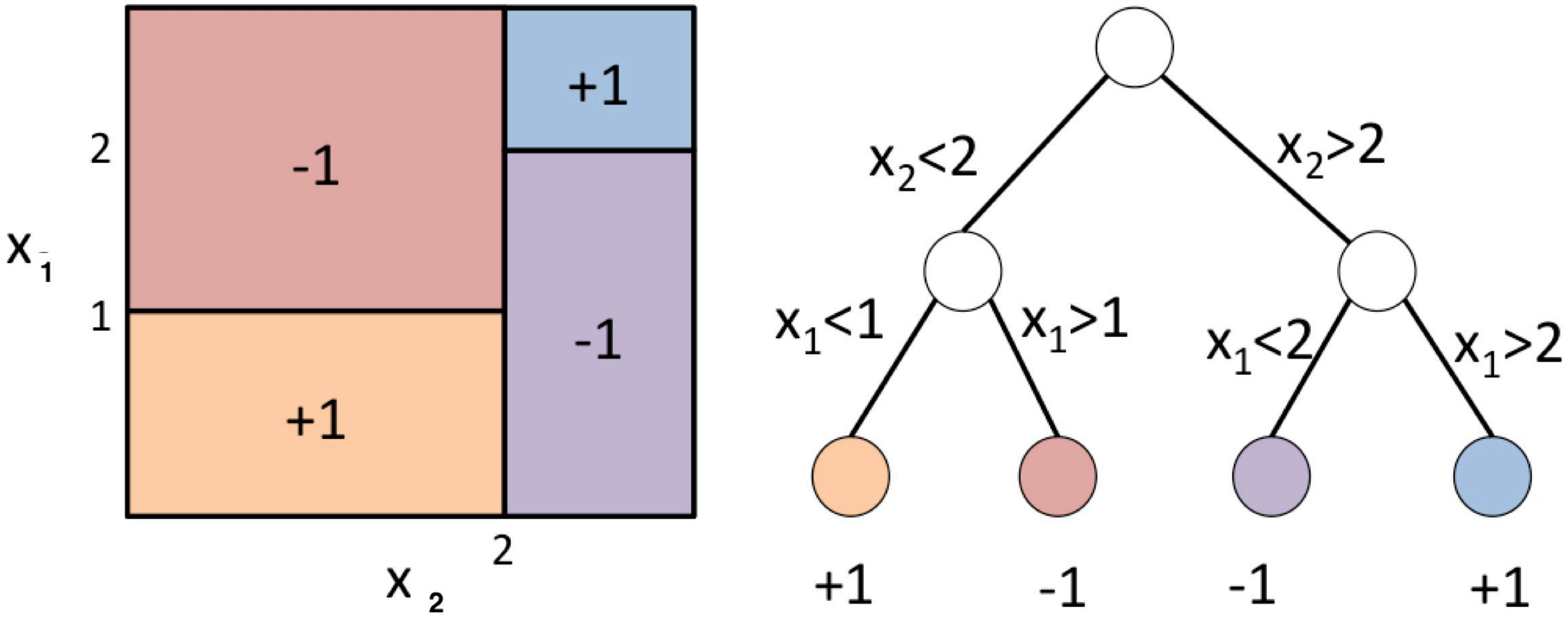
Lecture 11

Minhao Cheng

Decision Tree

Illustration

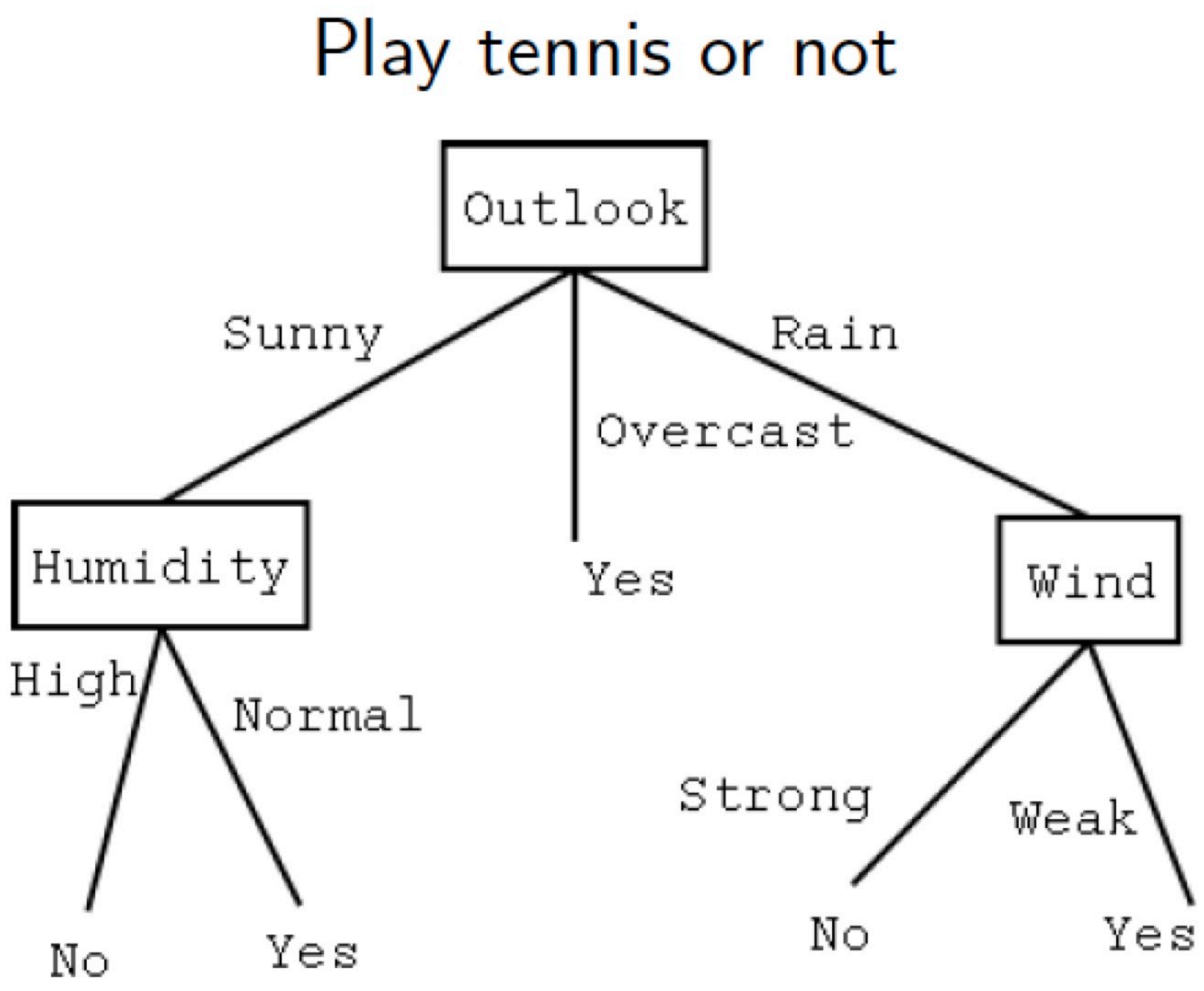
- Each node checks on feature x_i:
  - Go left if $x_i <$ threshold
  - Go right if $x_i >$ threshold



Decision Tree

A real example

- Each node checks on feature x_i:
  - Go left if $x_i <$ threshold
  - Go right if $x_i >$ threshold



Decision Tree

Pros

- Strength:
  - It’s a nonlinear classifier
  - Better interpretability
  - Can naturally handle categorical features

Decision Tree

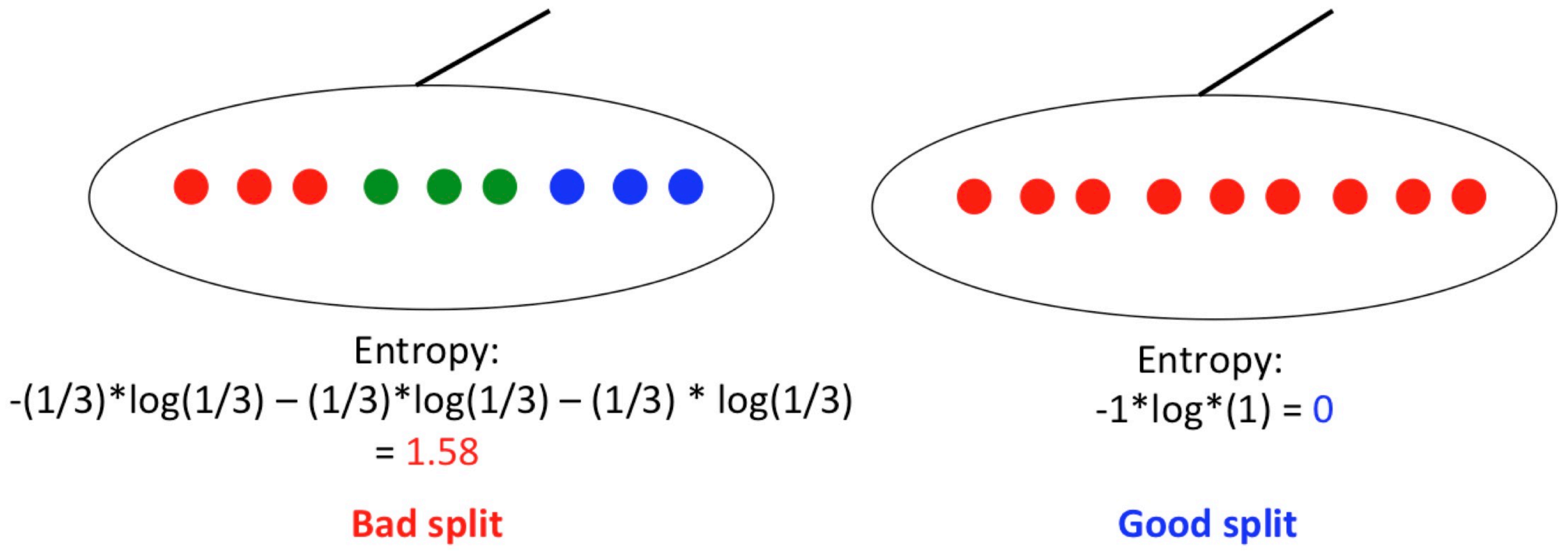
Pros

- Strength:
  - It's a nonlinear classifier
  - Better interpretability
  - Can naturally handle categorical features
- Computation:
  - Training: slow
  - Prediction: fast
    - h operations (h: depth of the tree, usually ≤ 15)

Decision Tree

Splitting the node

- Classification tree: Split the node to maximize entropy
- Let S be set of data points in a node, $c = 1,\ldots,C$ are labels:
  - entropy : $H(S) = -\sum_{c=1}^{C} p(c) \log p(c)$
    - Where $p(c)$ is the proportion of the data belong to class c
      - Entropy=0 if all samples are in the same class
      - Entropy is large if $p(1) = \ldots = p(C)$



Decision Tree

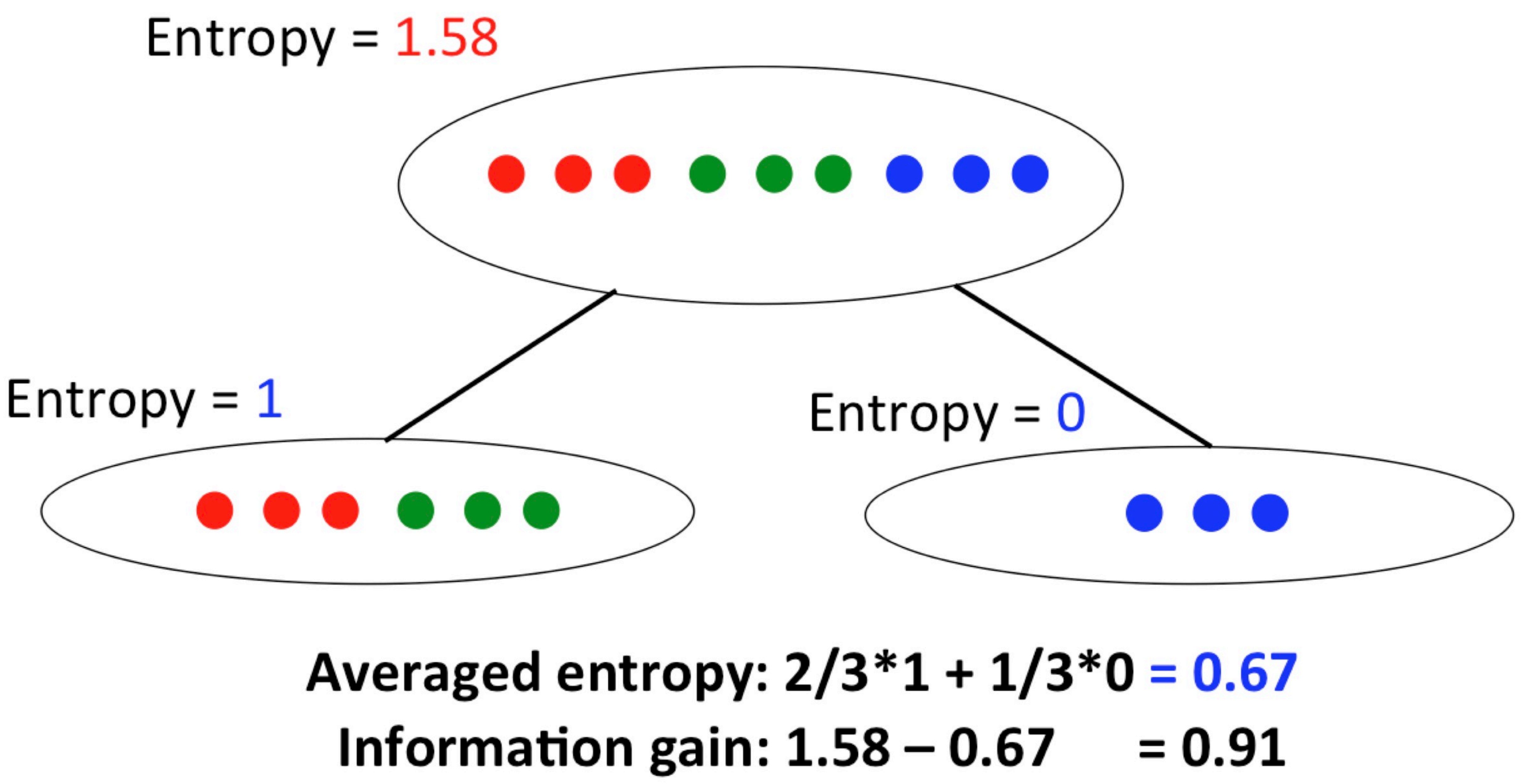
Information Gain

- The averaged entropy of a split $S \to S_1, S_2$
  - $\frac{|S_1|}{|S|}H(S_1) + \frac{|S_2|}{|S|}H(S_2)$
- Information gain: measure how good is the split
  - $H(S) - ((|S_1|/|S|)H(S_1) + (|S_2|/|S|)H(S_2))$

Decision Tree

Information Gain

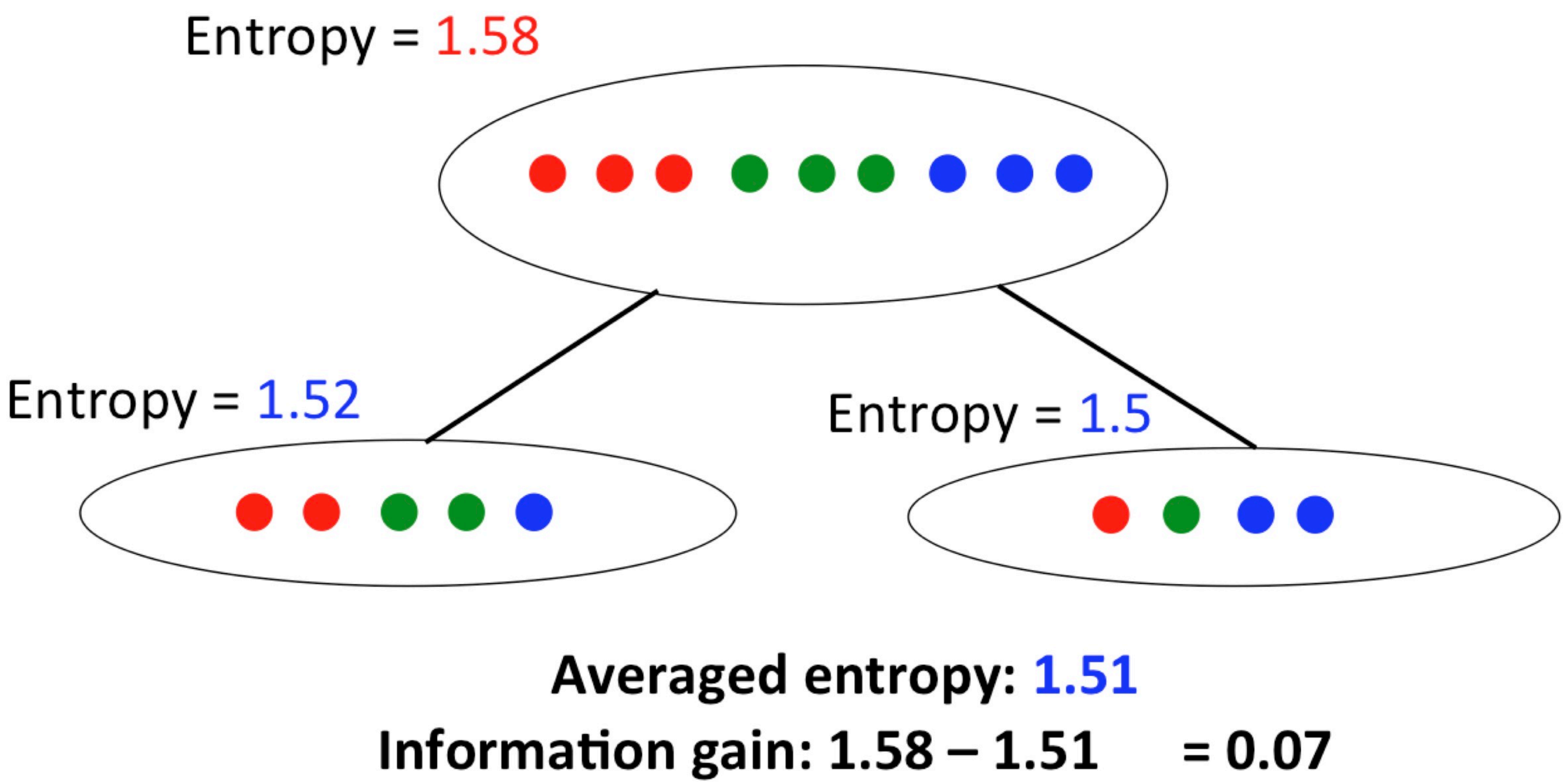
- The averaged entropy of a split $S \to S_1, S_2$
  - $\dfrac{|S_1|}{|S|}H(S_1) + \dfrac{|S_2|}{|S|}H(S_2)$
- Information gain: measure how good is the split
  - $H(S) - ((|S_1|/|S|)H(S_1) + (|S_2|/|S|)H(S_2))$



Decision Tree

Information Gain

- The averaged entropy of a split $S \rightarrow S_1, S_2$
  - $\dfrac{|S_1|}{|S|}H(S_1) + \dfrac{|S_2|}{|S|}H(S_2)$
- Information gain: measure how good is the split
  - $H(S) - ((|S_1|/|S|)H(S_1) + (|S_2|/|S|)H(S_2))$



Decision Tree

Splitting the node

- Given the current note, how to find the best split?

Decision Tree

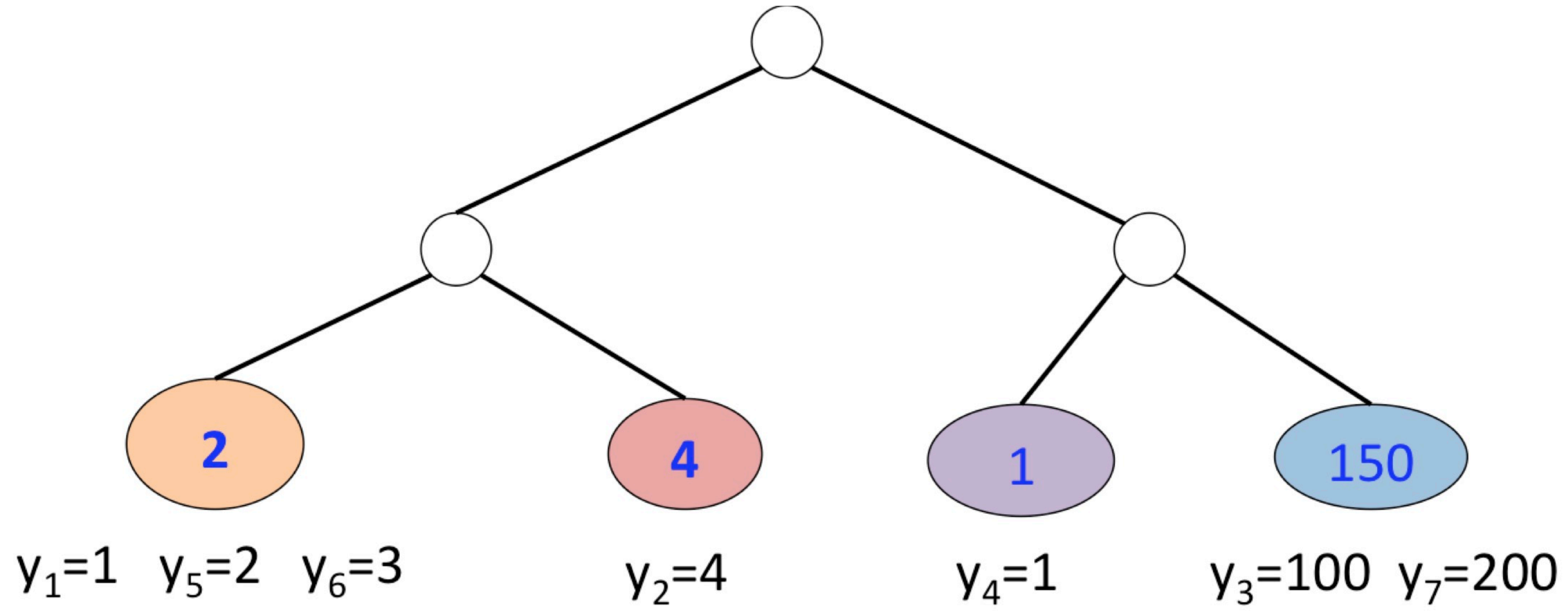
Splitting the node

- Given the current note, how to find the best split?
- For all the features and all the threshold
  - Compute the information gain after the split
  - Choose the best one (maximal information gain)

Decision Tree

Regression Tree

- Assign a real number for each leaf
- Usually average y values for each leaf (minimize square error)



Decision Tree

Regression Tree

- Objective function:
    - $\min_F \frac{1}{n} \sum_{i=1}^{n} (y_i - F(x_i))^2 +$ (Regularization)
- The quality of partition $S = S_1 \cup S_2$ can be computed by the objective function:
    - $\sum_{i \in S_1} (y_i - y^{(1)})^2 + \sum_{i \in S_2} (y_i - y^{(2)})^2,$
    - Where $y^{(1)} = \frac{1}{|S_1|} \sum_{i \in S_1} y_i, y^{(2)} = \frac{1}{|S_2|} \sum_{i \in S_2} y_i$

Decision Tree

Regression Tree

- Objective function:
    - $\min_{F} \frac{1}{n} \sum_{i=1}^{n} (y_i - F(x_i))^2 + \text{(Regularization)}$
- The quality of partition $S = S_1 \cup S_2$ can be computed by the objective function:
    - $\sum_{i \in S_1} (y_i - y^{(1)})^2 + \sum_{i \in S_2} (y_i - y^{(2)})^2,$
    - Where $y^{(1)} = \frac{1}{|S_1|} \sum_{i \in S_1} y_i,\ y^{(2)} = \frac{1}{|S_2|} \sum_{i \in S_2} y_i$
- Find the best split
    - Try all the features & thresholds and find the one with minimal objective function

Decision Tree

Parameters

- Maximum depth: (usually ≈ 10)
- Minimum number of nodes in each node: (10, 50, 100)

Decision Tree

Parameters

- Maximum depth: (usually ≈ 10)
- Minimum number of nodes in each node: (10, 50, 100)
- Single decision tree is not very powerful …
- Can we build multiple decision trees and ensemble them together?

Ensemble methods

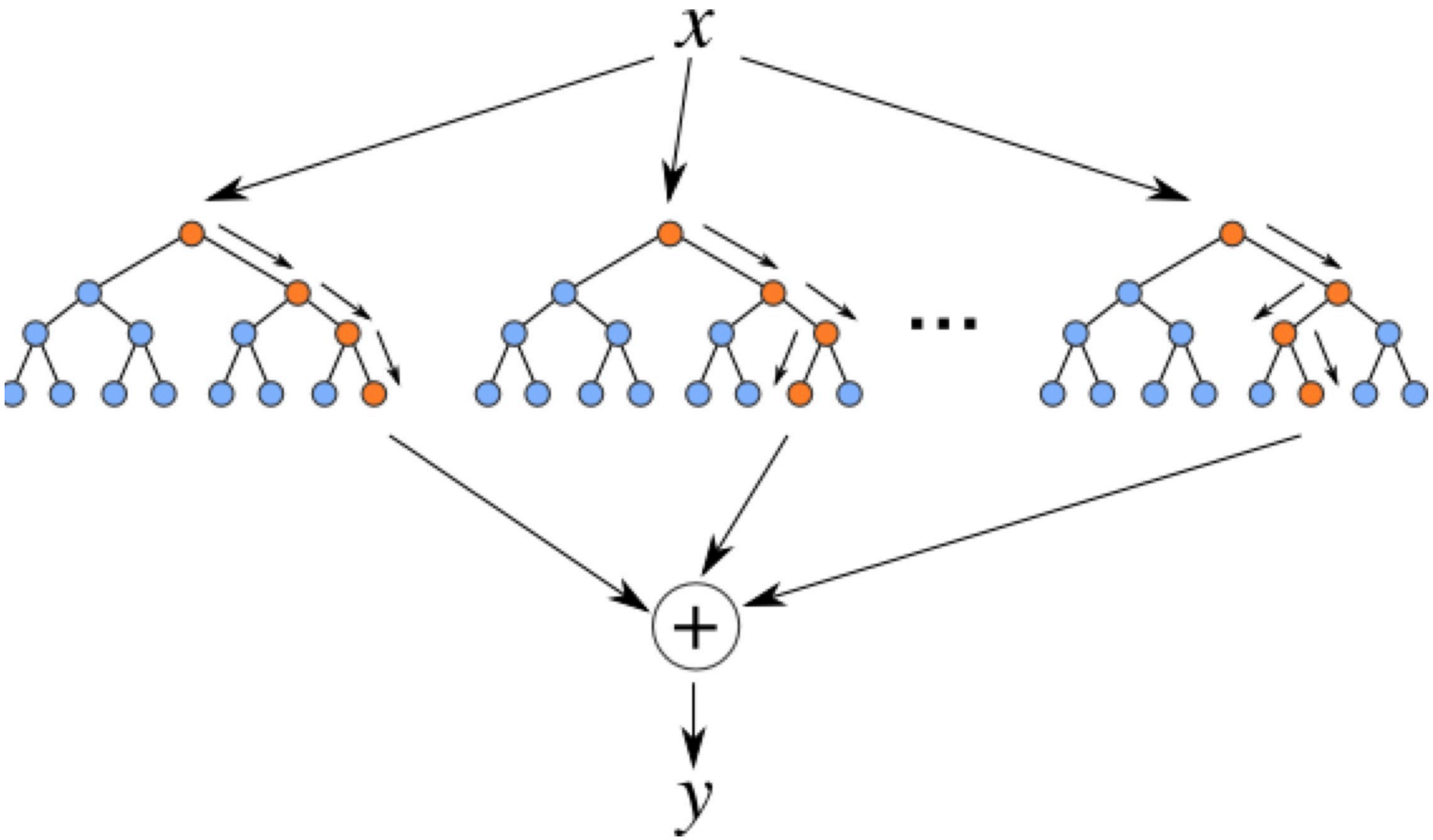
- Bagging
    - Random forest
- Boosting
    - Boosted decision tree

Random Forest
Definition

- Random Forest (Bootstrap ensemble for decision trees):
    - Create T trees
    - Learn each tree using a subsampled dataset S_i and subsampled feature set D_i
    - Prediction: Average the results from all the T trees
- Benefit:
    - Avoid over-fitting
    - Improve stability and accuracy
- Good software available:
    - R: “randomForest” package
    - Python: sklearn

Random Forest

Definition



Gradient Boosted Decision Tree

Boosted Decision Tree

- Minimize loss $\ell(y, F(x))$ with $F(\cdot)$ being ensemble trees
- $F^* = \arg\min_F \sum_{i=1}^{n} \ell(y_i, F(x_i))$ with $F(x) = \sum_{k=1}^{T} f_k(x)$
  - (Each f_k is a decision tree)

Gradient Boosted Decision Tree

Boosted Decision Tree

- Minimize loss $\ell(y, F(x))$ with $F(\cdot)$ being ensemble trees
  - $F^* = \arg\min_F \sum_{i=1}^{n} \ell(y_i, F(x_i))$ with $F(x) = \sum_{k=1}^{T} f_k(x)$
  - (Each f_k is a decision tree)
- Direct loss minimization: at each stage k, find the best function to minimize loss
  - Solve $f_k = \arg\min_{f_k} \sum_{i=1}^{N} \ell(y_i, F_{k-1}(x_i) + f_k(x_i))$
  - Update $F_k \leftarrow F_{k-1} + f_k$
- $F_k(x) = \sum_{j=1}^{k} f_j(x)$ is the prediction of x after k iterations

Gradient Boosted Decision Tree

Boosted Decision Tree

- Minimize loss $\ell(y, F(x))$ with $F(\cdot)$ being ensemble trees
  - $F^* = \arg\min_F \sum_{i=1}^{n} \ell(y_i, F(x_i))$ with $F(x) = \sum_{k=1}^{T} f_k(x)$
  - (Each f_k is a decision tree)
- Direct loss minimization: at each stage k, find the best function to minimize loss
  - Solve $f_k = \arg\min_{f_k} \sum_{i=1}^{N} \ell(y_i, F_{k-1}(x_i) + f_k(x_i))$
  - Update $F_k \leftarrow F_{k-1} + f_k$
- $F_k(x) = \sum_{j=1}^{k} f_j(x)$ is the prediction of x after k iterations
- Two problems:
  - Hard to implement for general loss
  - Tend to overfit training data

Gradient Boosted Decision Tree

Boosted Decision Tree

- Let $\hat{y}_i = \sum_{k=1}^{T} f_k(x_i), f_k \in F$
  - $\hat{y}_i^{(0)} = 0$
  - $\hat{y}_i^{(1)} = f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i)$
  - …
  - $\hat{y}_i^{(t)} = \sum_{k=1}^{t} f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)$

Gradient Boosted Decision Tree

Boosted Decision Tree

- Let $\hat{y}_i = \sum_{k=1}^{T} f_k(x_i), f_k \in F$
  - $\hat{y}_i^{(0)} = 0$
  - $\hat{y}_i^{(1)} = f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i)$
  - …
  - $\hat{y}_i^{(t)} = \sum_{k=1}^{t} f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)$
- Consider MSE error is used:
  - $\text{obj}^{(t)} = \sum_{i=1}^{n} (y_i - (\hat{y}_i^{(t-1)} + f_t(x_i)))^2 = \sum_{i=1}^{n} [2(\hat{y}_i^{(t-1)} - y_i)f_t(x_i) + f_t(x_i)^2] + \text{constant}$

Gradient Boosted Decision Tree

Boosted Decision Tree

- Let $\hat{y}_i = \sum_{k=1}^{T} f_k(x_i), f_k \in F$
- Consider MSE error is used:
  - $\text{obj}^{(t)} = \sum_{i=1}^{n} (y_i - (\hat{y}_i^{(t-1)} + f_t(x_i)))^2 = \sum_{i=1}^{n} [2(\hat{y}_i^{(t-1)} - y_i)f_t(x_i) + f_t(x_i)^2] + \text{constant}$

Gradient Boosted Decision Tree

Boosted Decision Tree

- Let $\hat{y}_i = \sum_{k=1}^{T} f_k(x_i), f_k \in F$
- Consider general loss
  - Use Taylor expansion
  - obj$^{(t)} = \sum_{i=1}^{n} [\ell(y_i, \hat{y}_i^{(t-1)})) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)]$ + constant
  - Where $g_i = \partial_{\hat{y}_i^{(t-1)}} \ell(y_i, \hat{y}_i^{(t-1)})$ is gradient, $h_i = \partial^2_{\hat{y}_i^{(t-1)}} \ell(y_i, \hat{y}_i^{(t-1)})$ is second order derivative

Gradient Boosted Decision Tree

Boosted Decision Tree

- Let $\hat{y}_i = \sum_{k=1}^{T} f_k(x_i), f_k \in F$
- Consider general loss
  - Use Taylor expansion
  - obj$^{(t)} = \sum_{i=1}^{n} [\ell(y_i, \hat{y}_i^{(t-1)})) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)]$ + constant
  - Where $g_i = \partial_{\hat{y}_i^{(t-1)}} \ell(y_i, \hat{y}_i^{(t-1)})$ is gradient, $h_i = \partial^2_{\hat{y}_i^{(t-1)}} \ell(y_i, \hat{y}_i^{(t-1)})$ is second order derivative
  - The object only depends on g_i, h_i

Gradient Boosted Decision Tree

Boosted Decision Tree

- Let $\hat{y}_i = \sum_{k=1}^{T} f_k(x_i), f_k \in F$
- Consider general loss
  - Use Taylor expansion
  - $\text{obj}^{(t)} = \sum_{i=1}^{n} [\ell(y_i, \hat{y}_i^{(t-1)})) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \text{constant}$
  - Where $g_i = \partial_{\hat{y}_i^{(t-1)}} \ell(y_i, \hat{y}_i^{(t-1)})$ is gradient, $h_i = \partial^2_{\hat{y}_i^{(t-1)}} \ell(y_i, \hat{y}_i^{(t-1)})$ is second order derivative
  - The object only depends on $f_t(x_i)$
  - Get rid of constant term
    - $\text{obj}^{(t)} = \sum_{i=1}^{n} [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \text{constant} = \sum_{i=1}^{n} \frac{h_i}{2} (f_t(x_i) - g_i/h_i)^2 + \text{constant}$

Gradient Boosted Decision Tree

Gradient Boosted Decision Tree (GBDT)

- Finding $f_k(x)$ by minimizing the loss function:
- $$\arg\min_{f_k} \sum_{i=1}^{N} [f_k(x_i) - g_i/h_i]^2 + R(f_k)$$
  - Reduce the training of any loss function to regression tree (just need to compute g_i for different functions)
  - $h_i = \alpha$ (fixed step size) for original GBDT
  - XGboost shows computing second order derivate yields better performance

Gradient Boosted Decision Tree

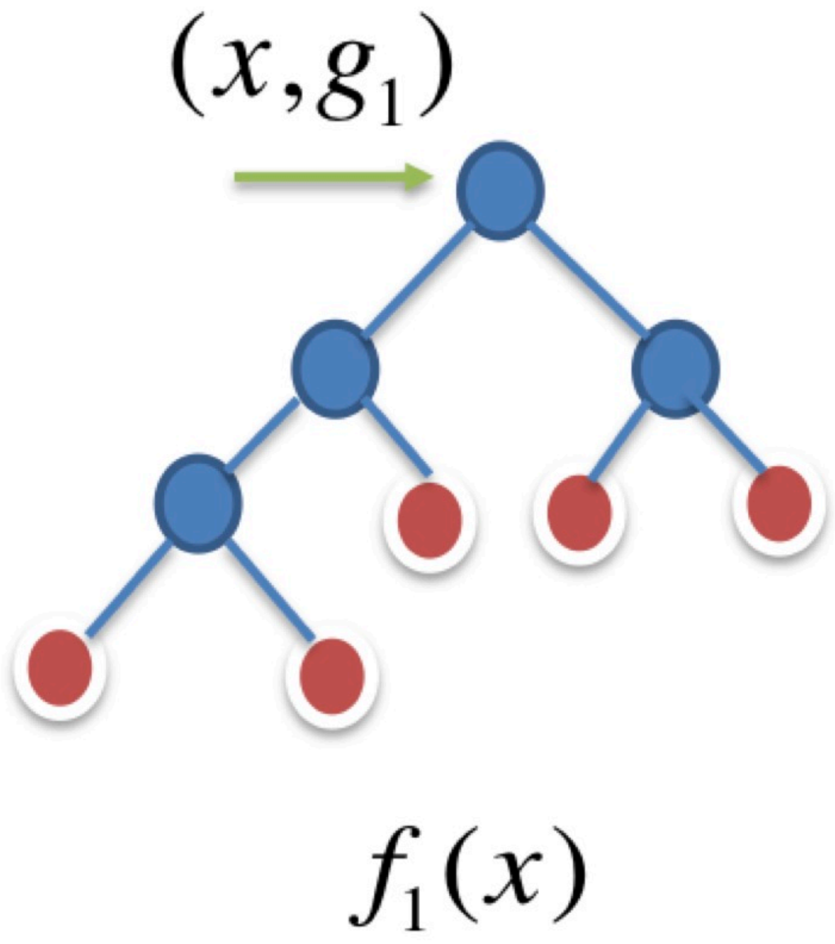
Gradient Boosted Decision Tree (GBDT)

- Finding $f_k(x)$ by minimizing the loss function:
  - $\arg\min_{f_k} \sum_{i=1}^{N} [f_k(x_i) - g_i/h_i]^2 + R(f_m)$
    - Reduce the training of any loss function to regression tree (just need to compute g_i for different functions)
    - $h_i = \alpha$ (fixed step size) for original GBDT
    - XGboost shows computing second order derivate yields better performance
- Algorithm:
  - Computing the current gradient for each $\hat{y}_i$
  - Building a base learner (decision tree) to fit the gradient
  - Updating current prediction $\hat{y}_i = F_k(x_i)$ for all i

Gradient Boosted Decision Tree

Gradient Boosted Decision Tree (GBDT)

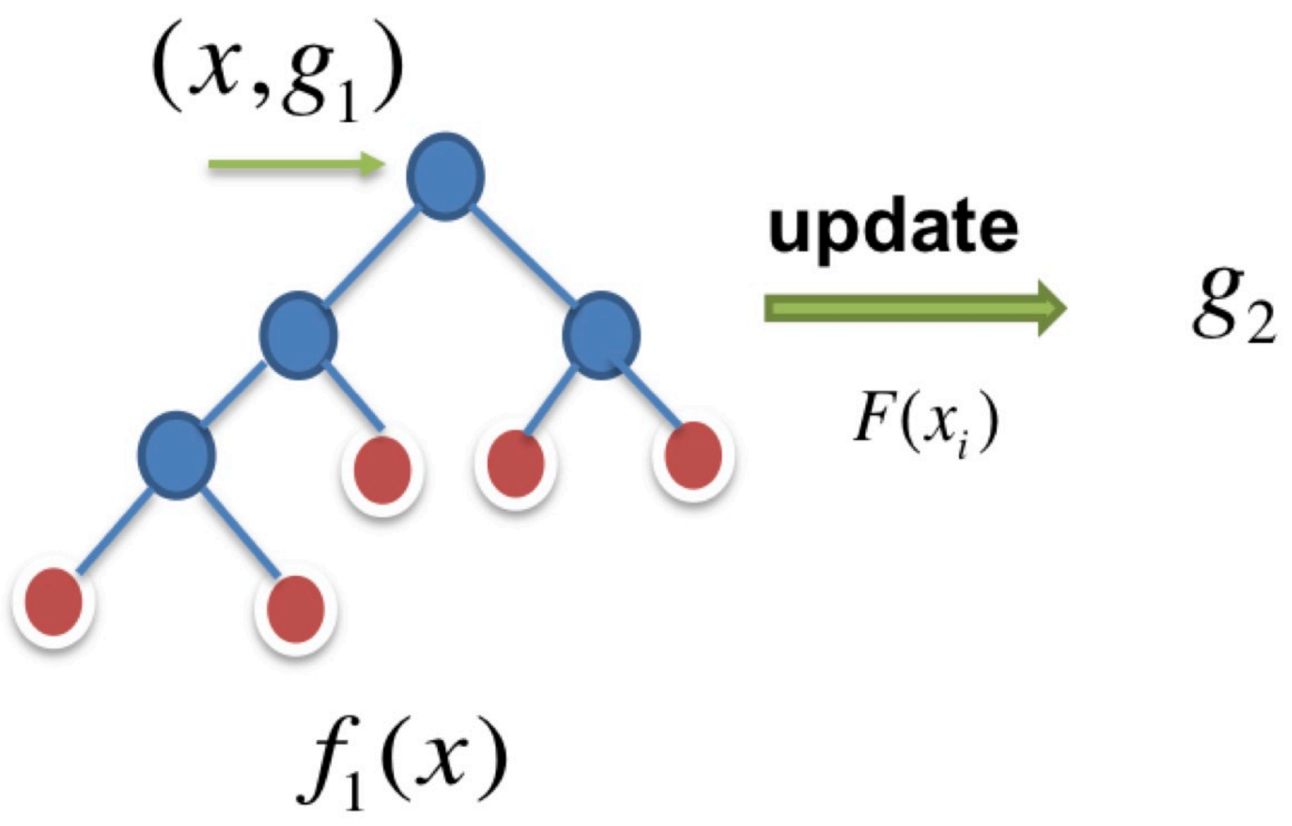
- Key idea:
  - Each base learner is a decision tree
  - Each regression tree approximates the functional gradient $\frac{\partial \ell}{\partial F}$



Gradient Boosted Decision Tree

Gradient Boosted Decision Tree (GBDT)

- Key idea:
    - Each base learner is a decision tree
    - Each regression tree approximates the functional gradient $\frac{\partial \ell}{\partial F}$

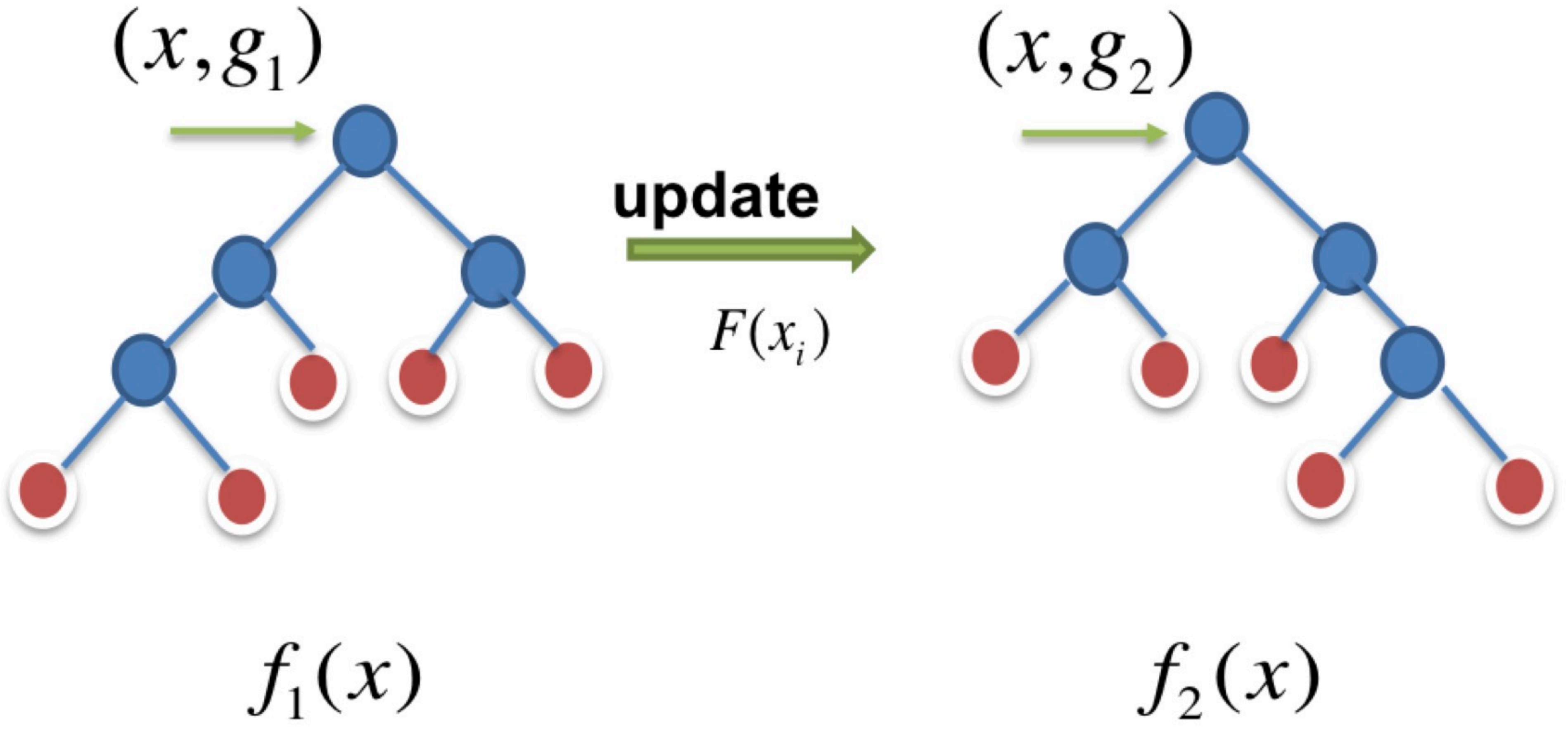


$$F_{m-1}(x_i) = \sum_{j=1}^{m-1} f_j(x_i) \qquad g_m(x_i) = \frac{\partial \ell(y_i, F(x_i))}{\partial F(x_i)}\bigg|_{F(x_i)=F_{m-1}(x_i)}$$

Gradient Boosted Decision Tree

Gradient Boosted Decision Tree (GBDT)

- Key idea:
  - Each base learner is a decision tree
  - Each regression tree approximates the functional gradient $\dfrac{\partial \ell}{\partial F}$

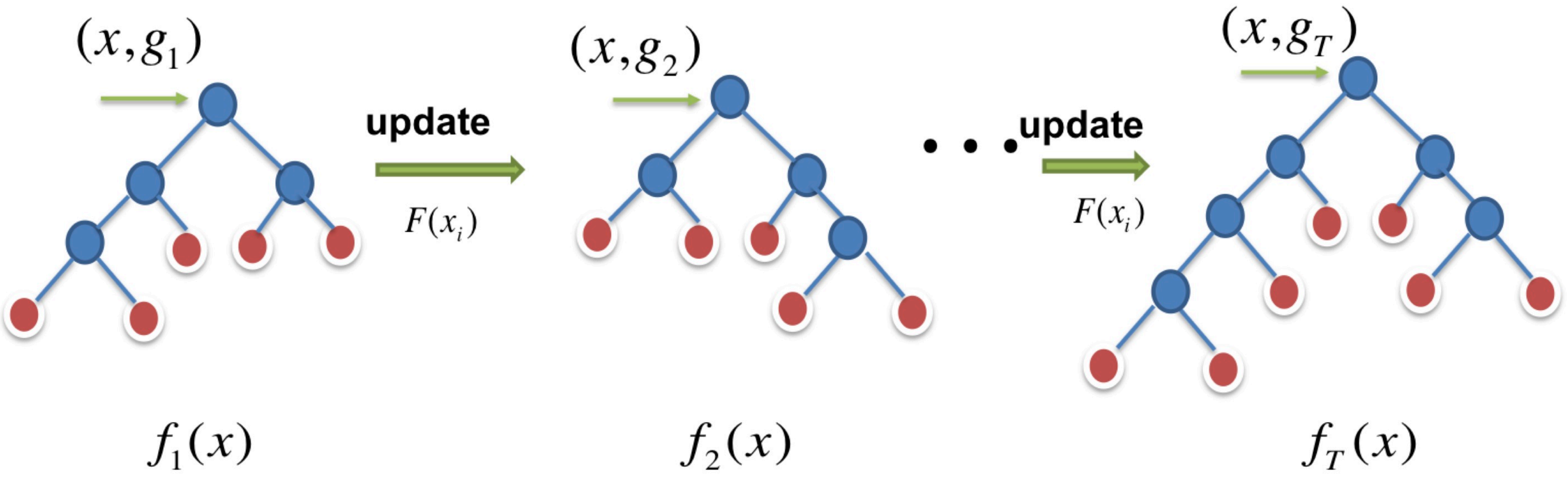


$$F_{m-1}(x_i) = \sum_{j=1}^{m-1} f_j(x_i) \qquad g_m(x_i) = \left.\frac{\partial \ell(y_i, F(x_i))}{\partial F(x_i)}\right|_{F(x_i)=F_{m-1}(x_i)}$$

Gradient Boosted Decision Tree

Gradient Boosted Decision Tree (GBDT)

- Key idea:
  - Each base learner is a decision tree
  - Each regression tree approximates the functional gradient $\dfrac{\partial \ell}{\partial F}$

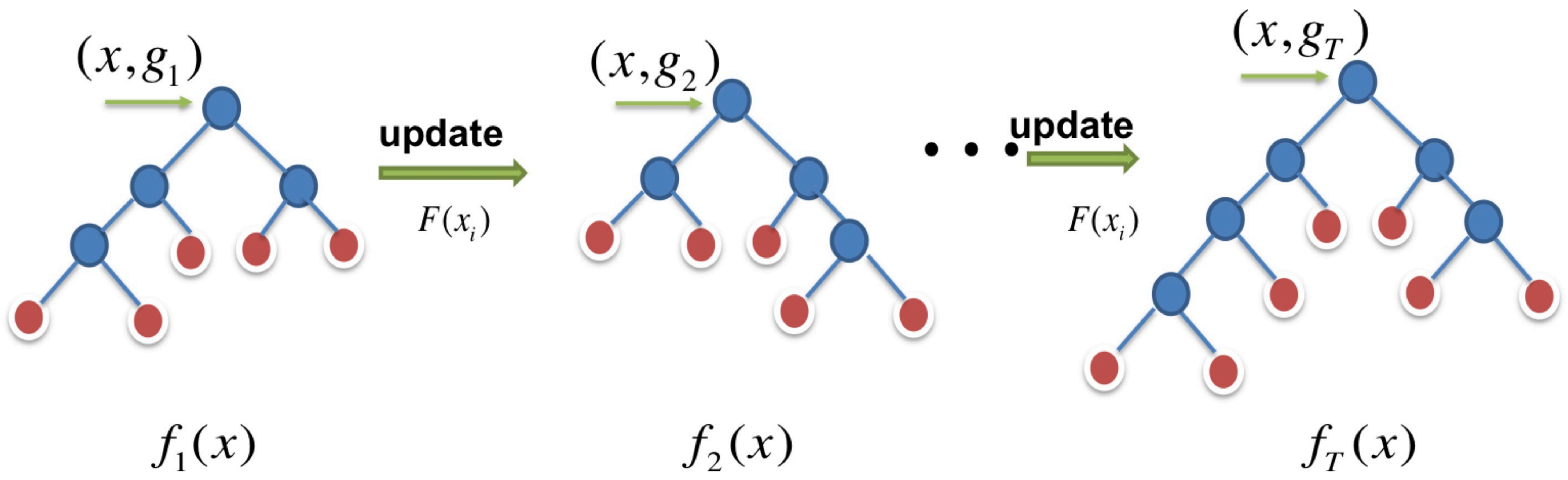


$$F_{m-1}(x_i) = \sum_{j=1}^{m-1} f_j(x_i) \quad g_m(x_i) = \left.\frac{\partial \ell(y_i, F(x_i))}{\partial F(x_i)}\right|_{F(x_i)=F_{m-1}(x_i)}$$

Gradient Boosted Decision Tree

Gradient Boosted Decision Tree (GBDT)

- Key idea:
  - Each base learner is a decision tree
  - Each regression tree approximates the functional gradient $\dfrac{\partial \ell}{\partial F}$



Final prediction $F(x_i) = \sum_{j=1}^{T} f_j(x_i)$

Gradient Boosted Decision Tree

Open source packages

- XGBoost: the first widely used tree-boosting software
- LightGBM: released by Microsoft
  - Histogram-based training approach — much faster than finding the best split