



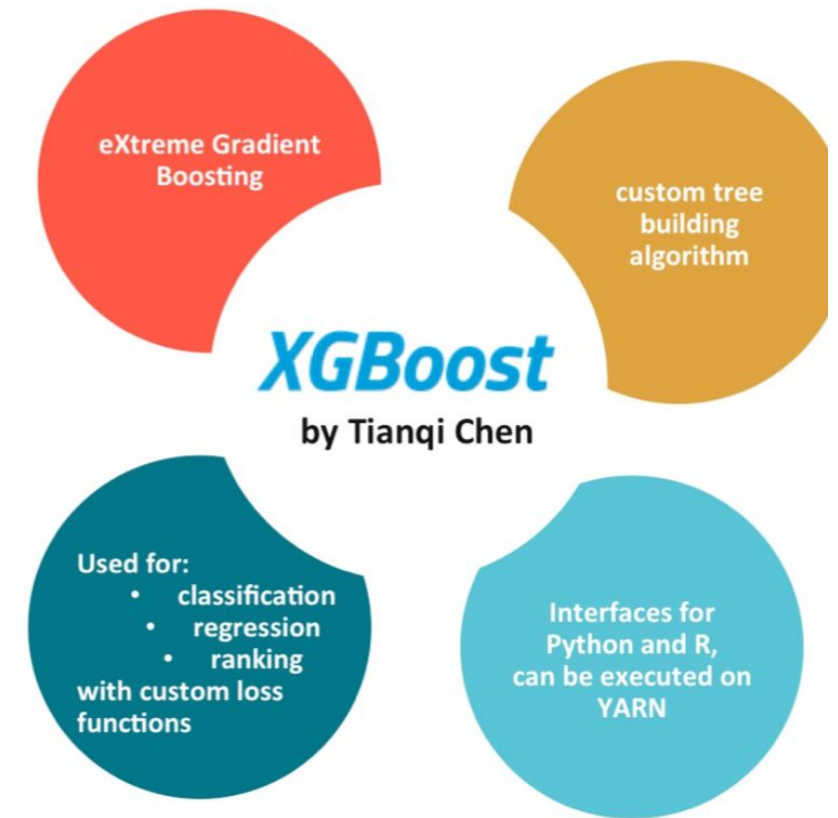
# XGBoost Explained (eXtreme Gradient Boosting)

Dr. Risman Adnan Mattotorang  
Telkom University

# What is XGBoost

---

- XGBoost = CART ensemble model
- Scalable ML system for tree boosting
- Widely used in Kaggle Competition
- Can be used to handle sparse data
- **Guide:** Focus on high level concepts.
  - Detail algorithm is on paper but ....
  - Better focus on using the library
- This lecture is to explain how **tree boosting** works before you hack it 😊
- If you need more explanation, look on references.



# Key Concepts

1. **Supervised Learning Summary**
  - Separation of Model, Parameters, Objective Function, Regularization
  - Generalization Trade Off (VC Dimension and Bias-Variance)
2. **Decision Tree and CART Ensemble Model**
3. **Gradient Boosting Algorithm**
  - Exact Greedy Algorithm
  - Sparsity-Aware Split Finding
  - Pruning and Regularization
4. **Summary**

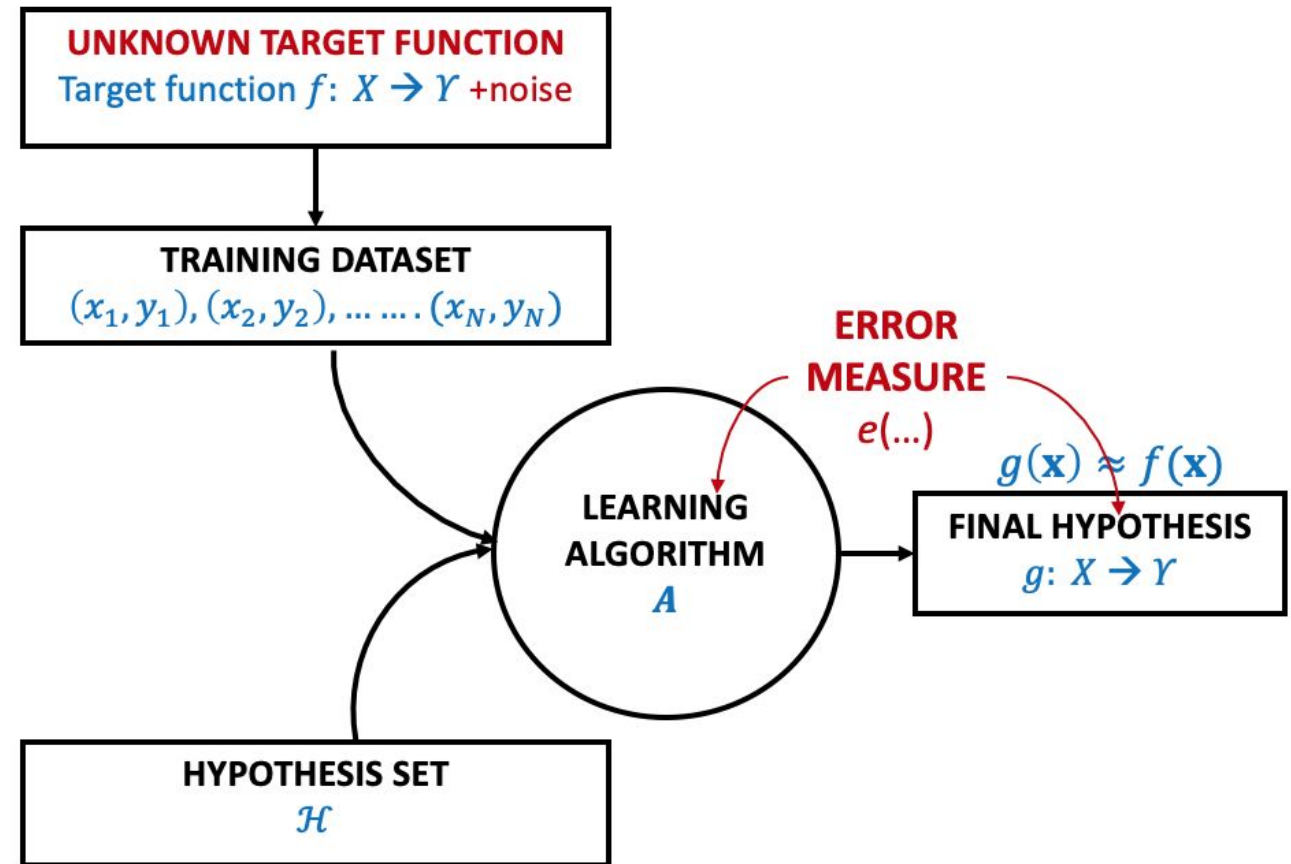
# Supervised Learning Problem

## Model

- Model = hypothesis set + algorithm
- Model makes prediction  $\hat{y}_i$  given  $x_i$
- Prediction  $\hat{y}_i$  can have many interpretations:
  - Regression:**  $\hat{y}_i$  is predicted values
  - Classifications:**  $\hat{y}_i$  is predicted class labels
  - Ranking:**  $\hat{y}_i$  is predicted score

## Parameter

- Things we need to learn from data
- Linear model:  $\hat{y}_i = \sum_j w_j x_{ij}$ ,  $\theta = \{w_j\}_{j=1}^d$
- We learn by optimizing **error measure!**



# Error Measure vs Objective Function

- What does  $h \approx f$  mean? Error measure:  $E(h, f)$
- Error measure is “cost of using  $h$  when we should use  $f$ ” or simply **cost function** (or loss)
- Almost always pointwise definition:  $e(h(\mathbf{x}), f(\mathbf{x}))$  for examples:
  - Squared Error** :  $e(h(\mathbf{x}), f(\mathbf{x})) = (h(\mathbf{x}) - f(\mathbf{x}))^2$
  - Binary Error** :  $e(h(\mathbf{x}), f(\mathbf{x})) = (h(\mathbf{x}) - f(\mathbf{x}))^2$
- Total error  $E(h, f) =$  average of pointwise error  $e(h(\mathbf{x}), f(\mathbf{x}))$ 
  - In-sample Error** :  $E_{in}(h) = \frac{1}{n} \sum_{i=1}^n e(h(\mathbf{x}_i), f(\mathbf{x}_i))$
  - Out-of-sample Error** :  $E_{out}(h) = \mathbb{E}_{\mathbf{x}}[e(h(\mathbf{x}), f(\mathbf{x}))]$
- As learning goal is to **minimize total cost**, sometime we called it **objective function**.

# Objective Function

- **Objective function** is “everywhere” in machine learning models

$$Obj(\Theta) = L(\Theta) + \Omega(\Theta)$$

**Training cost** measures how well model fit on training data

**Regularization** measures complexity of the models

- **Training Cost:**  $L = \sum_{i=1}^n e(f(x), h(x)) = \sum_{i=1}^n e(y_i, \hat{y}_i)$ 
  - Square Error :  $e(y_i, \hat{y}_i) = (y_i - \hat{y}_i)^2$
  - Logistic Error :  $e(y_i, \hat{y}_i) = y_i \ln(1 + e^{-\hat{y}_i}) + (1 - y_i) \ln(1 + e^{\hat{y}_i})$
- **Regularization:**
  - L2 Norm :  $\Omega(w) = \lambda \|w\|^2$
  - L1 Norm (Lasso) :  $\Omega(w) = \lambda \|w\|_1$



# Conceptual Examples

- **Ridge Regression:**  $\sum_{i=1}^n (y_i - w^T x_i)^2 + \lambda \|w\|^2$ 
  - Linear model, square cost, L2 regularization
- **Lasso Regression:**  $\sum_{i=1}^n (y_i - w^T x_i)^2 + \lambda \|w\|_1$ 
  - Linear model, square cost, L1 regularization
- **Logistic Regression:**  $\sum_{i=1}^n [y_i \ln(1 + e^{-w^T x_i}) + (1 - y_i) \ln(1 + e^{w^T x_i})] + \lambda \|w\|^2$ 
  - Linear model, logistic cost, L2 regularization
- **Engineering:** Define Objective (Cost, Regularization) and Optimize (Eq. SGD).

# Generalization Trade Off

- **Objective function** is “everywhere” in machine learning models

$$Obj(\Theta) = L(\Theta) + \Omega(\Theta)$$

**Training cost** measures how well model **fit** on training data

**Regularization** measures **complexity** of the models

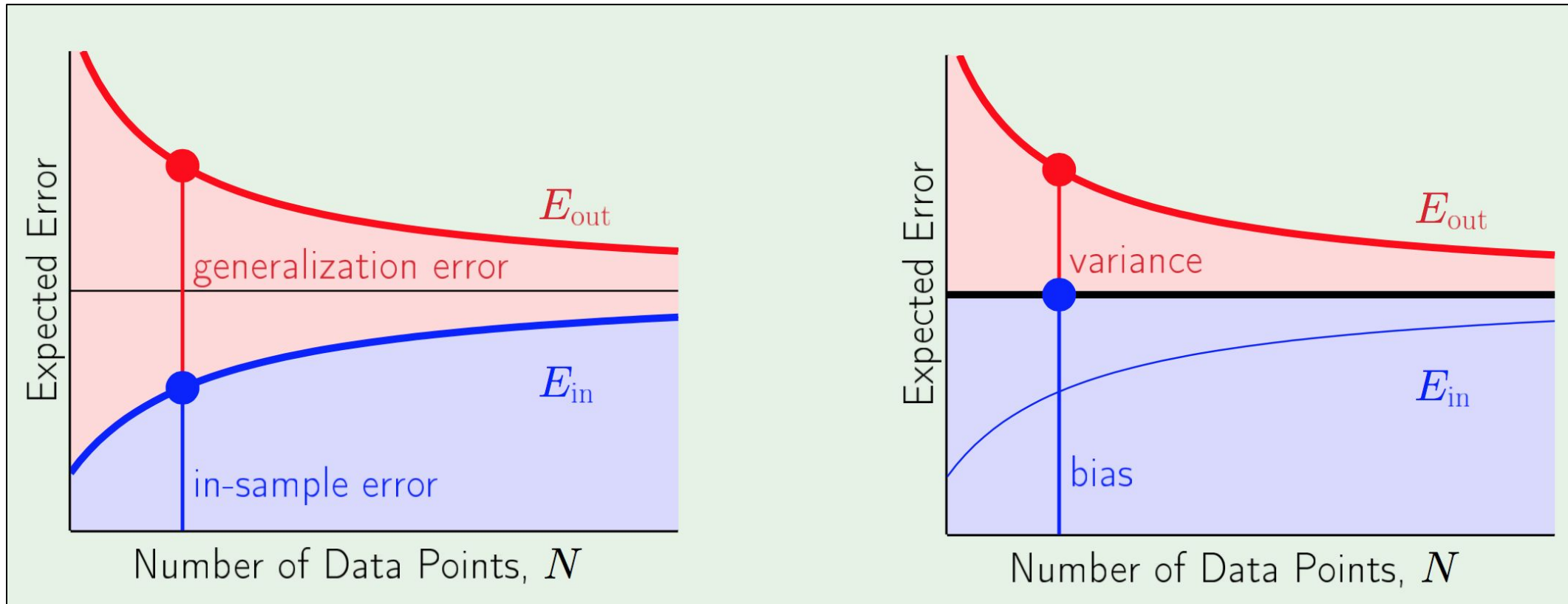
- **Why do we want to contain two component in objective function?**

- VC Dimension :  $E_{out} \leq E_{in} + \xi$  (complexity penalty)
- Bias Variance :  $E_{out} \approx Bias + Var$

- **Optimizing training cost** encourage predictive models (smaller bias)
- **Optimizing regularization** encourage simple models (smaller variance, stable)



# VC and Bias-Variance Trade Off



**VC Analysis**  $E_{out} \approx E_{in} + \Omega$

**Bias Variance**  $E_{out} \approx Bias + Var$

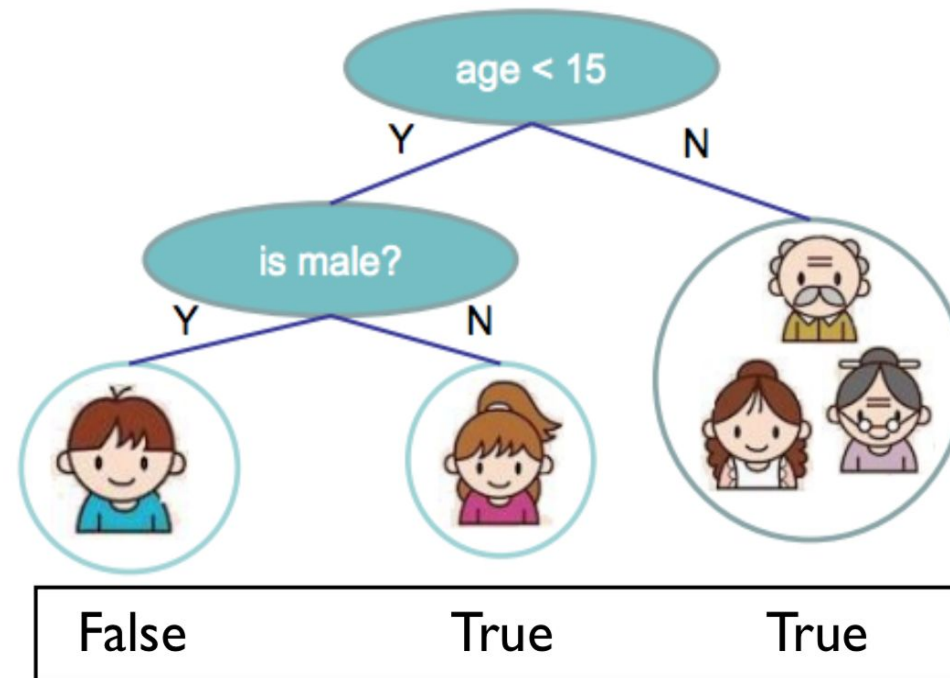
# Key Concepts

1. **Supervised Learning Summary**
  - Separation of Model, Parameters, Objective Function, Regularization
  - Generalization Trade Off (VC Dimension and Bias-Variance)
2. **Decision Tree and CART Ensemble Model**
3. **Gradient Boosting Algorithm**
  - Exact Greedy Algorithm
  - Sparsity-Aware Split Finding
  - Pruning and Regularization
4. **Summary**

# Decision Tree

**Input:** BMI, Age, Sex

**Output:** Has Diabetes?



True or False (1 or 0) in each leaf

# CART

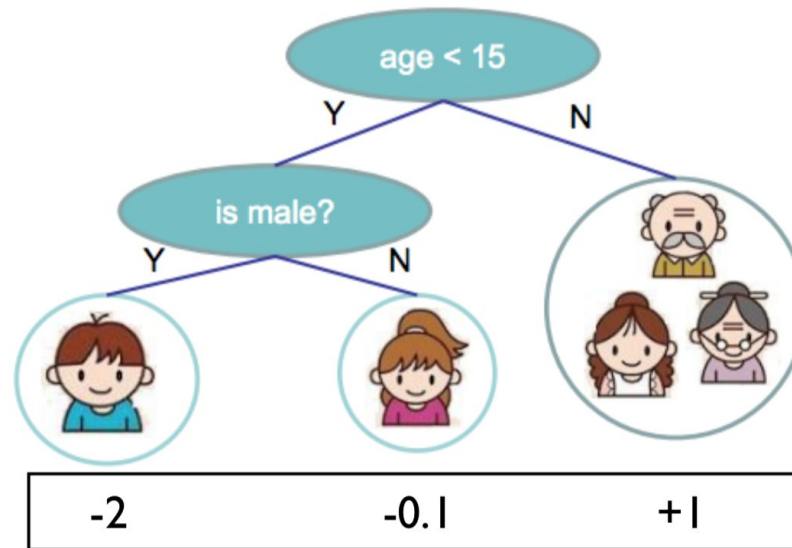
- **CART = Classification and Regression Tree**

- Decision rules same as decision tree

**Input:** BMI, Age, Sex



**Output:** Has Diabetes?

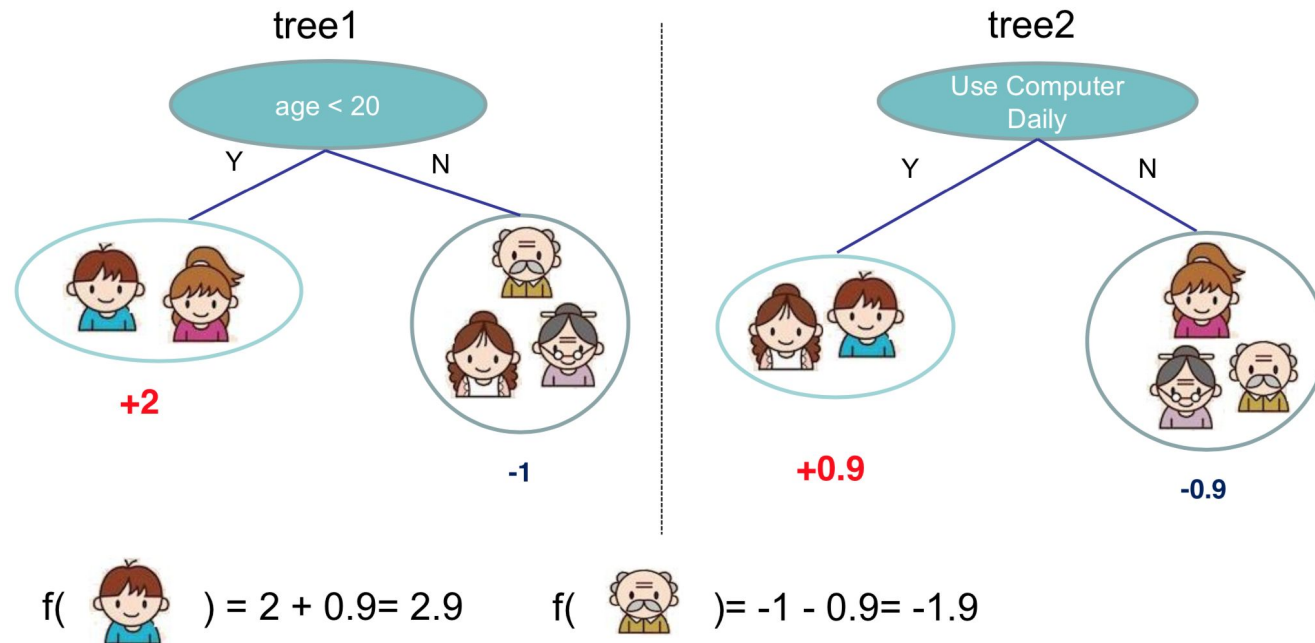


- Contains one score in each leaf value.
  - **Think:** A function that maps the attributes to the score.

# CART Ensemble Model

**Input:** age, gender, occupation..

**Output:** Like Minecraft?



## Benefits of Ensemble

- Widely used GBM, Random Forest, etc.
- Invariant to scaling of input.
- Don't need features normalization
- Learn higher order interaction of features
- Scalable and are used in industry

**Ensemble:** Use multiple models to get better performance. Expand Hypothesis Set.

**Example:** Bagging and Boosting (Such as XGBoost)

# Objective Function Formulation

- **Model:** Assuming we have  $K$  trees.  $\hat{y}_i = \sum_{k=1}^K f_k(x_i), \quad f_k \in \mathcal{F}$

Space of functions containing all Regression trees

- **Think:** Regression tree is a function that maps the attributes to the score.

- **Parameters:**

- Including structure of each tree, and the score in the leaf.
- Or simply use function as parameters  $\Theta = \{f_1, f_2, \dots, f_K\}$
- Instead of learning weights, we are learning functions (trees).

- **Objective Function:**  $Obj = \sum_{i=1}^n l(y_i, \hat{y}_i) + \sum_{k=1}^K \Omega(f_k)$   
Training loss                      Complexity of the Trees

# Complexity of CART Ensemble

- **Possible and obvious ways:**

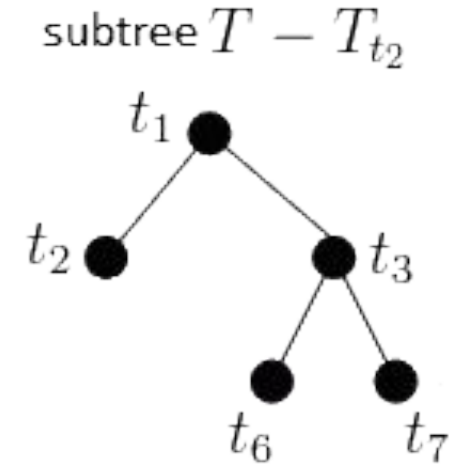
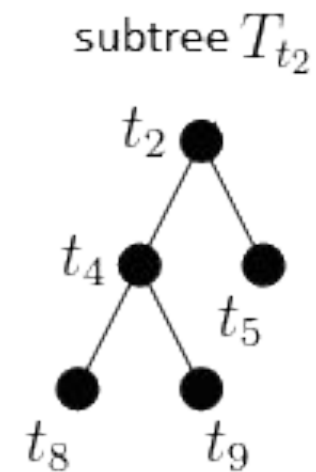
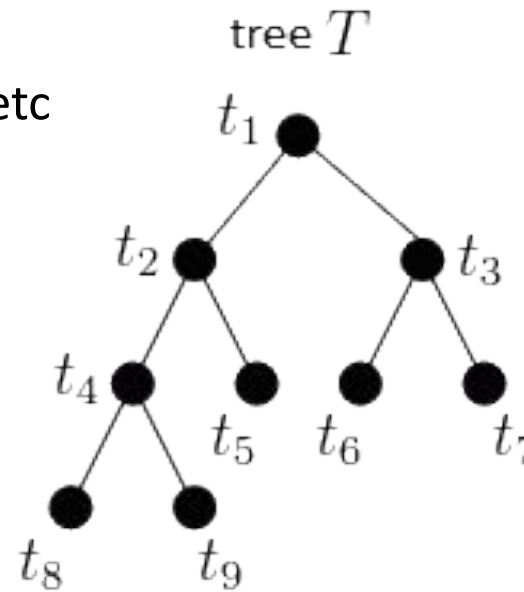
- Number of nodes in tree, depth, etc

- **Heuristic view of complexities:**

1. Split by information gain
2. Prune the tree
3. Maximum depth
4. Smooth the leaf values

- **Theoretical view of complexities:**

1. Training cost/loss
2. Regularization defined by #nodes
3. Constraint on function space
4. L2 regularization of leaf weight



QA

Training Cost + Regularization objective pattern applies for CART ensemble model. But how do we learn it?



# Key Concepts

1. **Supervised Learning Summary**
  - Separation of Model, Parameters, Objective Function, Regularization
  - Generalization Trade Off (VC Dimension and Bias-Variance)
2. **Decision Tree and CART Ensemble Model**
3. **Gradient Boosting Algorithm**
  - Exact Greedy Algorithm
  - Sparsity-Aware Split Finding
  - Pruning and Regularization
4. **Summary**

# So How Do We Learn?

- **Objective Function:**  $\sum_{i=1}^n l(y_i, \hat{y}_i) + \sum_k \Omega(f_k), f_k \in \mathcal{F}$
- **Cant use SGD since they are trees not numerical vectors!**

- **Solution: Additive Training (Boosting)**

- Start from constant prediction, add new function each time.

$$\hat{y}_i^{(0)} = 0$$

$$\hat{y}_i^{(1)} = f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i)$$

$$\hat{y}_i^{(2)} = f_1(x_i) + f_2(x_i) = \hat{y}_i^{(1)} + f_2(x_i)$$

...

$$\hat{y}_i^{(t)} = \sum_{k=1}^t f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)$$

← New function

Model at training round t

Keep functions added in previous round

# Additive Training (Boosting)

- **How we decide which  $f$  to add?**

- Optimize the objective!!

- **Prediction at round  $t$  is:**  $\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t(x_i)$

$$\begin{aligned} Obj^{(t)} &= \sum_{i=1}^n l(y_i, \hat{y}_i^{(t)}) + \sum_{i=1}^t \Omega(f_i) \\ &= \sum_{i=1}^n l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) + constant \end{aligned}$$

- **Consider square error:**

Goal: find  $f_t$  to minimize this

$$\begin{aligned} Obj^{(t)} &= \sum_{i=1}^n \left( y_i - (\hat{y}_i^{(t-1)} + f_t(x_i)) \right)^2 + \Omega(f_t) + const \\ &= \sum_{i=1}^n \left[ 2(\hat{y}_i^{(t-1)} - y_i)f_t(x_i) + f_t(x_i)^2 \right] + \Omega(f_t) + const \end{aligned}$$

This is usually called residual from previous round

# Taylor Expansion for Approximation

- **Goal:**  $Obj^{(t)} = \sum_{i=1}^n l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) + constant$

- Seems still complicated except for the case of square error

- **Take Taylor expansion of the objective**

- Recall :  $f(x + \Delta x) \simeq f(x) + f'(x)\Delta x + \frac{1}{2}f''(x)\Delta x^2$

- Define:  $g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}), \quad h_i = \partial_{\hat{y}^{(t-1)}}^2 l(y_i, \hat{y}^{(t-1)})$

- If you are not comfortable with this, think of square error:

$$Obj^{(t)} \simeq \sum_{i=1}^n \left[ l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t) + constant$$

- **Compare to what we got in previous slides.**

# New Objective Function

- **New objective with constant removed:**

$$\sum_{i=1}^n \left[ g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t)$$

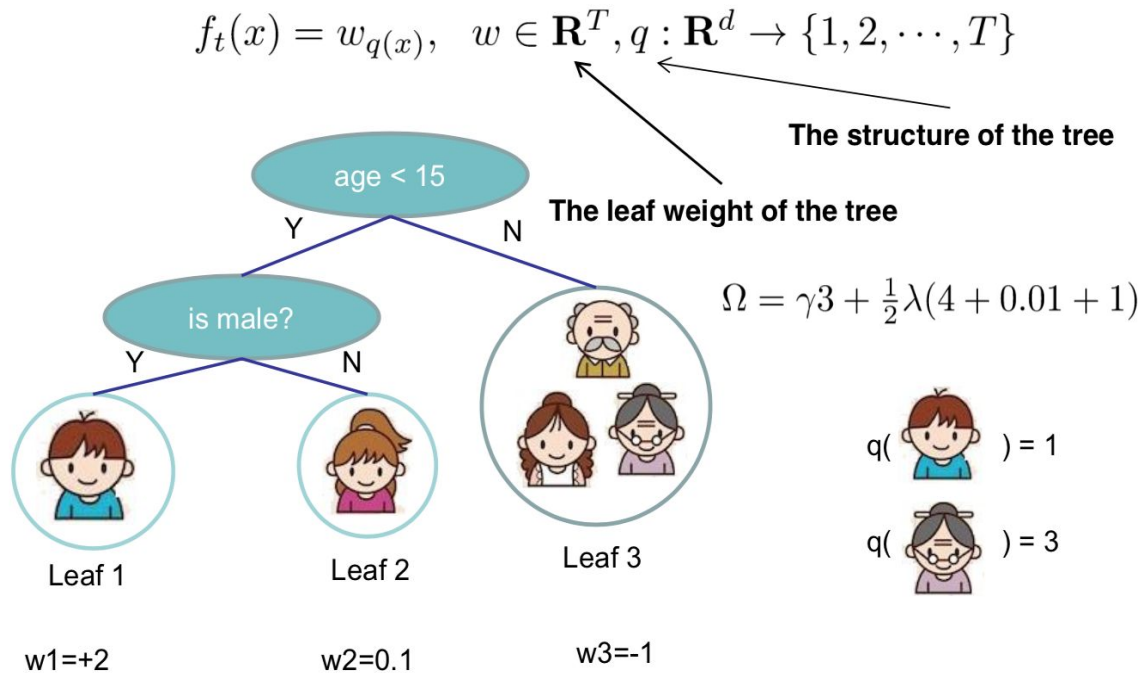
- **where**  $g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}), \quad h_i = \partial_{\hat{y}^{(t-1)}}^2 l(y_i, \hat{y}^{(t-1)})$

- **Why spending so much efforts to derive new objective? Why not just grow trees?**
  - **Theoretical Benefit:** To know what we are learning, convergence.
  - **Engineering Benefit:** To recall the elements of supervised learning
    - Both  $g$  and  $h$  come from definition of training cost function (gradient of error measure)
    - Think of how you can separate modules of your code when you are asked to implement boosted tree for both square and logistic costs

# Refine Definition of the Tree

- We define tree by **a vector of scores in leafs**, and a leaf index mapping function.

- Define complexity as (**this is not the only possible definition**):  $\Omega(f_t) = \underbrace{\gamma T}_{\text{Number of leaves}} + \underbrace{\frac{1}{2}\lambda \sum_{j=1}^T w_j^2}_{\text{L2 norm of leaf scores}}$



## ▪ New Objective:

$$\sum_{i=1}^n [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t)$$

$$g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}), \quad h_i = \partial_{\hat{y}^{(t-1)}}^2 l(y_i, \hat{y}^{(t-1)})$$

## ▪ Regroup by Leaf:

$$\begin{aligned} Obj^{(t)} &\simeq \sum_{i=1}^n [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t) \\ &= \sum_{i=1}^n [g_i w_{q(x_i)} + \frac{1}{2} h_i w_{q(x_i)}^2] + \gamma T + \lambda \frac{1}{2} \sum_{j=1}^T w_j^2 \\ &= \sum_{j=1}^T \left[ (\sum_{i \in I_j} g_i) w_j + \frac{1}{2} (\sum_{i \in I_j} h_i + \lambda) w_j^2 \right] + \gamma T \end{aligned}$$

- **A Sum of  $T$  independent quadratic function!**

# The Structure Score

- Define the instance set in leaf  $j$  as :  $I_j = \{i | q(x_i) = j\}$

- Two facts about single variable quadratic function:

$$\operatorname{argmin}_x Gx + \frac{1}{2}Hx^2 = -\frac{G}{H}, \quad H > 0 \quad \min_x Gx + \frac{1}{2}Hx^2 = -\frac{1}{2}\frac{G^2}{H}$$

- Define**  $G_j = \sum_{i \in I_j} g_i$   $H_j = \sum_{i \in I_j} h_i$  **then:**  $Obj^{(t)} = \sum_{j=1}^T \left[ (\sum_{i \in I_j} g_i)w_j + \frac{1}{2}(\sum_{i \in I_j} h_i + \lambda)w_j^2 \right] + \gamma T$   
 $= \sum_{j=1}^T \left[ G_j w_j + \frac{1}{2}(H_j + \lambda)w_j^2 \right] + \gamma T$

- Assume the structure of tree ( $q(x)$ ) is fixed, the **optimal weight** in each leaf, and the resulting **objective value** are:

$$w_j^* = -\frac{G_j}{H_j + \lambda} \quad Obj = -\frac{1}{2} \sum_{j=1}^T \frac{G_j^2}{H_j + \lambda} + \gamma T$$






  
 This measures how good a tree structure is!

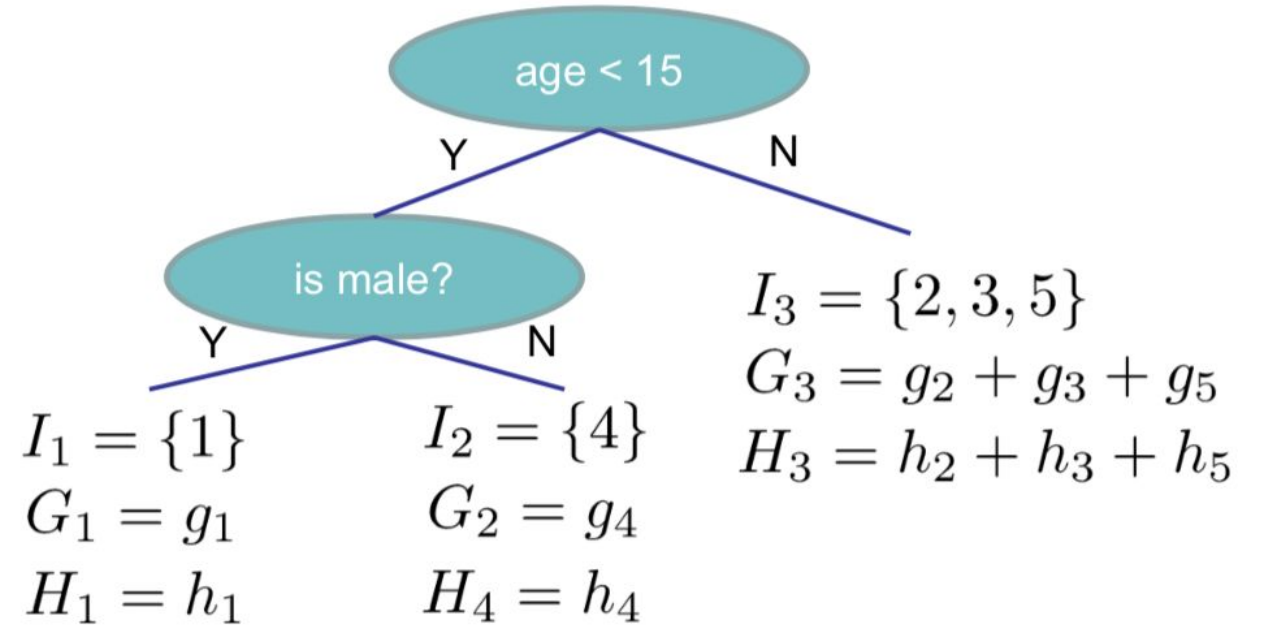
Can be used as scoring function to measure quality of a tree structure  $q$ , like impurity score.



# The Structure Score Calculation

Instance index      gradient statistics

1		$g_1, h_1$
2		$g_2, h_2$
3		$g_3, h_3$
4		$g_4, h_4$
5		$g_5, h_5$



$$Obj = - \sum_j \frac{G_j^2}{H_j + \lambda} + 3\gamma$$

The smaller the score is, the better the structure is

# Exact Greedy Algorithm

- Normally it is impossible to enumerate all the possible tree structures  $q$ .
- GBM used **a greedy algorithm** starts from a single leaf and iteratively add branches to the tree.
- Assume that  $I_L$  and  $I_R$  are the instance sets of left and right nodes after the split.
- **Loss/cost reduction after split is given by:**

$$\mathcal{L}_{split} = \frac{1}{2} \left[ \frac{(\sum_{i \in I_L} g_i)^2}{\sum_{i \in I_L} h_i + \lambda} + \frac{(\sum_{i \in I_R} g_i)^2}{\sum_{i \in I_R} h_i + \lambda} - \frac{(\sum_{i \in I} g_i)^2}{\sum_{i \in I} h_i + \lambda} \right] - \gamma$$

- **That formula is usually used in practices for evaluating split candidates.**
- In addition, to prevent overfitting two techniques from Friedman [1,2] are applied:
  - **Shrinkage.** It is like learning rate in SGD.
  - **Feature Subsampling.** It can speed up computation.

# Searching and Greedy Algorithms

## Searching Algorithm for Single Tree

---

- Enumerate the possible tree structures  $q$
- Calculate the structure score for the  $q$ , using the scoring eq.

$$Obj = -\frac{1}{2} \sum_{j=1}^T \frac{G_j^2}{H_j + \lambda} + \gamma T$$

- Find the best tree structure, and use the optimal leaf weight

$$w_j^* = -\frac{G_j}{H_j + \lambda}$$

- But... there can be infinite possible tree structures..

## Greedy Learning of the Tree

---

- In practice, we grow the tree greedily
  - Start from tree with depth 0
  - For each leaf node of the tree, try to add a split. The change of objective after adding the split is

$$Gain = \frac{1}{2} \left[ \frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} \right] - \gamma$$

Annotations for the Gain equation:

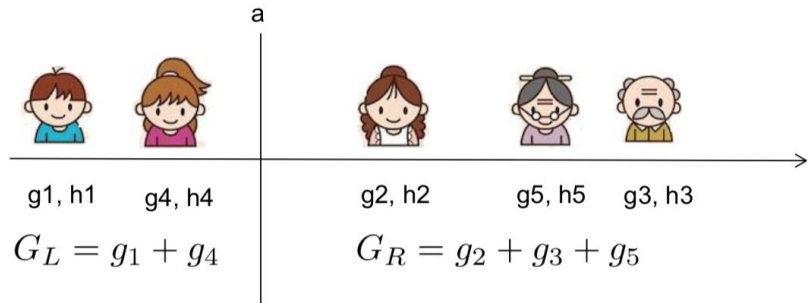
- the score of left child (points to  $\frac{G_L^2}{H_L + \lambda}$ )
- the score of right child (points to  $\frac{G_R^2}{H_R + \lambda}$ )
- the score of if we do not split (points to  $\frac{(G_L + G_R)^2}{H_L + H_R + \lambda}$ )
- The complexity cost by introducing additional leaf (points to  $-\gamma$ )

- Remaining question: how do we find the best split?

# Split Finding Algorithm

## Efficient Finding of the Best Split

- What is the gain of a split rule  $x_j < a$  ? Say  $x_j$  is age



- All we need is sum of  $g$  and  $h$  in each side, and calculate

$$Gain = \frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} - \gamma$$

- Left to right linear scan over sorted instance is enough to decide the best split along the feature

## An Algorithm for Split Finding

- For each node, enumerate over all features
  - For each feature, sorted the instances by feature value
  - Use a linear scan to decide the best split along that feature
  - Take the best split solution along all the features
- Time Complexity growing a tree of depth  $K$ 
  - It is  $O(n d K \log n)$ : or each level, need  $O(n \log n)$  time to sort. There are  $d$  features, and we need to do it for  $K$  level
  - This can be further optimized (e.g. use approximation or caching the sorted features)
  - Can scale to very large dataset

# Exact Greedy Algorithm

---

**Algorithm 1:** Exact Greedy Algorithm for Split Finding

---

**Input:**  $I$ , instance set of current node

**Input:**  $d$ , feature dimension

$gain \leftarrow 0$

$G \leftarrow \sum_{i \in I} g_i, H \leftarrow \sum_{i \in I} h_i$

**for**  $k = 1$  **to**  $m$  **do**

$G_L \leftarrow 0, H_L \leftarrow 0$

**for**  $j$  in sorted( $I$ , by  $\mathbf{x}_{jk}$ ) **do**

$G_L \leftarrow G_L + g_j, H_L \leftarrow H_L + h_j$

$G_R \leftarrow G - G_L, H_R \leftarrow H - H_L$

$score \leftarrow \max(score, \frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{G^2}{H + \lambda})$

**end**

**end**

**Output:** Split with max score

---

---

**Algorithm 2:** Approximate Algorithm for Split Finding

---

**for**  $k = 1$  **to**  $m$  **do**

    Propose  $S_k = \{s_{k1}, s_{k2}, \dots, s_{kl}\}$  by percentiles on feature  $k$ .

    Proposal can be done per tree (global), or per split(local).

**end**

**for**  $k = 1$  **to**  $m$  **do**

$G_{kv} \leftarrow \sum_{j \in \{j | s_{k,v} \geq \mathbf{x}_{jk} > s_{k,v-1}\}} g_j$

$H_{kv} \leftarrow \sum_{j \in \{j | s_{k,v} \geq \mathbf{x}_{jk} > s_{k,v-1}\}} h_j$

**end**

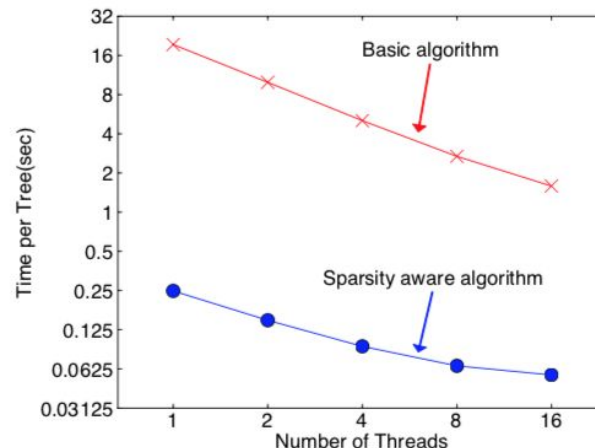
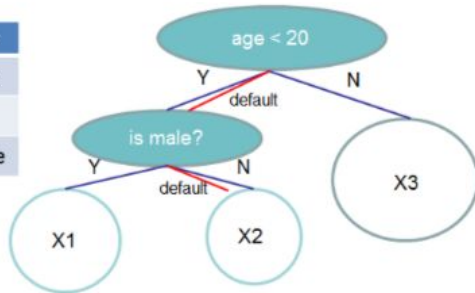
Follow same step as in previous section to find max score only among proposed splits.

---

# Sparsity-Aware Split Finding

- It is common data is sparse due to:
  - Presence of missing values
  - Frequent zero entries
  - One-hot encoding
- How to make algorithm aware of sparsity?
  - **Solution:** Tree structure with default direction

Data		
Example	Age	Gender
X1	?	male
X2	15	?
X3	25	female




---

## Algorithm 3: Sparsity-aware Split Finding

---

**Input:**  $I$ , instance set of current node

**Input:**  $I_k = \{i \in I | x_{ik} \neq \text{missing}\}$

**Input:**  $d$ , feature dimension

*Also applies to the approximate setting, only collect statistics of non-missing entries into buckets*

$gain \leftarrow 0$

$G \leftarrow \sum_{i \in I} g_i, H \leftarrow \sum_{i \in I} h_i$

**for**  $k = 1$  **to**  $m$  **do**

*// enumerate missing value goto right*

$G_L \leftarrow 0, H_L \leftarrow 0$

**for**  $j$  in sorted( $I_k$ , ascent order by  $\mathbf{x}_{jk}$ ) **do**

$G_L \leftarrow G_L + g_j, H_L \leftarrow H_L + h_j$

$G_R \leftarrow G - G_L, H_R \leftarrow H - H_L$

$score \leftarrow \max(score, \frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{G^2}{H + \lambda})$

**end**

*// enumerate missing value goto left*

$G_R \leftarrow 0, H_R \leftarrow 0$

**for**  $j$  in sorted( $I_k$ , descent order by  $\mathbf{x}_{jk}$ ) **do**

$G_R \leftarrow G_R + g_j, H_R \leftarrow H_R + h_j$

$G_L \leftarrow G - G_R, H_L \leftarrow H - H_R$

$score \leftarrow \max(score, \frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{G^2}{H + \lambda})$

**end**

**end**

**Output:** Split and default directions with max gain

---

# Pruning and Regularization

- The gain of split, it can be negative!

$$Gain = \frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} - \gamma$$

- When the training cost reduction is smaller than regularization
- Trade-off between simplicity and predictiveness

- Pre-stopping

- Stop split if the best split have negative gain
- But maybe a split can benefit future splits..

- Post-pruning

- Grow a tree to maximum depth, recursively prune all the leaf splits with negative gain



# Categorical Variables and Time Series Data

## Work with Categorical Variables

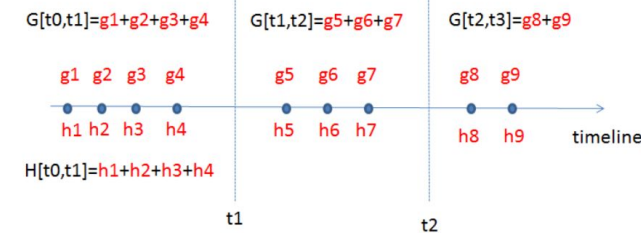
- Some tree learning algorithm handles categorical variable and continuous variable separately
  - We can easily use the scoring formula we derived to score split based on categorical variables.
- Actually it is not necessary to handle categorical separately.
  - We can encode the categorical variables into numerical vector using one-hot encoding. Allocate a #categorical length vector

$$z_j = \begin{cases} 1 & \text{if } x \text{ is in category } j \\ 0 & \text{otherwise} \end{cases}$$

- The vector will be sparse if there are lots of categories, the learning algorithm is preferred to handle sparse data

## Work with Time Series Data

- Time series problem



- All that is important is the structure score of the splits

$$Obj = -\frac{1}{2} \sum_{j=1}^T \frac{G_j^2}{H_j + \lambda} + \gamma T$$

- Top-down greedy, same as trees
- Bottom-up greedy, start from individual points as each group, greedily merge neighbors
- Dynamic programming, can find optimal solution for this case

# Recap: Boosted Tree Algorithm

- Add a new tree in each iteration
- Beginning of each iteration, calculate

$$g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}), \quad h_i = \partial_{\hat{y}^{(t-1)}}^2 l(y_i, \hat{y}^{(t-1)})$$

- Use the statistics to greedily grow a tree  $f_t(x)$

$$Obj = -\frac{1}{2} \sum_{j=1}^T \frac{G_j^2}{H_j + \lambda} + \gamma T$$

- Add  $f_t(x)$  to the model  $\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t(x_i)$ 
  - Usually, instead we do  $y^{(t)} = y^{(t-1)} + \epsilon f_t(x_i)$
  - $\epsilon$  is called step-size or shrinkage, usually set around 0.1
  - This means we do not do full optimization in each step and reserve chance for future rounds, it helps prevent overfitting

# Key Concepts

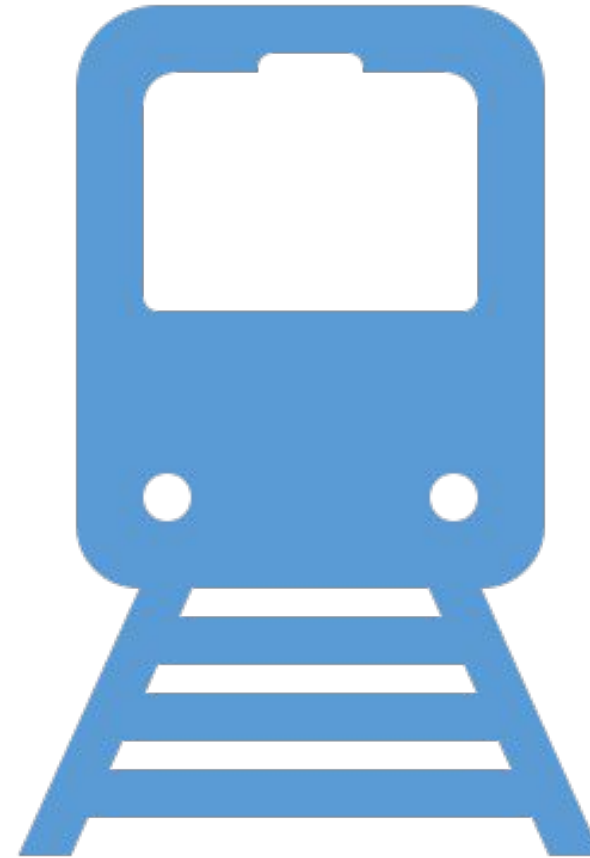
1. **Supervised Learning Summary**
  - Separation of Model, Parameters, Objective Function, Regularization
  - Generalization Trade Off (VC Dimension and Bias-Variance)
2. **Decision Tree and CART Ensemble Model**
3. **Gradient Boosting Algorithm**
  - Exact Greedy Algorithm
  - Sparsity-Aware Split Finding
  - Pruning and Regularization
4. **Summary**

# Recall: XGBoost Parameters

1. **max\_depth** (how deep is your tree?)
2. **learning\_rate** (shrinkage parameter, "speed of learning")
3. **max\_delta\_step** (additional cap on learning rate, needed in case of highly imbalanced classes)
4. **n\_estimators** (number of boosting rounds)
5. **booster** (as XGBoost is not only about trees, but it actually is)
6. **scale\_pos\_weight** (in binary classification whether to rebalance the weights)
7. **base\_score** (boosting starts with predicting 0.5 for all observations, you can change it here)
8. **seed / random\_state** (To reproduce results.)
9. **missing** (if you want to treat some other things as missing than np.nan)
10. **objective** (give callable that maps `objective(y_true, y_pred) -> grad, hes`)

# Limitation of XGBoost

1. Cant work properly with complex data. Coz it is based on Decision Tree.
2. Harder to train compare to Random Forest.
3. Inappropriate for time series data. Eq. How to do k-fold validation.



# Summary



THE SEPARATION BETWEEN MODEL, OBJECTIVE, PARAMETERS CAN BE HELPFUL FOR US TO UNDERSTAND AND CUSTOMIZE ML MODELS.



THE VC DIMENSION AND BIAS-VARIANCE TRADE-OFF APPLIES EVERYWHERE, INCLUDING LEARNING IN FUNCTIONAL SPACE.



WE CAN BE FORMAL ABOUT WHAT WE LEARN AND HOW WE LEARN. CLEAR UNDERSTANDING OF THEORY CAN BE USED TO GUIDE CLEANER ENGINEERING IMPLEMENTATION.

# Home Work: Make Your First XGBoost Case

XGBoost

TABLE OF CONTENTS

Installation Guide

Get Started with XGBoost

XGBoost Tutorials

Frequently Asked Questions

XGBoost User Forum

GPU support

XGBoost Parameters

Python package

R package

JVM package

Julia package

CLI interface

Contribute to XGBoost

Search...

Docs / XGBoost Documentation

XGBoost Documentation

**XGBoost** is an optimized distributed gradient boosting library designed to be highly **efficient**, **flexible** and **portable**. It implements machine learning algorithms under the [Gradient Boosting](#) framework. XGBoost provides a parallel tree boosting (also known as GBDT, GBM) that solve many data science problems in a fast and accurate way. The same code runs on major distributed environment (Hadoop, SGE, MPI) and can solve problems beyond billions of examples.

Contents

- [Installation Guide](#)
- [Get Started with XGBoost](#)
- [XGBoost Tutorials](#)
  - [Introduction to Boosted Trees](#)
  - [Distributed XGBoost with AWS YARN](#)
  - [Distributed XGBoost with XGBoost4J-Spark](#)
  - [DART booster](#)
  - [Monotonic Constraints](#)
  - [Feature Interaction Constraints](#)
  - [Text Input Format of DMatrix](#)
  - [Notes on Parameter Tuning](#)
  - [Using XGBoost External Memory Version \(beta\)](#)

v: latest



# Reference and Further Reading\*

1. StatQuest Gradient Boosting: [Part 1](#), [Part 2](#), [Part 3](#), [Part 4](#)
2. Mason et al., Boosting Algorithms as Gradient Descent in Function Space, 1999.
3. J.H. Friedman, Stochastic Gradient Boosting, 1999.
4. J.H. Friedman, Greedy function approximation a gradient boosting machine, 2001.
5. Tianqi Chen et al., XGBoost: A Scalable Tree Boosting System, arXiv:1603.02754v3, 2016.
6. Tianqi Chen XGBoost Documentation: <https://xgboost.readthedocs.io/en/latest/index.html>
7. Hastie et al., The Element of Statistical Learning, Chapter 10.