# Ensemble Method : XGBoost

Session 13 @ IIFT

Dr. Tanujit Chakraborty

www.ctanujit.org

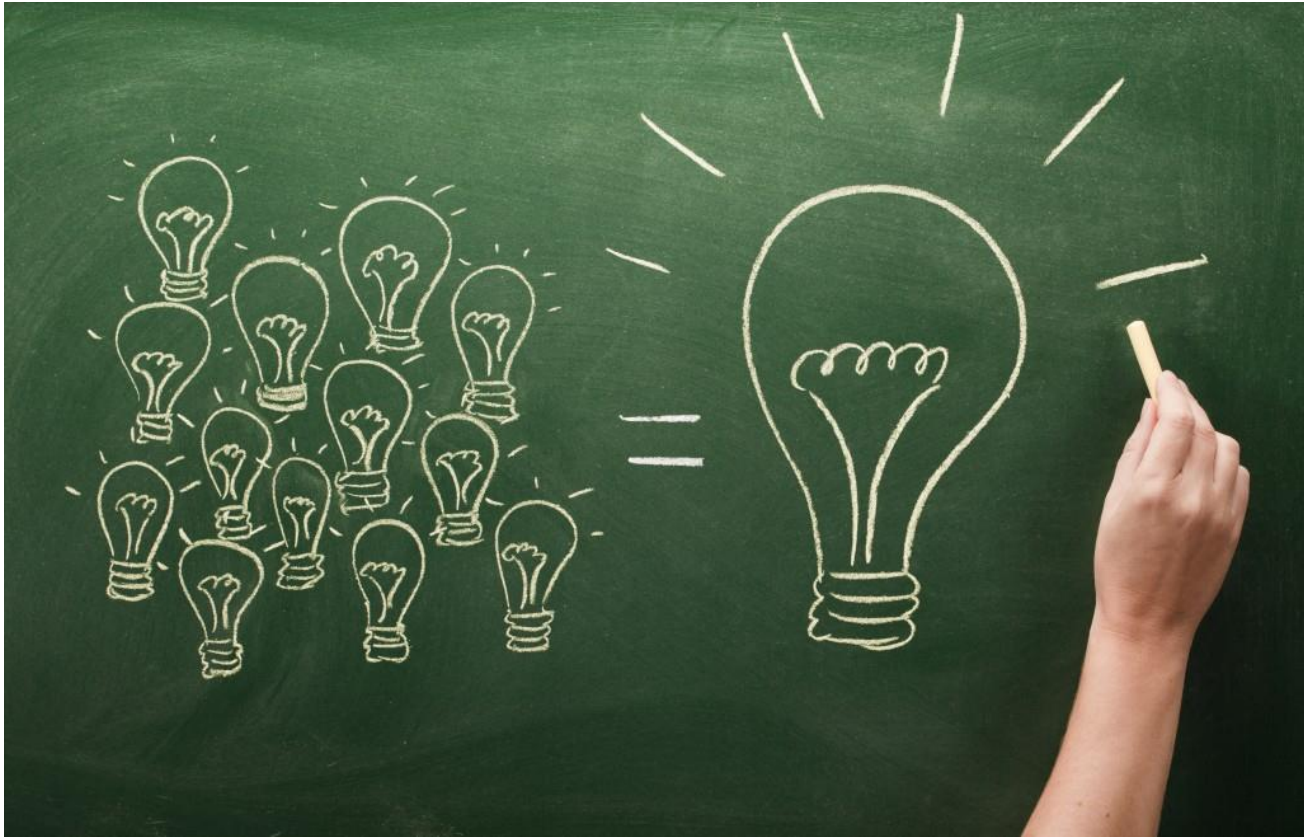
# Popular Ensemble Methods

Bagging

Random Forests

Boosting

# Wisdom of the crowds



# Ensemble methods

- A single decision tree does not perform well
- But, it is super fast
- What if we learn multiple trees?

We need to make sure they do not all just learn the same

# Variant of algorithms to learn Tree Ensembles

- Random Forest (*Breiman 1997)*
  - RandomForest packages in R and python

- Gradient Tree Boosting *(Friedman 1999)*
  - R GBM
  - sklearn.ensemble.GradientBoostingClassifier

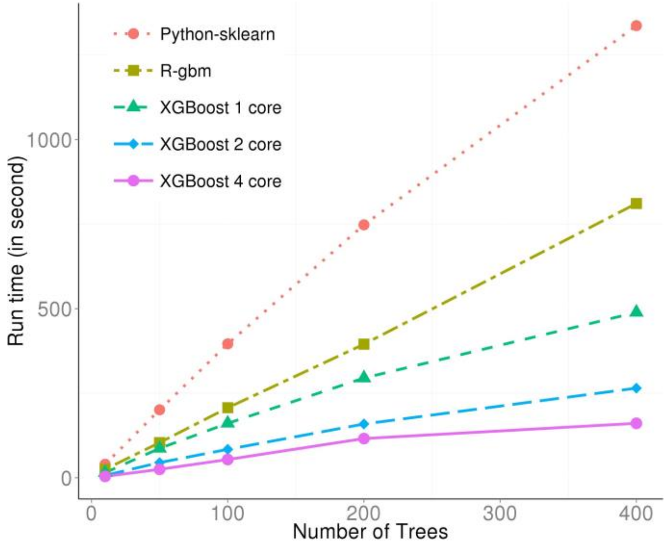
- Gradient Tree Boosting with Regularization (variant of original GBM)
  - Regularized Greedy Forest (RGF)
  - **XGBoost**

# Learning Trees : Advantage and Challenges

- Advantages of tree-based methods
  - Highly accurate: almost half of data science challenges are won by tree based methods.
  - Easy to use: invariant to input scale, get good performance with little tuning.
  - Easy to interpret and control
- Challenges on learning tree(ensembles)
  - Control over-fitting
  - Improve training speed and scale up to larger dataset

# What is XGBoost

- A Scalable System for Learning Tree Ensembles
    - Model improvement
        - Regularized objective for better model
    - Systems optimizations
        - Out of core computing
        - Parallelization
        - Cache optimization
        - Distributed computing
    - Algorithm improvements
        - Sparse aware algorithm
        - Weighted approximate quantile sketch.




- *In short, faster tool for learning better models*

# What does XGBoost learn

- A self-contained derivation of general gradient boosting algorithm
- Resembles the original GBM derivation by *Friedman*
- Only preliminary of calculus is needed

# Elements of Supervised Learning

- **Model**: how to make prediction $\hat{y}_i = f(x_i)$
  - Linear model: $\hat{y}_i = \sum_j w_j x_{ij}$
- **Parameters**: the things we need to learn from data
  - Linear model: $\Theta = \{w_j | j = 1, \cdots, d\}$
- **Objective Function**: $Obj(\Theta) = L(\Theta) + \Omega(\Theta)$

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

**Regularization**, measures complexity of model

  - Linear model: $L(\Theta) = \sum_i (\hat{y}_i - y_i)^2$ , $\Omega(\Theta) = \lambda \|w\|_2^2$

# Elements of Tree Learning

- 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 Regression trees

- Objective $Obj = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$

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

**Regularization**, measures complexity of trees

# Trade off in Learning

$$Obj = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

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

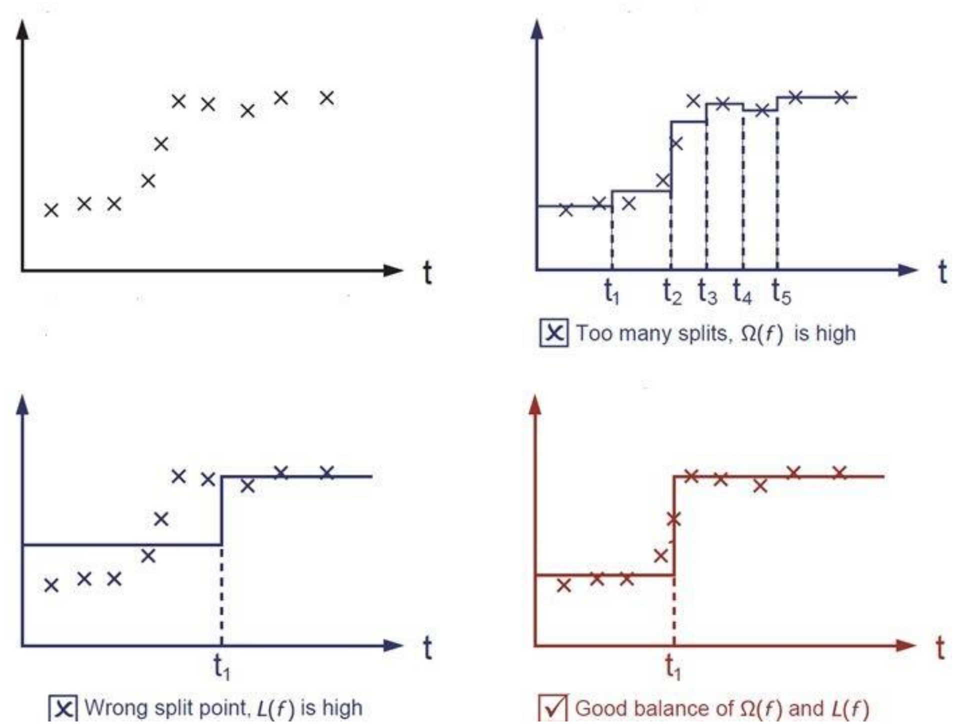
**Regularization**, measures complexity of trees

- Optimizing training loss encourages **predictive** models
  - Fitting well in training data at least get you close to training data which is hopefully close to the underlying distribution
- Optimizing regularization encourages **simple** models
  - Simpler models tends to have smaller variance in future predictions, making prediction **stable**

# Why do we need regularization

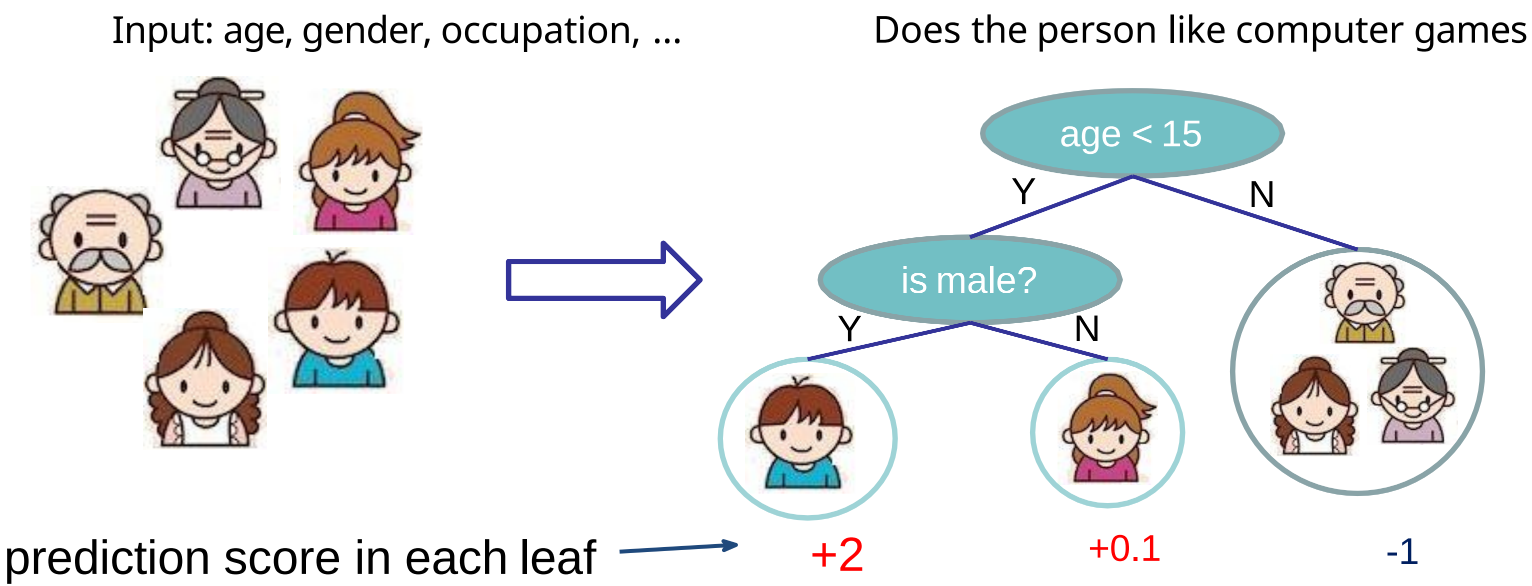
Consider the example of learning tree on a single variable t

Raw Data

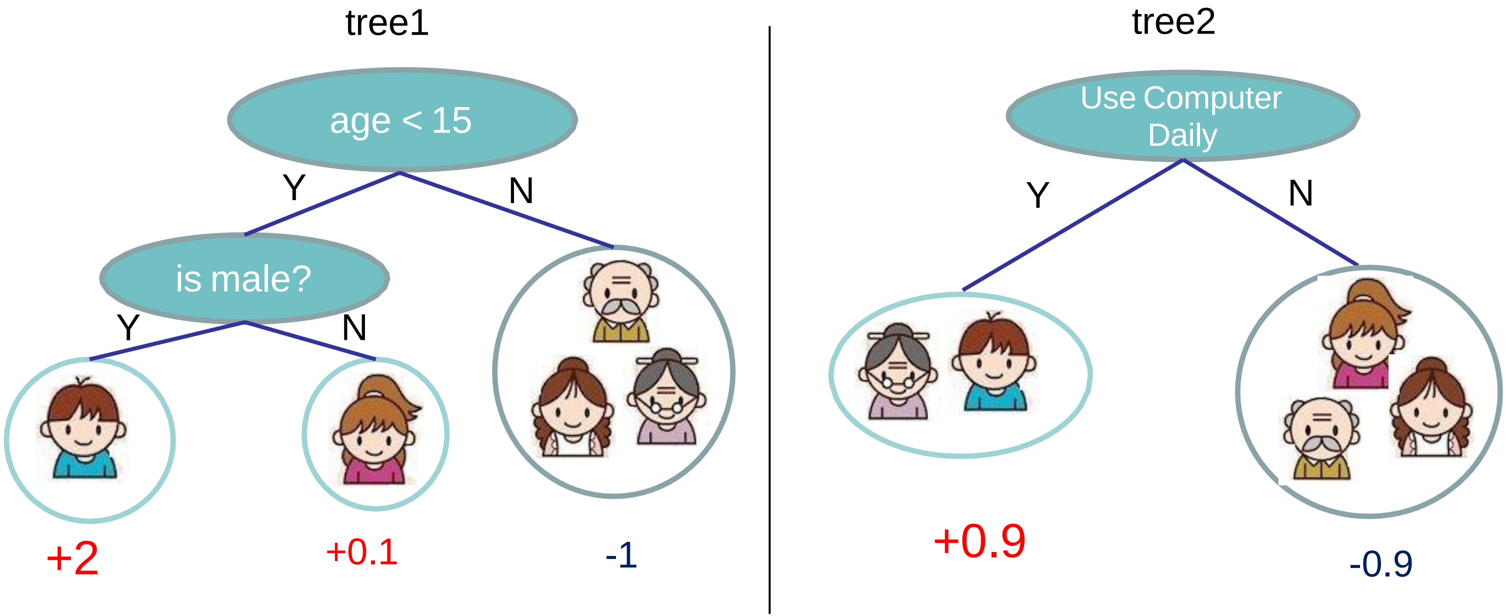


# Regression Tree

- Regression tree (also known as CART)
- This is what it would looks like for a commercial system



# When Trees forms a Forest (Tree Ensembles)



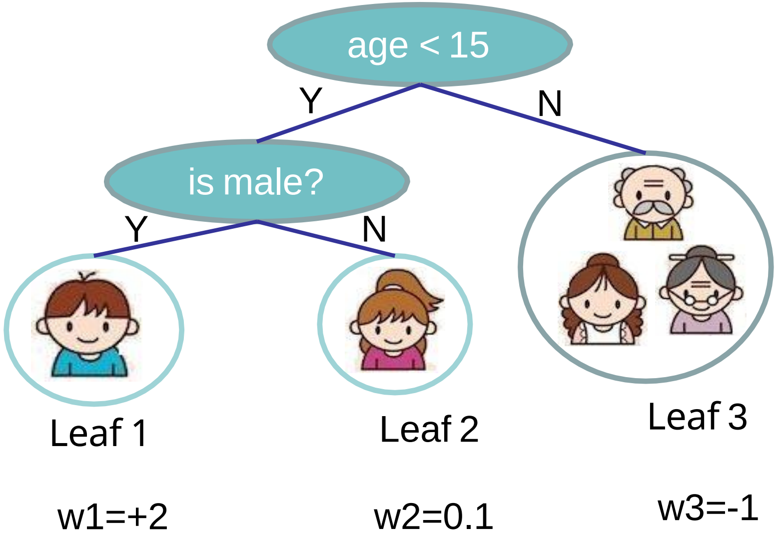

f(  ) = 2 + 0.9= 2.9

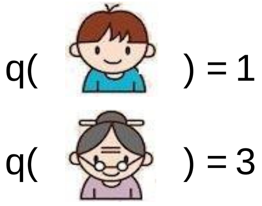
f(  )= -1 - 0.9= -1.9

# Define Complexity of a Tree

$$f_t(x) = w_{q(x)}, \quad w \in \mathbf{R}^T, q : \mathbf{R}^d \rightarrow \{1, 2, \cdots, T\}$$

**The leaf weight of the tree** **The structure of the tree**



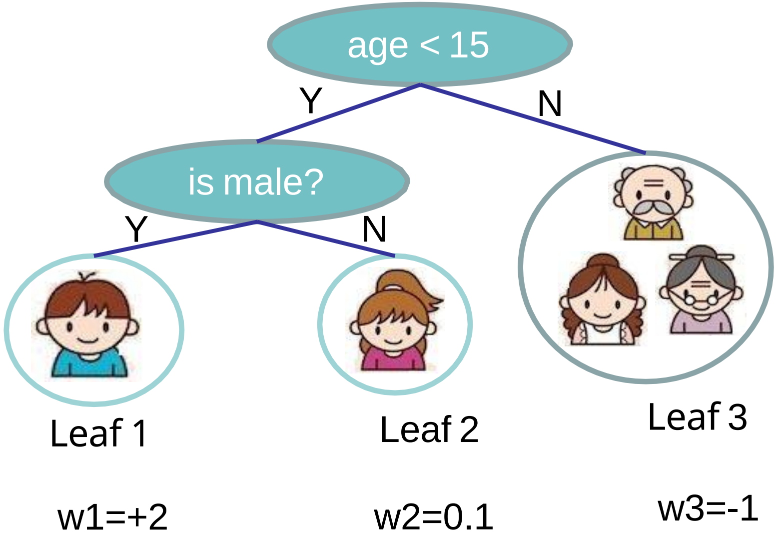



# Define Complexity of a Tree (cont’)

Objective in XGBoost

$$\Omega(f_t) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

**Number of leaves** **L2 norm of leaf scores**




$$\Omega = \gamma 3 + \frac{1}{2}\lambda(4 + 0.01 + 1)$$

# How can we learn tree ensembles

- Objective: $\sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_k \Omega(f_k), f_k \in \mathcal{F}$
- We can not use methods such as SGD.
- Solution: **Additive Training (Boosting)**
  - Start from constant prediction, add a new function each time

$$
\begin{aligned}
\hat{y}_i^{(0)} &= 0 \\
\hat{y}_i^{(1)} &= f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i) \\
\hat{y}_i^{(2)} &= f_1(x_i) + f_2(x_i) = \hat{y}_i^{(1)} + f_2(x_i) \\
&\dots \\
\hat{y}_i^{(t)} &= \sum_{k=1}^{t} f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)
\end{aligned}
$$

**New function**

**Model at training round t**

**Keep functions added in previous round**

# Additive Training

- How do we decide which f to add: Optimize the objective!
- The prediction at round t is $\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t(x_i)$

This is what we need to decide in round t

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

**Goal: find $f_t$ to minimize this**

- Consider square loss

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

**This is usually called residual from previous round**

# Taylor Expansion Approximation of Loss

- Goal $Obj^{(t)} = \sum_{i=1}^{n} l\left(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)\right) + \Omega(f_t) + constant$
- 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 $g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}), \quad h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$

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

- *In terms of square loss*

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

# Our New Goal

- Objective, with constants removed

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

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

- Define the instance set in leaf j as
    - Regroup the objective by leaf

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

    - This is sum of T independent quadratic function

# The Structure Score

- Two facts about single variable quadratic function

$$argmin_x \; Gx + \frac{1}{2}Hx^2 = -\frac{G}{H},\; H > 0 \qquad \min_x \; Gx + \frac{1}{2}Hx^2 = -\frac{1}{2}\frac{G^2}{H}$$

- Let us define $G_j = \sum_{i \in I_j} g_i \quad H_j = \sum_{i \in I_j} h_i$

$$\begin{aligned} Obj^{(t)} &= \sum_{j=1}^{T} \left[ (\sum_{i \in I_j} g_i) w_j + \frac{1}{2}(\sum_{i \in I_j} h_i + \lambda) w_j^2 \right] + \gamma T \\ &= \sum_{j=1}^{T} \left[ G_j w_j + \frac{1}{2}(H_j + \lambda) w_j^2 \right] + \gamma T \end{aligned}$$

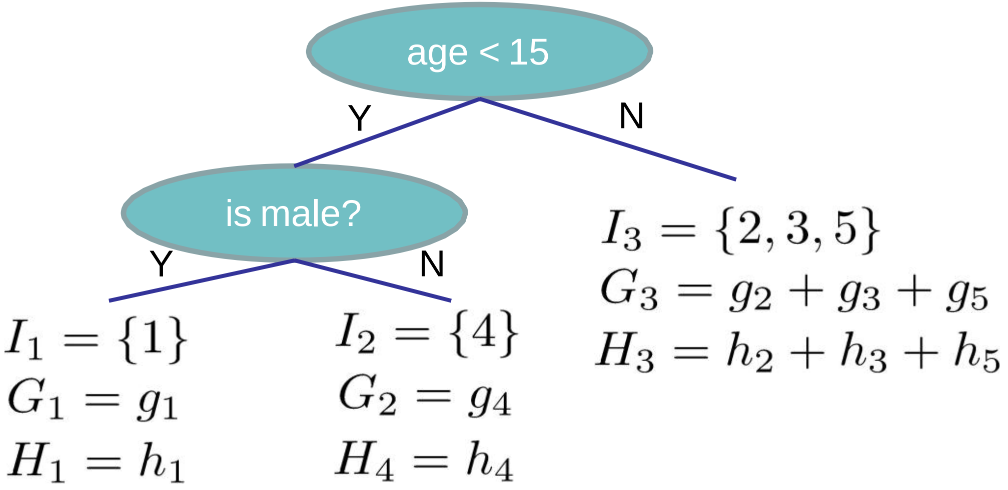
- Assume the structure of tree ( q(x) ) is fixed, the optimal weight in each leaf, and the resulting objective value are

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

**This measures how good a tree structure is!**

# The Structure Score Calculation

| Instance index | | gradient statistics |
|---|---|---|
| 1 |  | g1, h1 |
| 2 |  | g2, h2 |
| 3 |  | g3, h3 |
| 4 |  | g4, h4 |
| 5 |  | g5, h5 |




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

The smaller the score is, the better the structure is

# 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

The complexity cost by introducing additional leaf

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

the score of left child

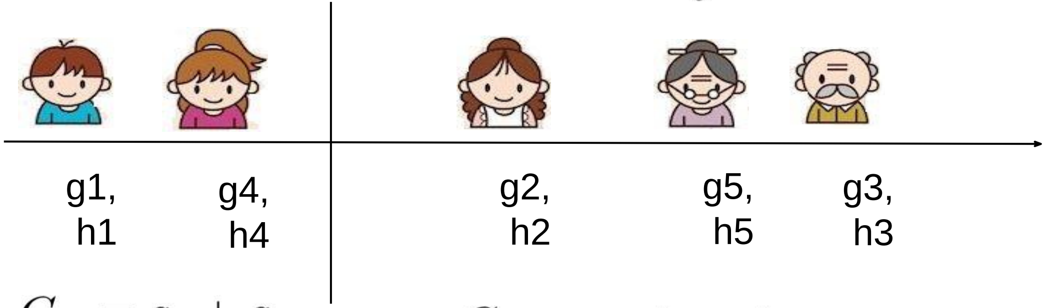
the score of right child

the score of if we do not split

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

# Efficient Finding of the Best Split

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




$$G_L = g_1 + g_4 \qquad G_R = g_2 + g_3 + g_5$$

- All we need is sum of g and h in each side, and calculate
- Left to right linear scan over sorted instance is enough to decide the best split along the feature

# Pruning and Regularization

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

# XGBoost Model Recap

- A regularized objective for better generalization

- Additive solution for generic objective function

- Structure score to search over structures.

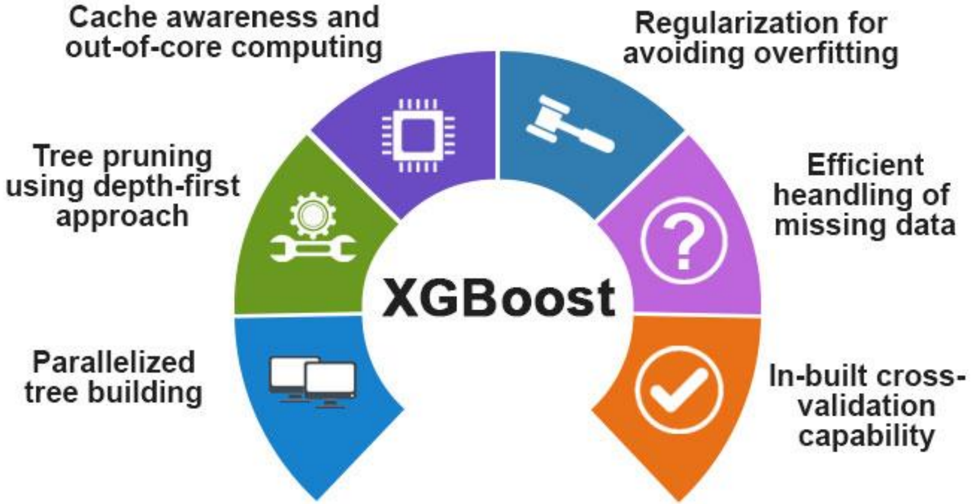
- Why take all the pain in deriving the algorithm
    - Know your model
    - Clear definitions in algorithm offers clear and extendible modules in software

# What can XGBoost can do for you

- Push the limit of computation resources to solve **one** problem
  - Gradient tree boosting
- Automatic handle missing value
- Interactive Feature analysis
- Extendible system for more functionalities
- Deployment on the Cloud

# What can XGBoost **cannot** do for you

- Feature engineering
- Hyper parameter tuning
- A lot more cases …



# Further reading

- The Elements of Statistical Learning BY Trevor Hastie, Robert Tibshirani, Jerome Friedman.

- Introduction to Statistical Learning Book: https://www.statlearning.com/