XGBoost

Weiwei Zhang, Zoey Lyu, Leyuan Yu

01

Key Concepts

Might just be a review :)

02

Regression Tree

What are we learning?

03

Gradient Boosting

How do we learn?

04

Others

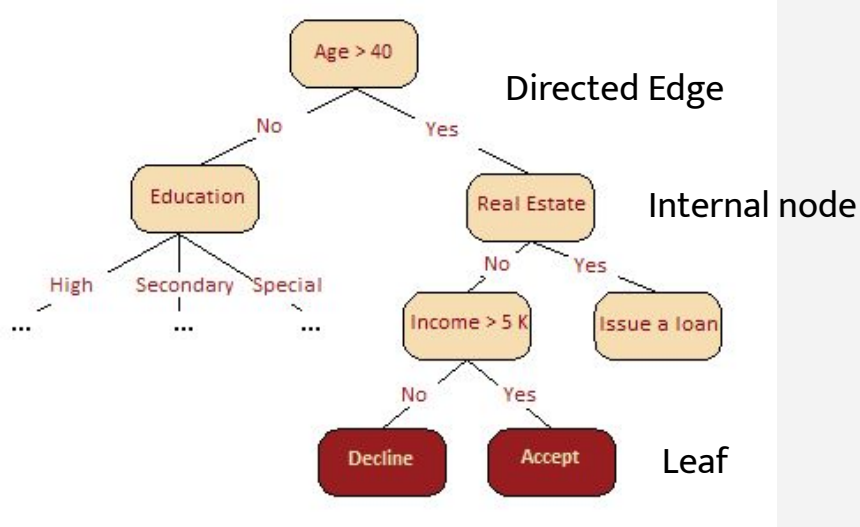
Some misc stuff.

01

Key concepts

Decision tree - recap

- A tree-structure layout of features/attributes for predicting the output, with components such as root, nodes and leafs.
- Algorithm: if - else if - else

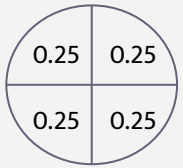


Entropy - recap

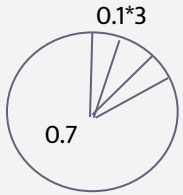
- Entropy concept originates from thermodynamics to describe molecular disorder.

- Shannon's information theory: Entropy measures the average information content of a message. Entropy is 0 when all messages are identical. In machine learning, entropy act as an impurity measure: a set's entropy is zero when it contains instances of only one class.

$$\mathcal{H}(X) = -\sum_{x} P(x) \log_2 P(x)$$



H = - 4 * 0.25 * $\log_2(0.25)$ = 2



H = - [0.7 * $\log_2(0.7)$ + 3 * 0.1 * $\log_2(0.1)$] = 1.357

Entropy - recap

- Gini impurity vs Entropy:

  Most of time there is little difference. Default selection is Gini impurity.

  When you want to isolate the most frequent class in its own branch of the tree, you should use Gini impurity. Entropy tends to produce slightly more balanced tree. [1]

Information Gain & Information Gain Ratio

Suppose we have dataset $\mathcal{D}$ and a corresponding set of attributes $\mathcal{A}$, to see how well a attribute $a \in \mathcal{A}$ to partition a corpus, we use **Information Gain**:

- Defined as: $g(\mathcal{D}, a) = \mathcal{H}(\mathcal{D}) - \mathcal{H}(\mathcal{D}|a)$ [1]
- For instance, the dataset has K classes. The entropy of the dataset is:

$$\mathcal{H}(\mathcal{D}) = -\sum_{k=1}^{K} \frac{|C_k|}{|\mathcal{D}|} \log \frac{|C_k|}{|\mathcal{D}|}$$

;

- The attribute a could partition the dataset into N sub-datasets. The conditional entropy is the weighted sum of the entropy of these sub-datasets:

$$\mathcal{H}(\mathcal{D}|a) = +\sum_{n=1}^{N} \frac{|\mathcal{D}_n|}{|\mathcal{D}|} \mathcal{H}(\mathcal{D}_n)$$

If use only information gain for deciding which attribute to use, it prefers attributes with more alternative values. To tackle this, we instead use **Information Gain Ratio**

$$g_R(\mathcal{D}, a) = \frac{g(\mathcal{D}, a)}{\mathcal{H}_a(\mathcal{D})}$$

where, suppose a has a set of alternative values V:

$$\mathcal{H}_a(\mathcal{D}) = -\sum_{v=1}^{V} \frac{|\mathcal{D}_v|}{|\mathcal{D}|} \log \frac{|\mathcal{D}_v|}{|\mathcal{D}|}$$

Decision tree construction

Base Cases:

- All samples in the samples in the list belong to the same class. When this happens, it simply creates a leaf node for the decision tree saying to choose that class
- None of the feature provide any information gain.

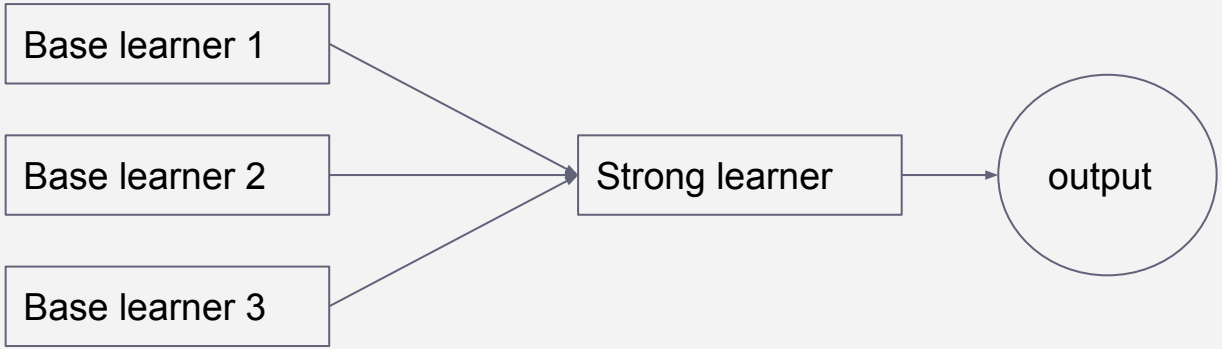
Algorithm:

- Check for the base cases
- For each attribute a, find the information gain (ID3) / information gain ratio (C4.5) from splitting on a.
- Let a be the attribute with the highest information gain (ID3) / information gain ratio (C4.5)
- Create a decision node that splits on a.
- Recurese on the sub-lists obtained by splitting on a, and add those nodes as children of node
-

Ensembles overview

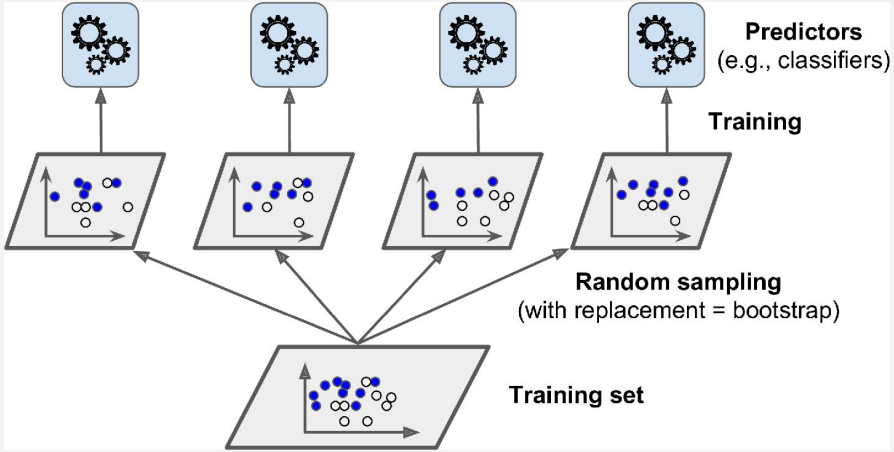
Ensemble learning:

- A method that is used to enhance the performance of Machine Learning model by combining several learners with improved efficiency and accuracy.



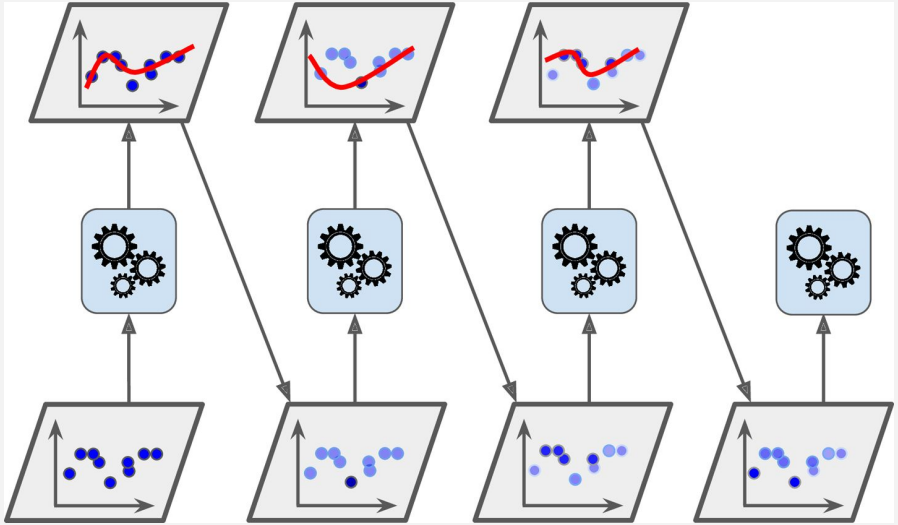
Ensembles - Bagging

- Bagging approach is to use the same training algorithm for every predictor and train them on different random subsets of the training set. When sampling is performed with replacement, this method is called bagging (short for bootstrap aggregating)

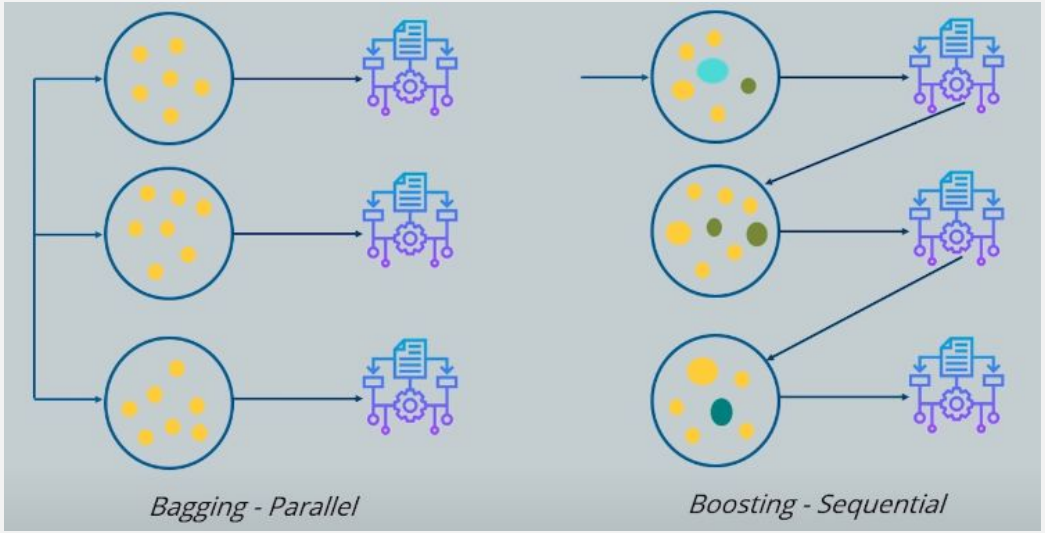


Ensembles - Boosting

- Boosting (originally called hypothesis boosting) refers to any Ensemble method that can combine several weak learners into a strong learner. The general idea of most boosting methods is to train predictors sequentially, each trying to correct its predecessor.



Bagging vs Boosting



02
Regression Tree & Ensemble

Objective Function & Bias Variance Tradeoff

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

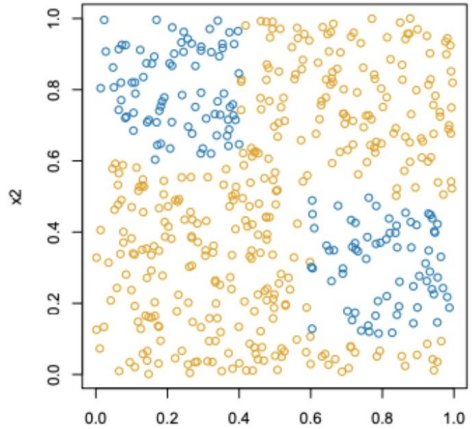
Training Loss measures how well model fit on training data

Regularization, measures complexity of model

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

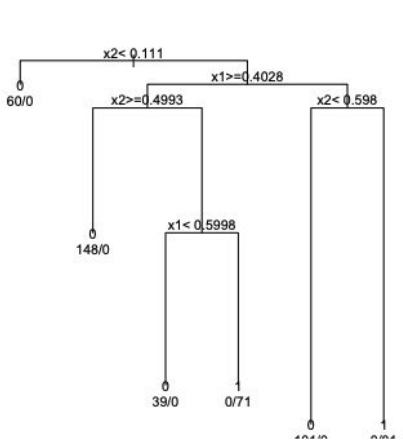
Classification and Regression Trees

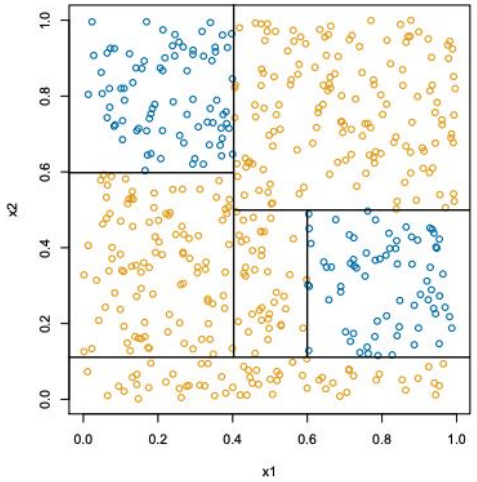
- Tree-based based methods operate by dividing up the feature space into rectangles **-> meaning tree-based methods non-linear**
- Each rectangle is like a *neighbourhood* in a Nearest-Neighbours method
- You predict using the *average* or classify using the *most common class* in each rectangle



- **Can handle numeric/categorical/ranking data**

Classification and Regression Trees





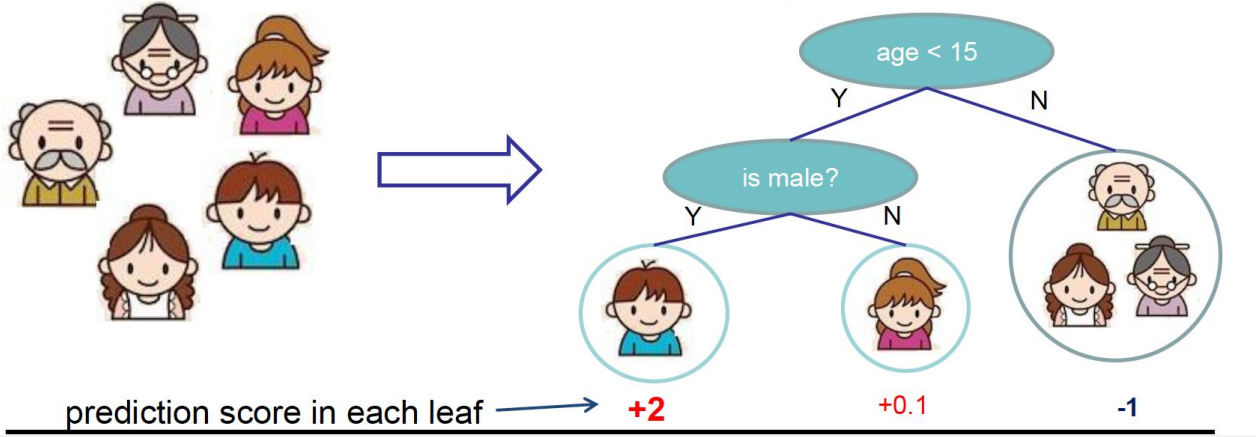
- Trees are built up via a greedy algorithm: Recursive binary partitioning
- At each step, you pick a new split by finding the input X_j and split point $\tilde{x}_j$ that best partitions the data
  - In prediction, you choose splits to minimize the RSS
  - In classification, choose splits to maximize node purity (minimize Gini index)

Classification and Regression Trees

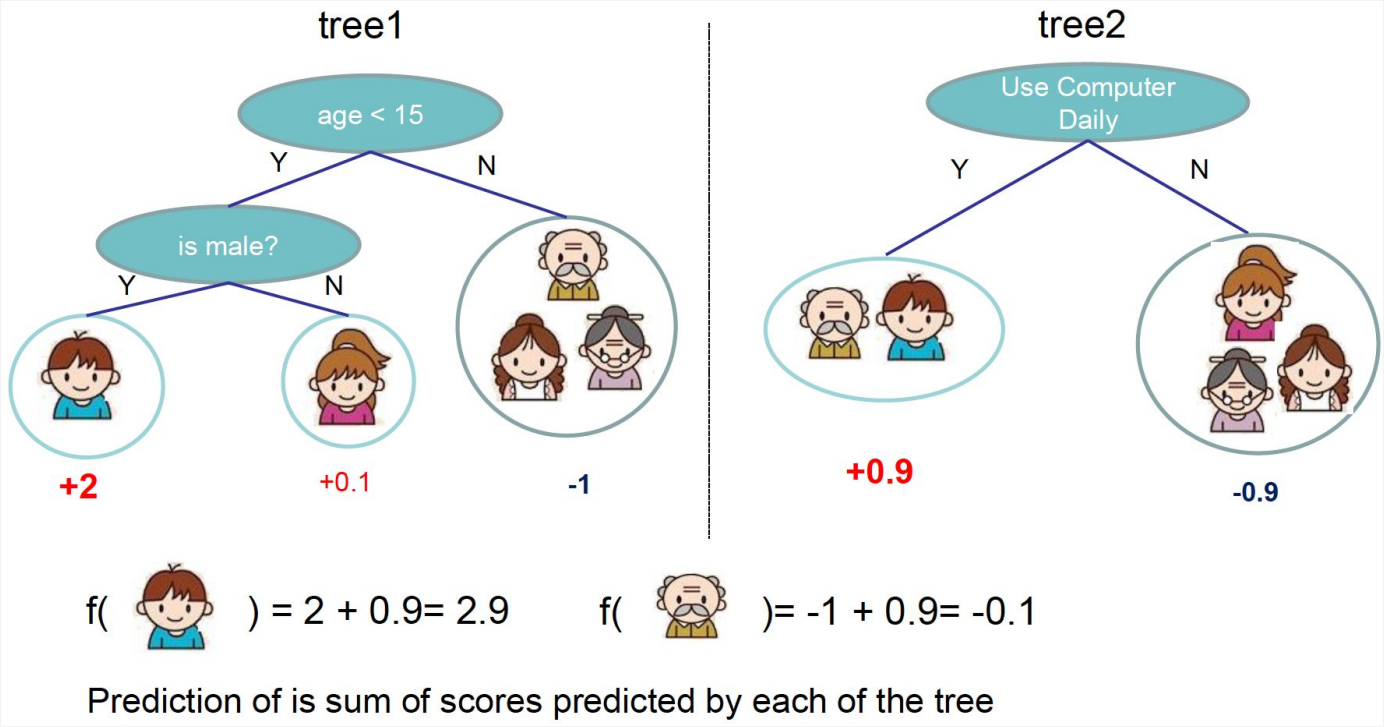
- regression tree (also known as classification and regression tree):
  - Decision rules same as in decision tree
  - Contains one score in each leaf value

Input: age, gender, occupation, …

Does the person like computer games



Regression Tree Ensemble



Regression Tree Ensemble: Models and parameters

- 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

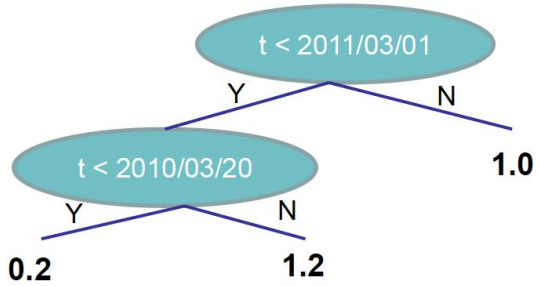
$$\Theta = \{f_1, f_2, \cdots, f_K\}$$

  - Instead learning weights in $\mathbf{R}^d$, we are learning functions(trees)

Learning on a single variable

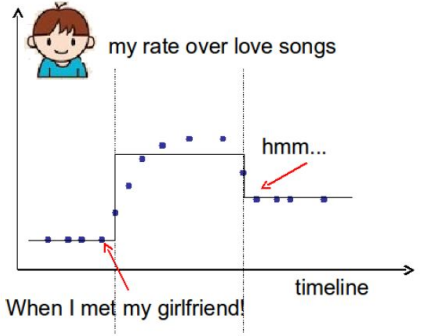
- How can we learn functions?
- Define objective (loss, regularization), and optimize it!!
- Example:
  - Consider regression tree on single input t (time)
  - I want to predict whether I like romantic music at time t

The model is regression tree that splits on time

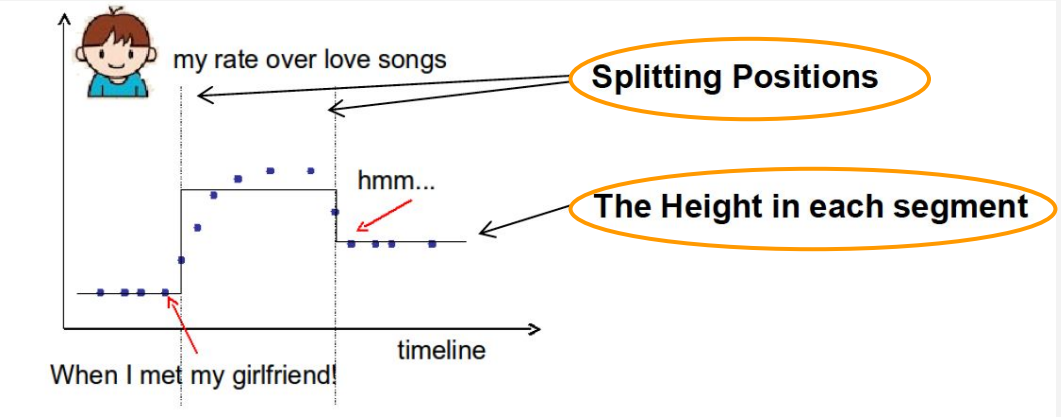


Equivalently

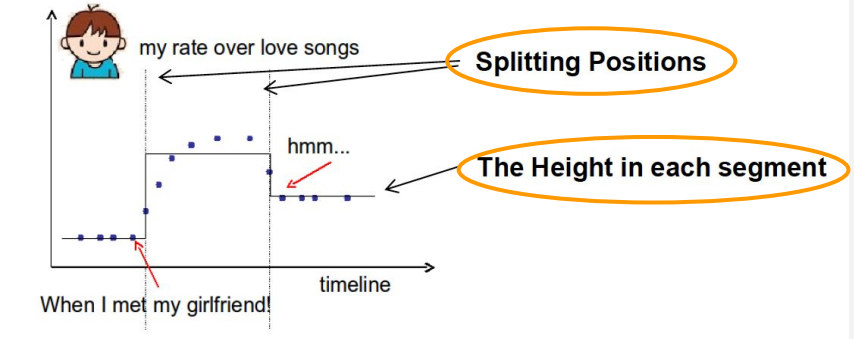
Piecewise step function over time



Learning on a single variable



Learning on a single variable



Model: assuming we have K trees

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i), \quad f_k \in \mathcal{F}$$

Objective

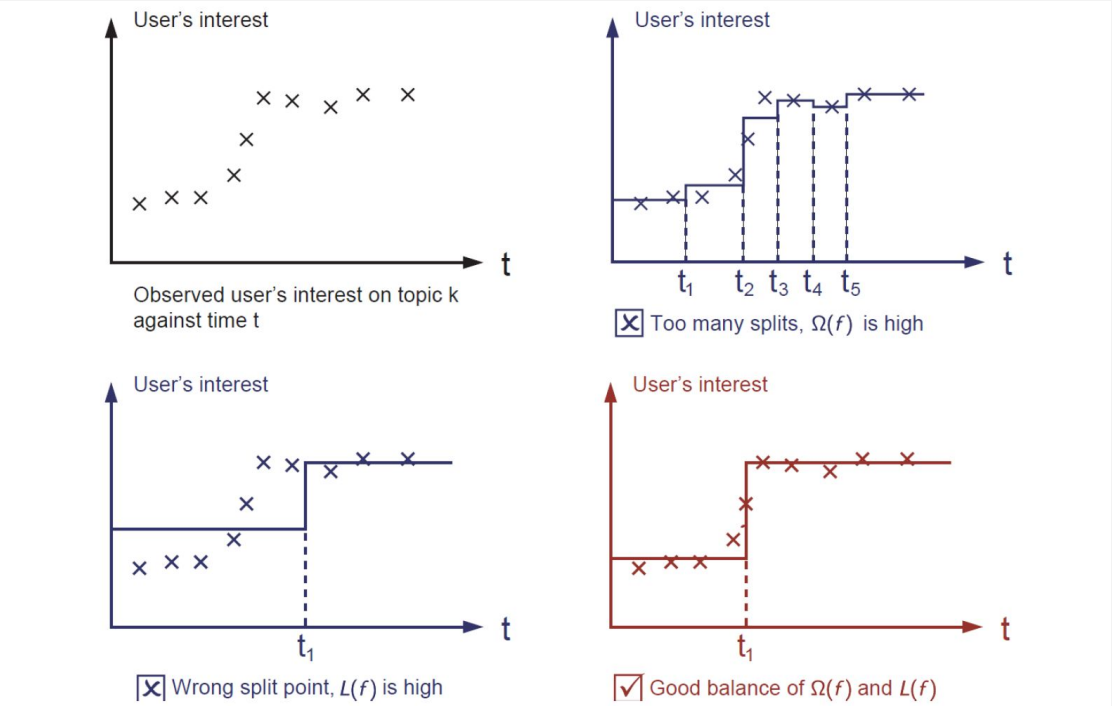
$$Obj = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

Training loss

Complexity of the Trees

How do we look at the bias-variance trade-off?

Learning on a single variable



03 GRADIENT BOOSTING

Understand how it learns.

Motivation

Like the other models, you can use XGBoost without understanding the thoughts behind, but:

- Curiosity is in our nature
- It is really fun and satisfactory after understanding details
- You need something to talk about in a job interview
- I am not good at Math, but even myself can understand how the model works, you definitely can do it too!

Content borrowed from the XGBoost author Tianqi's slide, also https://xgboost.readthedocs.io/en/latest/tutorials/model.html

Objective Function

- Objective function that is everywhere

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

Training Loss measures how well model fit on training data

Regularization, measures complexity of model

- Loss on training data: $L = \sum_{i=1}^{n} l(y_i, \hat{y}_i)$
  - Square loss: $l(y_i, \hat{y}_i) = (y_i - \hat{y}_i)^2$
  - Logistic loss: $l(y_i, \hat{y}_i) = y_i \ln(1 + e^{-\hat{y}_i}) + (1 - y_i) \ln(1 + e^{\hat{y}_i})$
- Regularization: how complicated the model is?
  - L2 norm: $\Omega(w) = \lambda \|w\|^2$
  - L1 norm (lasso): $\Omega(w) = \lambda \|w\|_1$

How it learns

- 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, to find f (since they are trees, instead of just numerical vectors)
- 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

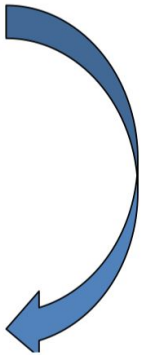
- Consider square loss

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

This is usually called residual from previous round

Taylor Expansion Approximation

- Goal $Obj^{(t)} = \sum_{i=1}^{n} l\left(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)\right) + \Omega(f_t) + constant$
  - Seems still complicated except for the case of square loss
- 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 $g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}), \quad h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$



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

- *If you are not comfortable with this, think of square loss*

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

- Compare what we get to previous slide

Taylor Series

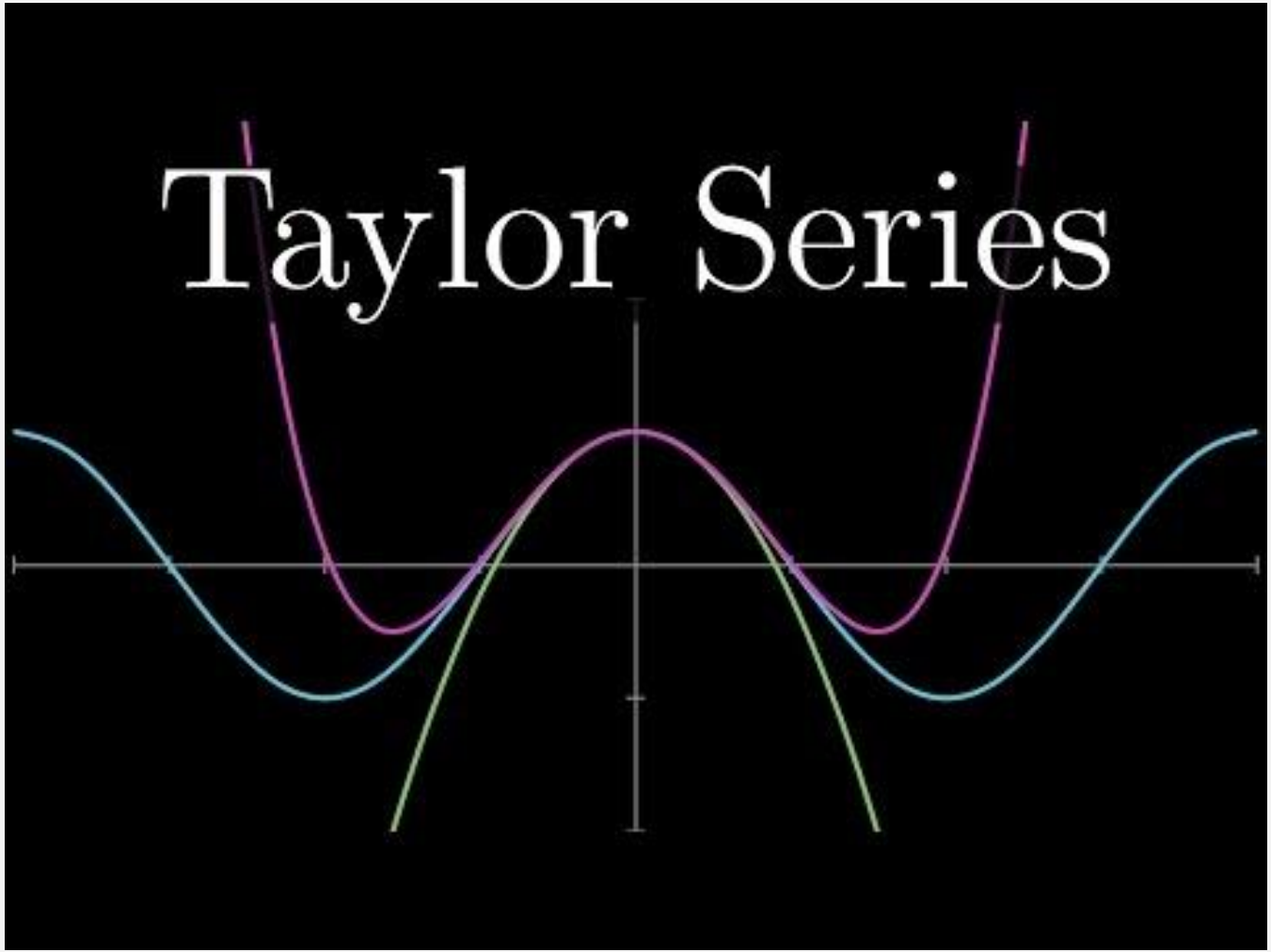
In mathematics, the **Taylor series** of a function is an infinite sum of terms that are expressed in terms of the function's derivatives at a single point. For most common functions, the function and the sum of its Taylor series are equal near this point.

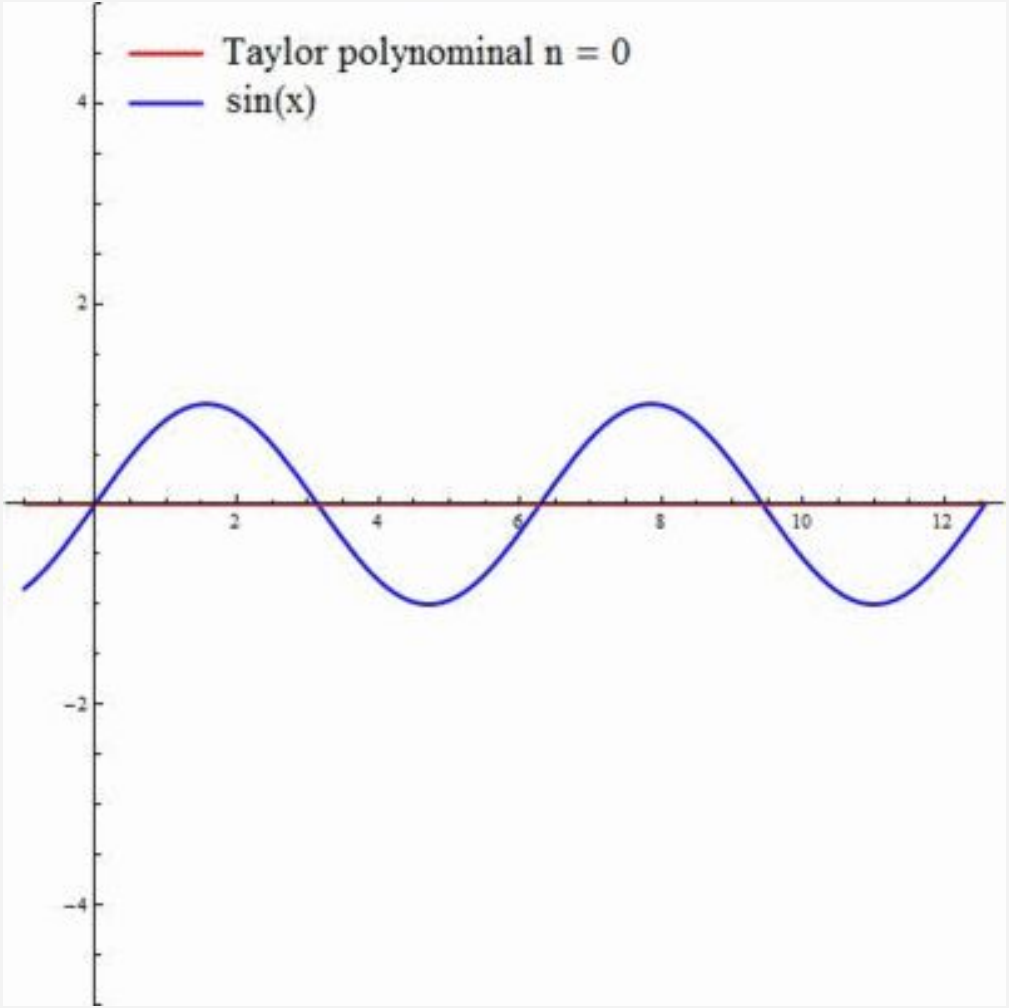
The Taylor series of a real or complex-valued function $f(x)$ that is infinitely differentiable at a real or complex number a is the power series

$$f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f'''(a)}{3!}(x-a)^3 + \cdots,$$

where $n!$ denotes the factorial of n. In the more compact sigma notation, this can be written as

$$\sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n,$$


Taylor Series


Taylor polynominal n = 0
sin(x)
4
2
-2
-4
2
4
6
8
10
12

Taylor Expansion Approximation

- Goal $Obj^{(t)} = \sum_{i=1}^{n} l\left(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)\right) + \Omega(f_t) + constant$
  - Seems still complicated except for the case of square loss
- 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 $g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}), \quad h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$

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

- *If you are not comfortable with this, think of square loss*

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

- Compare what we get to previous slide

Our New Goal

- Objective, with constants removed

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

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

- Why spending s much efforts to derive the objective, why not just grow trees …
  - Theoretical benefit: know what we are learning, convergence
  - **Engineering** benefit, recall the elements of supervised learning
    - g_i and h_i comes from definition of loss function
    - The learning of function only depend on the objective via g_i and h_i
    - Think of how you can separate modules of your code when you are asked to implement boosted tree for both square loss and logistic loss

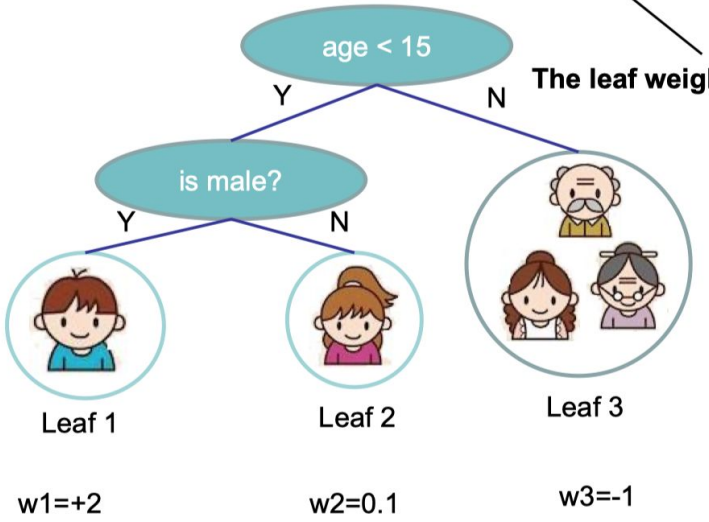
Re-define Tree

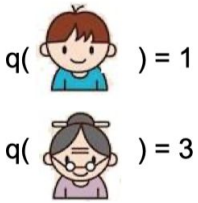
- We define tree by a vector of scores in leafs, and a leaf index mapping function that maps an instance to a leaf

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

The structure of the tree

The leaf weight of the tree



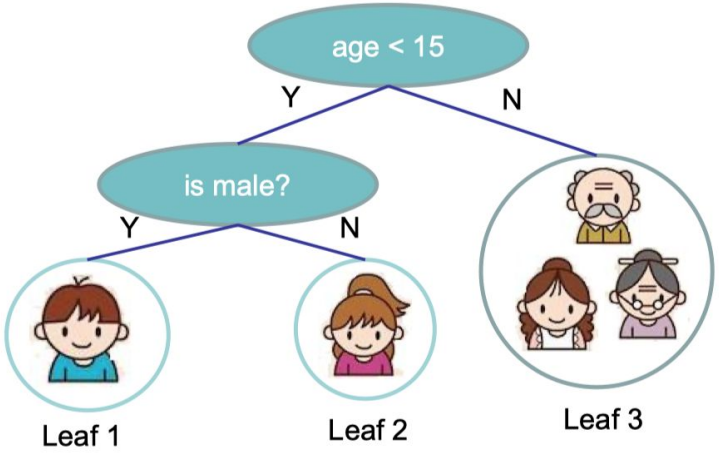


Complexity of a Tree

- Define complexity as (this is not the only possible definition)

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

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



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

Revisit the Objectives

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

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

- This is sum of T independent quadratic functions

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 $G_j = \sum_{i \in I_j} g_i \quad H_j = \sum_{i \in I_j} h_i$

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

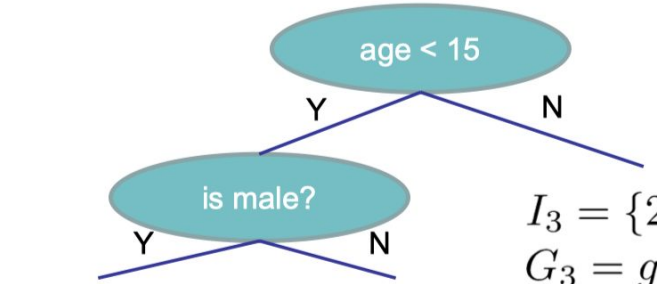
- Assume the structure of tree (q(x)) is fixed, the optimal weight in each leaf, and the resulting objective value are

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

This measures how good a tree structure is!

Structure Score Example

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



$$I_1 = \{1\} \quad I_2 = \{4\} \quad I_3 = \{2, 3, 5\}$$
$$G_1 = g_1 \quad G_2 = g_4 \quad G_3 = g_2 + g_3 + g_5$$
$$H_1 = h_1 \quad H_4 = h_4 \quad H_3 = h_2 + h_3 + h_5$$

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

The smaller the score is, the better the structure is

Searching Algorithm for a 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

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

The complexity cost by introducing additional leaf

the score of left child

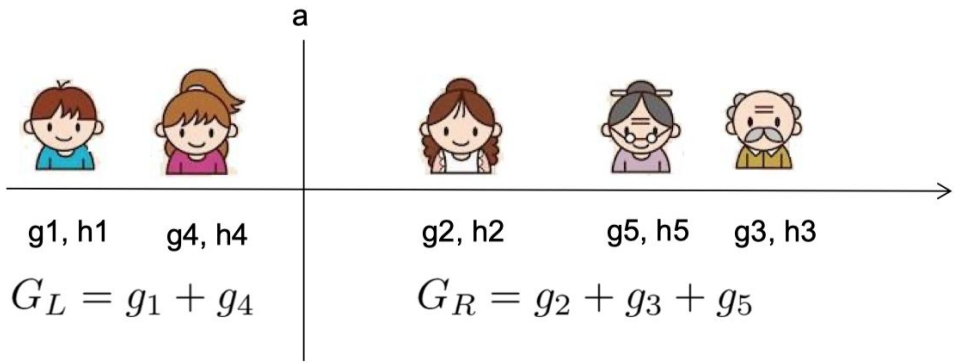
the score of right child

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



- 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

What about 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

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

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

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 predictivness

- 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

Recap: Boosted Tree Algorithm

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

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

- 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 $y^{(t)} = y^{(t-1)} + \epsilon f_t(x_i)$
  - ϵ 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

Summary

- The separation between model, objective, parameters can be helpful for us to understand and customize learning models
- The bias-variance trade-off applies everywhere, including learning in functional space

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

- We can be formal about what we learn and how we learn. Clear understanding of theory can be used to guide cleaner implementation.

04

Some other stuff

Other Things

Other than the core algorithm, the authors spend a lot of effort to make the model run efficiently

- Sparsity aware split finding
- Column block for parallel learning
- Cache-aware access
- Blocks for out-of core computation

Some Resources

- Paper
- Documentation
- For Classification
- Tuning Xgboost Model In Practice

Thank you!