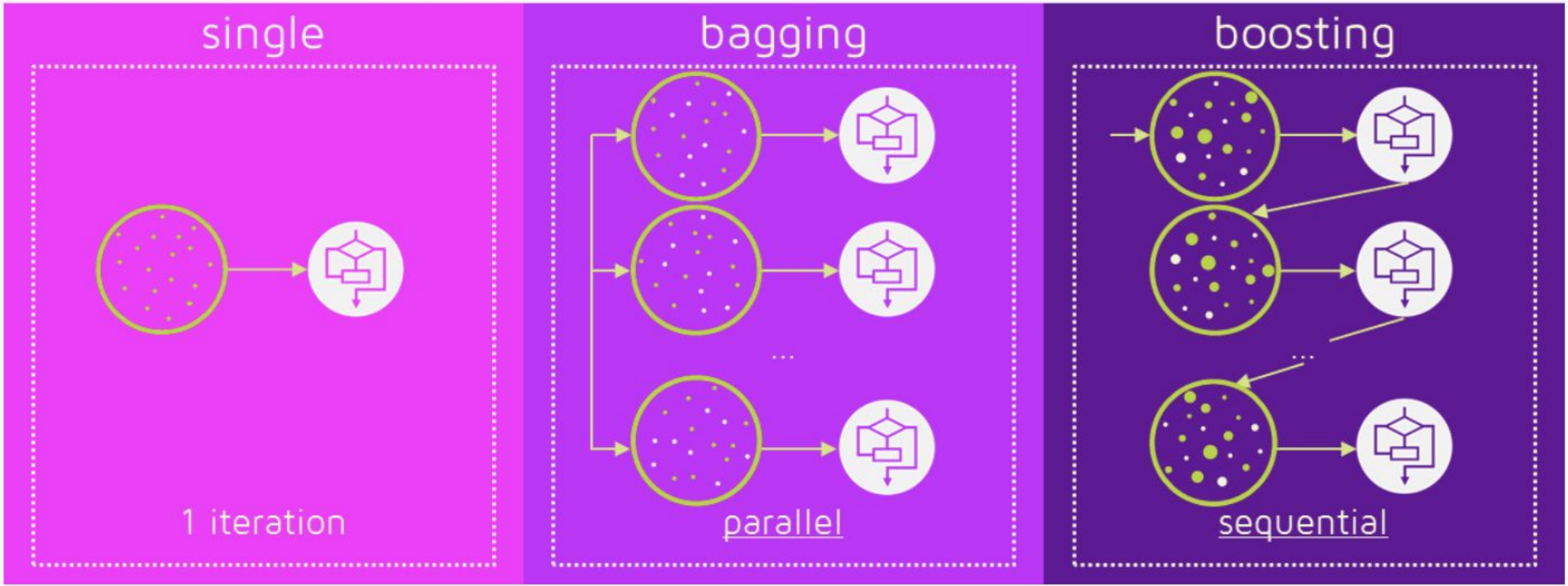
XGBoost: A Scalable Tree Boosting System

Lukáš Brchl

Boosting vs boosting



Boosting

- ML ensemble approach used for supervised learning
- It converts weak learners to strong ones
- A weak learner is a classifier that is only slightly correlated with the true classification
    - It can label examples better than random guessing
- A strong learner is a classifier well-correlated with the true classification

Boosting

- Iteratively learning weak classifiers with respect to a distribution and adding them to a final strong classifier
- The adding happens in weighted manner related to the weak learners accuracy
- After a weak learner is added, the data weights are "re-weighted".
- Misclassified input data gain a higher weight and correct ones lose weight
- Future weak learners focus more on the examples that previous weak learners misclassified
- Most known boosting algorithm is AdaBoost

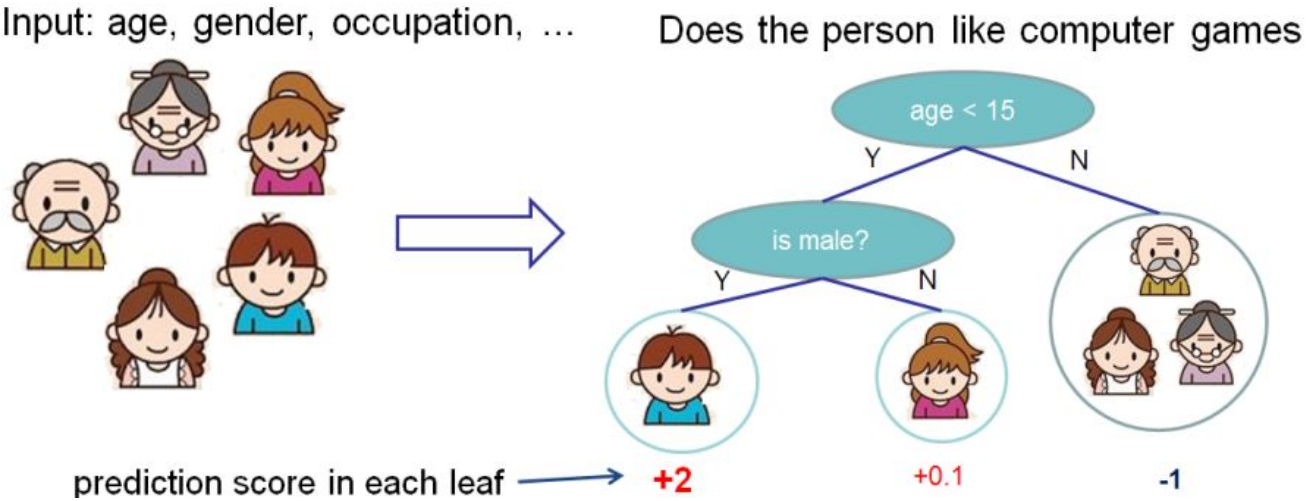
Gradient boosting

- Builds the model like other boosting methods do, but allowing optimization of an arbitrary differentiable loss function
- Can be interpreted as an optimization algorithm on a suitable cost function - suitable for gradient descent
- We iteratively choose a function (weak hypothesis) that points in the negative gradient direction

The difference is that instead of weighing the added individual classifiers, as in the AdaBoost, the gradient boosting trains the weak learners on the residues from the previous iteration to minimize the loss function

Gradient boosted trees

- Concrete implementation, which can be found in the scikit-learn package
- Most often used on CART trees
- Alternative names - Gradient boosted trees (GBT), Gradient Boosting Machine (GBM), Multiple Additive Regression Trees (MART)



Gradient boosted trees - terminology

- Set of training data with labels
  - $\{(x_i, y_i)\}_{i=1}^{n}$
- Boosted trees model prediction
  - $\hat{y} = F(x) = \sum_{i=0}^{M} F_i(x)$
- Loss function with continuous first derivative
  - $l(y, F(x))$
  - Example $l(y, F(x)) = \frac{1}{2}(y - F(x))^2$
- Number of algorithm iterations
  - M

Gradient boosted trees - algorithm

In each iteration m, $1 \leq m \leq M$ repeat:

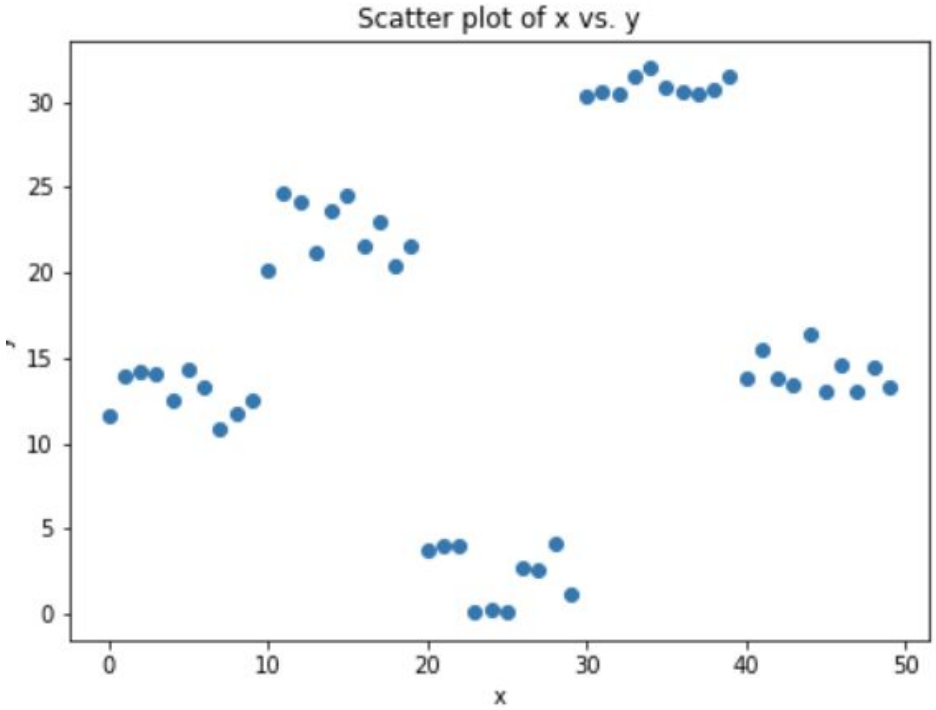
1. Calculate partial derivations
2. Train the weak learner tree on the residues
3. Calculate the multiplier γ that determines the ideal step size in the negative gradient direction
4. Create a new model that reduces the error from the previous iteration as

   $$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x)$$

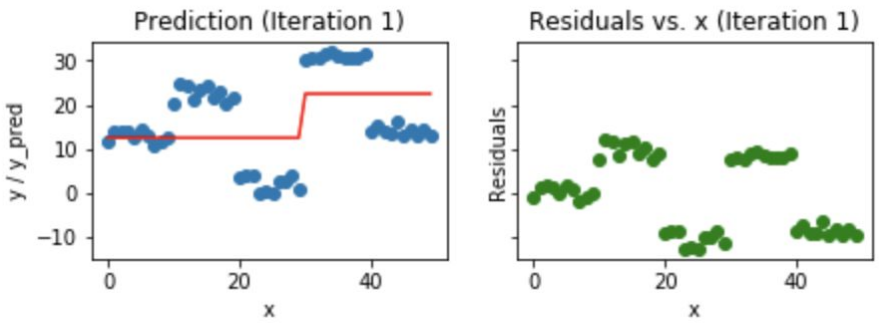
   Then, the final model is given as

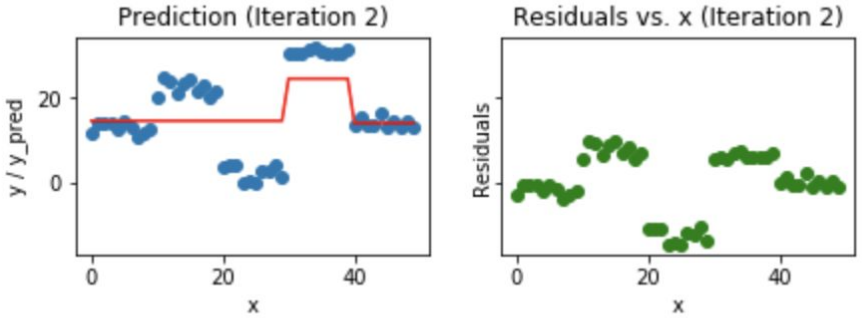
   $$F(x) = F_0(x) + \sum_{i=1}^{M} \gamma_i h_i(x)$$

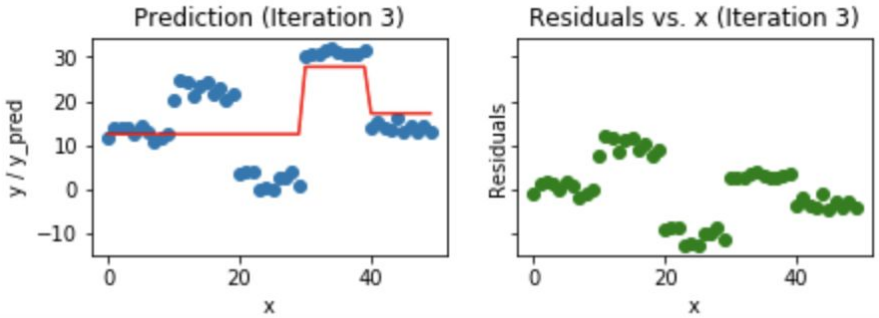
Gradient boosted trees - example



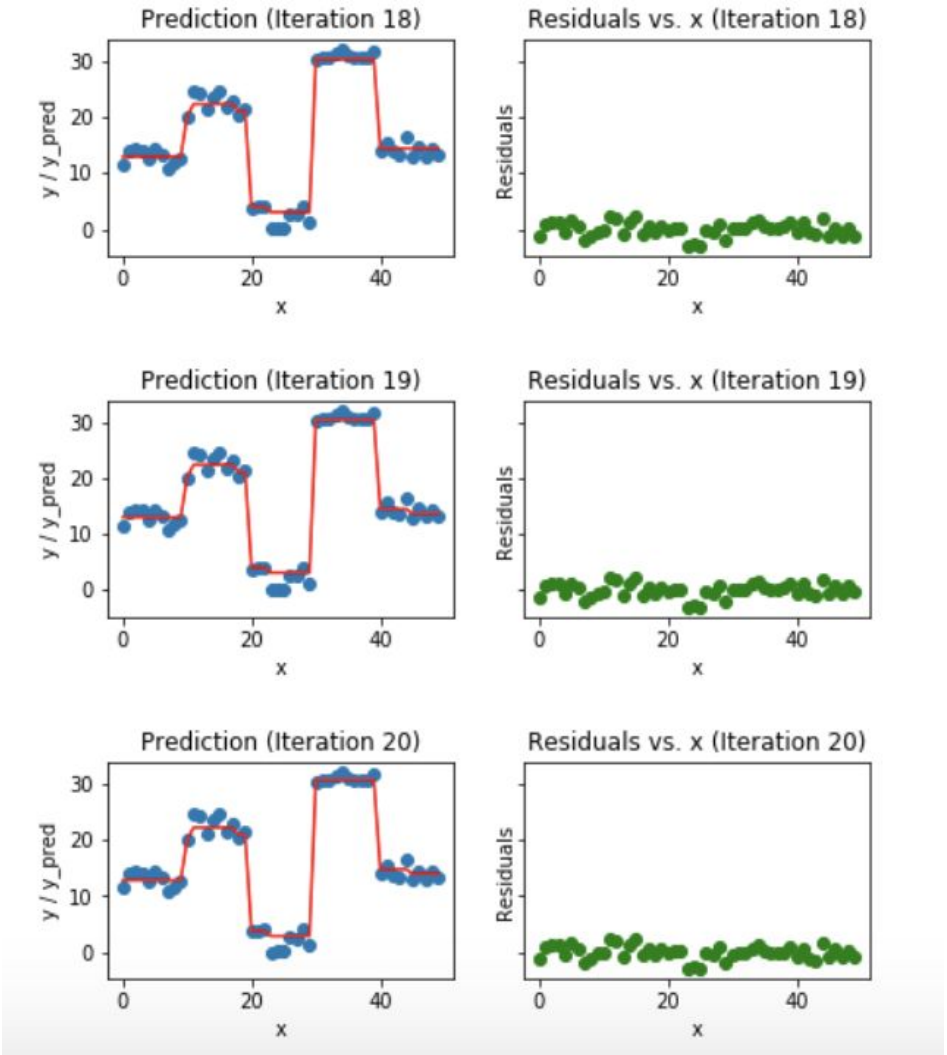
How do we choose the first model at iteration 0?







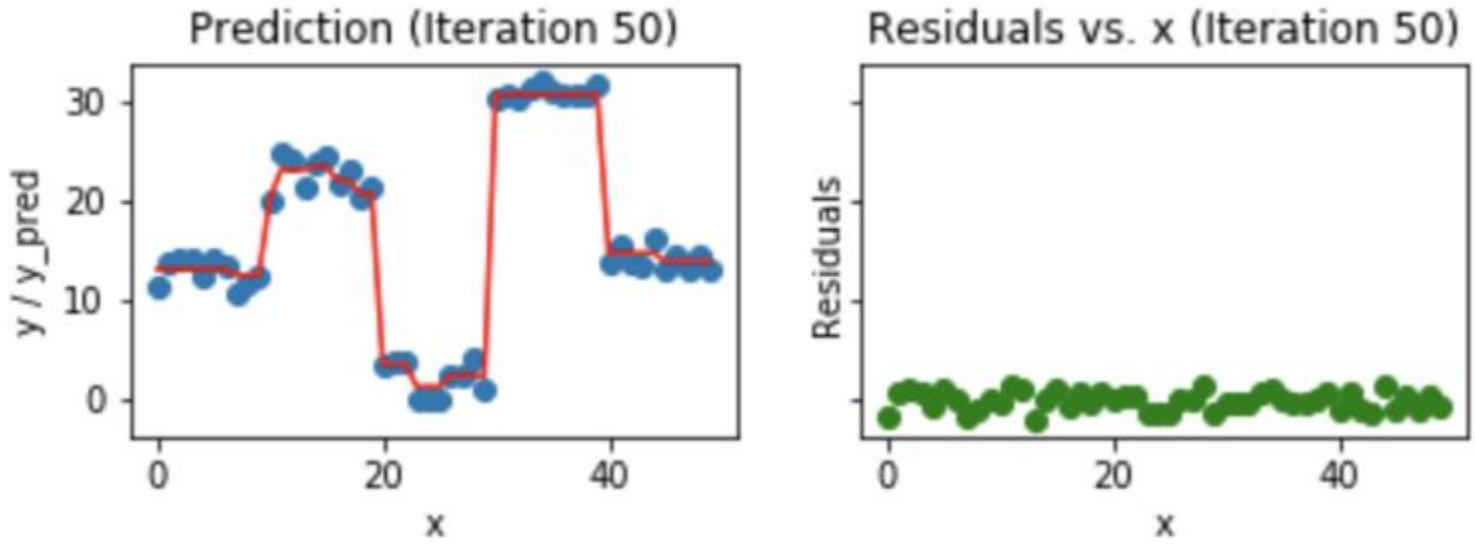
We can now see that residuals are randomly distributed



What happens if we won’t stop the training?

You are right, our favourite overfitting occurs!

How do we deal with the overfitting?



Gradient boosted trees - regularization

- Shrinkage
  - Adding multiple of α, 0 ≤ α ≤ 1 as a learning rate
  - $F_m(x) = F_{m-1}(x) + \alpha\gamma_m h_m(x)$

- Stochastic
  - The new model is not trained on the all data but only on a particular subset

- Model complexity penalization

What is XGBoost?

- An open-source library that implements a gradient boosting algorithm on trees
- Multiplatform and multilanguage - C++, Java, Python and R
- In 2015, 17 of the 29 Kaggle winners were using the XGBoost

The name XGBoost refers to the engineering goal to push the limit of computations resources for boosted tree algorithms

Why do we care about XGBoost?

Isn’t it easier to feed the data to the neural networks and pray?

Why do we care about XGBoost?

Isn’t it easier to feed the data to the neural networks and pray?

- Trees have proved to work well in most of the situations
- Very fast learning and inference
- Minimum data pre-processing needed
- Handling the missing data
- Scales up to billion instances
- Run on distributed systems using Hadoop and Spark libraries

- Custom loss functions, sparse-aware data approach, out-of-core computing, smart cache structure, parallelization….

XGBoost magic revealed

This is how we make predictions - $\hat{y}_i = \sum_{k=1}^{K} f_k(x_i), \quad x_i \in R^d, f_k \in F, \quad d, i, K \in \mathbb{N}$

This is objective function we try to minimize - $Obj = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$

It measures how our model is good with predictions, but also accounts the model complexity

We can choose any loss function, for e.g. - $l(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2$

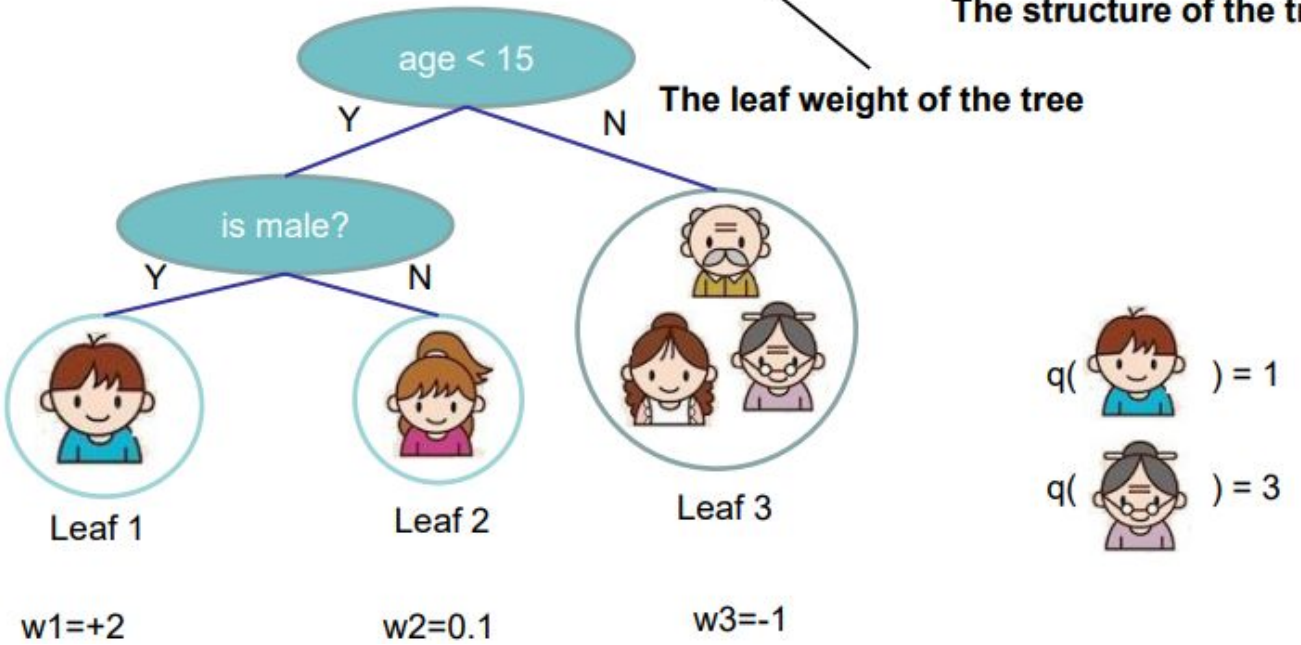
A good regularization that accounts the complexity is - $\Omega(f_k) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$

XGBoost magic revealed - tree structure

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

The structure of the tree

The leaf weight of the tree



XGBoost magic revealed - simplifying objective function

How do we find $f_t(x_i)$ that minimizes this expression?

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

Not that easy right? Because $f_t(x_i)$ is the tree

XGBoost magic revealed - simplifying objective function

How do we find $f_t(x_i)$ that minimizes this expression?

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

We can approximate the loss function with second order Taylor series

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

where

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

But why do we do that?

XGBoost magic revealed - simplifying objective function

We can drop some constants and substitute the regularization

$$Obj^{(t)} \simeq \sum_{i=1}^{n} \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \gamma T + \frac{1}{2} \lambda \sum_{j=1}^{T} w_j^2$$

XGBoost magic revealed - simplifying objective function

We can drop some constants and substitute the regularization

$$Obj^{(t)} \simeq \sum_{i=1}^{n} \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \gamma T + \frac{1}{2} \lambda \sum_{j=1}^{T} w_j^2$$

Let's put some magic into it and play with sums (read handout)

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

And then, just another substitution to make it lovely

$$Obj^{(t)} \simeq \sum_{j=1}^{T} \left[G_j w_j + \frac{1}{2} (H_j + \lambda) w_j^2 \right] + \gamma T$$

XGBoost magic revealed - simplifying objective function

Now, how do we find the minimum of the parabola?

$$Obj^{(t)} \simeq \sum_{j=1}^{T} \left[G_j w_j + \frac{1}{2}(H_j + \lambda) w_j^2 \right] + \gamma T$$

XGBoost magic revealed - simplifying objective function

Now, how do we find the minimum of the parabola?

$$Obj^{(t)} \simeq \sum_{j=1}^{T} \left[G_j w_j + \frac{1}{2}(H_j + \lambda) w_j^2 \right] + \gamma T$$

I hope you remember this from BI-ZMA

$$\frac{\partial (G_j w_j + \frac{1}{2}(H_j + \lambda) w_j^2)}{\partial w_j} = 0$$

$$G_j + (H_j + \lambda) w_j = 0$$

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

XGBoost magic revealed - simplifying objective function

Now, how do we find the minimum of the parabola?

$$Obj^{(t)} \simeq \sum_{j=1}^{T} \left[G_j w_j + \frac{1}{2}(H_j + \lambda) w_j^2 \right] + \gamma T$$

I hope you remember this from BI-ZMA

$$\frac{\partial (G_j w_j + \frac{1}{2}(H_j + \lambda) w_j^2)}{\partial w_j} = 0$$

$$G_j + (H_j + \lambda) w_j = 0$$

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

XGBoost magic revealed - simplifying objective function

After substituting the optimal weights, we finally get the score $Obj^{(t)} \simeq -\frac{1}{2}\sum_{j=1}^{T}\frac{G_j^2}{H_j+\lambda} + \gamma T$

which is the final form expressing the minimum value of the objective function in the iteration t for the predefined tree structure

How do we find the optimal tree structure?

XGBoost magic revealed - simplifying objective function

After substituting the optimal weights, we finally get the score $Obj^{(t)} \simeq -\frac{1}{2}\sum_{j=1}^{T}\frac{G_j^2}{H_j+\lambda} + \gamma T$

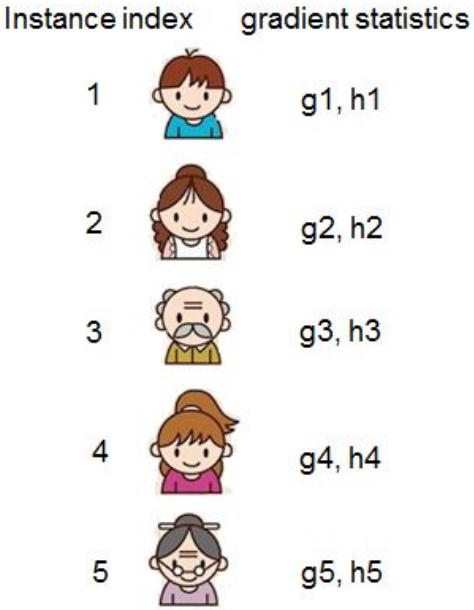
which is the final form expressing the minimum value of the objective function in the iteration t for the predefined tree structure

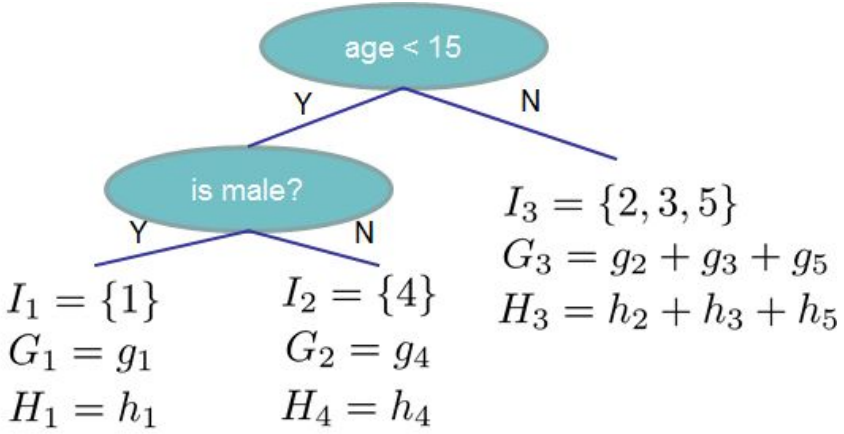
How do we find the optimal tree structure?

We use greedy approach, but go read the handout!

XGBoost magic revealed - simplifying objective function

After substituting the optimal weights, we finally get the score $Obj^{(t)} \simeq -\frac{1}{2}\sum_{j=1}^{T}\frac{G_j^2}{H_j+\lambda} + \gamma T$





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

The smaller the score is, the better the structure is

And the last bit to achieve overall satisfaction

XGBoost is faster compared to **20 years old** GBM from scikit

Method	Time per Tree (sec)	Test AUC
XGBoost	0.6841	0.8304
XGBoost (colsample=0.5)	0.6401	0.8245
scikit-learn	28.51	0.8302
R.gbm	1.032	0.6224

Table 1: Performance comparison of Exact Greedy Methods with 500 trees on Higgs-1M data

References

https://arxiv.org/abs/1603.02754

https://statweb.stanford.edu/~jhf/ftp/trebst.pdf

https://medium.com/mlreview/gradient-boosting-from-scratch-1e317ae4587d