

Lecture 25: Boosting and Gradients

Big Data and Machine Learning for Applied Economics

Econ 4676

Ignacio Sarmiento-Barbieri

Universidad de los Andes

November 2, 2021

Announcements

- ▶ Problem Set 4: Next Friday presentations
- ▶ Thursday you need to submit a .csv it at 8:00 pm.
 - ▶ Please upload it to your repo don't forget to follow the instructions, if you have questions ask before hand, **not at 7:30pm before submission!**
 - ▶ The lowest the better the score (smaller loss)
 - ▶ If you forget to send me the number of parameters I'll assign 100,000
 - ▶ If I can't grab you predictions file from your repo with grep you won't get credit for the problem set.
- ▶ I've uploaded the final presentation schedule

Announcements

- ▶ I've uploaded the final presentation schedule
- ▶ Presentations will be in the following order:
 - ▶ November 30th
 - 1 Bares y Rendimiento Educativo (Arrieta y Montero)
 - 2 Startup Failure (Rodriguez)
 - 3 Covid-19 (Saenz)
 - 4 Precios Propiedades (Gonzalez)
 - ▶ December 2nd
 - 1 Cambio Estructural (Rengifo)
 - 2 Booktopia (Agudelo, Cepeda, Cifuentes, y Mosquera) ✓
 - 3 Rendimiento Educativo (Salazar, Cortes, Rojas, y Peña) ✓
 - ▶ December 3rd
 - 1 Pobreza Multidimensional (Miranda)
 - 2 Demanda de Energía (Ramírez y Castro)
 - 3 Basketball (Segura, Prieto y Navarro)

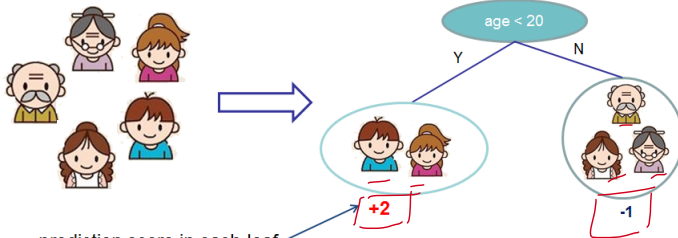
Agenda

- 1 Announcements
- 2 Recap: Trees, Forests, and Boosting
- 3 Boosting
 - Motivation
 - Boosting Trees
- 4 XGBoost
- 5 Review & Next Steps
- 6 Further Readings

Trees

Input: age, gender, occupation, ...

Like the computer game X



prediction score in each leaf

source: <https://xgboost.readthedocs.io/en/latest/tutorials/model.html>

Forests

$$\boxed{\text{orig}}_N \quad \boxed{b-1}_N \quad - \quad \boxed{3=\beta}_N$$

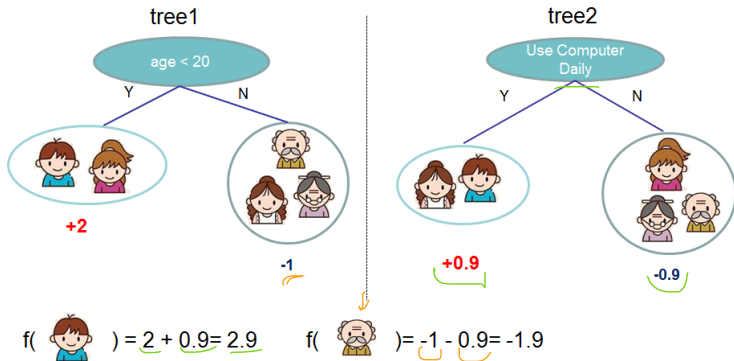
- ▶ We can improve performance a lot using either bootstrap aggregation (bagging), random forests, or boosting.
- ▶ Bagging & Random Forests:
 - ▶ Repeatedly draw bootstrap samples $(X_i^b, Y_i^b)_{i=1}^N$ from the observed sample.
 - ▶ For each bootstrap sample, fit a regression tree $\hat{f}^b(x)$
 - ▶ Bagging: full sample *set of predictors*
 - ▶ Random Forests: subset of predictors \sqrt{p} (breaks high correlation)
 - ▶ Average across bootstrap samples to get the predictor

$$\hat{f}_{bag} = \frac{1}{B} \sum_{b=1}^B \hat{f}^b(x)$$

$$\begin{aligned} V(x) &= \sigma^2 \\ V(\bar{x}) &= \frac{\sigma^2}{n} \end{aligned} \quad (1)$$

- ▶ Basically we are smoothing predictions.
- ▶ Idea: the variance of the average is less than that of a single prediction.

Forests



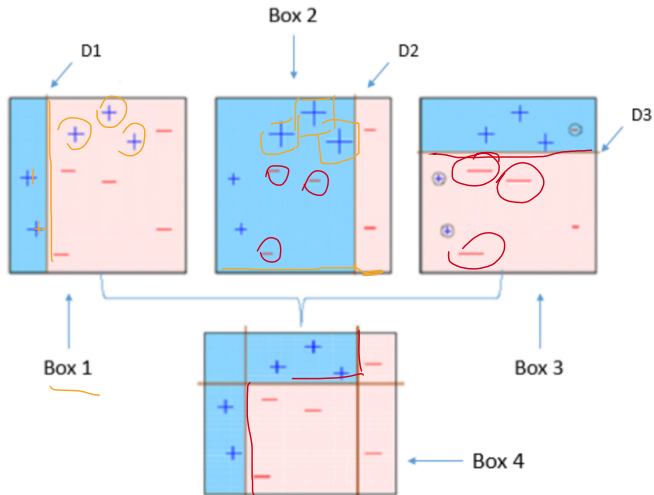
source: <https://xgboost.readthedocs.io/en/latest/tutorials/model.html>

Boosting: Motivation

- ▶ Boosting is one of the most powerful learning ideas introduced in the last twenty years.
- ▶ It was originally designed for classification problems, but can be extended to regression as well.
- ▶ The motivation for boosting was a procedure that combines the outputs of many “weak” classifiers to produce a powerful ensemble, “committee.”

Boosting: Motivation

AdaBoost



Boosting: Motivation

- ▶ The idea of ensemble learning is to build a prediction model by combining the strengths of a collection of simpler base models.
- ▶ Bagging and random forests form committee of trees, where each cast a vote for the predicted class.
- ▶ Boosting is like as a committee method as well, although unlike random forests, the committee of weak learners evolves over time, and the members cast a weighted vote.

Boosting Trees

- ▶ The goal here is to solve something which looks like

$$\textcircled{f^*} = \operatorname{argmin}_{f \in \mathcal{F}} \left\{ \sum_{i=1}^n L(y_i, f(\mathbf{x}_i)) \right\} \quad (2)$$

- ▶ for some loss function L , and for some set of predictors \mathcal{F} .
- ▶ This is an optimization problem.
- ▶ Note that here in a function space, so we are solving for a function not a point.

Boosting Trees

- From from a numerical perspective, optimization is solve using gradient descent (this is why this technique is also called gradient boosting).

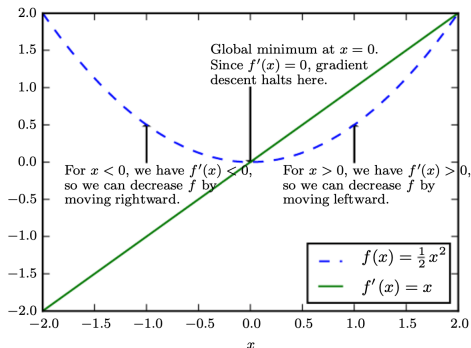


Figure 4.1: Gradient descent. An illustration of how the gradient descent algorithm uses the derivatives of a function to follow the function downhill to a minimum.

Source: Goodfellow, I., Bengio, Y., Courville, A., & Bengio, Y. (2016). Deep learning. Cambridge: MIT press.

Boosting Trees

- ▶ Again, the optimum is not some some real value x^* , but some function f^* .
- ▶ Thus, here we will have something like

$$f^m = f^{m-1} + \underset{\phi \in H}{\operatorname{argmin}} \sum_{i=1}^n \mathcal{L}(y_i, f^{m-1}(x_i) + \phi(x_i)) \quad (3)$$

- ▶ If we consider using mean squared error (MSE) as our loss function, the objective becomes

$$\sum_{i=1}^N (f_i^m - \hat{f}_i^{m-1} + \phi_m(x_i))^2 \quad (4)$$

- ▶ For other losses of interest (for example, logistic loss), it is not so easy to get such a nice form.

Boosting Trees: Algorithm explained

- So in the general case, we take the Taylor expansion of the loss function:

$$obj^m = \sum_{i=1}^N [L(f_i, \hat{f}_i^{(m)}) + g_{im} \phi_m(x_i)] \quad (5)$$

where

$$g_{im} = - \left. \frac{\partial L(f_i, \phi(\mathbf{x}_i))}{\partial \phi(\mathbf{x}_i)} \right|_{\phi(\mathbf{x}_i) = \phi^{(m-1)}(\mathbf{x}_i)} \quad (6)$$

- The goal is to fit a model so that $g_{im} = \phi^*(x_i)$
- when we have that optimal function, set $f^m = f^{m-1} + \phi^*(x_i)$

Boosting Trees: Algorithm explained

- ▶ After we remove all the constants, the specific objective at step m becomes

$$obj^m = \sum_{i=1}^N [g_{im}\phi_m(x_i)] \quad (7)$$

- ▶ This becomes our optimization goal for the new tree. One important advantage of this definition is that the value of the objective function only depends on g_i

Boosting Trees

Implementations of Gradient Boosting

► Gradient Tree Boosting Algorithm

- 1 Initialize $f_0(x) = 0$ and $g_i = y_i$ for all i
- 2 for $m = 1$ to M :
 - 1 For $i = 1, 2, \dots, N$ compute g_{im}
 - 2 Fit a regression tree to the targets g_{im} giving terminal regions R_{jm} $j = 1, 2, \dots, J_m$
 - 3 For $j = 1, 2, \dots, J_m$ compute

$$c_{jm} = \underset{c}{\operatorname{argmin}} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + c) \quad (8)$$

- 4 Update $f_m(x) = f_{(m-1)}(x) + \sum_{j=1}^{J_m} c_{jm} I(x \in R_{jm})$

- 3 Output $\hat{f}(x) = \sum_i^M f_m(x)$

Boosting Trees: Algorithm explained

- ▶ Learning tree structure is much harder than traditional optimization problem where you can simply take the gradient.
- ▶ It is intractable to learn all the trees at once.
- ▶ Instead, we use an additive strategy

Boosting Trees: Algorithm explained

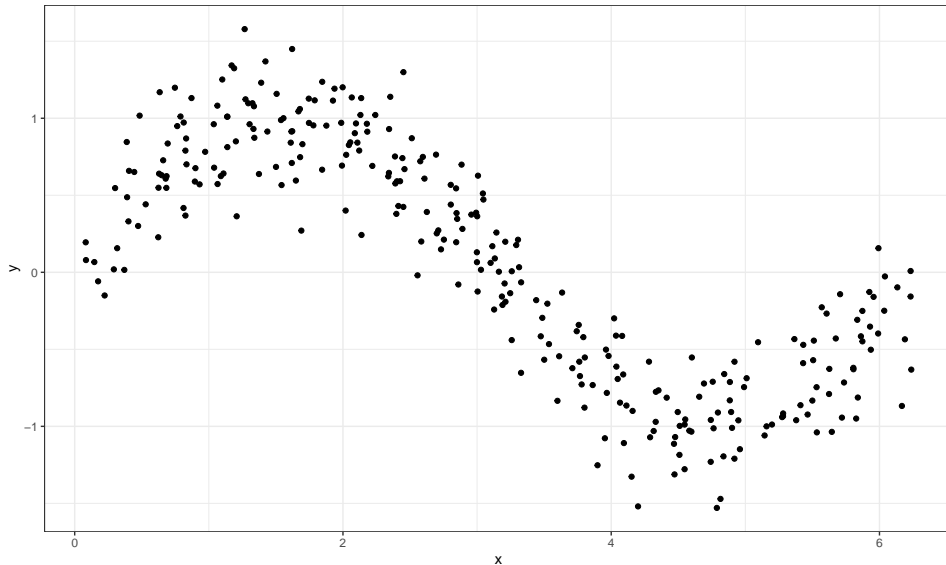
- Instead, we use an additive strategy: fix what we have learned, and add one new tree at a time. We write the prediction value at step m as \hat{y}_i^m .

$$\begin{aligned}\hat{y}_i^0 &= 0 \\ \hat{y}_i^1 &= \hat{y}_i^0 + f_1(x_i) \\ &\dots \\ \hat{y}_i^M &= \sum_{m=1}^M f_m(x_i) = \hat{y}_i^{m-1} + f_m(x_i)\end{aligned}\tag{9}$$

- Which tree do we want at each step? Add the one that optimizes our objective.

$$obj^m = \sum_{i=1}^N L(y_i, \hat{y}_i^{(m)}) = \sum_{i=1}^N [g_{im} \phi_m(x_i)]\tag{10}$$

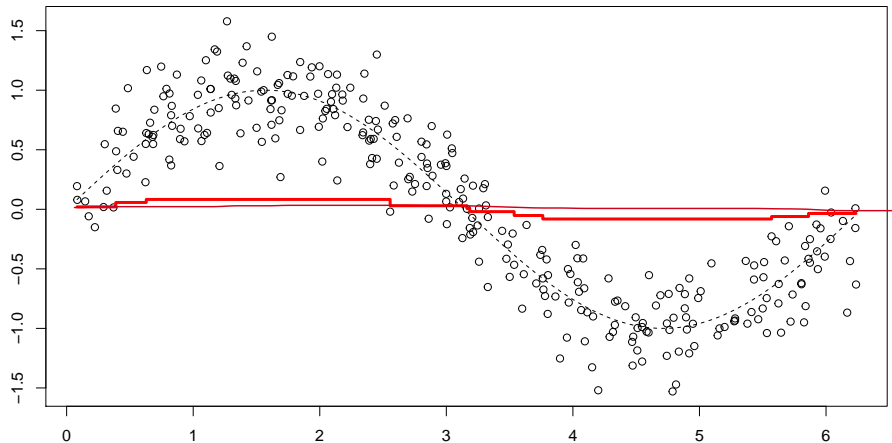
Boosting Trees: Example



Boosting Trees: Example

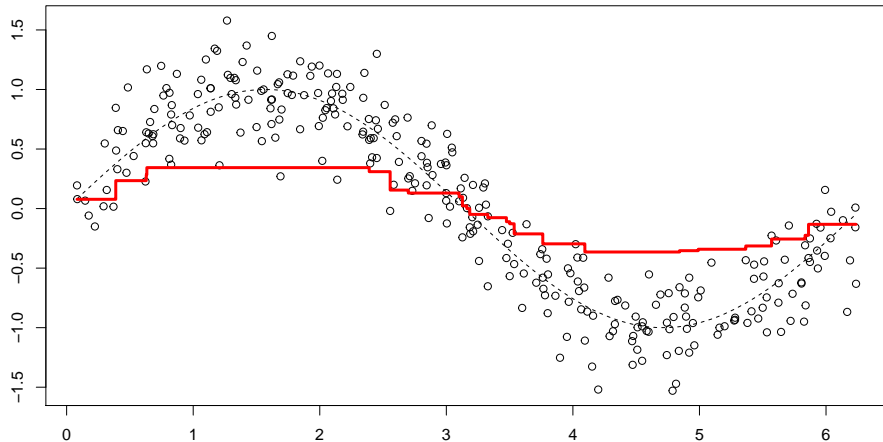
► Algorithm:

$M < -2$



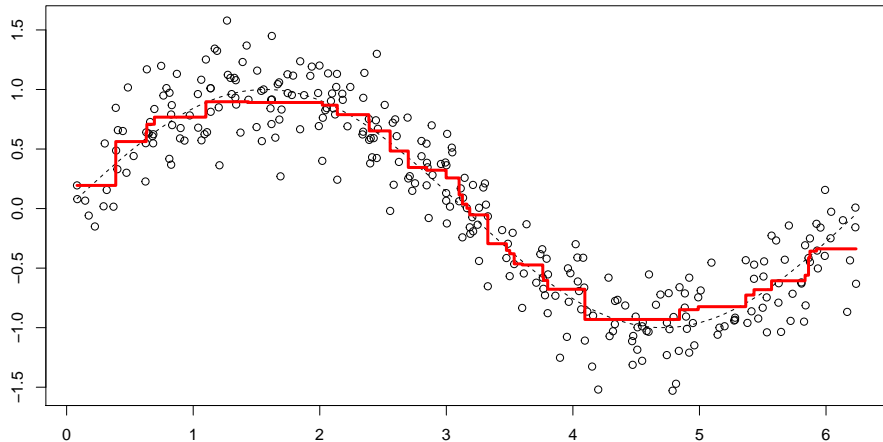
Boosting Trees: Example

$M < -10$



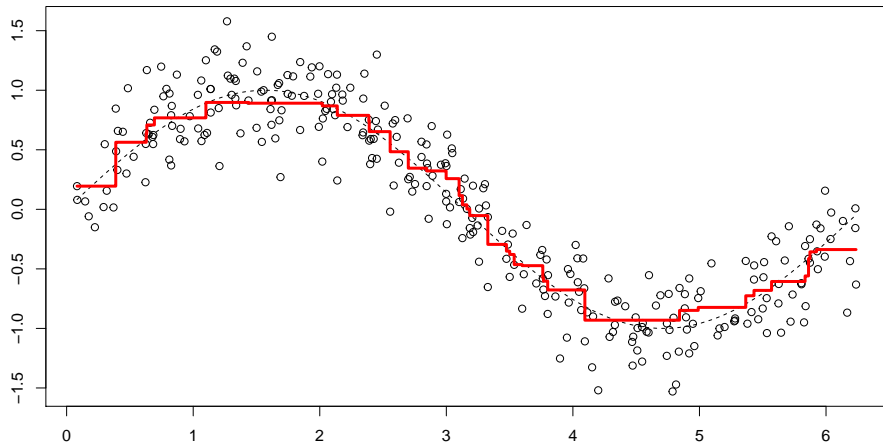
Boosting Trees: Example

$M < -100$



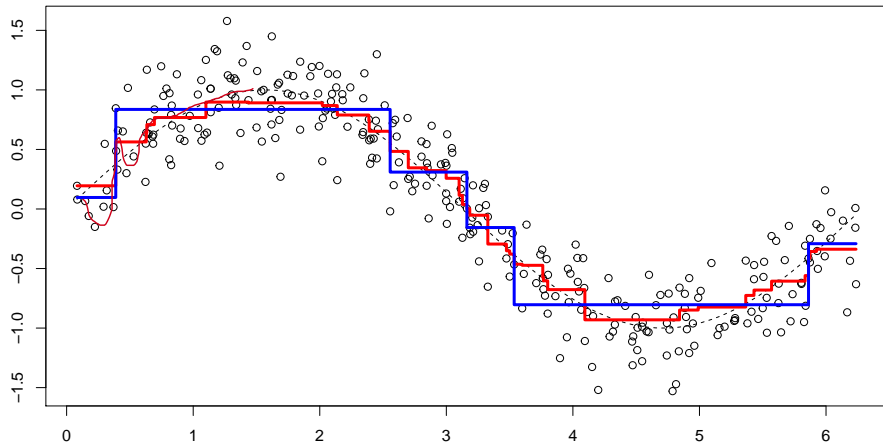
Boosting Trees: Example

$M < -300$



Boosting Trees: Example

- Simple tree (blue), boosted tree (red)



Boosting Trees: Parameters

- ▶ How many iterations (M)?
 - ▶ Each iteration usually reduces the training risk $L(\cdot)$, so that for M large enough this risk can be made arbitrarily small.
 - ▶ However, fitting the training data too well can lead to overfitting, which degrades the risk on future predictions.
 - ▶ We use cross-validation

Boosting Trees: Parameters

► Can we have shrinkage?

- The simplest implementation of shrinkage in the context of boosting is to scale the contribution of each tree by a factor $\nu \in (0, 1)$ when it is added to the current approximation. That is, we replace step

$$\hat{y}_m(x) = f_{(m-1)}(x) + \nu \phi_m(x_i) \quad (11)$$

- Empirically it has been found that smaller values of ν favor better test error, and require correspondingly larger values of M .
- the best strategy appears to be to set ν to be very small ($\nu < 0.1$) 0, 0.01
- This yields dramatic improvements (over no shrinkage $\nu = 1$)



Boosting Trees: Parameters



- ▶ How deep should my trees be?
 - ▶ Often using stumps works well (Tree with a single split)
 - ▶ In this case, the boosted ensemble is fitting an additive model, since each term involves only a single variable.
- ▶ Can I subsample? Yes, rows and columns
 - ▶ With stochastic gradient boosting, at each iteration we sample a fraction η of the training observations (without replacement), and grow the next tree using that subsample.
 - ▶ Reduces the computing time by the same fraction η , and some cases improves prediction
 - ▶ Same for columns and breaks high correlation (like forests)

XGBoost: Motivation

- ▶ Now that you understand decision trees and gradient boosting, XGBoost is a walk in the park
- ▶ It is a gradient boosting algorithm that uses decision trees
- ▶ Beyond that, its implementation was specifically engineered for optimal performance and speed.
- ▶ Why talk about XGBoost?
 - ▶ Among the 29 challenge winning solutions published in Kaggle's blog during 2015, 17 solutions used XGBoost.
 - ▶ Among these solutions, eight solely used XGBoost to train the model, while most others combined XGBoost with neural nets in ensembles. (The second most popular method, deep neural nets, was used in 11 solutions)
 - ▶ The success of the system was also witnessed in 2015 Data Mining and Knowledge Discovery competition organized by ACM (KDD Cup) , where XGBoost was used by every winning team in the top-10.
 - ▶ Historically, XGBoost has performed quite well for structured, tabular data. But, if you are dealing with non-structured data such as images, neural networks are usually a better option (more on this later)



XGBoost is a Boosting Tree

- ▶ Which tree do we want at each step?
- ▶ Add the one that optimizes our objective.

$$\mathcal{L} = \underbrace{\sum_{i=1}^N L(y_i, \hat{y}_i)}_{\text{loss}} + \underbrace{\sum_{k=1}^m \Omega(f_k)}_{\text{penalty}} \quad (12)$$

- ▶ $L(\cdot)$ is a differentiable convex loss function that measures the difference between the prediction \hat{y}_i and the target y_i .
- ▶ The second term $\Omega(f)$ penalizes the complexity of the model, where

$$\Omega(f) = \underbrace{\gamma T}_{\text{number of nodes}} + \frac{1}{2} \lambda \underbrace{\|\omega\|_2}_{(\omega - 0)^2} \quad (13)$$

XGBoost is a Boosting Tree

- ▶ The function above cannot be optimized using traditional optimization methods
- ▶ Let \mathcal{L}^m be the loss at step m , then \hat{y}_i^m is the prediction of row i at the m iteration
- ▶ We need to add f_m to minimize the following objective

$$\mathcal{L}^m = \sum_{i=1}^N (y_i - \hat{y}_i^{m-1} + f_m(x_i))^2 + \Omega(f_t) \quad (14)$$

- ▶ So in the general case, we take a second order Taylor expansion of the loss function:

$$\mathcal{L}^m = \sum_{i=1}^N \left[L(y_i, \hat{y}_i^{(m-1)}) + g_{im} f_m(x_i) + \frac{1}{2} h_{im} f_m^2(x_i) \right] + \Omega(f_t) \quad (15)$$

where $g_{im} = \frac{\partial L(y_i, \hat{y}_i^{(m-1)})}{\partial f^{(m-1)}(x_i)}$ and $h_{im} = \frac{\partial^2 L(y_i, \hat{y}_i^{(m-1)})}{\partial (f^{(m-1)}(x_i))^2}$

XGBoost is a Boosting Tree

- ▶ After we remove all the constants, the specific objective at step m becomes

$$\mathcal{L}^m = \sum_{i=1}^N \left[\underbrace{g_{im}f_m(x_i)} + \underbrace{\frac{1}{2}h_{im}f_m^2(x_i)} \right] + \underbrace{\Omega(f_m)} \quad (16)$$

- ▶ This becomes our optimization goal for the new tree.
- ▶ One important advantage of this definition is that the value of the objective function only depends on g_{im} and h_{im}

XGBoost Model Complexity

- ▶ Let's talk about the regularization term
- ▶ We need to define the complexity of the tree $\Omega(f)$.
- ▶ But us first refine the definition of the tree $f(x)$ as

$$f_t(x) = \underline{w}_{q(x)}, w \in R^T, q : R^d \rightarrow \{1, 2, \dots, T\}. \quad (17)$$

- ▶ Here w is the vector of predictions on leaves, q is a function assigning each data point to the corresponding leaf, and T is the number of leaves.

XGBoost Model Complexity

$Q \rightarrow T$

- In XGBoost, they define the complexity as

$$\Omega(f) = \underbrace{\gamma T}_{\text{penalty}} + \underbrace{\frac{1}{2} \lambda \sum_{j=1}^T w_j^2}_{\text{regularization}} \quad (w - 0)^2 \quad (18)$$

- The regularization is one part most tree packages treat less carefully, or simply ignore.
- This was because the traditional treatment of tree learning only emphasized improving impurity, while the complexity control was left to heuristics.
- By defining it formally, we can get a better idea of what we are learning and obtain models that perform well in the wild.

XGBoost Model Complexity

- ▶ With the above notation we can rewrite the objective function as

$$\mathcal{L}^{(t)} \approx \sum_{i=1}^n [g_i w_{q(x_i)} + \frac{1}{2} h_i w_{q(x_i)}^2] + \gamma T + \frac{1}{2} \lambda \sum_{j=1}^T w_j^2 \quad (19)$$

$$= \sum_{j=1}^T [(\sum_{i \in I_j} g_i) w_j + \frac{1}{2} (\sum_{i \in I_j} h_i + \lambda) w_j^2] + \gamma T \quad (20)$$

- ▶ where $I_j = \{i | q(x_i) = j\}$ set of indicators that assign to the j -th leaf.
- ▶ Note the following in the second line, we changed the index of the summation because all the data points on the same leaf get the same score.
- ▶ Defining $G_j = \sum_{i \in I_j} g_i$ and $H_j = \sum_{i \in I_j} h_i$

$$\mathcal{L}^{(t)} = \sum_{j=1}^T [G_j w_j + \frac{1}{2} (H_j + \lambda) w_j^2] + \gamma T \quad (21)$$

XGBoost Model Complexity

- ▶ In the above equation, w_j are independent with respect to each other,
- ▶ the form $G_j w_j + \frac{1}{2}(H_j + \lambda)w_j^2$ is quadratic
- ▶ then the best w_j for a given structure tree structure $q(x)$ get is:

$$w_j^* = -\frac{G_j}{H_j + \lambda} \quad (22)$$

(23)

- ▶ and the best objective reduction we can get

$$\mathcal{L}^* = -\frac{1}{2} \sum_{j=1}^T \frac{G_j^2}{H_j + \lambda} + \gamma T \quad (24)$$






Improvement (handwritten red text above the sum term)

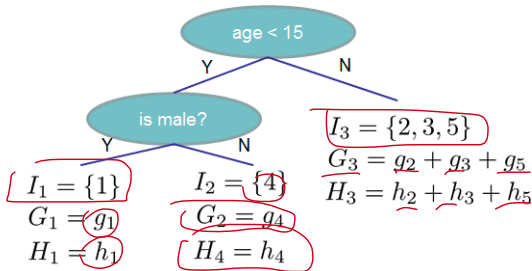
penalization (handwritten red text pointing to the γT term)

XGBoost Model Complexity: Example

$T \approx 3$

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

source: <https://xgboost.readthedocs.io/en/latest/tutorials/model.html>

Learn the tree structure

- ▶ Now that we have a way to measure how good a tree is,
- ▶ Ideally we would enumerate all possible trees and pick the best one.
- ▶ In practice this is intractable, so we will try to optimize one level of the tree at a time.
- ▶ Specifically we try to split a leaf into two leaves, and the score it gains is

$$Gain = \frac{1}{2} \left[\underbrace{\frac{G_L^2}{H_L + \lambda}} + \underbrace{\frac{G_R^2}{H_R + \lambda}} - \underbrace{\frac{(G_L + G_R)^2}{H_L + H_R + \lambda}} \right] \underbrace{- \gamma}_{\gamma} \quad (25)$$

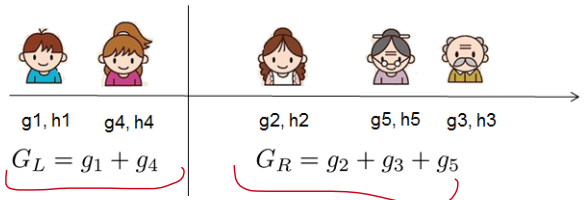
Learn the tree structure

$$Q - \frac{\alpha T}{2}$$

- ▶ This formula can be decomposed as
 - 1 the score on the new left leaf
 - 2 the score on the new right leaf
 - 3 the score on the original leaf
 - 4 regularization on the additional leaf.
- ▶ We can see an important fact here: if the gain is smaller than γ , we would do better not to add that branch. This is exactly the pruning techniques in tree based models
- ▶ For real valued data, we usually want to search for an optimal split. To efficiently do so, we place all the instances in sorted order

Learn the tree structure

- ▶ A left to right scan is sufficient to calculate the structure score of all possible split solutions, and we can find the best split efficiently



source: <https://xgboost.readthedocs.io/en/latest/tutorials/model.html>

Review & Next Steps

- ▶ Gradient Based Optimization
- ▶ Boosting Trees
 - ▶ How it works
- ▶ XGBoost
 - ▶ How it works
- ▶ Preview next class: superlearners and text
- ▶ Questions? Questions about software?

Further Readings

- ▶ Charpentier, Arthur (2018). Classification from scratch, boosting.
<https://freakonometrics.hypotheses.org/tag/xgboost>
- ▶ Chen, T., & Guestrin, C. (2016, August). Xgboost: A scalable tree boosting system. In Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining (pp. 785-794).
- ▶ Friedman, J., Hastie, T., & Tibshirani, R. (2001). The elements of statistical learning (Vol. 1, No. 10). New York: Springer series in statistics.
- ▶ Goodfellow, I., Bengio, Y., Courville, A., & Bengio, Y. (2016). Deep learning. Cambridge: MIT press.

XGBoost: Demo

```
#Load the required packages
```

```
library("McSpatial") #loads the package
```

```
library("dplyr") #for data wrangling
```

```
library("caret")
```

```
data(matchdata) #loads the data set
```

```
set.seed(123) #set the seed for replication purposes
```

```
#linear regression
```

```
model_lm<-train(lnprice~.+.:latitude+.:longitude+.:latitude:longitude,  
                data = matchdata,  
                trControl = trainControl(method = "cv", number = 5),  
                method = "lm")
```

XGBoost: Demo

```
model_lm
```

```
## Linear Regression
##
## 3204 samples
## 17 predictor
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 2562, 2564, 2561, 2564, 2565
## Resampling results:
##
## RMSE      Rsquared   MAE
## 0.2058519  0.8472526  0.1442374
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

XGBoost: Demo

- ▶ From Caret's manual
- ▶ eXtreme Gradient Boosting

- ▶ method = 'xgbTree' (L n)
- ▶ Type: Regression, Classification

- ▶ Tuning parameters:

nrounds (# Boosting Iterations) ^M

max_depth (Max Tree Depth) = 1, = 2, = 3

eta (Shrinkage) = 0.001

gamma (Minimum Loss Reduction)

colsample_bytree (Subsample Ratio of Columns) 0.20

* min_child_weight (Minimum Sum of Instance Weight)

subsample (Subsample Percentage) 0.20

booster

- ▶ Required packages: xgboost, plyr
- ▶ A model-specific variable importance metric is available.

XGBoost: Demo

```
tune_grid <- expand.grid(  
M nrounds = seq(from = 200, to = 1000, by = 50),  
  eta = c(0.025, 0.05, 0.1, 0.3),  
  max_depth = c(2, 3, 4, 5, 6),  
  gamma = 0,  
  colsample_bytree = 1,  
  min_child_weight = 1,  
  subsample = 1  
)  
  
tune_control <- caret::trainControl(  
  method = "cv", # cross-validation  
  number = 3, # with n folds  
  #index = createFolds(tr_treated$Id_clean), # fix the folds  
  verboseIter = FALSE, # no training log  
  allowParallel = TRUE # FALSE for reproducible results  
)
```

XGBoost: Demo

```
input_x <- select(matchdata, -lnprice)
input_x$year <- as.factor(input_x$year)
input_x$careia <- as.factor(input_x$careia)
require("Matrix")
```

```
input_x <- sparse.model.matrix(~., data=input_x)
input_y <- matchdata$lnprice
```

```
model_xgb <- caret::train(
  x = input_x,
  y = input_y,
  trControl = tune_control,
  tuneGrid = tune_grid,
  method = "xgbTree",
  verbose = TRUE
)
```

```
model_xgb$bestTune
```

```
##      nrounds max_depth  eta gamma colsample_bytree min_child_weight subsample
## 93      550         2 0.05    0              1              1              1
```

XGBoost: Demo

