Gradient Boosting
XGBoost

Prof. Dr. Stephan Trahasch

Based on slides from Tianqi Chen

Outline

- Introduction
- What does XGBoost learn?
- What can XGBoost System do for you?
- Summary

Machine Learning Algorithms and Common Use-cases

- Linear Models for Ads Clickthrough
- Factorization Models for Recommendation
- Deep Neural Nets for Images, Audios etc.

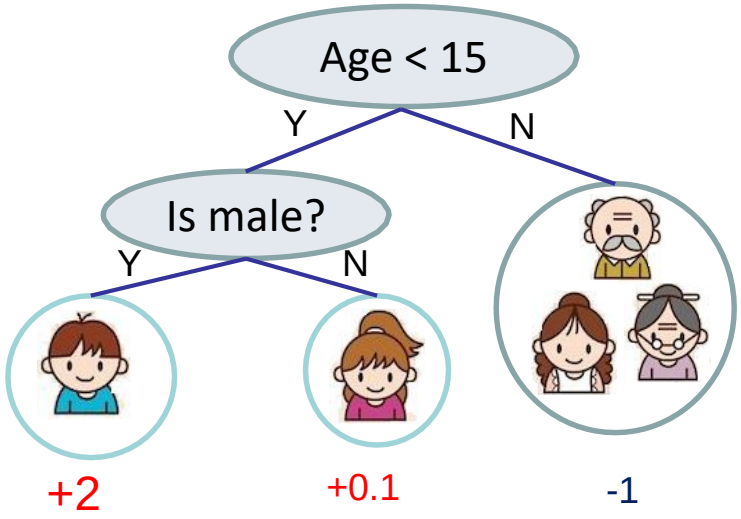
- Trees for tabular data: the secret sauce in machine learning
  - Anomaly detection
  - Ads clickthrough
  - Fraud detection
  - Insurance risk estimation

Regression Tree

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

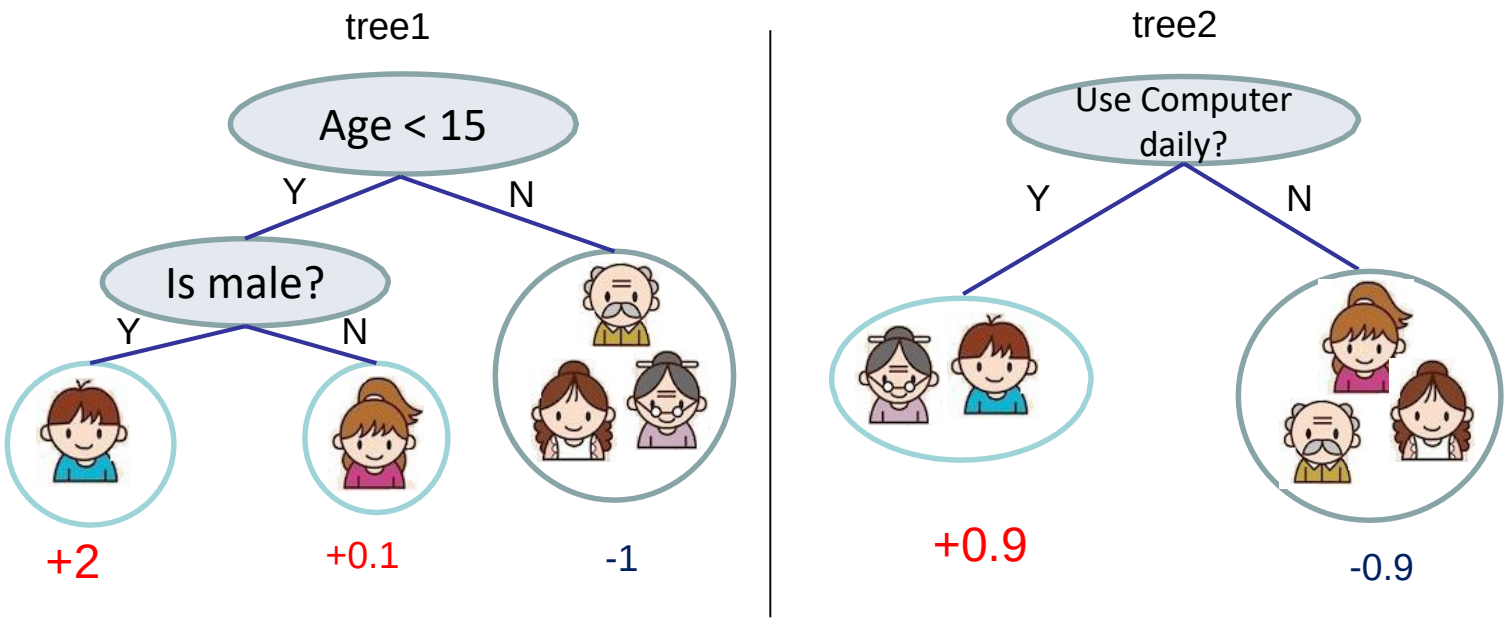
Input: age, gender, occupation, …

Does the person like computer games



prediction score in each leaf

When Trees forms a Forest (Tree Ensembles)



f( ) = 2 + 0.9= 2.9 f( )= -1 - 0.9= -1.9

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**

Base Learners

- XGBoost involves creating a meta-model that is composed of many individual model that combine to give a final prediction
- Individual models = base learners
- Base learners -> non-linear
- Each base learner should be good at distinguishing or predicting different parts of the dataset
- Tree Base Learner
  - Decision Tree
  - Boosted model is weighted sum of decion trees

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

XGBoost
Machine Learning Challenge Winning Solutions

- Maksims Volkovs, Guangwei Yu and Tomi Poutanen, 1st place of the 2017 ACM RecSys challenge. Link to paper.
- Vlad Sandulescu, Mihai Chiru, 1st place of the KDD Cup 2016 competition. Link to the arxiv paper.
- Marios Michailidis, Mathias Müller and HJ van Veen, 1st place of the Dato Truely Native? competition. Link to the Kaggle interview.
- Vlad Mironov, Alexander Guschin, 1st place of the CERN LHCb experiment Flavour of Physics competition. Link to the Kaggle interview.
- Josef Slavicek, 3rd place of the CERN LHCb experiment Flavour of Physics competition. Link to the Kaggle interview.
- Mario Filho, Josef Feigl, Lucas, Gilberto, 1st place of the Caterpillar Tube Pricing competition. Link to the Kaggle interview.
- Qingchen Wang, 1st place of the Liberty Mutual Property Inspection. Link to the Kaggle interview.
- Chenglong Chen, 1st place of the Crowdflower Search Results Relevance. Link to the winning solution.
- Alexandre Barachant (“Cat”) and Rafał Cycoń (“Dog”), 1st place of the Grasp-and-Lift EEG Detection. Link to the Kaggle interview.
- Halla Yang, 2nd place of the Recruit Coupon Purchase Prediction Challenge. Link to the Kaggle interview.
- Owen Zhang, 1st place of the Avito Context Ad Clicks competition. Link to the Kaggle interview.
- Keiichi Kuroyanagi, 2nd place of the Airbnb New User Bookings. Link to the Kaggle interview.
- Marios Michailidis, Mathias Müller and Ning Situ, 1st place Homesite Quote Conversion. Link to the Kaggle interview.

Machine Learning Challenge Winning Solutions

The most frequently used tool by data science competition winners

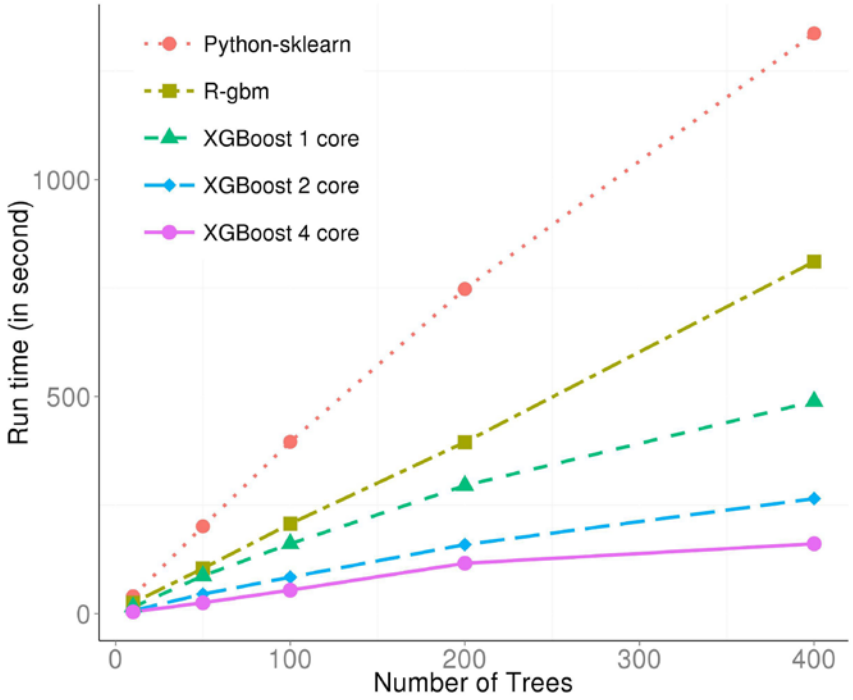
- *17 out of 29 winning solutions* in kaggle in 2015 used XGBoost
- Solve wide range of problems: store sales prediction; high energy physics event classification; web text classification; customer behavior prediction; motion detection; ad click through rate prediction; malware classification; product categorization; hazard risk prediction; massive online course dropout rate prediction

- Present and Future of KDDCup. Ron Bekkerman (KDDCup 2015 chair): *"Something dramatic happened in Machine Learning over the past couple of years. It is called XGBoost – a package implementing Gradient Boosted Decision Trees that works wonders in data classification. Apparently, every winning team used XGBoost, mostly in ensembles with other classifiers. Most surprisingly, the winning teams report very minor improvements that ensembles bring over a single well- configured XGBoost.."*

- A lot contributions from the kaggle community

XGBoost is 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.

Outline

- Introduction
- What does XGBoost learn?
- What can XGBoost System do for you?
- Summary

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

ML 101: Elements of Supervised Learning

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

- **Objective Function:** $\text{Obj}(\Theta) = L(\Theta) + \Omega(\Theta)$

Training Loss measures how well model fit on training data

Regularization, measures complexity of model

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

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

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

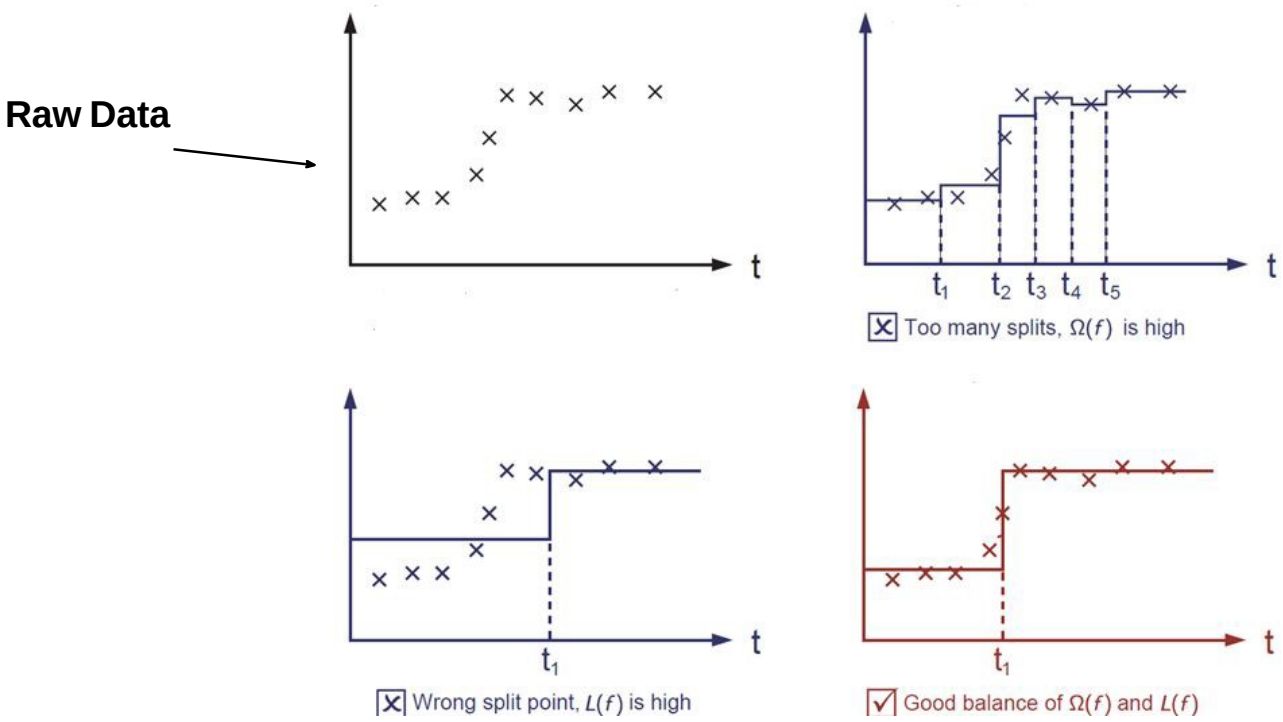
Training Loss measures how well model fit on training data

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

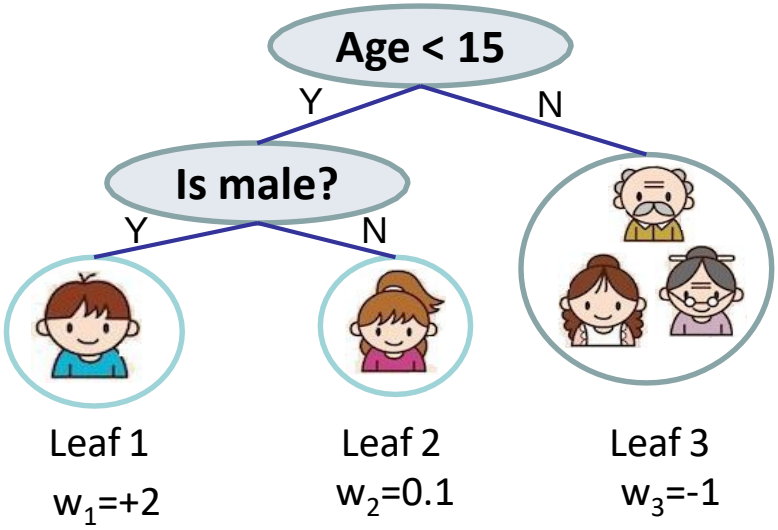


Define Complexity of a Tree

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

The leaf weight of the tree

The structure of the tree



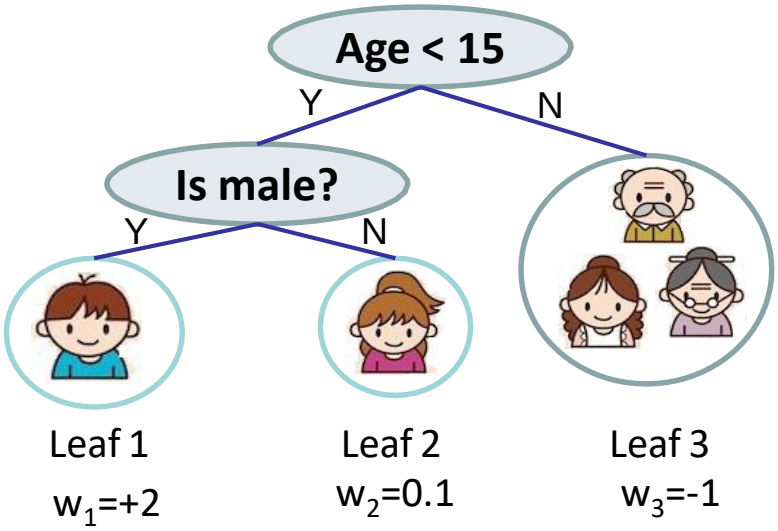
q( ) = 1

q( ) = 3

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**



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

Common Loss Functions in XGBoost

- Reg:linear
  use for regression problems
- Re:logistix
  use for classification problems when you want just decision, not probabilitiy
- Binary:logistic
  when you want probability rather than just decision

Regularization in XGBoost

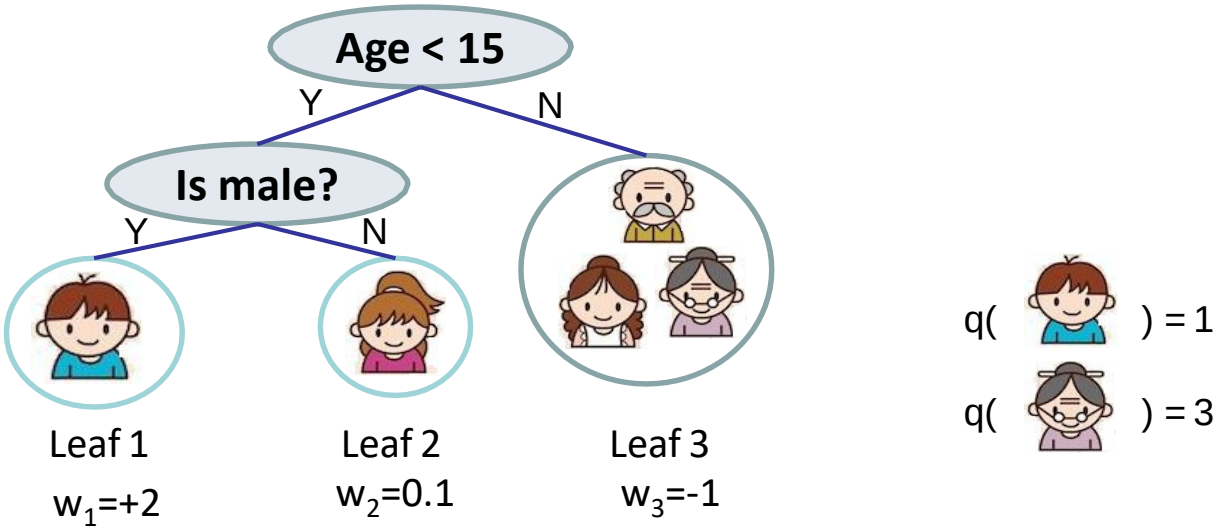
- Regularization is a control on model complexity
- Want models that are both accurate and as simple as possible
- Regularization parameters in XGBoost
  - Gamma: minimum loss reduction allowed for a split occur
  - Alpha: l1 regularization on leaf weights, larger values mean more regularization
  - Lambda: l2 regularization on leaf weights

Learning a Tree Ensemble in Three Slides

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

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

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



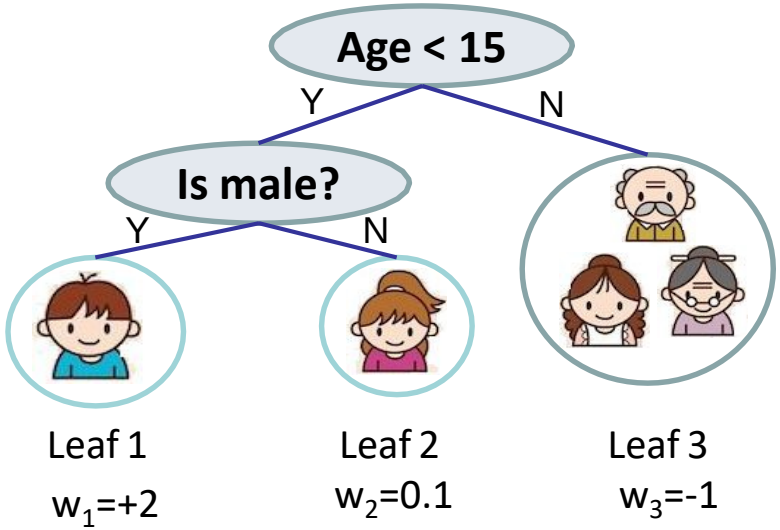
Learning a Tree Ensemble in Three Slides

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

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

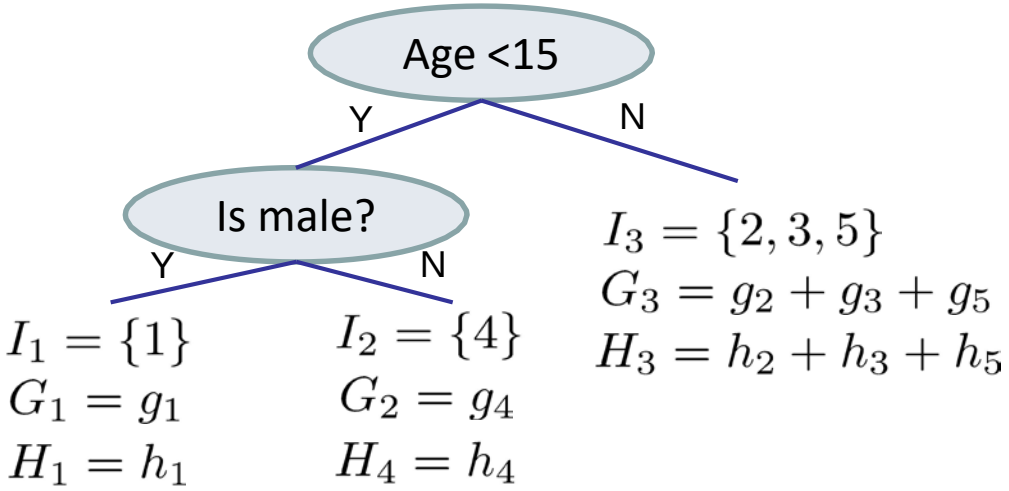


$\Omega = \gamma 3 + \frac{1}{2}\lambda(4+0.01+1)$

Learning a Tree Ensemble in Three Slides

Score for a new tree

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



$I_1 = \{1\}$
$G_1 = g_1$
$H_1 = h_1$

$I_2 = \{4\}$
$G_2 = g_4$
$H_4 = h_4$

$I_3 = \{2, 3, 5\}$
$G_3 = g_2 + g_3 + g_5$
$H_3 = h_2 + h_3 + h_5$

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

The smaller the score is, the better the structure is.

Gradient Statistics

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

How can we learn tree ensembles?

- Objective: $\text{Obj} = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_k \Omega(f_k) \quad , f_k \in \mathcal{F}$
- 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

$$\text{Obj}^{(t)} = \sum_{i=1}^{n} l\left(y_i, \hat{y}_i^{(t)}\right) + \sum_{i=1}^{t} \Omega(f_i)$$

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

Goal: find f_t to minimize this.

Taylor Expansion Approximation of Loss

Goal $\text{Obj}^{(t)} \cong \sum_{i=1}^{n} \left[l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) \right] + \Omega(f_t) + \text{constant}$

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

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

In terms of square loss $g_i = \partial_{\hat{y}^{(t-1)}} (\hat{y}^{(t-1)} - y_i)^2 = 2(\hat{y}^{(t-1)} - y_i)$

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

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 $I_j=\{i|q(x_i)=j\}$

Regroup the objective by leaf

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

This is sum of T independent quadratic 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 $G_j = \sum_{i \in I_j} g_i \quad H_j = \sum_{i \in I_j} h_i$

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

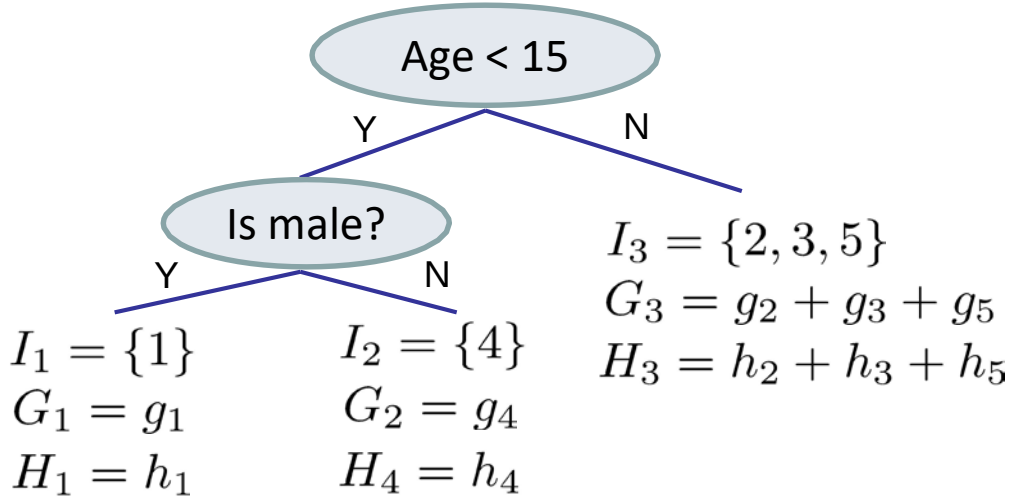
Assume the structure of tree (q(x)) is fixed, the optimal weight in each leaf, and the resulting objective value are

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

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

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

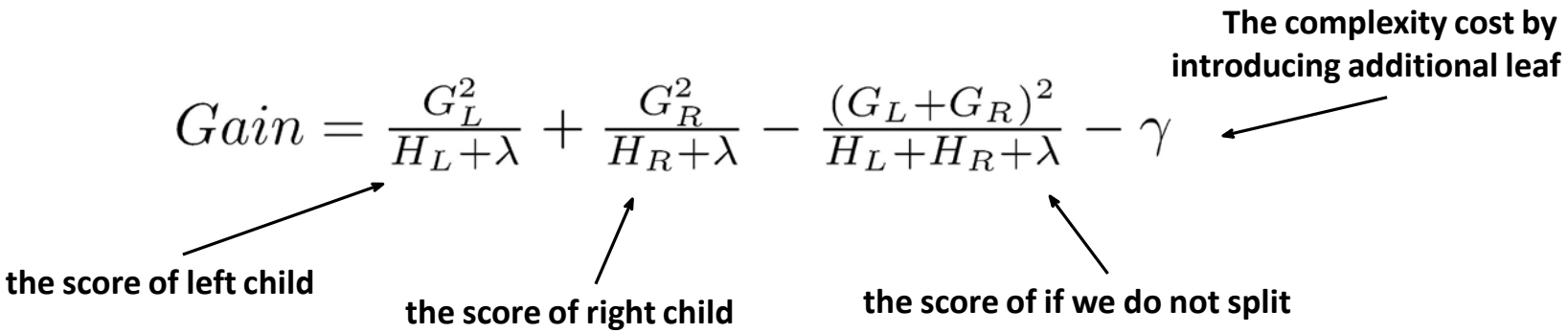
But… there can be infinite possible tree structures…

Greedy Learning of the Tree

In practice, we grow the tree greedily.

Start from tree with depth 0.

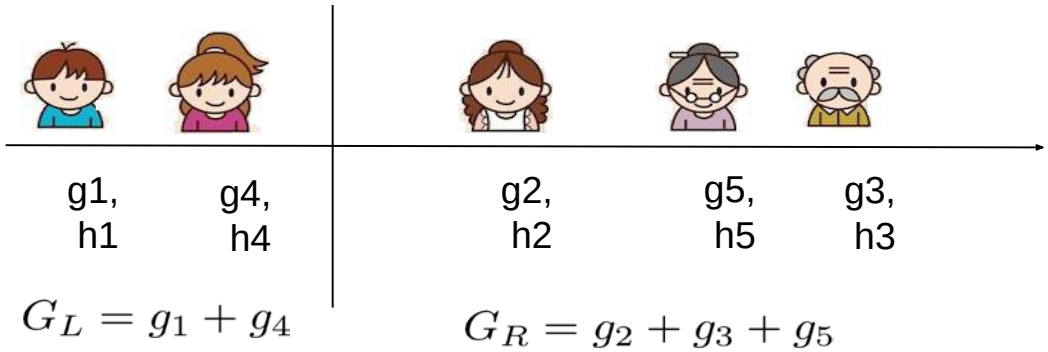
For each leaf node of the tree, try to add a split. The change of objective after adding the split is



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.

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

Outline

- Introduction
- What does XGBoost learn?
- What can XGBoost System do for you?
- Summary

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

Getting Started (R)

```
# load data
data(agaricus.train,
package='xgboost')
data(agaricus.test,
package='xgboost')  train <-
agaricus.train
test <- agaricus.test


# fit model
bst <- xgboost(data = train$data, label = train$label, max.depth = 2, eta
            = 1, nround = 2,  nthread = 2, objective = "binary:logistic")
# predict
pred <- predict(bst, test$data)
```

Common tree hyperparameters

- Learning rate: learning rate/eta
- Gamma: min loss reduction to create new tree split
- Lambda: L2 reg on leaf weights
- Alpha: L1 reg on leaf weights
- Max_depth: max depth per tree
- Subsample: % samples used per tree
- Colsample_bytree: % features used per tree
- …

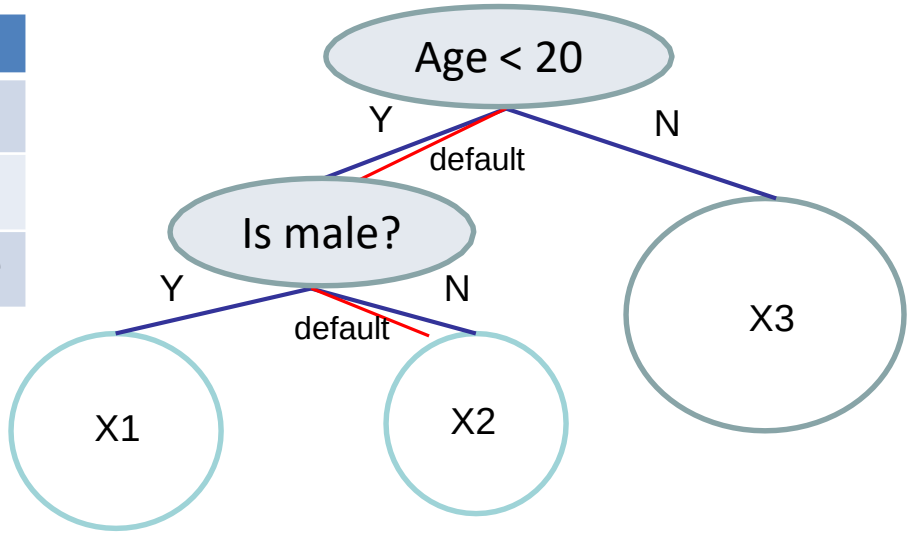
https://xgboost.readthedocs.io/en/latest/parameter.html

Automatic Missing Value Handling

XGBoost learns the best direction for missing values

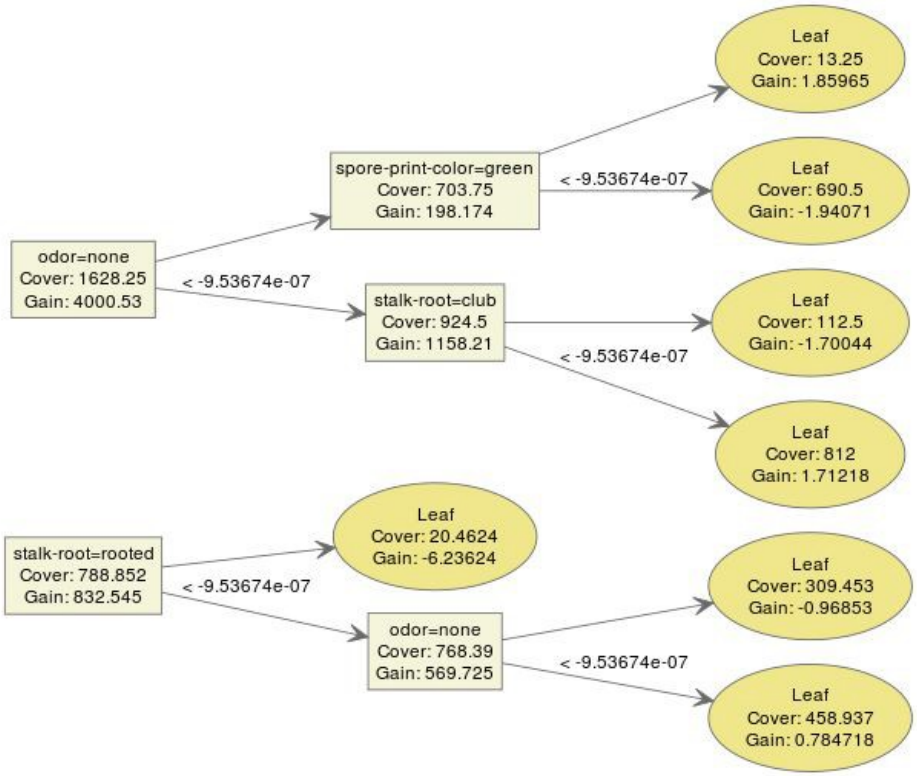
Data

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



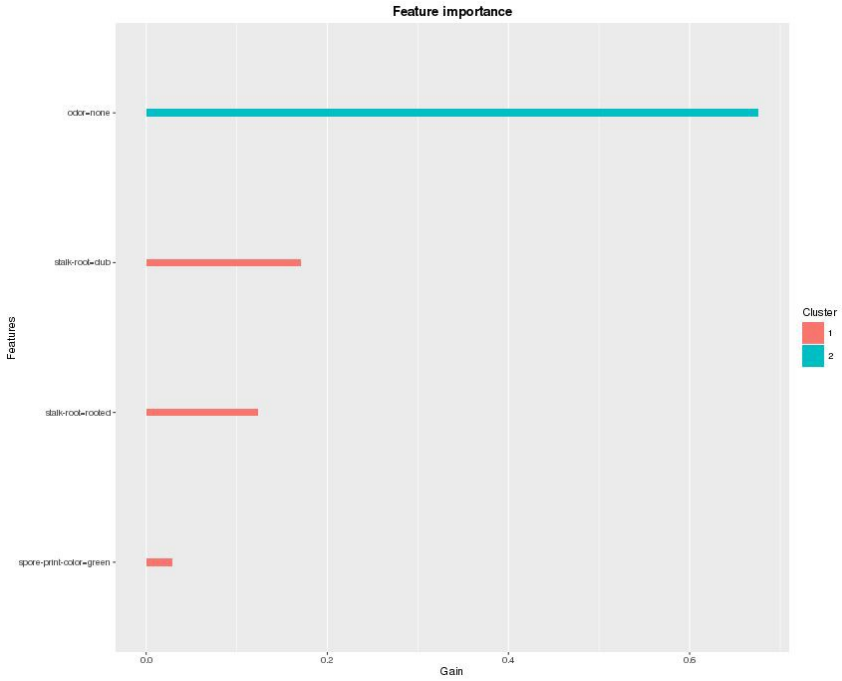
Inspect Your Models

```
bst <- xgboost(data = train$data, label = train$label, max.depth = 2,
       eta = 1, nthread = 2, nround = 2, objective = "binary:logistic")
xgb.plot.tree(feature_names = agaricus.train$data@Dimnames[[2]],
       model = bst)
```



Feature Importance Analysis

```
bst <- xgboost(data = train$data, label = train$label, max.depth = 2,
        eta = 1, nthread = 2, nround = 2,
        objective = "binary:logistic")
importance_matrix <- xgb.importance(agaricus.train$data@Dimnames[[2]],
        model = bst)
xgb.plot.importance(importance_matrix)
```

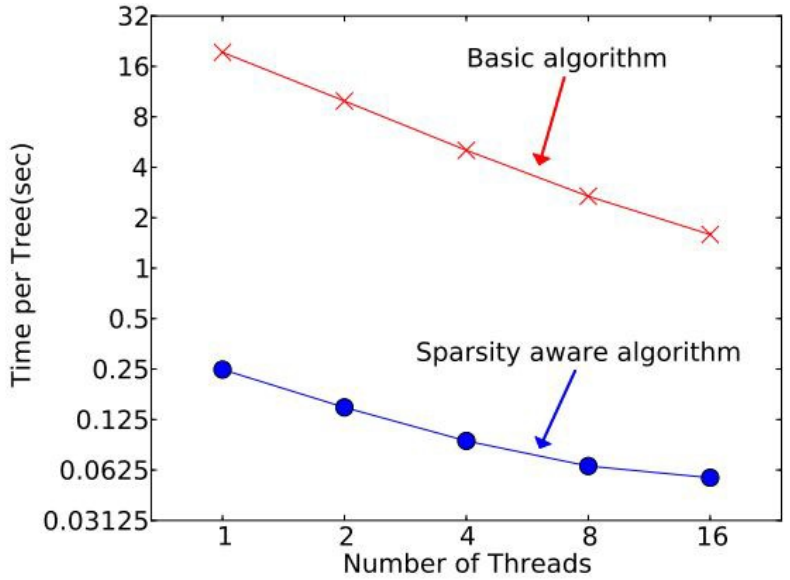


Automatic Sparse Data Optimization

Useful for categorical encoding and other cases (e.g. Bag of words)

User do not need to worry about large sparse matrices

Impact of sparse aware vs basic algorithm on allstate dataset



Extendibility: Customized Objective Function

XGBoost solves wide range of objectives

- Binary classification
- Ranking
- Multi-class classification

Customize objective function

```
loglossobj <- function(preds, dtrain) {
 # dtrain is the internal format of the training data
 # We extract the labels from the training data
 labels <- getinfo(dtrain, "label")
 # We compute the 1st and 2nd gradient, as grad and hess
 preds <- 1/(1 + exp(-preds))
 grad <- preds - labels
 hess <- preds * (1 - preds)
 # Return the result as a list
 return(list(grad = grad, hess = hess))
}

model <- xgboost(data = train$data, label = train$label, nrounds = 2,
           objective = loglossobj, eval_metric =  "error")
```

Extendibility: Modular Library

Plugin system

- Enable you to plugin customized data loader, metrics, learners
- Optionally build with some of the plugins
- https://github.com/dmlc/xgboost/tree/master/plugin

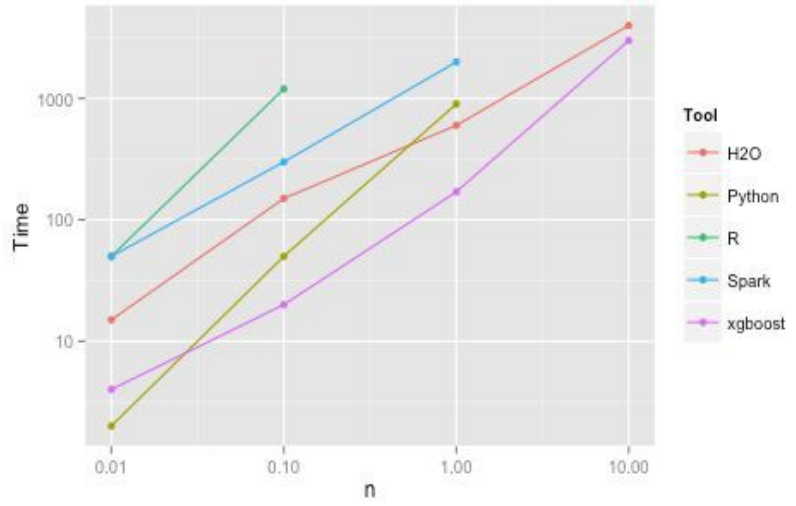
Modular library to for even more extensions

- Recent pull request of supporting DART (dropout in tree boosting)
- Reuse of all data loading and tree learning modules
- Around 300 lines of additional code

Faster Training Speed via Parallel Training

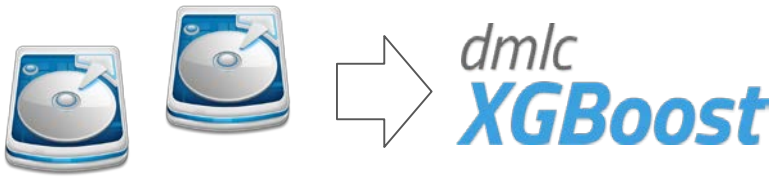
- Push limit of machine in all cases
- Low memory footprint
- Hackable native codes
  - Instead of everything in backend
  - Early-stopping
  - Checkpointing
  - Customizable objective

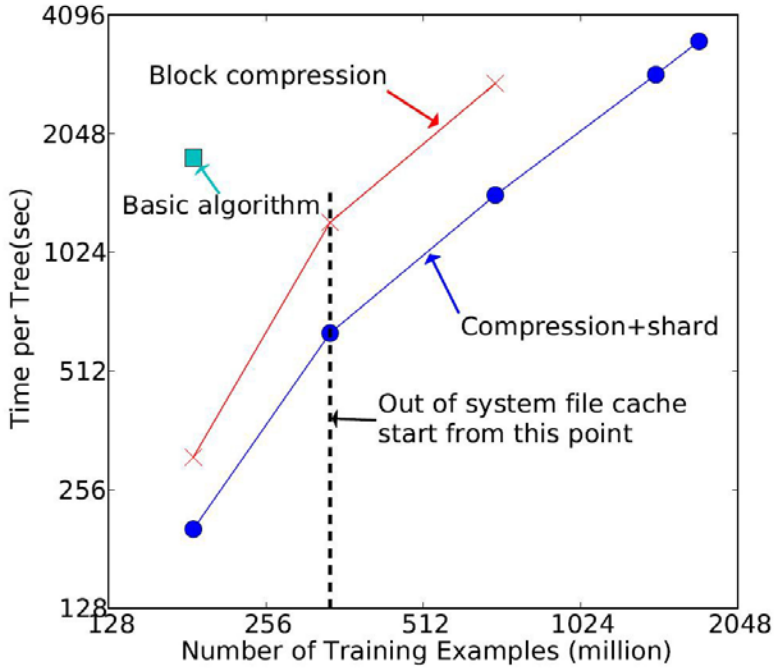
Minimum benchmark from szilard/benchm-ml



XGBoost with Out of Core Computation

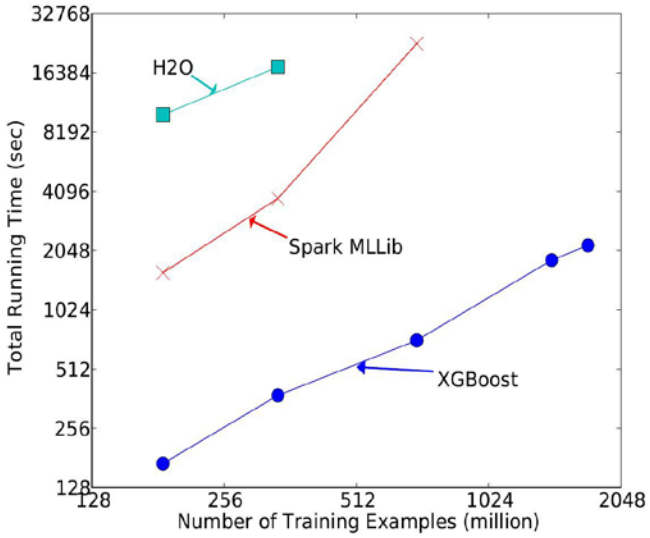
- Impact of out of core optimizations
- On a single EC2 machine with two SSD



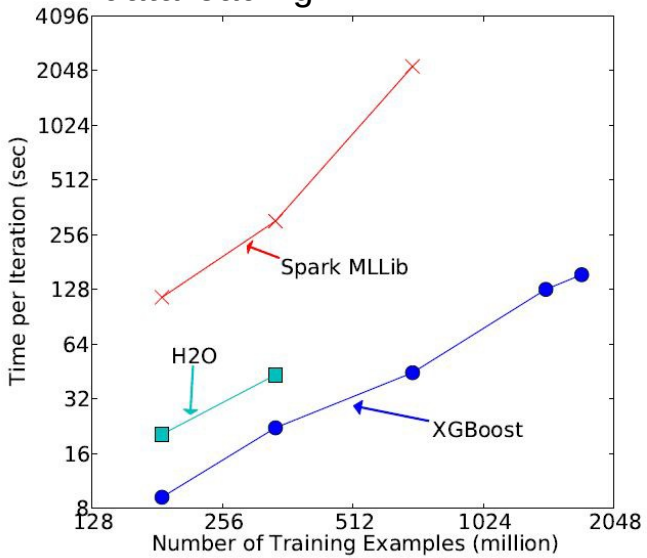


Distributed XGBoost vs Other Solutions

End to end cost include data loading



Per iteration cost exclude data loading



- 16 AWS m3.2xlarge machines
- Missing data points are due to out of memory

What can XGBoost cannot do for you

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

Unix Philosophy in Machine Learning

- XGBoost focuses on one thing and does its best for you.
- In order to be useful for the users, XGBoost also have to be open and integrate well with other systems by common interface.
- Always be modular, extendible, so we can keep up to the state of art easily.

Outline

- Introduction
- What does XGBoost learn?
- What can XGBoost System do for you?
- Summary

DMLC: Distributed Machine Learning Common

DMLC is a group to collaborate on open-source machine learning projects, with a goal of making cutting-edge large-scale machine learning widely available. The contributors includes researchers, PhD students and data scientists who are actively working on the field.

dmlc XGBoost for effective tree boosting

dmlc mxnet for deep learning

Support of other core system components for largescale ML

Contribute to XGBoost and Other DMLC Projects

- Contribute code to improve the package
- Create tutorials on how the usecases
- Share your experience
- Awesome XGBoost
  https://github.com/dmlc/xgboost/tree/master/demo

Acknowledgement

- XGBoost Committers
  Tong He, Bing Xu, Michael Benesty, Yuan Tang, Scott Lundberg

- Contributors of XGBoost

- Users in the XGBoost Community