XGBOOST: A SCALABLE TREE BOOSTING SYSTEM

(T. CHEN, C. GUESTRIN, 2016)

NATALLIE BAIKEVICH

HARDWARE ACCELERATION FOR DATA PROCESSING SEMINAR

ETH ZÜRICH

MOTIVATION

- ✓ **Effective** statistical models
- ✓ **Scalable** system
- ✓ **Successful** real-world applications

XGBoost

eXtreme Gradient Boosting

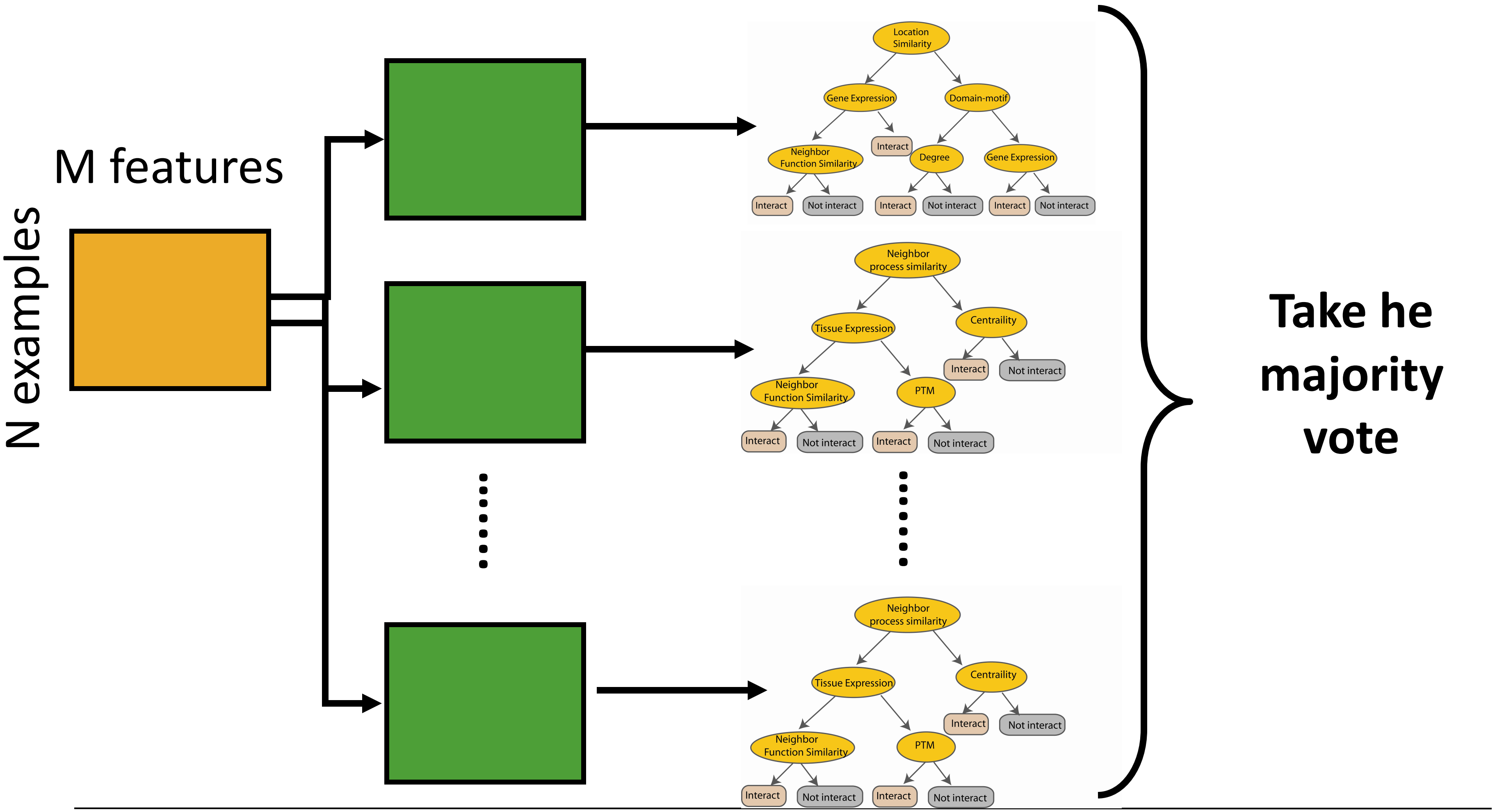
Model Averaging

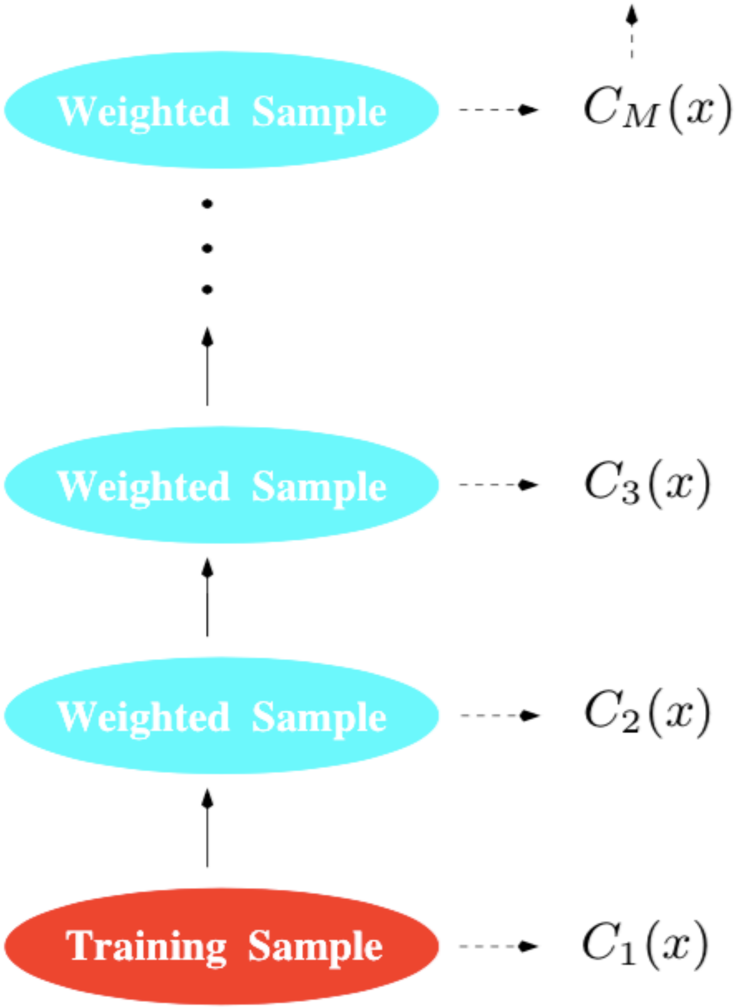
Classification trees can be simple, but often produce noisy (bushy) or weak (stunted) classifiers.

- Bagging (Breiman, 1996): Fit many large trees to bootstrap-resampled versions of the training data, and classify by majority vote.
- Boosting (Freund & Shapire, 1996): Fit many large or small trees to reweighted versions of the training data. Classify by weighted majority vote.
- Random Forests (Breiman 1999): Fancier version of bagging.

In general Boosting $\succ$ Random Forests $\succ$ Bagging $\succ$ Single Tree.

Random Forest Classifier





Boosting

- Average many trees, each grown to re-weighted versions of the training data.
- Final Classifier is weighted average of classifiers:

$$C(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m C_m(x)\right]$$

AdaBoost (Freund & Schapire, 1996)

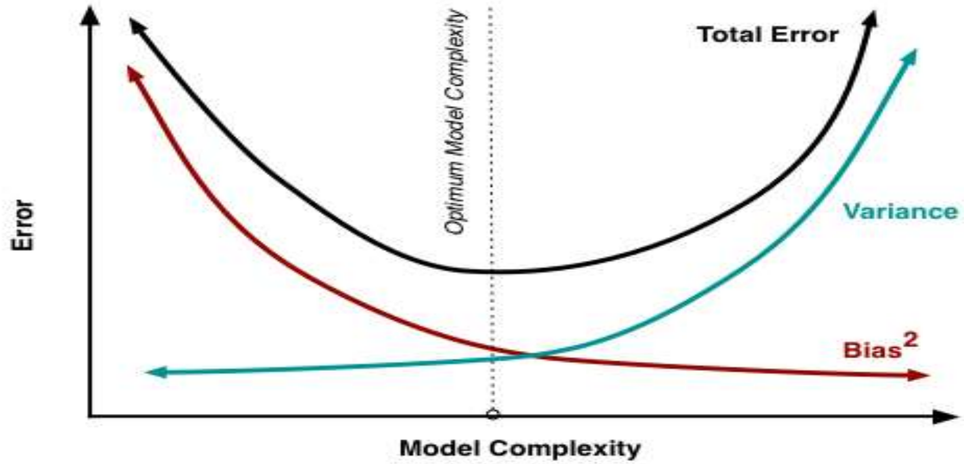
1. Initialize the observation weights $w_i = 1/N,\ i = 1, 2, \ldots, N$.
2. For $m = 1$ to M repeat steps (a)–(d):
   (a) Fit a classifier $C_m(x)$ to the training data using weights w_i.
   (b) Compute weighted error of newest tree

$$\mathrm{err}_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq C_m(x_i))}{\sum_{i=1}^{N} w_i}.$$

The smaller the error of a tree, the higher the weight for this tree

   (c) Compute $\alpha_m = \log[(1 - \mathrm{err}_m)/\mathrm{err}_m]$.
   (d) Update weights for $i = 1, \ldots, N$:
   $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq C_m(x_i))]$
   and renormalize to w_i to sum to 1.
3. Output $C(x) = \mathrm{sign}\left[\sum_{m=1}^{M} \alpha_m C_m(x)\right]$.

BIAS-VARIANCE TRADEOFF



Random Forest

Variance ↓

Boosting

Bias ↓

+

+

Voting

A BIT OF HISTORY

AdaBoost, 1996

Random Forests, 1999

Gradient Boosting Machine, 2001

A BIT OF HISTORY

AdaBoost, 1996

Random Forests, 1999

Gradient Boosting Machine, 2001

Various improvements in tree boosting

XGBoost package

A BIT OF HISTORY

AdaBoost, 1996

Random Forests, 1999

Gradient Boosting Machine, 2001

Various improvements in tree boosting

XGBoost package

1st Kaggle success: Higgs Boson Challenge

17/29 winning solutions in 2015

WHY DOES XGBOOST WIN "EVERY" MACHINE LEARNING COMPETITION?

- (MASTER THESIS, D. NIELSEN, 2016)

- 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.

Source: https://github.com/dmlc/xgboost/tree/master/demo#machine-learning-challenge-winning-solutions

Regression Trees

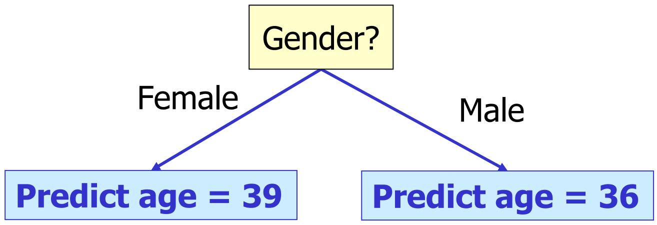
- “Decision trees for regression”

A regression tree leaf

Predict age = 47

Mean age of records matching this leaf node

A one-split regression tree



Choosing the attribute to split on

Gender	Rich?	Num. Children	Num. Beany Babies	Age
Female	No	2	1	38
Male	No	0	0	24
Male	Yes	0	5+	72
:	:	:	:	:

- We can't use information gain.
- What should we use?

Choosing the attribute to split on

Gender	Rich?	Num. Children	Num. Beany Babies	Age
Female	No	2	1	38
Male	No	0	0	24
Male	Yes	0	5+	72
:	:	:	:	:

MSE(Y|X) = The expected squared error if we must predict a record's Y value given only knowledge of the record's X value

If we're told $x=j$, the smallest expected error comes from predicting the mean of the Y-values among those records in which $x=j$. Call this mean quantity $\mu_y^{x=j}$

Then…

$$MSE(Y \mid X) = \frac{1}{R}\sum_{j=1}^{N_X} \sum_{(k \text{ such that } x_k = j)} (y_k - \mu_y^{x=j})^2$$

Choosing the attribute to split on

Gender	Rich?	Num.	Num. Beany	Age
Female	N			
Male	N			
Male	Y			
:	:			

Regression tree attribute selection: greedily choose the attribute that minimizes MSE(Y|X)

Guess what we do about real-valued inputs?

Guess how we prevent overfitting

MSE(Y|X) = Th…

value given only knowledge of the record's X value

If we're told $x=j$, the smallest expected error comes from predicting the mean of the Y-values among those records in which $x=j$. Call this mean quantity $\mu_y^{x=j}$

Then…

$$MSE(Y \mid X) = \frac{1}{R}\sum_{j=1}^{N_X} \sum_{(k \text{ such that } x_k = j)} (y_k - \mu_y^{x=j})^2$$

Pruning Decision

...property-owner = Yes

Do I deserve to live?

Gender?

Female | Male

Predict age = 39

property-owning females = 56712
Mean age among POFs = 39
Age std dev among POFs = 12

Predict age = 36

property-owning males = 55800
Mean age among POMs = 36
Age std dev among POMs = 11.5

Use a standard Chi-squared test of the null-hypothesis "these two populations have the same mean" and Bob's your uncle.

Linear Regression Trees

Also known as "Model Trees"

...property-owner = Yes

Gender?

Female | Male

Predict age =

26 + 6 * NumChildren - 2 * YearsEducation

Predict age =

24 + 7 * NumChildren - 2.5 * YearsEducation

Leaves contain linear functions (trained using linear regression on all records matching that leaf)

Split attribute chosen to minimize MSE of regressed children.

Pruning with a different Chi-squared

Linear Regression Trees

...property-owner = Yes

Also known as "M... l Trees"

Gender?

Female

Predict age =

26 + 6 * N...
2 * Year...

Detail: You typically ignore any categorical attribute that has been tested on higher up in the tree during the regression. But use all untested attributes, and use real-valued attributes even if they've been tested above

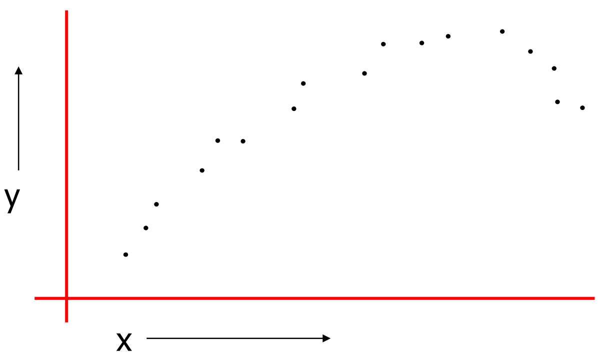
Leaves contain ... functions (traine... linear regression ... records matching t...)

... chosen to minimize ... regressed children.

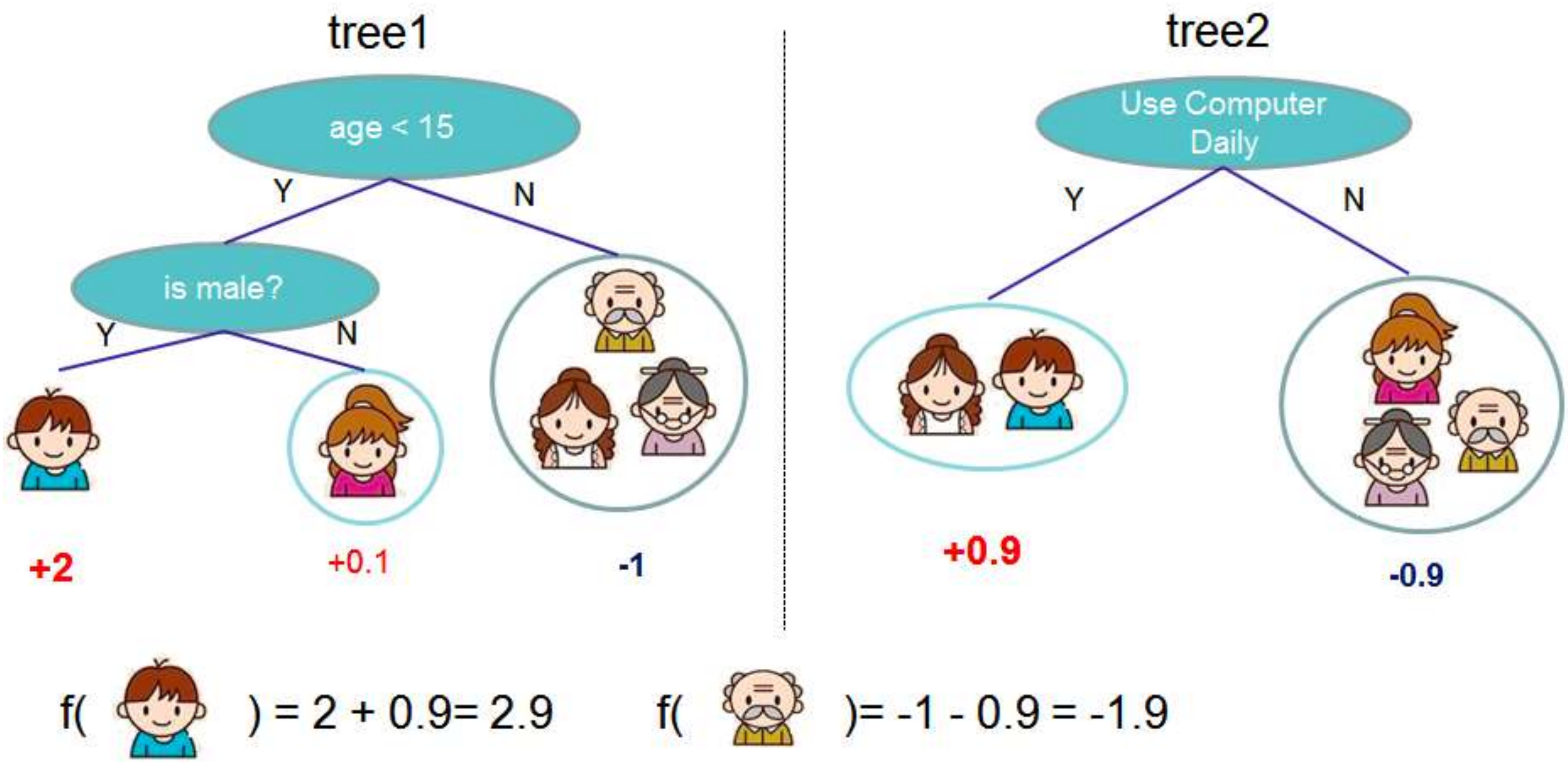
Pruning with a different Chi-squared

Test your understanding

Assuming regular regression trees, can you sketch a graph of the fitted function $y^{est}(x)$ over this diagram?



TREE ENSEMBLE

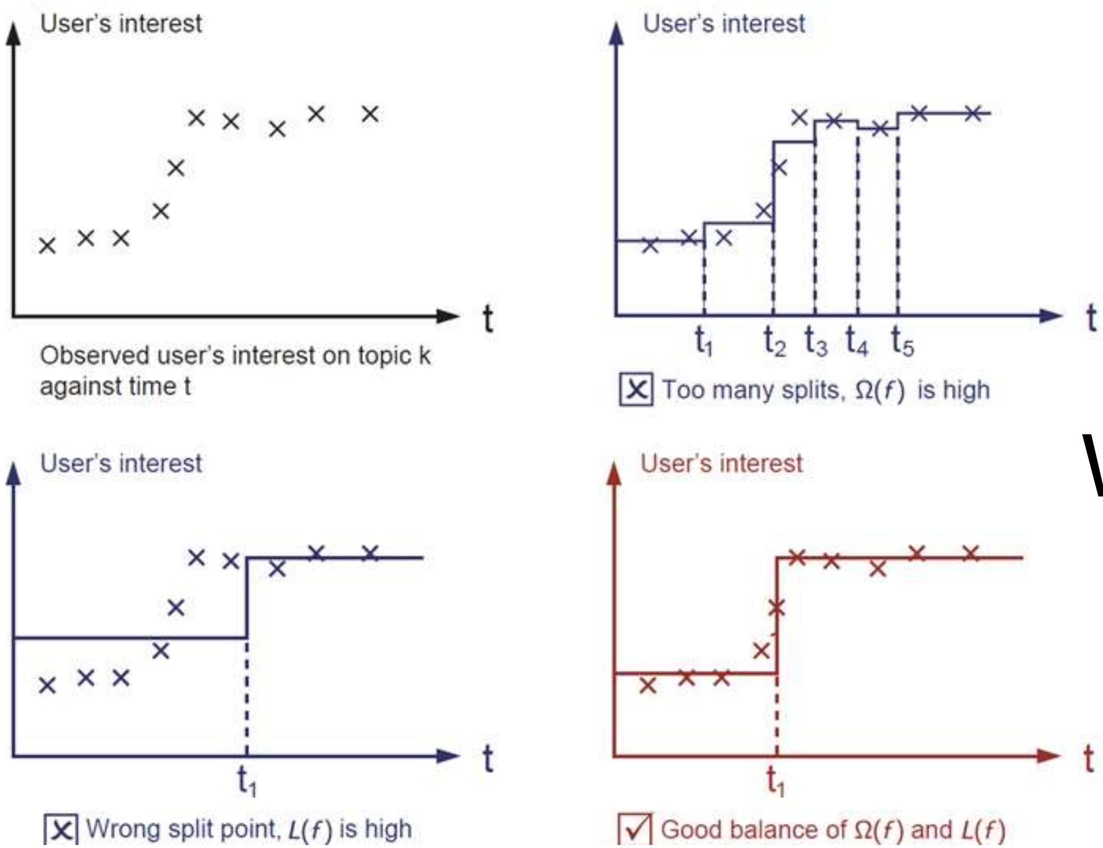


REGULARIZED LEARNING OBJECTIVE

$$L = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k)$$

loss

regularization



$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i)$$

$$\Omega(f) = \gamma T + \frac{1}{2} \lambda \|w\|^2$$

# of leaves

Source: http://xgboost.readthedocs.io/en/latest/model.html

So How do we Learn?

- 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 of Loss

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

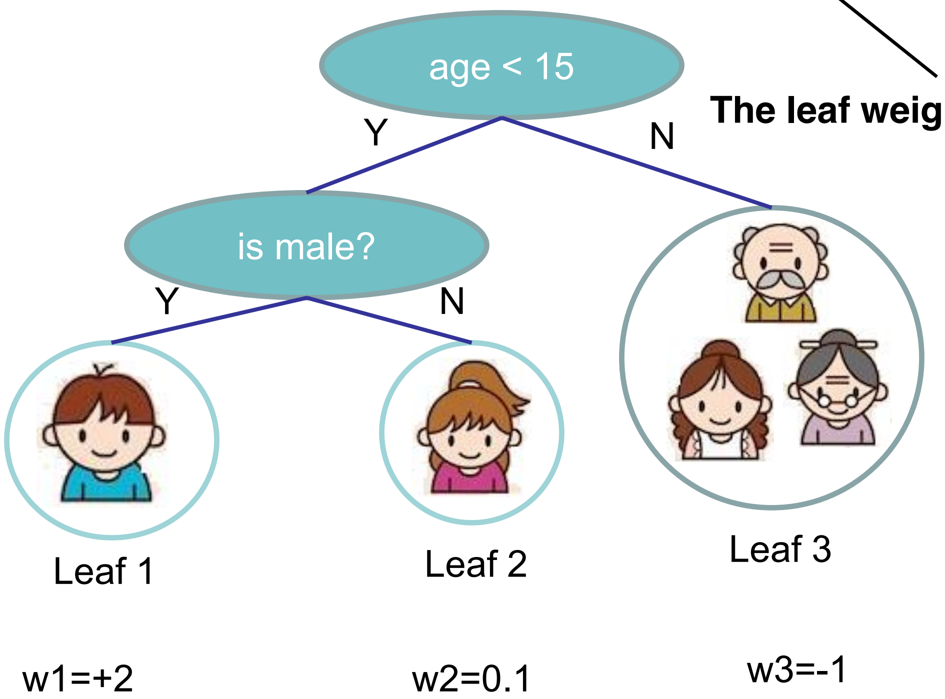
Refine the definition of tree

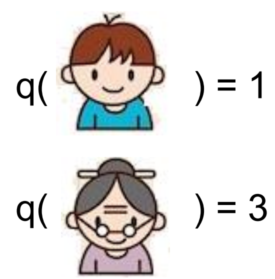
- We define tree by a vector of scores in leafs, and a leaf index mapping function that maps an instance to a leaf

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

The structure of the tree

The leaf weight of the tree



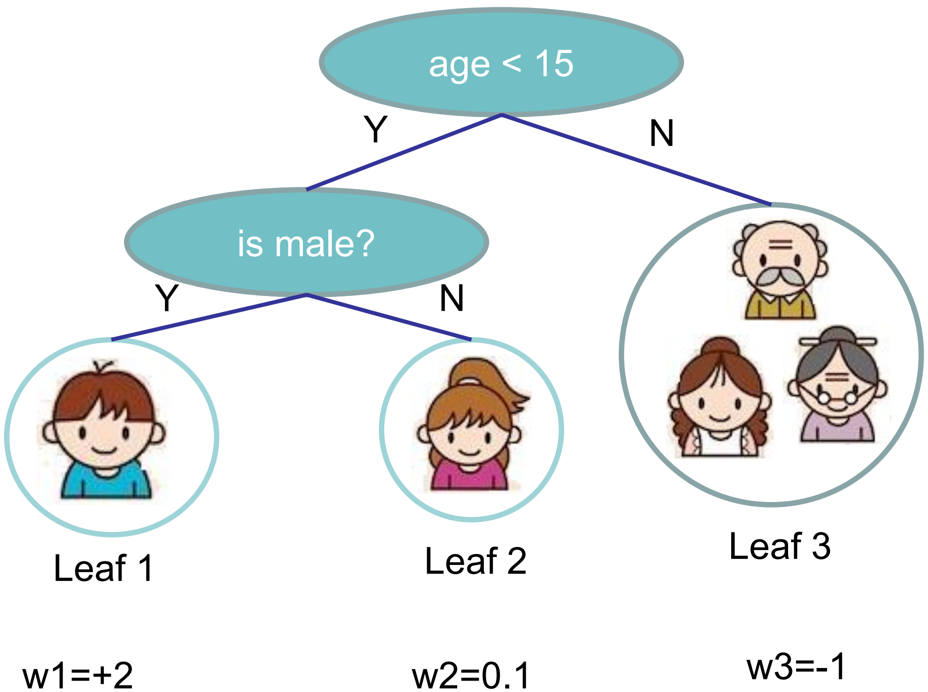


Define Complexity of a Tree (cont’)

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

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

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

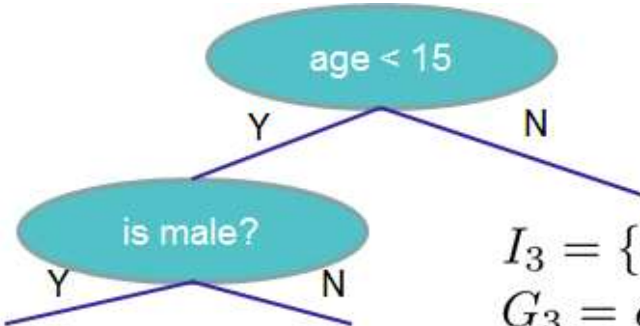
$$w_j^* = -\frac{G_j}{H_j + \lambda} \qquad Obj = -\frac{1}{2}\sum_{j=1}^{T} \frac{G_j^2}{H_j + \lambda} + \gamma T$$

This measures how good a tree structure is!

SCORE CALCULATION

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

1st order gradient **2nd order gradient**



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

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

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

Statistics for each leaf

Score

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

The smaller the score is, the better the structure is

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 $f_t(x)$

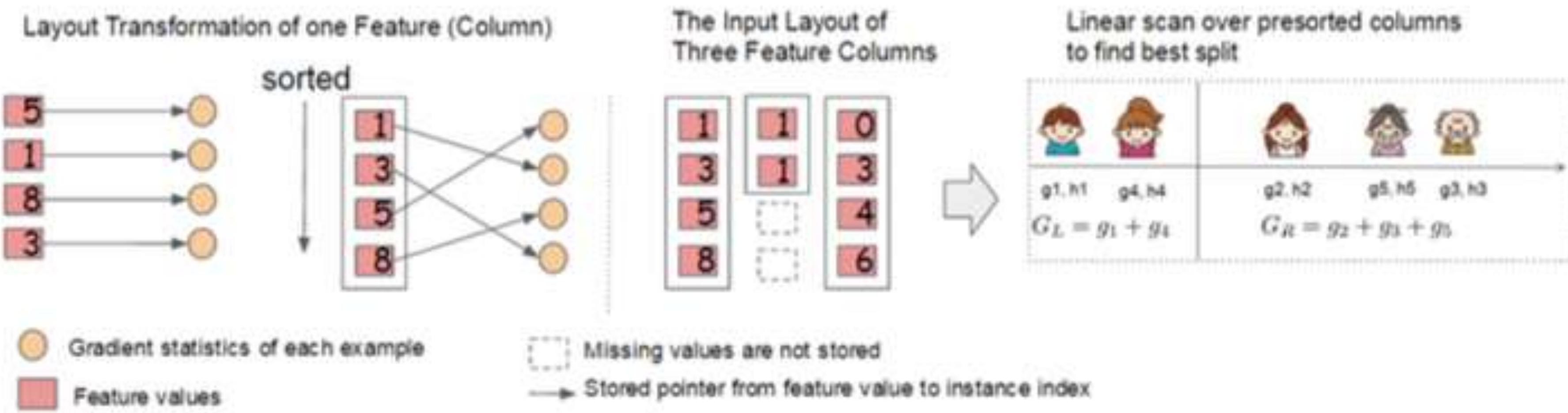
$$Obj = -\frac{1}{2} \sum_{j=1}^{T} \frac{G_j^2}{H_j + \lambda} + \gamma T$$

- 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

ALGORITHM FEATURES

- ✓ **Regularized** objective
- ✓ **Shrinkage** and column **subsampling**
- ✓ Split finding: exact & **approximate**, global & local
- ✓ **Weighted** quantile sketch
- ✓ **Sparsity**-awareness

SYSTEM DESIGN: BLOCK STRUCTURE



Max depth

$$O(Kd\|x\|_0 \log n) \rightarrow O(Kd\|x\|_0 + \|x\|_0 \log B)$$

Sorted structure –> linear scan

trees   # non-missing entries

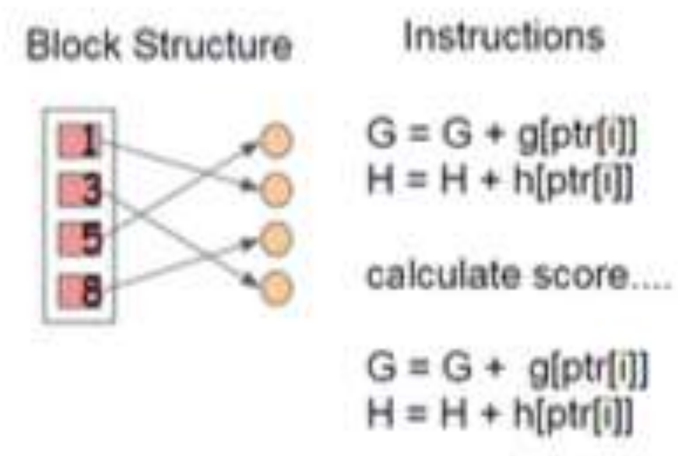
Blocks can be

- ✓ **Distributed across machines**
- ✓ **Stored on disk in out-of-core setting**

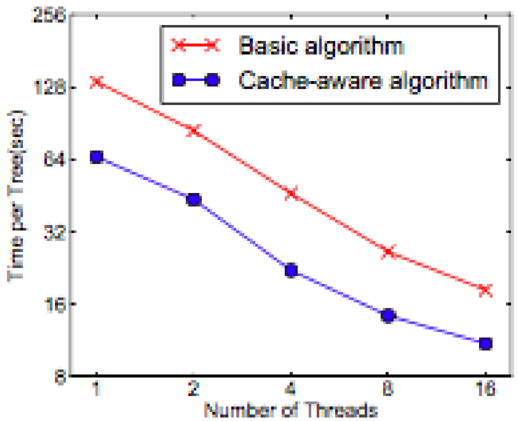
SYSTEM DESIGN: CACHE-AWARE ACCESS

Improved split finding

Non-continuous memory access

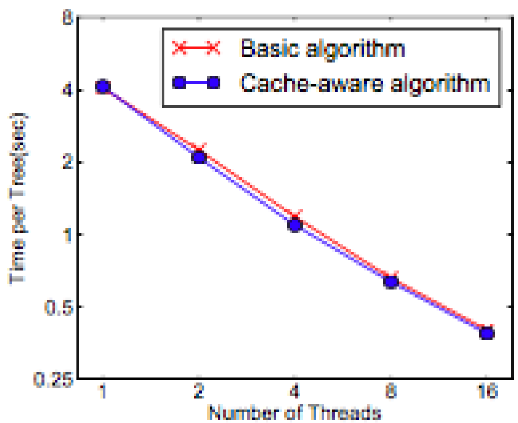


✓ Allocate internal buffer
✓ **Prefetch** gradient statistics



(b) Higgs 10M

Datasets:
Larger vs Smaller



(d) Higgs 1M

SYSTEM DESIGN: BLOCK STRUCTURE

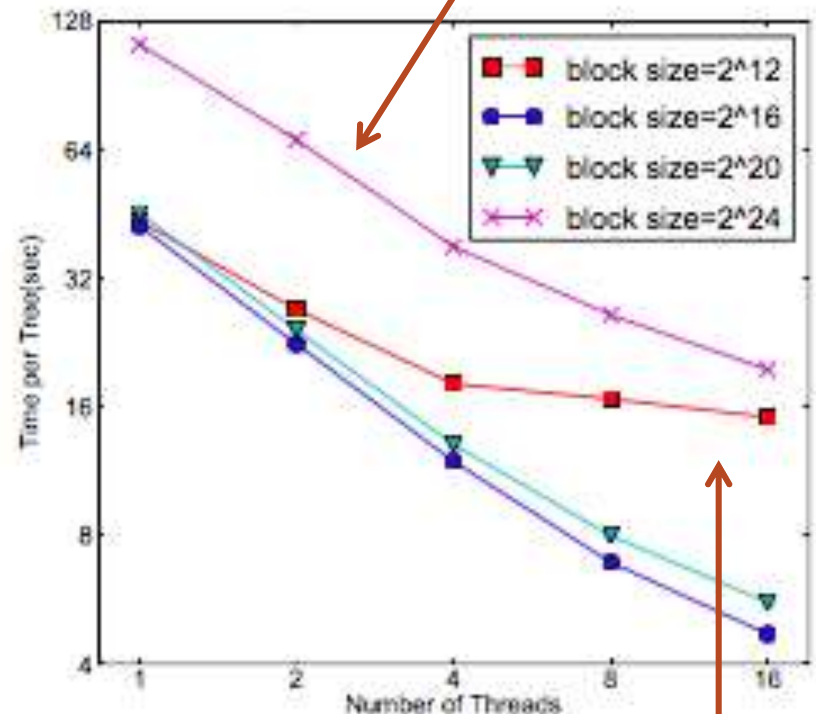
Prefetch
in independent thread

Compression by columns (CSC):

Decompression
vs
Disk Reading

Block sharding:
Use multiple disks

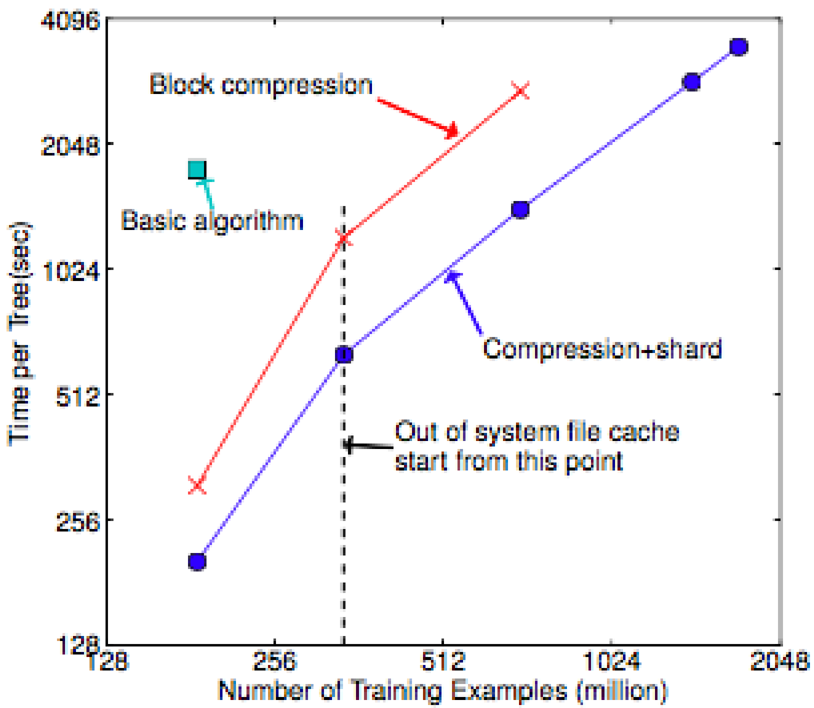
Too large blocks, cache misses

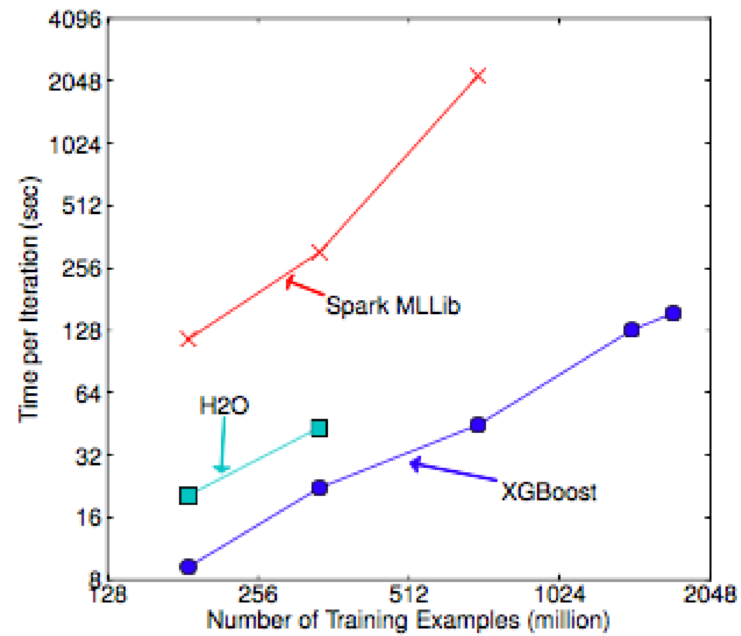


(a) Allstate 10M

Too small, inefficient parallelization

EVALUATION





(b) Per iteration cost exclude data loading

AWS c3.8xlarge machine:
32 virtual cores, 2x320GB SSD, 60 GB RAM

32 m3.2xlarge machines, each:
8 virtual cores, 2x80GB SSD, 30GB RAM

DATASETS

Dataset	n	m	Task
Allstate	10M	4227	Insurance claim classification
Higgs Boson	10M	28	Event classification
Yahoo LTRC	473K	700	Learning to rank
Criteo	1.7B	67	Click through rate prediction

WHAT'S NEXT?

XGBoost

Scalability

Weighted quantiles

Sparsity-awareness

Cache-awarereness

Data compression

→

Tuning

Hyperparameter optimization

Parallel Processing

GPU

FPGA

Model Extensions

DART (+ Dropouts)

LinXGBoost

More Applications

The RGF algorithm is a variation of GBDT in which the structure search and the optimization are decoupled. More specifically, the main differences are given as follows:

- RGF introduces an explicit regularization term that takes advantage of individual tree structures.

$$\hat{h} = argmin_{h \in H}[\ell(h(\mathbf{x}); y) + R(h)] \tag{4}$$

- RGF employs a *fully-corrective* greedy algorithm which iteratively modifies the weights of all the leaf nodes (decision rules) currently obtained while new rules are added into the forest by greedy search. Here, an explicit regularization is also included to avoid overfitting and very large models.

- RGF utilizes the concept of structured sparsity to perform greedy search directly over the forest nodes based on the forest structure.

Algorithm 2 Regularized Greedy Forest framework

$F \leftarrow \{\}$
while stopping criterion not met **do**
  Fix weights and adjust forest structure s:
  $\hat{s} \leftarrow argmin_{s \in S(F)} Q(s(F))$ (the optimum s that minimizes $Q(F)$ among all the structures that can be obtained by applying one structure-changing operation to F).
  if some criterion is met **then**
    Fix the structure and change the weights in F s.t. the loss is minimized in $Q(F)$ (it can be optimized using a standard procedure (such as coordinate descent) if the regularization penalty is standard e.g., *L2-loss*
  end if
end while
Optimize leaf weights in F to minimize loss in $Q(F)$
return $h_F(\mathbf{x})$