# History of Boosting Algorithm: Adaboost, GBM, XGBoost

Seungyeop Hyun

Seoul National University

July 28, 2022

## Contents

## Boosting Algorithm

- Boosting algorithm is proposed by Freund and Schapire(1996).
- Boosting is an ensemble method that combines multiple weak learners into a strong learner.
- It trains each weak learner sequentially, while the bagging uses independent models.
- Weak learner is one whose error rate is slightly better than random guessing.

## Boosting Algorithm

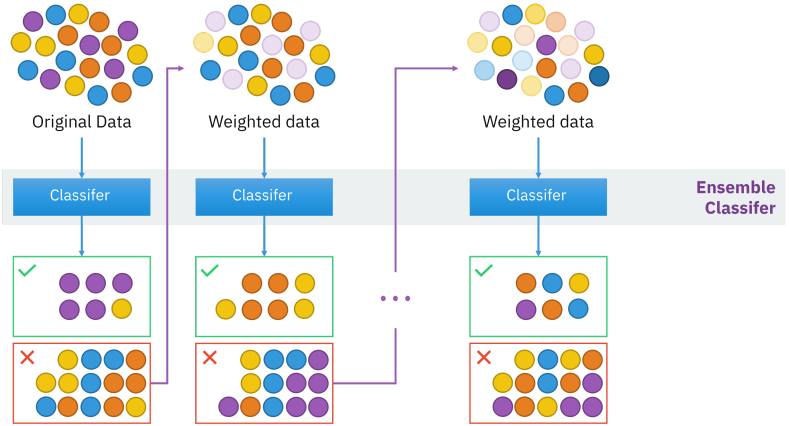


Figure 1: An illustration presenting the intuition behind the boosting algorithm, consisting of the parallel learners and weighted dataset.

## Contents

## AdaBoost

- Freund and Schapire proposed the adaptive boosting, called AdaBoost in 1996.
- AdaBoost repeatedly updates weights and learns the weak learner using modified versions of the data in every iteration.
- AdaBoost uses the stump as weak learner.

## Schematic of AdaBoost

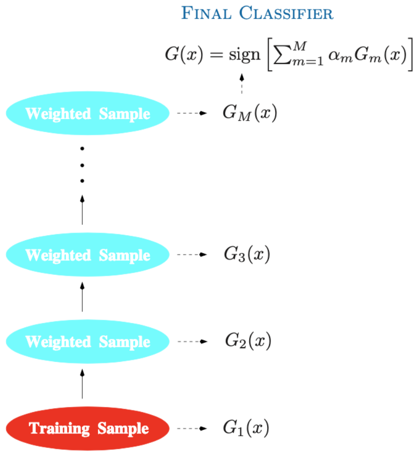
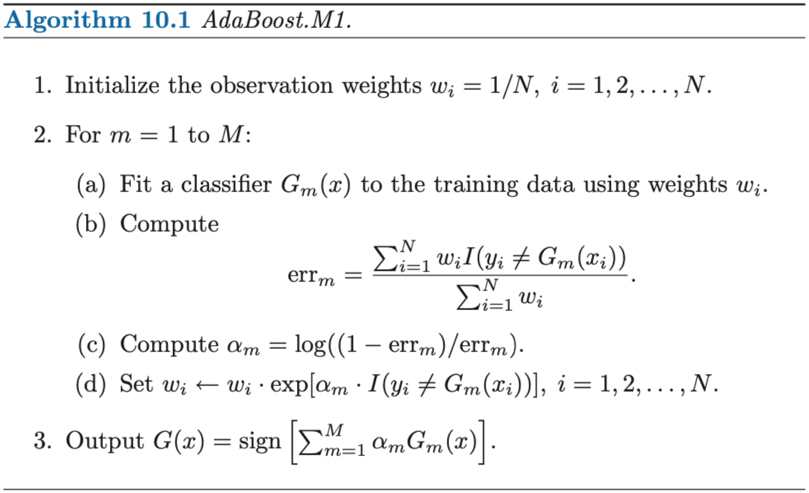


Figure 2: Classifiers are trained on weighted versions of the dataset, and then combined to produce a final prediction.

## Algorithm of AdaBoost



**Algorithm 10.1** *AdaBoost.M1.*

```
1. Initialize the observation weights $w_i = 1/N,\ i = 1, 2, \ldots, N$.
2. For $m = 1$ to $M$:
    (a) Fit a classifier $G_m(x)$ to the training data using weights $w_i$.
    (b) Compute
        $$\mathrm{err}_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}.$$
    (c) Compute $\alpha_m = \log((1 - \mathrm{err}_m)/\mathrm{err}_m)$.
    (d) Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))],\ i = 1, 2, \ldots, N$.
3. Output $G(x) = \mathrm{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$.
```

## Algorithm of AdaBoost

- Observations that were misclassified by the weak learner $G_{m-1}(x)$ at the previous step have increased weights, whereas the weights are decresaed for those that were classified correctly.
- $G_m(x)$ is fitted using weighted data from the previous step.
- The weighted error rate is calculated with the classifier $G_m(x)$.
- $\alpha_m$ is used for updating weights. And it works for the weight of each weak learner when we make the final classifier.

## Why it works?

- We can understand AdaBoost with statistical learning mechanism.
- From the classifier $G(x) = \text{sign}[\sum_m^M \alpha_m G_m(x)]$, we can think the boosting as a way of fitting an additive expansion in a set of basis function.
- Basis function expansions take the form

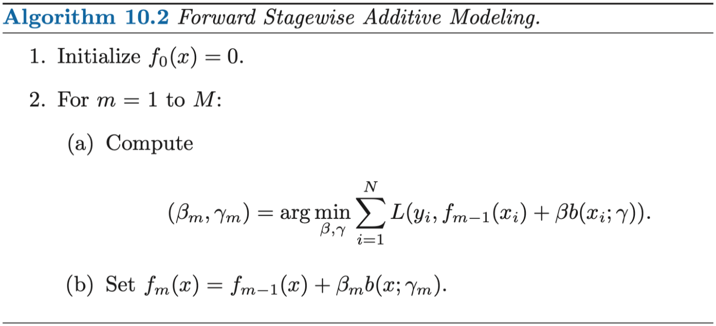
$$f(x) = \sum_{m=1}^{M} \beta_m b(x; \gamma_m)$$

where $\beta_m$ are the coefficients for basis function $b(x;\gamma) \in \mathbb{R}$, characterized by $\gamma_m$

- Here the $b(x;\gamma)$ are the individual learner $G_m(x) \in \{-1, 1\}$

## Forward Stagewise Additive Modeling

- We can find the solution with forward stagewise additive modeling algorithm.
- It finds the answer by sequentially adding new basis functions to the expansion with the functions that have already been added are fixed.



**Algorithm 10.2** *Forward Stagewise Additive Modeling.*

```
1. Initialize $f_0(x) = 0$.
2. For $m = 1$ to $M$:
    (a) Compute
        $$(\beta_m, \gamma_m) = \arg\min_{\beta,\gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma)).$$
    (b) Set $f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m)$.
```

## Equivalence between two algorithms

- We can see that AdaBoost is equivalent to forward stagewise addtive modeling using the 'exponential' loss function

$$L(y, f(x)) = \exp(-yf(x))$$

- We have to solve

$$(\beta_m, G_m) = \arg\min_{\beta, G} \sum_{i=1}^{n} \exp\left(-y_i(f_{m-1}(x_i) + \beta G(x_i))\right)$$

and we can rewrite it with $w_i^{(m)} = \exp(-y_i f_{m-1}(x_i))$,

$$(\beta_m, G_m) = \arg\min_{\beta, G} \sum_{i=1}^{n} w_i^{(m)} \exp(-\beta y_i G(x_i)) \tag{2.1}$$

## Equivalence between two algorithms

- For fixed $\beta > 0$, $G_m$ in (2.1) is calculated by

$$G_m = \arg\min_G \sum_{i=1}^{N} w_i^{(m)} I(y_i \neq G(x_i))$$

  and it minimizes the weighted error rate

$$\text{err}_m = \frac{\sum_{i=1}^{N} w_i^{(m)} I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i^{(m)}}$$

- We can get the solution of $\beta$ by plugging $G_m$ into (2.1)

$$\beta_m = \frac{1}{2} \log \frac{1 - \text{err}_m}{\text{err}_m}$$

## Equivalence between two algorithms

- Then the $f_m(x)$ is updated

$$f_m(x) = f_{m-1}(x) + \beta_m G_m(x)$$

- And the weights for the next step automatically changed

$$w_i^{(m+1)} = w_i^{(m)} e^{-\beta_m y_i G_m(x_i)} \tag{2.2}$$

- With the fact that $-y_i G_m(x_i) = 2 \cdot I(y_i \neq G_m(x_i)) - 1$, (2.2) becomes

$$w_i^{(m+1)} = w_i^{(m)} e^{\alpha_m I(y_i \neq G_m(x_i))} \cdot e^{-\beta_m} \tag{2.3}$$

where $\alpha_m = 2\beta_m$ from AdaBoost.

## Equivalence between two algorithms

- Because the $e^{-\beta_m}$ is multiplied to all the weights by same value, it has no effect on fitting the classifier.
- The weight update step in AdaBoost is equivalent to (2.3).
- We can see that the AdaBoost gets an answer by solving the optimization problem on loss functions.

## Contents

## Introduction

- Forward stagewise boosting(or AdaBoost) is a very greedy strategy that follows the optimal direction to minimize loss function.
- However, we can't find a solution when we don't use some loss functions like exponential loss.
- Friedman (2001) extended AdaBoost for general loss functions which are differentiable.
- Purpose doesn't change. We want to solve the problem below.

$$(\beta_m, \gamma_m) = \arg\min_{\gamma,\beta} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta h(x_i; \gamma))$$

## Gradient Boosting

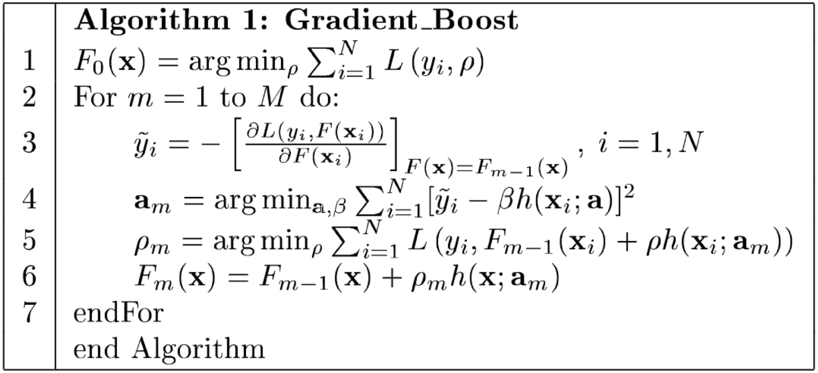
- Gradient boosting uses the gradient of loss function

$$g_m(x_i) = \left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F=F_{m-1}}$$

- But the best steepest descent direction $-\mathbf{g}_m = \{-g_m(x_i)\}_1^N$ is defined only at the data points $\{x_i\}_1^N$.
- Gradient boosting chooses the basis function $h(x_i; \gamma_m)$ that produces $\mathbf{h}_m = \{h(x_i; \gamma_m)\}$ most parallel to $-\mathbf{g}_m \in \mathbb{R}^N$.

$$\gamma_m = \arg\min_{\gamma,\beta} \sum_{i=1}^{n} [-g_m(x_i) - \beta h(x_i; \gamma)]^2$$

# Algorithm of Gradient Boosting



```
Algorithm 1: Gradient_Boost
$F_0(\mathbf{x}) = \arg\min_\rho \sum_{i=1}^N L(y_i, \rho)$
For $m = 1$ to $M$ do:
    $\tilde{y}_i = -\left[\frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}\right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})}, \ i = 1, N$
    $\mathbf{a}_m = \arg\min_{\mathbf{a},\beta} \sum_{i=1}^N [\tilde{y}_i - \beta h(\mathbf{x}_i; \mathbf{a})]^2$
    $\rho_m = \arg\min_\rho \sum_{i=1}^N L(y_i, F_{m-1}(\mathbf{x}_i) + \rho h(\mathbf{x}_i; \mathbf{a}_m))$
    $F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \rho_m h(\mathbf{x}; \mathbf{a}_m)$
endFor
end Algorithm
```

## Application: Gradeint Tree Boosting

- Basis function $h$, regression tree with $J$ terminal nodes, can be expreesed the additive form

$$h(x; \{b_j, R_j\}_1^J) = \sum_{j=1}^{J} w_j I(x \in R_j)$$

where $w_j$ is the weight for the terminal node $R_j$.

- From the gradient boosting algorithm, the model update step is

$$\begin{aligned} f_m(x) &= f_{m-1}(x) + \rho_m \sum_{j=1}^{J} w_{jm} I(x \in R_{jm}) \\ &= f_{m-1}(x) + \sum_{j=1}^{J} \gamma_{jm} I(x \in R_{jm}) \end{aligned}$$

where $\gamma_{jm} = \rho_m w_{jm}$

## Application: Gradeint Tree Boosting

- We can optimize $\gamma_{jm}$

$$
\begin{aligned}
& (\gamma_{1m}, \cdots, \gamma_{Jm}) = \arg \min_{\gamma_{1m}, \cdots, \gamma_{Jm}} \sum_{i=1}^{N} L(y_i, F_{m-1}(x_i) + \sum_{j=1}^{J} \gamma_j I(x \in R_{jm})) \\
\iff \quad & \gamma_{jm} = \arg \min_{\gamma} \sum_{x_i \in R_{jm}} L(y_i, F_{m-1}(x_i) + \gamma), \quad j = 1, \cdots, J
\end{aligned}
$$

## Contents

## eXtreme Gradient Boosting

- Tianqi Chen and Carlos Guestrin(2016) proposed XGBoost, eXtreme Gradient Boosting, which is based on gradient boosting.
- XGboost is faster and more accurate than gradient boosting algorithm.
- It improves overfitting problem of gradient boosting.

## Tree Ensemble Models

- For a data set with $n$ examples and $m$ features $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}$ $(|\mathcal{D}| = n,\ \mathbf{x}_i \in \mathbb{R}^m.\ y_i \in \mathbb{R})$, a tree ensemble model uses $K$ additive functions to predict the output.

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

  where $\mathcal{F} = \{f(\mathbf{x}) = w_{q(\mathbf{x})}\}(q : \mathbb{R}^m \to T, w \in \mathbb{R}^T)$ is the space of basic tree model.

# Tree Ensemble Models

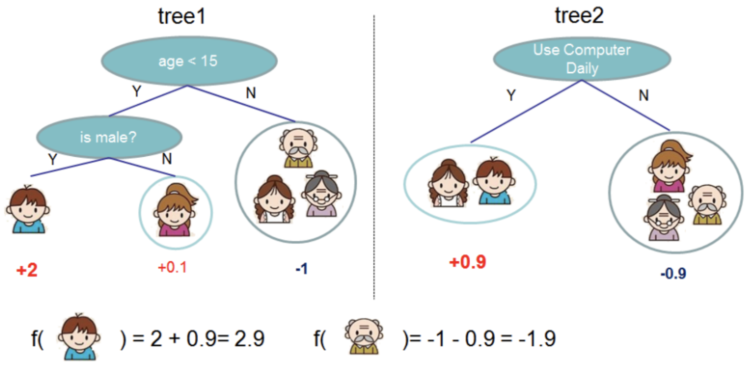


Figure 3: Tree Ensemble Model. The final prediction for a given example is the sum of predictionsfrom each tree.

## Regularized Learning Objective

- To learn the individual trees in XGBoost, we minimize the *regularized* objective.

$$\mathcal{L}^{(t)}(\phi) = \sum_i l(y_i, \hat{y}_i) + \Omega(f_t)$$

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

- $l$ is a differentiable convex loss function.
- $\Omega$ penalized the complexity of the trees. It helps to make simple functions and smooth the weights for leaf nodes to avoid overfitting.
- If the second term $\Omega$ goes to 0, it is equal to objective used in traditional gradient boosting.

## Gradient Tree Boosting

- In XGBoost, the model is trained in an additive manner. We learn the each tree sequentially.
- Let the $\hat{y}_i^{(t)}$ be the prediction value of the $i$-th examples at the $t$-th iteration. We have to find optimal $f_t$ in below,

$$\mathcal{L}^{(t)}(\phi) = \sum_i l(y_i, \hat{y}_i^{(t-1)} + f_t(\mathbf{x}_i)) + \Omega(f_t)$$

- We greedily add the $f_t$ which minimizes the objective.

## Gradient Tree Boosting

- Using second order approximation, we can quickly optimize the objective.

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

  where $g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)}),\ h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)})$ are the first and second derivative of loss function.

- And we can erase the constant terms to simplify the objective at step $t$.

$$\tilde{\mathcal{L}}^{(t)} = \sum_{i=1}^{n} \left[ g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i) \right] + \Omega(f_t)$$

## Gradient Tree Boosting

- Set $I_j = \{i | q(\mathbf{x}_i) = j\}$ as the observation set of the leaf $j$. The equation can be rewritten By expanding $\Omega$

$$\begin{aligned}\tilde{\mathcal{L}}^{(t)} &= \sum_{i=1}^{n} \left[ g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i) \right] + \gamma T + \frac{1}{2} \lambda \sum_{j=1}^{T} w_j^2 \\ &= \sum_{j=1}^{T} \left[ (\sum_{i \in I_j} g_i) w_j + \frac{1}{2} (\sum_{i \in I_j} h_i + \lambda) w_j^2 \right] + \gamma T\end{aligned}$$

## Gradient Tree Boosting

- Then we can compute the optimal weight and value of the simplified objective.

$$w_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} + \lambda}$$

$$\tilde{\mathcal{L}}^{(t)}(q) = -\frac{1}{2} \sum_{j=1}^{T} \frac{(\sum_{i \in I_j} g_i)^2}{\sum_{i \in I_j} h_i + \lambda} + \gamma T$$

- $\tilde{\mathcal{L}}^{(t)}(q)$ used as a scoring function to measure the quality of a tree structure $q$.
- At an any single node, the loss reduction after the split is given By

$$\mathcal{L}_{split} = \frac{1}{2} \left[ \frac{(\sum_{i \in I_L} g_i)^2}{\sum_{i \in I_L} h_i + \lambda} + \frac{(\sum_{i \in I_R} g_i)^2}{\sum_{i \in I_R} h_i + \lambda} - \frac{(\sum_{i \in I} g_i)^2}{\sum_{i \in I} h_i + \lambda} \right] - \gamma$$

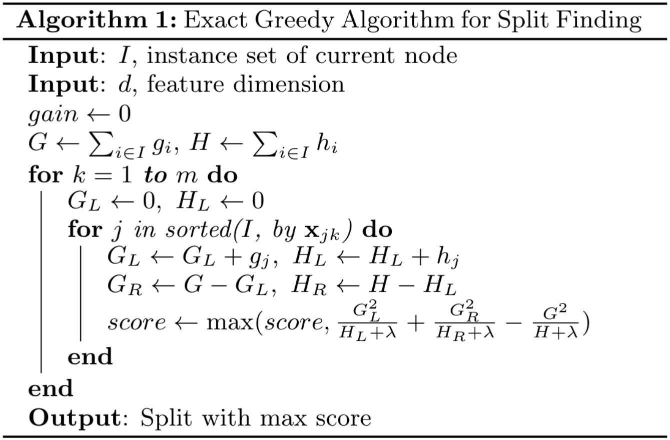
where $I_L, I_R$ are the observation sets of separated nodes.

## Shrinkage and Column Subsampling

- Shrinkage and column subsampling are two additional techniques to prevent overfitting.
- Shrinkage factor $\eta$ is multiplied to each tree model. It reduces the influence of each tree and gives a chance to improve the whole model for the trees at future steps.
- Column subsampling presents randomness for learning each tree. It is from Random Forest.

## Split Finding Algorithm

- Finding the best split is important issue in tree learning.
- Exact greedy algorithm enumerates over all the possible splits on all the features.

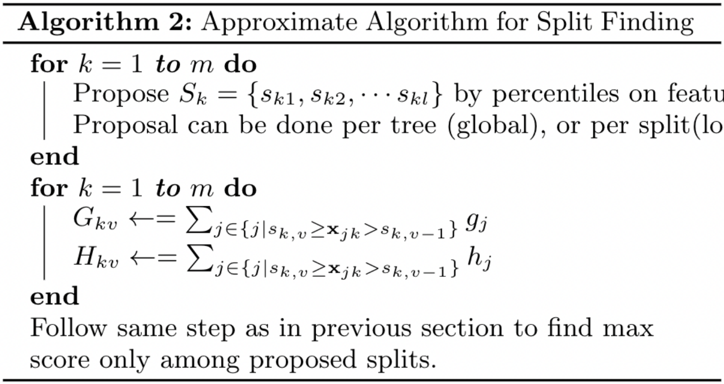


**Algorithm 1:** Exact Greedy Algorithm for Split Finding

```
Input: $I$, instance set of current node
Input: $d$, feature dimension
$gain \leftarrow 0$
$G \leftarrow \sum_{i \in I} g_i$, $H \leftarrow \sum_{i \in I} h_i$
for $k = 1$ to $m$ do
    $G_L \leftarrow 0,\ H_L \leftarrow 0$
    for $j$ in sorted($I$, by $\mathbf{x}_{jk}$) do
        $G_L \leftarrow G_L + g_j,\ H_L \leftarrow H_L + h_j$
        $G_R \leftarrow G - G_L,\ H_R \leftarrow H - H_L$
        $score \leftarrow \max(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$
    end
end
Output: Split with max score
```

## Approximate Algorithm

- Exact greedy algorithm is powerful, but it is computationally impossible to find all candidates on continuous features.



**Algorithm 2:** Approximate Algorithm for Split Finding

```
for $k = 1$ to $m$ do
    Propose $S_k = \{s_{k1}, s_{k2}, \cdots s_{kl}\}$ by percentiles on feature $k$.
    Proposal can be done per tree (global), or per split(local).
end
for $k = 1$ to $m$ do
    $G_{kv} \leftarrow= \sum_{j \in \{j | s_{k,v} \geq \mathbf{x}_{jk} > s_{k,v-1}\}} g_j$
    $H_{kv} \leftarrow= \sum_{j \in \{j | s_{k,v} \geq \mathbf{x}_{jk} > s_{k,v-1}\}} h_j$
end
Follow same step as in previous section to find max
score only among proposed splits.
```

## Approximate Algorithm

- The algorithm first suggests candidate split points according to percentiles of feature distribution through weighted quantile sketch algorithm.
- Then, it puts the features into the groups split by the candidate points calculated in advance.
- There are two variants, global and local variant.
  1. Global : proposing all the cadidate splits at the initial phase of learning.
  2. Local : re-proposing the points after each split.

# Approximate Algorithm

- Global variant needs more candidates because the split points are not changed during the training. And the local variant is more proper for deeper trees.

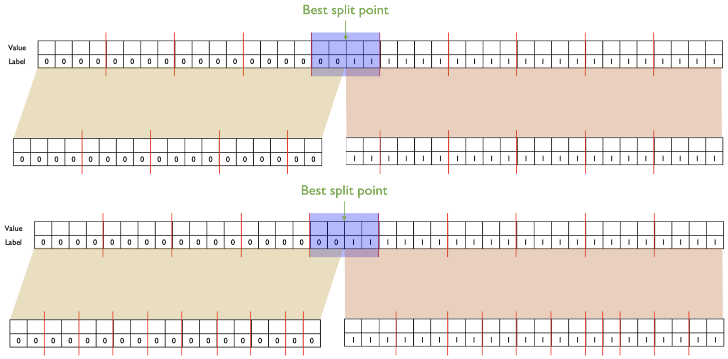


Figure 4: Global and local proposal.

# Approximate Algorithm

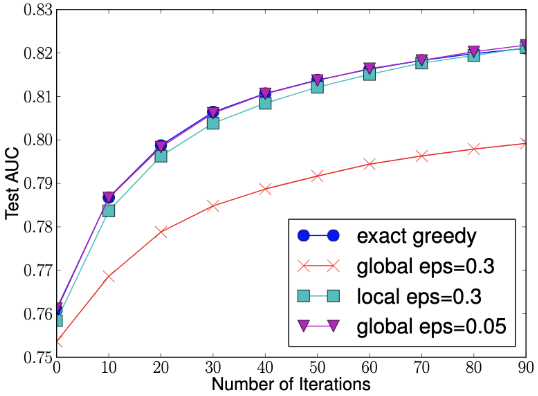


Figure 5: The eps parameter corresponds to the accuracy of the approximate sketch. This roughly translates to 1 / eps buckets in the proposal.We find that local proposals require fewer buckets,because it refine split candidates

## Weighted Quantile Sketch

- Proposing the candidate split points is importants step in the approximate algorithm. XGBoost uses *weighted quantile sketch* algorithm in this stage.
- Let $\mathcal{D}_k = \{(x_{1k}, h_1), \cdots, (x_{nk}, h_k)\}$ means $k$-th feature values and second order gradient statistics of each training examples.
- Define the rank functions $r_k : \mathbb{R} \to [0, +\infty)$ as

$$r_k(z) = \frac{1}{\sum_{(x,h)\in\mathcal{D}_k} h} \sum_{(x,h)\in\mathcal{D}_k, x<z} h$$

- It represents the proportion of instances.

## Weighted Quantile Sketch

- Our goal is find candidate split points $\{s_{k1}, s_{k2}, \cdots, s_{kl}\}$ satisfying

$$|r_k(s_{k,j}) - r_k(s_{k,j+1})| < \epsilon, \quad s_{k1} = \min_i \mathbf{x}_{ik}, s_{kl} = \max_i \mathbf{x}_{ik}$$

  where $\epsilon$ is an approximation factor.
- Roughly $1/\epsilon$ candidate points are picked up.
- $h_i$ is considered as weight of each data point. The objective at step $t$, $\tilde{\mathcal{L}}^{(t)}$, can be rewritten. It shows the role of $h_i$

$$\sum_{i=1}^{n} \frac{1}{2} h_i (f_t(\mathbf{x}_i) - g_i/h_i)^2 + \Omega(f_t) + constant$$

## Sparsity-aware Split Finding

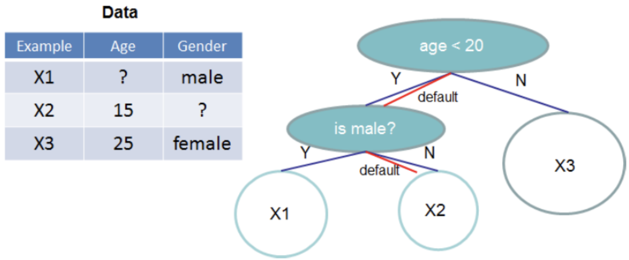
- Sparsity-aware split finding algorithm is used in XGBoost to handel a sparsity pattern in data.
- There are various causes for sparsity in real world problems.
  1. presence of missing values
  2. frequent zero entries
  3. artifacts of feature engineering such as one-hot encoding
- Using a default direction in each node, missing values are classified.

## Sparsity-aware Split Finding

**Algorithm 3:** Sparsity-aware Split Finding

```
Input: $I$, instance set of current node
Input: $I_k = \{i \in I | x_{ik} \neq \text{missing}\}$
Input: $d$, feature dimension
Also applies to the approximate setting, only collect
statistics of non-missing entries into buckets
$gain \leftarrow 0$
$G \leftarrow \sum_{i \in I}, g_i, H \leftarrow \sum_{i \in I} h_i$
for $k = 1$ to $m$ do
    // enumerate missing value goto right
    $G_L \leftarrow 0,\ H_L \leftarrow 0$
    for $j$ in sorted($I_k$, ascent order by $\mathbf{x}_{jk}$) do
        $G_L \leftarrow G_L + g_j,\ H_L \leftarrow H_L + h_j$
        $G_R \leftarrow G - G_L,\ H_R \leftarrow H - H_L$
        $score \leftarrow \max(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$
    end
    // enumerate missing value goto left
    $G_R \leftarrow 0,\ H_R \leftarrow 0$
    for $j$ in sorted($I_k$, descent order by $\mathbf{x}_{jk}$) do
        $G_R \leftarrow G_R + g_j,\ H_R \leftarrow H_R + h_j$
        $G_L \leftarrow G - G_R,\ H_L \leftarrow H - H_R$
        $score \leftarrow \max(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$
    end
end
Output: Split and default directions with max gain
```

# Sparsity-aware Split Finding



**Data**

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

Figure 6: Tree structure with default directions. An example will be classified into the default direction when the feature needed for the split is missing.

## Sparsity-aware Split Finding

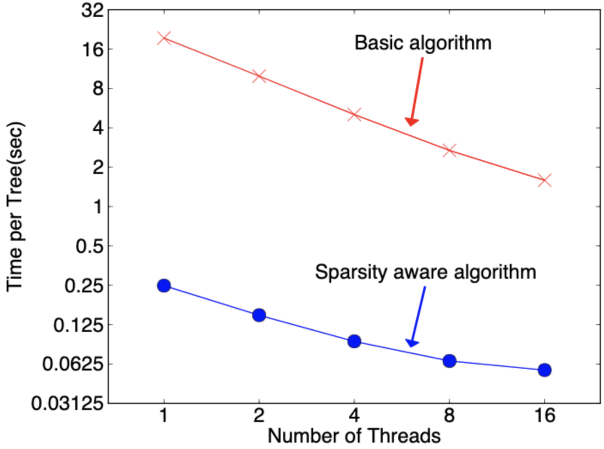


Figure 7: Impact of the aprsity aware algorithm. The dataset is aprse mainly due to one-hot encoding.

- Recently, faster and stronger methods like LightGBM and CatBoost are proposed.
- LightGBM improves the learning speed dramatically and CatBoost works very well with categorical features.
- Boosting is still powerful comparing to the deep learning algorithms.

## References

- Chen T and Guestrin C. XGBoost: A Scalable Tree Boosting System. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, p.785-794, 2016.
- Pilsung Kang. Business Analytics - IME654, Korea University
- Yoav Freund and Robert E. Schapire. a decision theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, p.119-139, 1996.
- Hastie T., Tibshirani R., Friedman J. H. (2001). The elements of statistical learning: Data mining, inference, and prediction.
- Friedman J. H.(2001), Greedy Function Approximation: A Gradient Boosting Machine, *The Annals of Statistics*