XGBOOST AS A GRADIENT TREE BOOSTING ALGORITHM

MARIUSZ BUDZIŃSKI

Abstract

"Statistics starts with data." Those are the first words of Leo‘s Breiman famous paper [4] which summarize main differences between statistical and machine learning. Machine learning has this advatage over statistical learning that it do not assume nothing about how data set was collected and/or nothing about its distribution. Whole family of such algorithms which also constitute one of the most powerful machine learning aproaches are tree-based algorithms. Starting from decision and regression tree next step is to combine many such models to recive more powerful comite. Very recently because in 2014 there was introduced such algorithm called **XGBoost** [5] which I will describe in this report. This consist the goal of this paper.

1. Introduction

Before we introduce with details mentioned algorithm I would like to give brief overview of tree-based algorithms.

1.1. **Tree based algorithm‘s overview.** The essential idea of tree-based algorithms is to divide feature space into p$-$ dimensional boxes (p is dimension of a feature space) and to apply at each box the same, simple model. To see why does it work recall few basic facts about conditional expectation. Let $\mathbf{X}$ be a random variable defined on probability space $(\Omega, \mathcal{F}, \mathbb{P})$ and consider finite disjoint partition of Ω, literally $\{\mathrm{A}_i\}_{i\in 1:p}$, where there exist some A_i having positive probability and let $\sigma = \sigma(\{\mathrm{A}_i\}_{i\in 1:p})$. Then of course

$$\int_{A_i} \mathbb{E}(\mathbf{X}|\sigma)d\mathbb{P} = \int_{A_i} X d\mathbb{P}$$

for every $i \in 1:p$ and

$$\mathbb{E}(\mathbf{X}|\sigma)(\omega) = \begin{cases} \mathbb{E}(\mathbf{X}|\mathrm{A}_i) & \omega \in \mathrm{A}_i, \mathbb{P}(\mathrm{A_i}) > 0 \\ a_i \in \mathbb{R} & \omega \in \mathrm{A}_i, \mathbb{P}(\mathrm{A_i}) = 0 \end{cases}$$

Based on the above we can treat $\mathbb{E}(\mathbf{X}|\sigma)(\omega)$ as an approximation of $\mathbf{X}$, which is constant on each partition set A_i. Define now

$$\hat{\mathbf{X}} \coloneqq \mathbb{E}(\mathbf{X}|\sigma)$$

Notice that such approximator is unbiased since $\mathbb{E}(\mathbb{E}(\mathbf{X}|\sigma)) = \mathbb{E}(\mathbf{X})$. To check how good this approximation is let consider its L_2 error

$$L_2(\mathbf{X}, \hat{\mathbf{X}}) = \mathrm{E}((\mathbf{X} - \hat{\mathbf{X}})^2) = \mathbb{E}(\mathbf{X}^2) + \mathbb{E}(\hat{\mathbf{X}}^2) - 2\int_\Omega \hat{\mathbf{X}}\mathbf{X} d\mathbb{P}$$

Notice that the last integral is equal to

$$\sum_{i=1}^{p} \mathbb{E}^2(\mathbb{X}|\mathrm{A_i})\mathbb{P}(\mathrm{A}_i).$$

We get the following

$$L_2(\mathbf{X}, \hat{\mathbf{X}}) = \mathbb{E}(\mathbf{X}^2) - \mathbb{E}(\hat{\mathbf{X}}^2).$$

Since $\mathbb{E}^2(\mathbf{X}) = \mathbb{E}^2(\hat{\mathbf{X}}) \leq \mathbb{E}(\hat{\mathbf{X}}^2)$ we see that $L_2(\mathbf{X}, \hat{\mathbf{X}})$ is smaller than $Var(\mathbf{X})$. Consider now another, bigger, disjoint partition $\{\mathrm{B}_i\}_{i\in 1:p+1}$ of Ω, say $\mathrm{A}_1 = \mathrm{B}_1 \bigcup \mathrm{B}_2$ and for $3 \leq i$, $\mathrm{B}_i = \mathrm{A}_{i-1}$. Similarly define

$$\bar{\mathbf{X}} := \mathbb{E}(\mathbf{X}|\sigma(\{\mathrm{B}_i\}_i)).$$

Then we have:

$$\mathbb{E}(\bar{\mathbf{X}}^2) - \mathbb{E}(\hat{\mathbf{X}}^2) = \mathbb{E}^2(\mathbb{X}|\mathrm{B}_1)\mathbb{P}(\mathrm{B}_1) + \mathbb{E}^2(\mathbb{X}|\mathrm{B}_2)\mathbb{P}(\mathrm{B}_2) - \mathbb{E}^2(\mathbb{X}|\mathrm{A}_1)\mathbb{P}(\mathrm{A}_1)$$

Using that

$$\mathbb{E}(\mathbb{X}|\mathrm{A}_1)\mathbb{P}(\mathrm{A}_1) = \mathbb{E}(\mathbb{X}|\mathrm{B}_1)\mathbb{P}(\mathrm{B}_1) + \mathbb{E}(\mathbb{X}|\mathrm{B}_2)\mathbb{P}(\mathrm{B}_2)$$

we revice the following

$$\mathbb{E}(\bar{\mathbf{X}}^2) - \mathbb{E}(\hat{\mathbf{X}}^2) = \frac{\mathbb{P}(\mathrm{B}_1)\mathbb{P}(\mathrm{B}_2)}{\mathbb{P}(\mathrm{A}_1)}(\mathbb{E}(\mathbb{X}|\mathrm{B}_1) - \mathbb{E}(\mathbb{X}|\mathrm{B}_2))^2 > 0$$

and

$$L_2(\mathbf{X}, \hat{\mathbf{X}}) - L_2(\mathbf{X}, \bar{\mathbf{X}}) > 0.$$

We conclude that the bigger partition is then the smaller L_2 error is. This can be rewriten for $\sigma-$algebras in general. Since $\mathbb{E}(\mathbf{X}|\sigma)$ minimize

$$\mathbb{E}(\mathbf{X} - \mathbf{Y})^2$$

among all $\mathbf{Y}$ being σ-measurable. So if we have two sigma algebras σ_1, σ_2 such that $\sigma_2 \supset \sigma_1$ then

$$\mathbb{E}(\mathbf{X} - \mathbb{E}(\mathbb{X}|\sigma_2))^2 < \mathbb{E}(\mathbf{X} - \mathbb{E}(\mathbb{X}|\sigma_1))^2.$$

Conlusion is simple:

Observation 1. *The bigger σ is the better aproximation of $\mathbf{X}$ by $\mathbb{E}(\mathbb{X}|\sigma)$ we get.*

1.1.1. *Problem formulation.* Let $\mathbf{Y}$ be our target variable and $\mathbf{X} = \{\mathbf{X}\}_{i\in 1:\bar{p}\in \mathrm{N}}$ be set of explanatory variables:

$$(\Omega, \mathcal{F}, \mathbb{P}) \xrightarrow{\mathbf{Y}} (\mathcal{Y}, \mathcal{F}_{\mathcal{Y}}, \mathbb{P}_{\mathcal{Y}}),$$
$$(\Omega, \mathcal{F}, \mathbb{P}) \xrightarrow{\mathbf{X}} (\mathcal{X}, \mathcal{F}_{\mathcal{X}}, \mathbb{P}_{\mathcal{X}}),$$

where $(\Omega, \mathcal{F}, \mathbb{P})$ remains unknown the same as $\mathbb{P}_{\mathcal{X}}$ and $\mathbb{P}_{\mathcal{Y}}$ if $|\mathcal{Y}| \neq 2$. For now let $\mathbf{Y}$ be continuous valued. Our task is based on data set $(\mathrm{X}_i, \mathrm{Y}_i)_{i\in 1:n}$, which is n times realization of random vector $(\mathbf{X}, \mathbf{Y})$, to build predictive model $\mathcal{M}_{(\mathrm{X}_i,\mathrm{Y}_i)_{i\in 1:n}}(\cdot)$ that for any realization of $\mathbf{X}$, literally X, gives an answear $\mathcal{M}_{(\mathrm{X}_i,\mathrm{Y}_i)_{i\in 1:n}}(\mathrm{X})$ as close as possible to real one. Notice that we are not making any assumption about $\mathbf{X}$, where usualy in statistical learning it is iid. Now, in **Observation 1**, to approximate $\mathbf{Y}$ we can use $\sigma(\mathbf{X}) = \sigma(\mathbf{X}_1, \ldots, \mathbf{X}_p)$, and assume for now that all of $\mathbf{X}_i$ are categorical. Since $\sigma(\mathbf{X}) = \sigma(\bigcup_{i=1}^p \sigma(\mathbf{X}_i))$ we have that $\sigma(\mathbf{X})$ is generated by set-theoretic operations of elements from $\{A_{ij}\}_{i\in 1:p, j\in 1:n_i}$, where for each i and j, $A_{ij} \in \sigma(\mathbf{X}_i)$. Of course, for each i, $\sigma(\mathbf{X}_i)$ is generated by all unions of elements from $\{\mathbf{X}_i^{-1}(x_j)\}_{j\in 1:n_i}$, where n_i states number of possible values that $\mathbf{X}_i$ can admit.

Now our regression model can be written in the following way

$$\mathcal{M}_{(\mathrm{X}_i,\mathrm{Y}_i)_{i\in 1:n}}(\mathbf{X}(\omega)) = \mathbb{E}(\mathbf{Y}|\sigma(\mathbf{X}_1, \ldots, \mathbf{X}_p))(\omega) = \mathbb{E}(\mathbf{Y}|\mathrm{A}_i) \tag{1}$$

where $\mathbf{X}(\omega) \in \mathrm{A}_i$, $\mathrm{A}_i \in \sigma(\mathbf{X})$ and $\bigcup_i \mathrm{A}_i = \mathcal{X}$. In classification tasks we can for each k-th class of $\mathbf{Y}$ find

$$\mathbb{E}(\mathbb{1}_{\mathbf{Y}==k}|\sigma(\mathbf{X}_1, \ldots, \mathbf{X}_p))(\omega) = \mathbb{E}(\mathbb{1}_{\mathbf{Y}==k}|\mathrm{A}_i) = \mathbb{P}(\mathbf{Y} == k|\mathrm{A}_i) \tag{2}$$

and then decision rule of our classifier $d(\mathbf{X})$ become

$$d(\mathbf{X}(\omega)) = \operatorname*{argmax}_k \mathbb{E}(\mathbb{1}_{\mathbf{Y}==k}|\sigma(\mathbf{X}_1, \ldots, \mathbf{X}_p))(\omega), \tag{3}$$

where n_i, A_i have the same meaning as above. In (1) and (2) try to use empirical measure of training sample instead to recive approximators of given qunatities. Finaly receiving:
in (1)

$$\mathbb{E}(\mathbf{Y}|\mathrm{A}_i) = \frac{N_i}{N}\sum_{j=1}^{N}\mathbb{1}_{\mathrm{X}_j \in A_i}\mathrm{Y}_j$$

and in (2)

$$\mathbb{P}(\mathbf{Y} == k|\mathrm{A}_i) = \frac{1}{N_i}\sum_{j=1}^{N}\mathbb{1}_{\mathrm{X}_j \in A_i}\mathbb{1}_{\mathrm{Y}_j == k}$$

Now if some of the predictors, let say $\mathbf{X}_k$, is continuous valued then when building $\sigma(\mathbf{X})$, elements of $\sigma(\mathbf{X}_k)$ can be replaced by set made by all set-theoretic operation of elements from $\{\mathbf{X}_k^{-1}(a_i)\}_i$, where $\{a_i\}_i$ is partition of $\mathbb{R}$ into non overlapping intervals. The maximum number of such intervals is $N+1$, N is number of learning examples. Nautral question appears how to find proper $\{A_i\}_i \in \sigma(\mathbf{X})$ in (1) and (3) and $\{a_i\}_i$ for all continuous valued variables. In regression tasks by

$$\theta^* = \underset{\theta}{\operatorname{argmax}} -\mathbb{E}(\mathcal{M}^{\theta}_{(\mathrm{X}_i,\mathrm{Y}_i)_{i\in 1:n}}(\mathbf{X}) - \mathbf{Y})^2$$

and in classification tasks by

$$\theta^* = \underset{\theta}{\operatorname{argmax}} -\mathbb{E}(\mathcal{M}^{\theta}_{(\mathrm{X}_i,\mathrm{Y}_i)_{i\in 1:n}}(\mathbf{X}) \neq \mathbf{Y}), \tag{4}$$

in practise using empirical measure to approximate $\mathbb{P}_{\mathbf{X},\mathbf{Y}}$ when taking $\mathbb{E}$, where θ means $\{A_i\}_i$ and all partitions $\{a_i\}_i$ for continuous variables. Note that classification task can be seen as modified *Karp's cover set problem* [6] since from finite set $\sigma(\mathbb{X})$ we need to find subcollection that covers whole space and each found set of this partition is as most homogeneous as possible by a value of $\mathbf{Y}$. Also notice that classification problem is ill posed since the solution might not be unique. Further notice that number of elements of $\sigma(\mathbf{X})$ grows rapidly:

$$\begin{aligned} |\sigma(\mathbf{X}_i)| &= \sum_{k=0}^{n_i}\binom{n_i}{k} = 2^{n_i}, \\ |\sigma(\bigcup_{i=1}^{p}\sigma(\mathbf{X}_i))| &> 2^{\sum_i^p n_i}, \end{aligned} \tag{5}$$

where n_i is number of classes of categorical variable or number of elements of partition for continuous variable. Because it would be hard to check all possibilities we need to have an efficient algorithm to find such regions. Tree-based algorithms perform at each step partition of input space based on some criterium in greedy fashion. Such solution is suboptimal. One of the most popular tree-based algorithms are C.4.5 [10] and CART [1]. I will skip details since they are easily accessible in the given references. It appears that trees are highly unstable, that means the small changes in input data might have very big impact on the final tree structure. Next step in tree-based algorithms is to build essemble of trees to increase their predictive power. One of the first such algorithms was *bagging* [2] and introduced later by the same author *random forests* [3]. First of them decrease variance of final prediction by averaging many trees. Second produce many as how it is possible uncorrelated trees and combine recived results. Since as autors showed that the more uncorelated and the more powerful each tree is then the smaller generalization error become. More details can be found in given references. Another way of combining trees is *boosting*, which is method of combining many 'weak' classifiers to produce powerful 'committe'. Firstly, it was developed for classification tasks but it was also extended to regression. Most comonly used is *AdaBoost* [7], which is sequentially applying a classification algorithm to reweighted versions of the training data and then taking a weighted majority vote of the sequnce of classifiers thus produced. Now in next section we will introduce **XGBoost**, algorithm of our interests, which is also a *boosting algorithm*.

2. Article presentation

Using notation from [5], the boosted tree model is sum of trees

$$\phi(\mathbf{X}) = \sum_{k=1}^{K} f_k(\mathbf{X}), \tag{6}$$

where $\wedge_k f_k \in \mathcal{F} = \{f(\mathbf{X}) = w_{q(\mathbf{X})} |\ q\colon\ \mathbb{R}^p \to \mathbb{R}^T\}$ is the space of regression trees and q reprezents structure of each tree that maps example $\mathbf{X}(\omega)$ to specific leaf of the tree, T is number of terminal leafs. Each terminal leaf is determined by its weights w. Unlike decision trees, each regression tree contains a continuous score on each of the leaf. To find proper f_k in (6) instead of minimizing (4) we can use general risk function $l(\mathbf{Y}, \hat{\mathbf{Y}})$ and additional *regularization term* $\Omega(f)$ receiving the following minimization criterium

$$\begin{gathered}\phi^* = \arg\min_{\phi} \left\{ \mathcal{L}(\phi) = \sum_i l(\mathrm{y}_i, \hat{\mathrm{y}}_i) + \sum_k \Omega(f_k) \right\}, \\ \text{where } \Omega(f_k) = \gamma T + \frac{1}{2}\lambda \left\| w \right\|^2 .\end{gathered} \tag{7}$$

Here $l(\cdot,\cdot)$ is a differentiable convex loss function and the regularization term is given to avoid over-fitting. If parameter λ is set to 0 we are receiving usual gradient tree boosting framework. The minimization in (7) is performed on boosted tree models (6) space, so cannot be optimized using traditional optimization methods in Euclidean space. Instead the model is trained in an additive manner. Formally, let $\hat{y}_i(t)$ be the prediction of the i-th observation at t-step, we will add f_t to (6) such that minimize the following

$$\mathcal{L}^{(t)} = \sum_i^n l(\mathrm{y}_i, \hat{\mathrm{y}}_i^{(t-1)}) + f_t(\mathrm{x}_i) + \Omega(f_t).$$

That means at each step t we are greedily adding solution that that most improves our model. We can use second order Taylor expantion of $l(\cdot,\cdot)$ with $h_i = \partial_{\hat{\mathrm{y}}^{(t-1)}} l(y_i, \hat{\mathrm{y}}^{(t-1)})$ and $g_i = \partial^2_{\hat{\mathrm{y}}^{(t-1)}} l(y_i, \hat{\mathrm{y}}^{(t-1)})$

$$\mathcal{L}^{(t)} \sim \sum_i^n [l(\mathrm{y}, \hat{\mathrm{y}}^{(t-1))}) + g_i(\mathrm{x}_i) f_t(\mathrm{x}_i) + \frac{1}{2} h_i(\mathrm{x}_i) f_t^2(\mathrm{x}_i)] + \Omega(f_t) \tag{8}$$

and forgetting about constant term right side of the (8) become as the following

$$\sum_i^n [g_i(\mathrm{x}_i) f_t(\mathrm{x}_i) + \frac{1}{2} h_i(\mathrm{x}_i) f_t^2(\mathrm{x}_i)] + \Omega(f_t). \tag{9}$$

Denoting $I_j\colon = \{i | q(x_i) = j\}$, we can rewrite the (9) in the following form

$$\begin{gathered}\sum_i^n [g_i(\mathrm{x}_i) f_t(\mathrm{x}_i) + \frac{1}{2} h_i(\mathrm{x}_i) f_t^2(\mathrm{x}_i)] + \gamma T + \frac{1}{2}\lambda \left\| w \right\|^2 = \\ \sum_{j=1}^{T} [(\sum_{i \in I_j} g_i) w_j + \frac{1}{2}(\sum_{i \in I_j} h_i + \lambda) w_j^2] + \gamma T.\end{gathered} \tag{10}$$

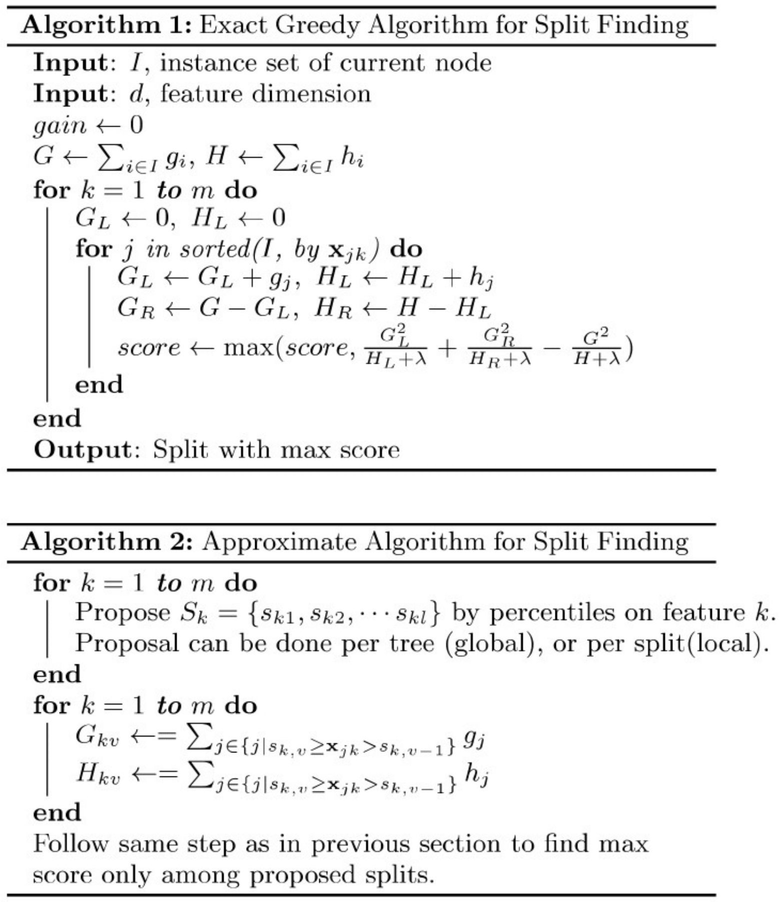
For fixed strucutre $q(x)$, by computeing gradient and finding point that minimize (10)we receive that the optimal weight w_j^* of j-th leaf is

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

and for such weights (10), named as $\tilde{\mathcal{L}^{(t)}}$, become

$$\tilde{\mathcal{L}^{(t)}} = -\frac{1}{2} \sum_{j=1}^{T} \frac{(\sum_{i \in I_j} g_i)^2}{\sum_{i \in I_j} h_i + \lambda} + \gamma T. \tag{11}$$

The above (11) can be used as a measure of goodness of tree structure q and since that to assess goodness of the split when a tree is build in greedy fashion, just by comparing (11) before and after spliting. In order to find the best split at each time, a split fnding algorithm enumerates over all the possible splits on all the features. In [5] this procedure is called *exact greedy algorithm* and it is supported by the single machine version of **XGBoost**. Direct formulation of the *exact greedy algorithm* can be found in [5]. However, when the data does not fit entirely into memory we can use *approximate algorithm*. Where candidates for splitting points are given according to percentiles of feature distribution. Further algorithm maps the continuous features into buckets, aggregates the statistics and finds the best solution among all proposals based on the aggregated statistics. There exist two versions of the algorithm: global and local. In first version proposals are given one time at the beggining of tree building process, in second one they are given at each split. **XGboost** for single machine supports *exact greedy algorithm* and *approximate algorithm* for weighted and non weighted data. Both of them are given in the following Figure 1.



```
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
```

```
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.
```

FIGURE 1. Figure taken from [5]

To comment how splitting candidates are found notice that (9) can be rewritten as the following

$$\sum_{i}^{n} \frac{1}{2} h_i(f_t(\mathrm{x}_i) + \frac{g_i(\mathrm{x}_i)}{h_i(\mathrm{x}_i)})^2 + \Omega(f_t) + \text{ const.},$$

which is exactly weighted squared loss with labels g_i/h_i and weights h_i. Now we can defne for each k-th numeric feature $\mathcal{D}_k = \{(x_{ik}, h_{ik})\}_{i \in 1:n}$ a rank functions $r_k \colon \mathbb{R} \to [0; \infty]$ as

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

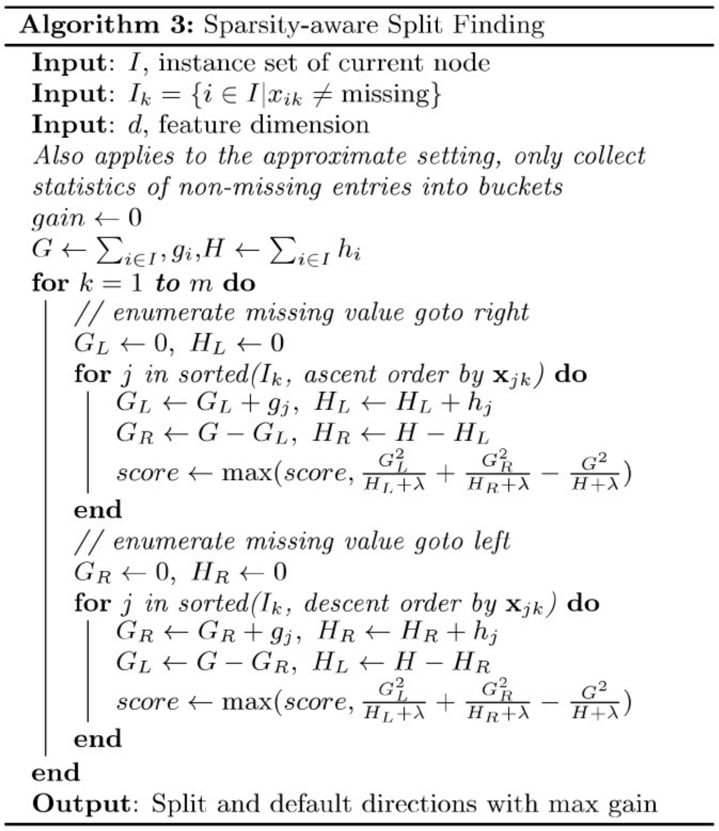
which represents the proportion of instances whose feature value k is smaller than z. The goal is to find candidate split points $\{s_{k1}, s_{k2}, .., s_{kl}\}$ such that

$$|r_k(s_{k,j}) - r_k(s_{k,j+1})| < \epsilon, \text{where } s_{k,1} = \min_i x_{ik} \text{ and } s_{k,l} = max_i x_{ik}.$$

When dataset is large and it is not weghted such proposal as described above can be found by usage of *quantile sketch* [8] but there is no existing quantile sketch for the weighted datasets. To solve this problem autors of [5] proposed a novel distributed weighted quantile sketch algorithm that can handle weighted data with a *provable theoretical guarantee* but with no references. Another issue present in real world data is sparsity, literally when data set is sparse. Such situation can be present due to:

(1) presence of missing values,
(2) frequent zero entries in the statistics,
(3) one-hot encoding or other feature engineering operations.

The solution given in [5] is simple: add default direction in each tree node for missing values in sparse data set matrix. Two directions are checked: left and right, default direction will be such that gives better decrease in (11) after splitting. The *Sparsity-Aware Split Finding algorithm* is attached in the following Figure 2:



```
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
```

FIGURE 2. Figure taken from [5]

Also **XGBoost** supports shrinkage and subsampling. The first technique scales newly added weights by a factor after each step of tree boosting. Since that shrinkage reduces the infuence of each individual tree and leaves space for future trees to improve the model. The second technique is column (feature) subsampling. This technique is used in RandomForest [3]. Using column sub-sampling prevents over-fitting.

Rest of the paper [5] is devoted to purely computational issues, literally, how data are stored in computer memory and how they are processed by computer cores. That is why I am skipping this details. Noteworthy is that purely computional ideas given by authors dramatically speed up computations in compare to other existing boosting algorithms, especially when a data set is large. Summary is given in the following Fugure 3.

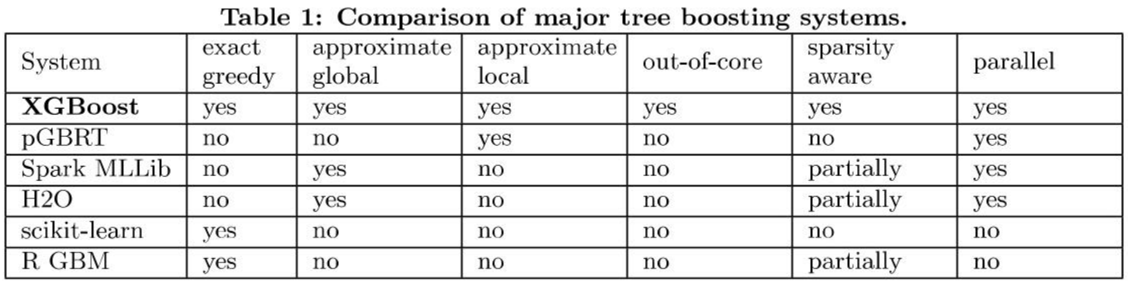


Table 1: Comparison of major tree boosting systems.

System	exact greedy	approximate global	approximate local	out-of-core	sparsity aware	parallel
XGBoost	yes	yes	yes	yes	yes	yes
pGBRT	no	no	yes	no	no	yes
Spark MLLib	no	yes	no	no	partially	yes
H2O	no	yes	no	no	partially	yes
scikit-learn	yes	no	no	no	no	no
R GBM	yes	no	no	no	partially	no

FIGURE 3. Figure taken from [5]

From the above Figure 3. we see that **XGBoost** supports *out-of-core* as well as *parallel* computations, respecting also *cache-aware* access. And to sum up **XGBoost** is optimized system that at the end is *gradient tree boosting algorithm.*

3. Future directions

About application of **XGBoost** and its remarkable successes. Look at machine learning competition site Kaggle. Among the 29 challenge winning solutions 3 published at Kaggle's blog during 2015, 17 solutions used XGBoost. For comparison, the second most popular method, deep neural nets, was used in 11 solutions. The success of the system was also witnessed in KDDCup 2015, where XGBoost was used by every winning team in the top-10. Moreover, the winning teams reported that ensemble methods outperform a well-configured XGBoost by only a small amount

Although **XGBoost** was a big step forward in big data processing, the efficiency and scalability are still unsatisfactory when the feature dimension is high and data size is large. This is because it is needed to scan all the data instances to estimate splitting statistics for all possible spliting points, which is very time consuming. To handle this issue in 2017 was introduced **LightGBM** [9] were two novel techniques were proposed: Gradient-based One-Side Sampling (GOSS) and Exclusive Feature Bundling (EFB). GOSS exclude a significant proportion of data instances with small gradients, and only use the rest to estimate the information gain. In paper, athors proved that, since the data instances with larger gradients play a more important role in the computation of information gain, GOSS can obtain quite accurate estimation of the information gain with a much smaller data size. Second algorithm, EFB bundle mutually exclusive features (i.e., that rarely take nonzero values simultaneously), to reduce the number of features. Authors proved that finding the optimal bundling of exclusive features is NP-hard, but a greedy algorithm can achieve quite good approximation ratio (and thus can effectively reduce the number of features without hurting the accuracy of split point determination by much). Experiments on multiple public datasets shown in [9] show that, LightGBM speeds up the training process of conventional GBDT by up to over 20 times while achieving almost the same accuracy.

As we see progress in *tree boosting algorithms* was made mainly by purely computational changes. How data are processed by computer. All mathematics behind stays more or less unchanged. Certainly we will be surprised how purely comutational tricks will improve well know machine learning algorthms, just like it was with *gradient tree boosting approach.*

References

[1] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA, 1984.

[2] Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[3] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[4] Leo Breiman. Statistical modeling: The two cultures (with comments and a rejoinder by the author). *Statistical Science*, 16(3):199–231, 08 2001.

[5] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In Balaji Krishnapuram, Mohak Shah, Alexander J. Smola, Charu Aggarwal, Dou Shen, and Rajeev Rastogi, editors, *KDD*, pages 785–794. ACM, 2016.

[6] Stephen A. Cook. The complexity of theorem-proving procedures. In Michael A. Harrison, Ranan B. Banerji, and Jeffrey D. Ullman, editors, *STOC*, pages 151–158. ACM, 1971.

[7] J. Friedman, T. Hastie, and R. Tibshirani. Additive Logistic Regression: a Statistical View of Boosting. *The Annals of Statistics*, 38(2), 2000.

[8] M. Greenwald and S. Khanna. Space-efficient online computation of quantile summaries. *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*.

[9] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 3146–3154. Curran Associates, Inc., 2017.

[10] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.