Gradient Boosting Decision Trees

Milan Straka

December 07, 2020

Charles University in Prague
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics

Gradient Boosting Decision Trees

The gradient boosting decision trees also train a collection of decision trees, but unlike random forests, where the trees are trained independently, in GBDT they are trained sequentially to correct the errors of the previous trees.

If we denote y_t as the prediction function of the t^{th} tree, the prediction of the whole collection is then

$$y(\boldsymbol{x}_i) = \sum_{t=1}^{T} y_t(\boldsymbol{x}_i; \boldsymbol{W}_t),$$

where $\boldsymbol{W}_t$ is a vector of parameters (leaf values, to be concrete) of the t^{th} tree.

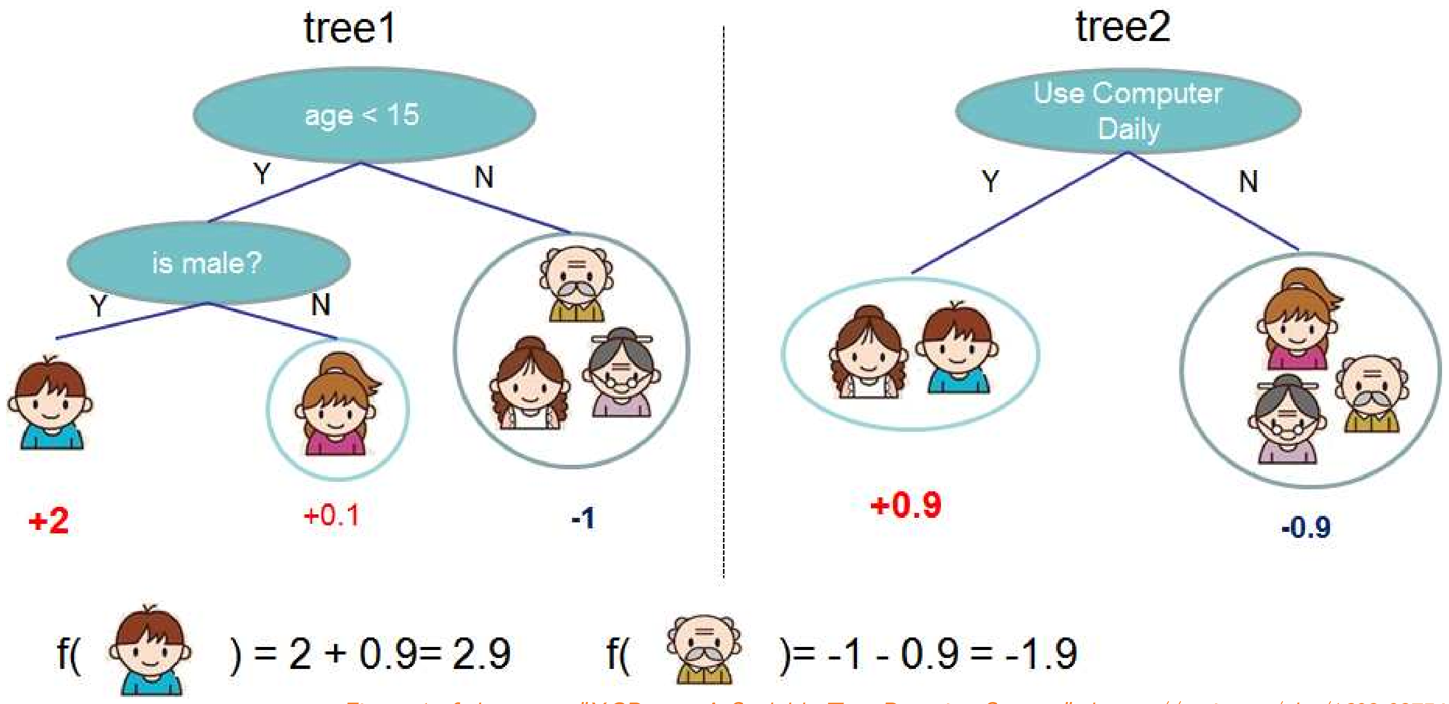


Figure 1 of the paper "XGBoost: A Scalable Tree Boosting System", https://arxiv.org/abs/1603.02754

Gradient Boosting for Regression

Considering a regression task first, we define the overall loss as

$$\mathcal{L}(\boldsymbol{W}) = \sum_i \ell\big(t_i, y(\boldsymbol{x}_i; \boldsymbol{W})\big) + \sum_{t=1}^{T} \frac{1}{2}\lambda \big\|\boldsymbol{W}_t\big\|^2,$$

where

- $\boldsymbol{W} = (\boldsymbol{W}_1, \ldots, \boldsymbol{W}_T)$ are the parameters (leaf values) of the trees;
- $\ell\big(t_i, y(\boldsymbol{x}_i; \boldsymbol{W})\big)$ is an per-example loss, $(t_i - y(\boldsymbol{x}_i; \boldsymbol{W}))^2$ for regression;
- the λ is the usual L_2 regularization strength.

Gradient Boosting for Regression

To construct the trees sequentially, we extend the definition to

$$\mathcal{L}^{(t)}(\boldsymbol{W}_t; \boldsymbol{W}_{1..(t-1)}) = \sum_i \Big[\ell\big(t_i, y^{(t-1)}(\boldsymbol{x}_i; \boldsymbol{W}_{1..(t-1)}) + y_t(\boldsymbol{x}_i; \boldsymbol{W}_t)\big)\Big] + \frac{1}{2}\lambda\big\|\boldsymbol{W}_t\big\|^2.$$

In the following text, we drop the parameters of $y^{(t-1)}$ and y_t for brevity.

The original idea of gradient boosting was to set $y_t(\boldsymbol{x}_i) \propto -\frac{\partial\ell\big(t_i, y^{(t-1)}(\boldsymbol{x}_i)\big)}{\partial y^{(t-1)}(\boldsymbol{x}_i)}$ as a direction minimizing the residual loss and then finding a suitable constant γ_t which would minimize the loss $\sum_i \Big[\ell\big(t_i, y^{(t-1)}(\boldsymbol{x}_i) + \gamma_t y_t(\boldsymbol{x}_i)\big)\Big] + \frac{1}{2}\lambda\big\|\boldsymbol{W}_t\big\|^2$.

Gradient Boosting

However, a more principled approach was later suggested. Denoting

$$g_i = \frac{\partial \ell\big(t_i, y^{(t-1)}(\boldsymbol x_i)\big)}{\partial y^{(t-1)}(\boldsymbol x_i)}$$

and

$$h_i = \frac{\partial^2 \ell\big(t_i, y^{(t-1)}(\boldsymbol x_i)\big)}{\partial y^{(t-1)}(\boldsymbol x_i)^2},$$

we can expand the objective $\mathcal{L}^{(t)}$ using a second-order approximation to

$$\mathcal{L}^{(t)}(\boldsymbol W_t; \boldsymbol W_{1..(t-1)}) \approx \sum_i \Big[\ell\big(t_i, y^{(t-1)}(\boldsymbol x_i)\big) + g_i y_t(\boldsymbol x_i) + \frac{1}{2} h_i y_t^2(\boldsymbol x_i)\Big] + \frac{1}{2}\lambda \big\|\boldsymbol W_t\big\|^2.$$

Gradient Boosting

Recall that we denote the indices of instances belonging to a node $\mathcal{T}$ as $I_\mathcal{T}$, and let us denote the prediction for the node $\mathcal{T}$ as $w_\mathcal{T}$. Then we can rewrite

$$\begin{aligned}\mathcal{L}^{(t)}(\boldsymbol{W}_t; \boldsymbol{W}_{1..(t-1)}) &\approx \sum_i \Big[g_i y_t(\boldsymbol{x}_i) + \frac{1}{2} h_i y_t^2(\boldsymbol{x}_i)\Big] + \frac{1}{2}\lambda \big\|\boldsymbol{W}_t\big\|^2 + \mathrm{const} \\ &\approx \sum_\mathcal{T} \Big[\Big(\sum_{i \in I_\mathcal{T}} g_i\Big) w_\mathcal{T} + \frac{1}{2}\Big(\lambda + \sum_{i \in I_\mathcal{T}} h_i\Big) w_\mathcal{T}^2\Big] + \mathrm{const}\end{aligned}$$

By setting a derivative with respect to $w_\mathcal{T}$ to zero, we get the optimal weight for a node $\mathcal{T}$:

$$w^*_\mathcal{T} = -\frac{\sum_{i \in I_\mathcal{T}} g_i}{\lambda + \sum_{i \in I_\mathcal{T}} h_i}.$$

Gradient Boosting

Substituting the optimum weights to the loss, we get

$$\mathcal{L}^{(t)}(\boldsymbol{W}) \approx -\frac{1}{2}\sum_{\mathcal{T}} \frac{\left(\sum_{i\in I_\mathcal{T}} g_i\right)^2}{\lambda + \sum_{i\in I_\mathcal{T}} h_i} + \text{const},$$

which can be used as a splitting criterion.

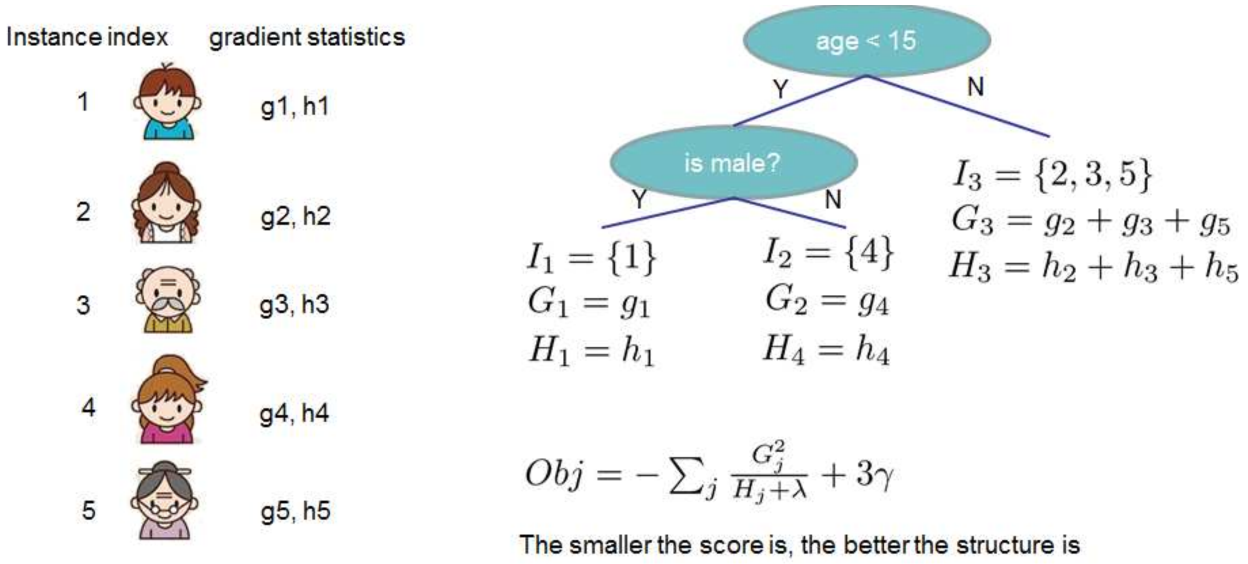


Figure 2 of the paper "XGBoost: A Scalable Tree Boosting System", https://arxiv.org/abs/1603.02754

Gradient Boosting

When splitting a node, the criterions of all possible splits can be effectively computed using the following algorithm:

Algorithm 1: Exact Greedy Algorithm for Split Finding

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

Modified from Algorithm 1 of the paper "XGBoost: A Scalable Tree Boosting System", https://arxiv.org/abs/1603.02754

Gradient Boosting

Furthermore, gradient boosting trees frequently use:

- data subsampling: either bagging, or (even more commonly) utilize only a fraction of the original training data for training a single tree (with 0.5 a common value),
- feature bagging;
- shrinkage: multiply each trained tree by a learning rate α, which reduces influence of each individual tree and leaves space for future optimization.

Binary Classification with Gradient Boosting Decision Trees

To perform classification, we train the trees to perform the linear part of a generalized linear model.

Specifically, for a binary classification, we perform prediction by

$$\sigma\big(y(\boldsymbol{x}_i)\big) = \sigma\left(\sum_{t=1}^T y_t(\boldsymbol{x}_i; \boldsymbol{W}_t)\right),$$

and the per-example loss is defined as

$$\ell\big(t_i, y(\boldsymbol{x}_i)\big) = -\log\Big[\sigma\big(y(\boldsymbol{x}_i)\big)^{t_i}\big(1-\sigma\big(y(\boldsymbol{x}_i)\big)\big)^{1-t_i}\Big].$$

Multiclass Classification with Gradient Boosting Decision Trees

For multiclass classification, we need to model the full categorical output distribution. Therefore, for each "timestep" t, we train K trees $\boldsymbol{W}_{t,k}$, each predicting a single value of the linear part of a generalized linear model.

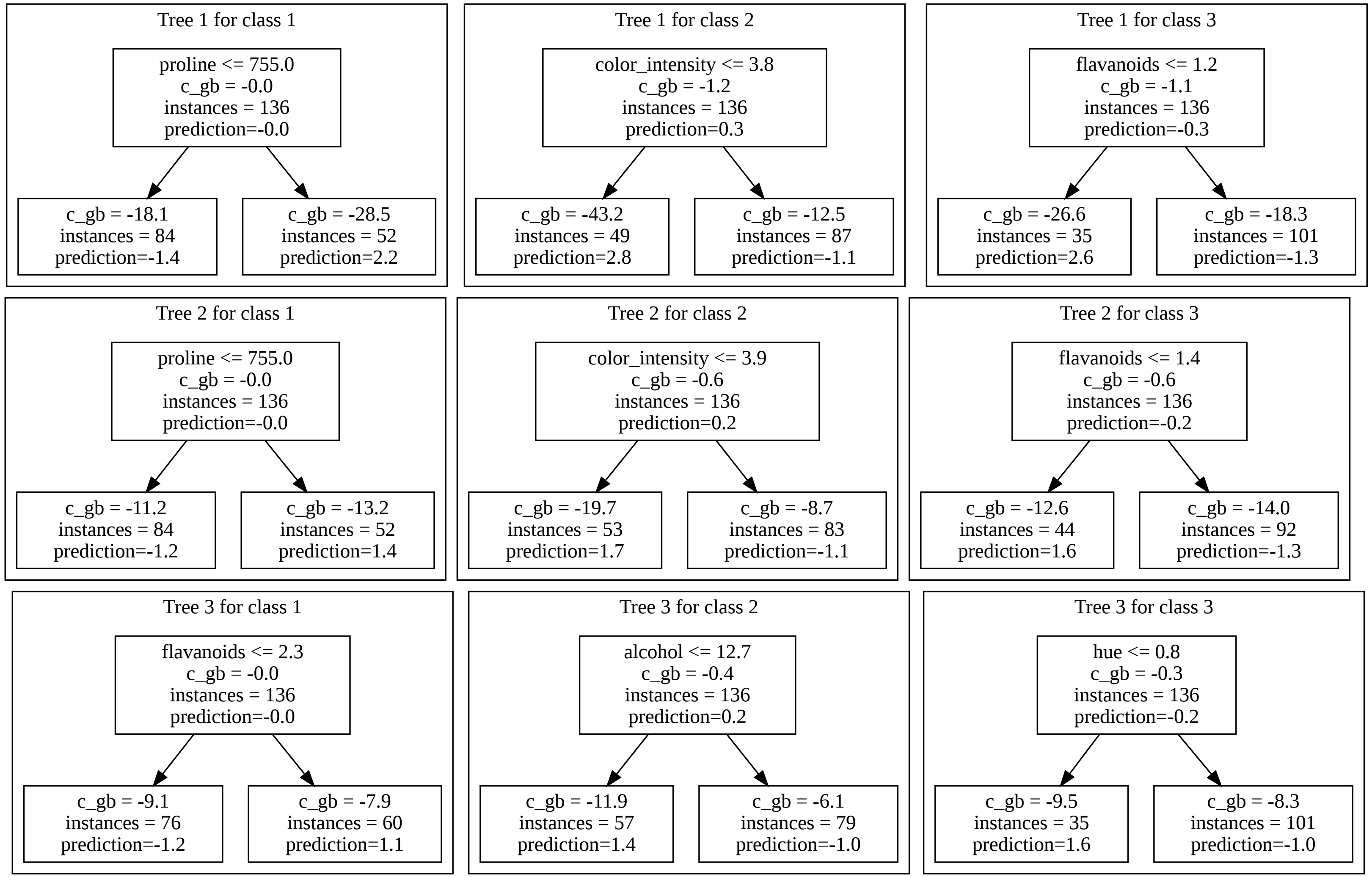
Then, we perform prediction by

$$\operatorname{softmax}\big(\boldsymbol{y}(\boldsymbol{x}_i)\big) = \operatorname{softmax}\left(\sum\nolimits_{t=1}^T y_{t,1}(\boldsymbol{x}_i; \boldsymbol{W}_{t,1}), \ldots, \sum\nolimits_{t=1}^T y_{t,K}(\boldsymbol{x}_i; \boldsymbol{W}_{t,K})\right),$$

and the per-example loss is defined analogously as

$$\ell\big(t_i, \boldsymbol{y}(\boldsymbol{x}_i)\big) = -\log\Big(\operatorname{softmax}\big(\boldsymbol{y}(\boldsymbol{x}_i)\big)_{t_i}\Big).$$

Multiclass Classification with Gradient Boosting Decision Trees



Gradient Boosting Demo and Implementations

Playground

You can explore the Gradient Boosting Trees playground.

Implementations

Scikit-learn offers an implementation of gradient boosting decision trees, `sklearn.ensemble.GradientBoostingClassifier` for classification and `sklearn.ensemble.GradientBoostingRegressor` for regression.

There are additional efficient implementations, capable of distributed processing of data larger than available memory:

- XGBoost,
- LightGBM, both offering scikit-learn interface, among others.

Supervised Machine Learning

This concludes the **supervised machine learning** part of our course.

We have encountered:

- parametric models
  - generalized linear models: perceptron algorithm, linear regression, logistic regression, multinomial (softmax) logistic regression, Poisson regression
    - linear models, but manual feature engineering allows solving non-linear problems
  - multilayer perceptron: non-linear model according to Universal approximation theorem
- non-parametric models
  - k-nearest neighbors
  - kernelized linear regression
  - support vector machines
- decision trees
  - can be both parametric or non-parametric depending on the constraints
- generative models
  - naive Bayes