Model Combinations, Gradient Boosted Trees, Naive Bayess

Milan Straka

December 09, 2019

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

Decision Trees

The idea of decision trees is to partition the input space into usually cuboid regions and solving each region with a simpler model.

We focus on **Classification and Regression Trees** (CART; Breiman et al., 1984), but there are additional variants like ID3, C4.5, ...

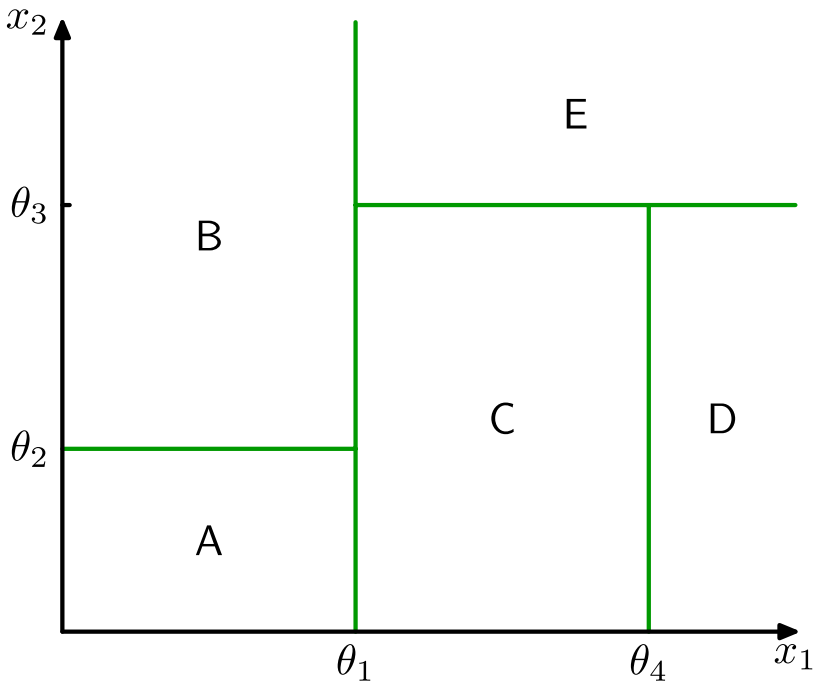


Figure 14.6 of Pattern Recognition and Machine Learning.

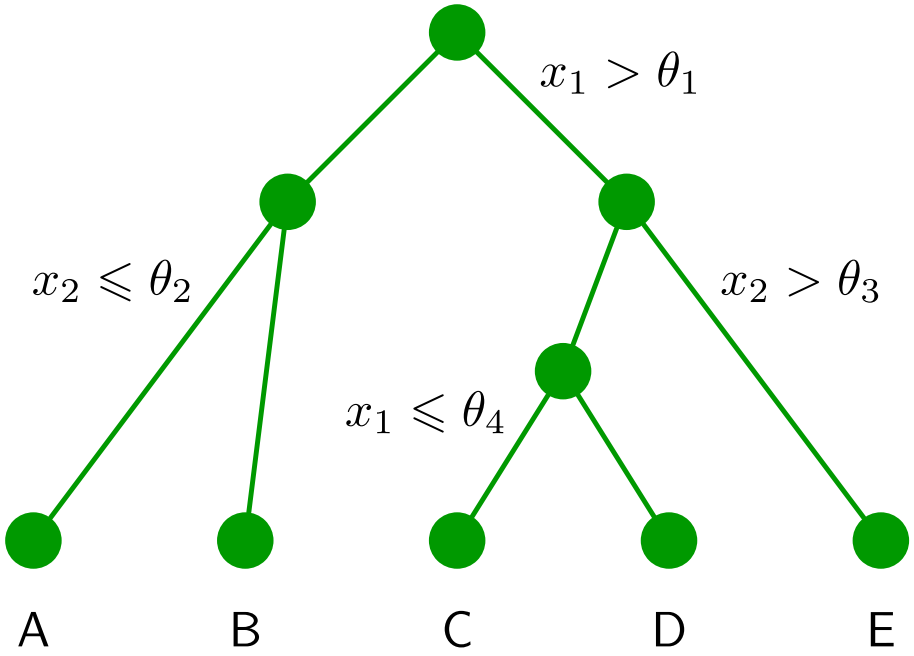


Figure 14.5 of Pattern Recognition and Machine Learning.

Regression Decision Trees

Assume we have an input dataset $\boldsymbol{X} \in \mathbb{R}^{N \times D}$, $\boldsymbol{t} \in \mathbb{R}^{N}$. At the beginning, the decision tree is just a single node and all input examples belong to this node. We denote $I_{\mathcal{T}}$ the set of training example indices belonging to a leaf node $\mathcal{T}$.

For each leaf, our model will predict the average of the training examples belonging to that leaf, $\hat{t}_{\mathcal{T}} = \frac{1}{|I_{\mathcal{T}}|} \sum_{i \in I_{\mathcal{T}}} t_i$.

We will use a *criterion* $c_{\mathcal{T}}$ telling us how *uniform* or *homogeneous* are the training examples belonging to a leaf node $\mathcal{T}$ – for regression, we will employ the sum of squares error between the examples belonging to the node and the predicted value in that node; this is proportional to variance of the training examples belonging to the leaf node $\mathcal{T}$, multiplied by the number of the examples. Note that even if it not *mean* squared error, it is sometimes denoted as MSE.

$$c_{\mathrm{SE}}(\mathcal{T}) \stackrel{\mathrm{def}}{=} \sum_{i \in I_{\mathcal{T}}} (t_i - \hat{t}_{\mathcal{T}})^2, \text{ where } \hat{t}_{\mathcal{T}} = \frac{1}{|I_{\mathcal{T}}|} \sum_{i \in I_{\mathcal{T}}} t_i.$$

Tree Construction

To split a node, the goal is to find a feature and its value such that when splitting a node $\mathcal{T}$ into $\mathcal{T}_L$ and $\mathcal{T}_R$, the resulting regions decrease the overall criterion value the most, i.e., the difference $c_{\mathcal{T}_L} + c_{\mathcal{T}_R} - c_{\mathcal{T}}$ is the lowest.

Usually we have several constraints, we mention on the most common ones:

- **maximum tree depth**: we do not split nodes with this depth;
- **minimum examples to split**: we only split nodes with this many training examples;
- **maximum number of leaf nodes**

The tree is usually built in one of two ways:

- if the number of leaf nodes is unlimited, we usually build the tree in a depth-first manner, recursively splitting every leaf until some above constraint is invalidated;
- if the maximum number of leaf nodes is give, we usually split such leaf $\mathcal{T}$ where the criterion difference $c_{\mathcal{T}_L} + c_{\mathcal{T}_R} - c_{\mathcal{T}}$ is the lowest.

Classification Decision Trees

For multi-class classification, we predict such class most frequent in the training examples belonging to a leaf $\mathcal{T}$.

To define the criterions, let us denote the average probability for class k in a region $\mathcal{T}$ at $p_{\mathcal{T}}(k)$.

For classification trees, one of the following two criterions is usually used:

- **Gini index**:

$$c_{\text{Gini}}(\mathcal{T}) \stackrel{\text{def}}{=} |I_{\mathcal{T}}| \sum_k p_{\mathcal{T}}(k)\big(1 - p_{\mathcal{T}}(k)\big)$$

- **Entropy Criterion**

$$c_{\text{entropy}}(\mathcal{T}) \stackrel{\text{def}}{=} |I_{\mathcal{T}}| H(p_{\mathcal{T}}) = -|I_{\mathcal{T}}| \sum_k p_{\mathcal{T}}(k) \log p_{\mathcal{T}}(k)$$

Binary Gini as (M)SE Loss

Recall that $I_\mathcal{T}$ denotes the set of training example indices belonging to a leaf node $\mathcal{T}$, let $n_\mathcal{T}(0)$ be the number of examples with target value 0, $n_\mathcal{T}(1)$ be the number of examples with target value 1, and let $p_\mathcal{T} = \frac{1}{|I_\mathcal{T}|} \sum_{i \in I_\mathcal{T}} t_i$.

Consider sum of squares loss $\mathcal{L}(p) = \sum_{i \in I_\mathcal{T}} (p - t_i)^2$.

By setting the derivative of the loss to zero, we get that the p minimizing the loss fulfils $|I_\mathcal{T}| p = \sum_{i \in I_\mathcal{T}} t_i$, i.e., $p = p_\mathcal{T}$.

The value of the loss is then

$$\begin{aligned}
\mathcal{L}(p_\mathcal{T}) &= \sum_{i \in I_\mathcal{T}} (p_\mathcal{T} - t_i)^2 = n_\mathcal{T}(0)(p_\mathcal{T} - 0)^2 + n_\mathcal{T}(1)(p_\mathcal{T} - 1)^2 \\
&= \frac{n_\mathcal{T}(0) n_\mathcal{T}(1)^2}{\big(n_\mathcal{T}(0) + n_\mathcal{T}(1)\big)^2} + \frac{n_\mathcal{T}(1) n_\mathcal{T}(0)^2}{\big(n_\mathcal{T}(0) + n_\mathcal{T}(1)\big)^2} = \frac{n_\mathcal{T}(0) n_\mathcal{T}(1)}{n_\mathcal{T}(0) + n_\mathcal{T}(1)} \\
&= \big(n_\mathcal{T}(0) + n_\mathcal{T}(1)\big)(1 - p_\mathcal{T}) p_\mathcal{T} = |I_\mathcal{T}| p_\mathcal{T}(1 - p_\mathcal{T})
\end{aligned}$$

Entropy as NLL Loss

Again let $I_{\mathcal{T}}$ denote the set of training example indices belonging to a leaf node $\mathcal{T}$, let $n_{\mathcal{T}}(c)$ be the number of examples with target value c, and let $p_{\mathcal{T}}(c) = \frac{n_{\mathcal{T}}(c)}{|I_{\mathcal{T}}|} = \frac{1}{|I_{\mathcal{T}}|} \sum_{i \in I_{\mathcal{T}}} [t_i = c]$.

Consider a distribution $\boldsymbol{p}$ on K classes and non-averaged NLL loss $\mathcal{L}(\boldsymbol{p}) = \sum_{i \in I_{\mathcal{T}}} -\log p_{t_i}$.

By setting the derivative of the loss with respect to p_c to zero (using a Lagrangian with constraint $\sum_c p_c = 1$), we get that the $\boldsymbol{p}$ minimizing the loss fulfils $p_c = p_{\mathcal{T}}(c)$.

The value of the loss with respect to $p_{\mathcal{T}}$ is then

$$\begin{aligned}
\mathcal{L}(p_{\mathcal{T}}) &= \sum_{i \in I_{\mathcal{T}}} -\log p_{t_i} \\
&= -\sum_c n_{\mathcal{T}}(c) \log p_{\mathcal{T}}(c) \\
&= -|I_{\mathcal{T}}| \sum_c p_{\mathcal{T}}(c) \log p_{\mathcal{T}}(c) = |I_{\mathcal{T}}| H(p_{\mathcal{T}})
\end{aligned}$$

Ensembling

Ensembling is combining several models with a goal of reaching higher performance.

The simplest approach is to train several independent models and then averaging their output.

Given that for independent identically distributed random values X_i we have

$$\mathrm{Var}(\sum X_i) = \sum \mathrm{Var}(X_i)$$
$$\mathrm{Var}(a \cdot X) = a^2 \, \mathrm{Var}(X),$$

we get that

$$\mathrm{Var}\left(\frac{1}{n}\sum X_i\right) = \frac{1}{n}\,\mathrm{Var}(X_1).$$

Therefore, if the models exhibit independent errors, these errors will cancel out with more models.

Bagging

For neural network models, the simple ensembling is usually enough, given that the loss has many local minima, so the models tend to be quite independent just when using different initialization.

However, algorithms with a convex loss functions usually converge to the same optimum independent on randomization.

In these cases, we can use **bagging**, which stands for **bootstrap aggregation**.

In bagging, we construct a different dataset for every model to be trained. We construct it using **bootstrapping** – we sample as many training instances as the original dataset has, but **with replacement**.

Such dataset is sampled using the same empirical data distribution and has the same size, but is not identical.

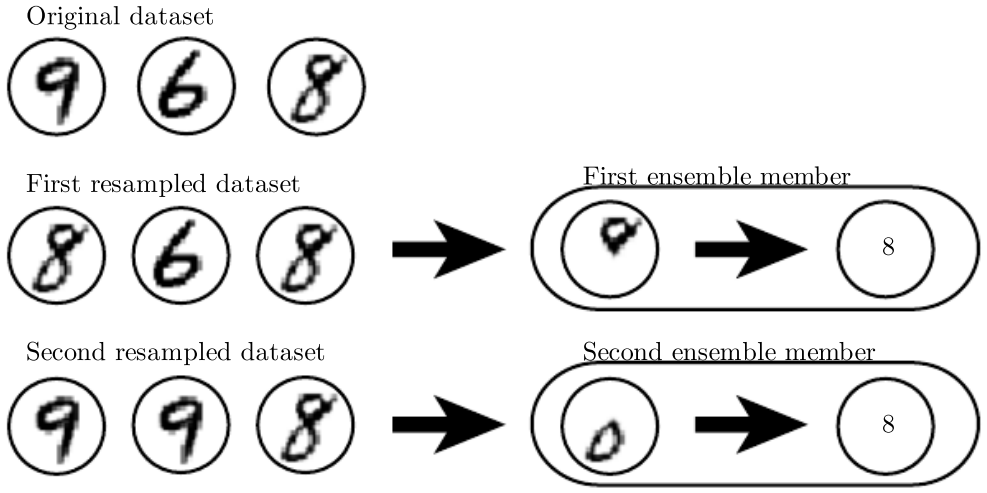
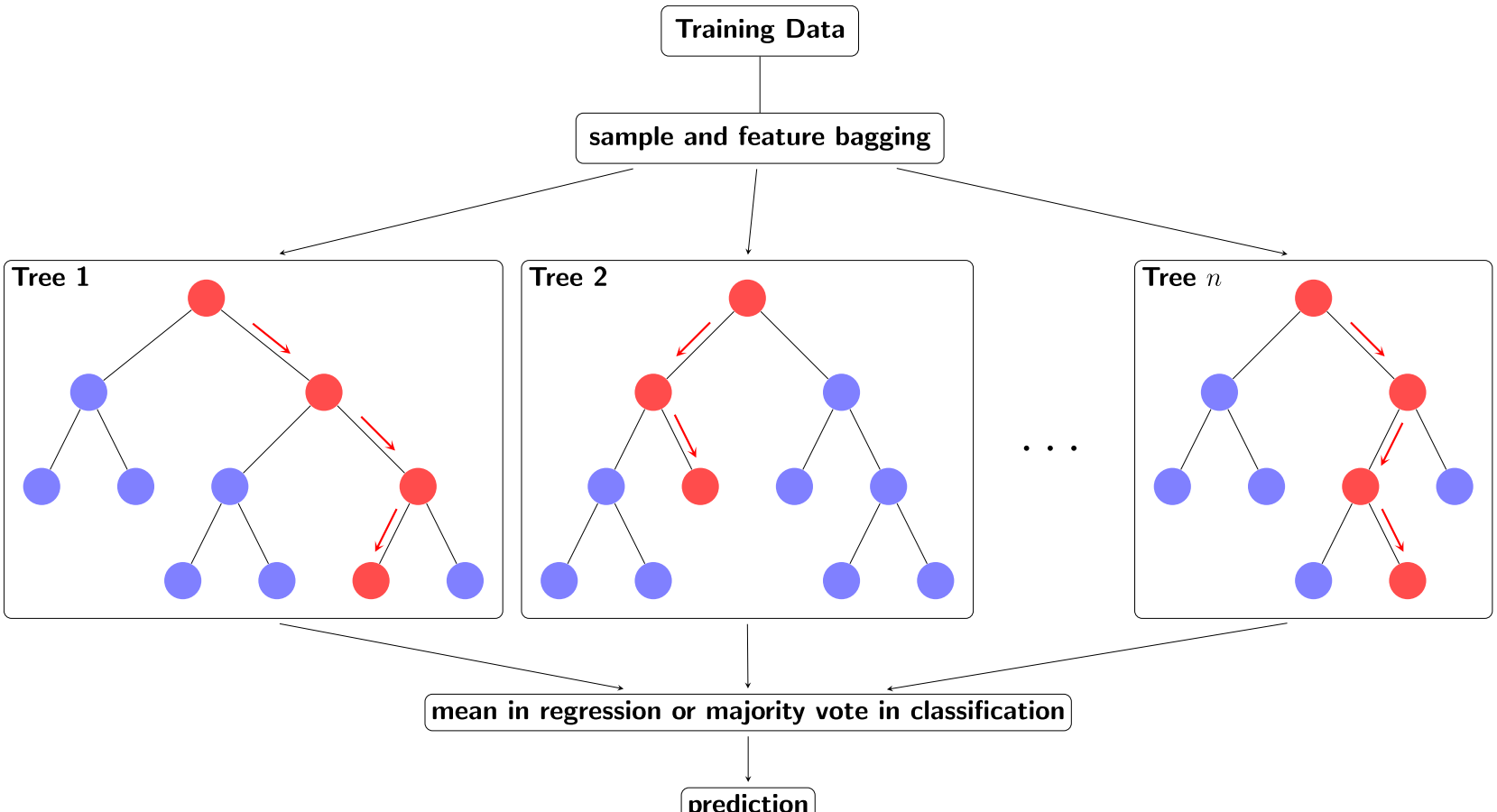


Figure 7.5, page 257 of Deep Learning Book, http://deeplearningbook.org

Random Forests

Bagging of data combined with random subset of features (sometimes called *feature bagging*).



https://tex.stackexchange.com/questions/503883/illustrating-the-random-forest-algorithm-in-tikz

Random Subset of Features

During each node split, only a random subset of features is considered when finding a best split. A fresh random subset is used for every node.

Extra Trees

The so-called extra trees are even more randomized, not finding the best possible feature value when choosing a split, but considering only boundaries with a uniform distribution within a feature's empirical range (minimum and maximum in the training data).

Gradient Boosting

$$y(\boldsymbol{x}_i) = \sum_k f_k(\boldsymbol{x}_i; \boldsymbol{W}_k)$$

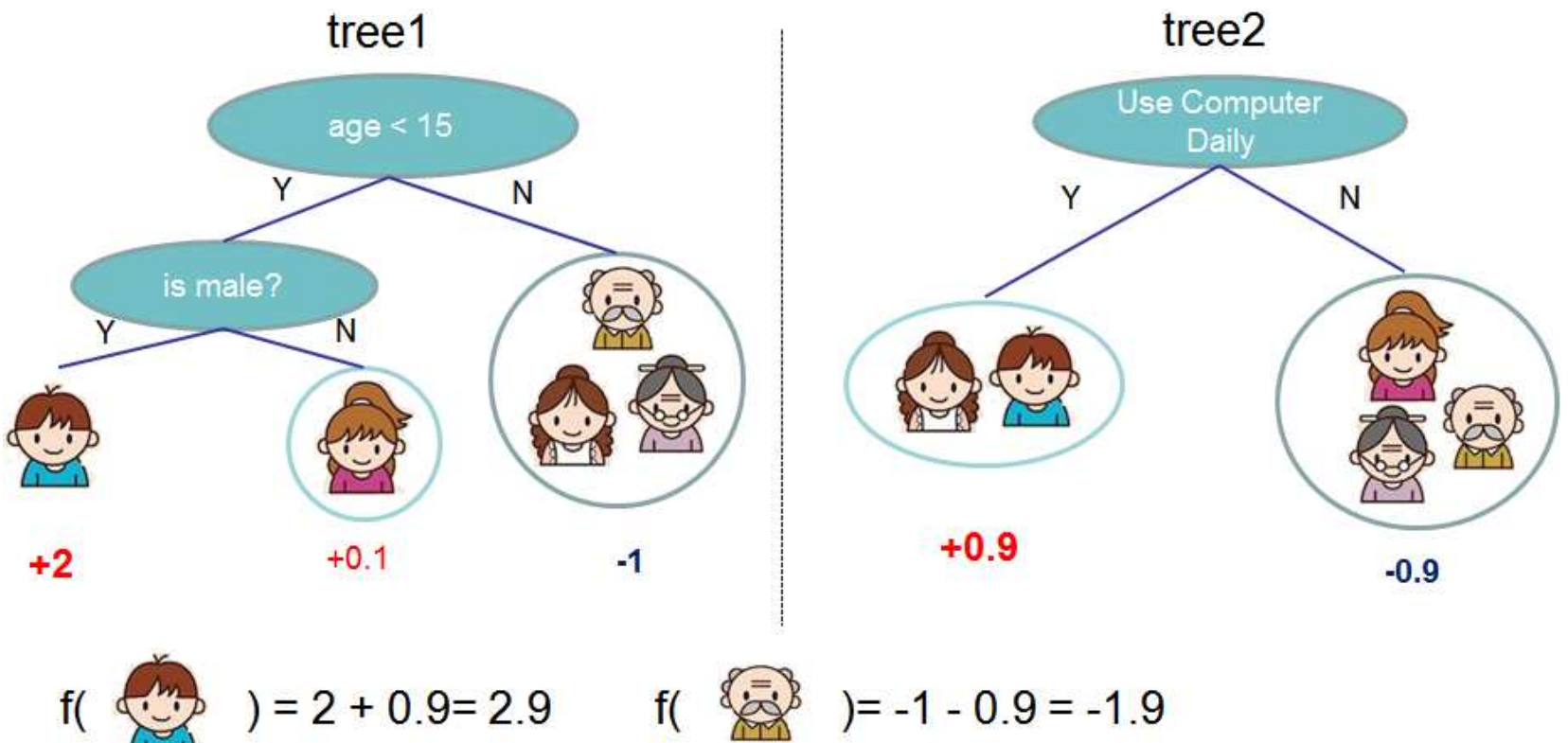


Figure 1 of the paper "XGBoost: A Scalable Tree Boosting System", https://arxiv.org/abs/1603.02754

Gradient Boosting

$$\mathcal{L}(\boldsymbol{W}) = \sum_i \ell\big(t_i, y(\boldsymbol{x}_i)\big) + \sum_k \frac{1}{2}\lambda||\boldsymbol{W}_k||^2$$

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

The original idea was to set $f_t(\boldsymbol{x}_i) \approx -\frac{\partial \ell\left(t_i, y^{(t-1)}(\boldsymbol{x}_i)\right)}{\partial y^{(t-1)}(\boldsymbol{x}_i)}$ as a direction minimizing the residual loss and then finding a suitable constant γ_t so that $\sum_i \ell\big(t_i, y^{(t-1)} + \gamma_t f_t(\boldsymbol{x}_i)\big)$ is as small as possible.

Gradient Boosting

However, a more principled approach was suggested later.

Denoting

$$g_i = \frac{\partial \ell\left(t_i, y^{(t-1)}(\boldsymbol{x}_i)\right)}{\partial y^{(t-1)}(\boldsymbol{x}_i)}$$
$$h_i = \frac{\partial^2 \ell\left(t_i, y^{(t-1)}(\boldsymbol{x}_i)\right)}{\partial y^{(t-1)}(\boldsymbol{x}_i)^2}$$

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

$$\mathcal{L}^{(t)}(\boldsymbol{W}) \approx \sum_i \left[\ell\left(t_i, y^{(t-1)}\right) + g_i f_t(\boldsymbol{x}_i) + \frac{1}{2} h_i f_t^2(\boldsymbol{x}_i)\right] + \frac{1}{2}\lambda ||\boldsymbol{W}_t||^2.$$

Gradient Boosting

We recall that we denote 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

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

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

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

Substituting the optimum weights to the loss, we get

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

which can be used as a splitting criterion.

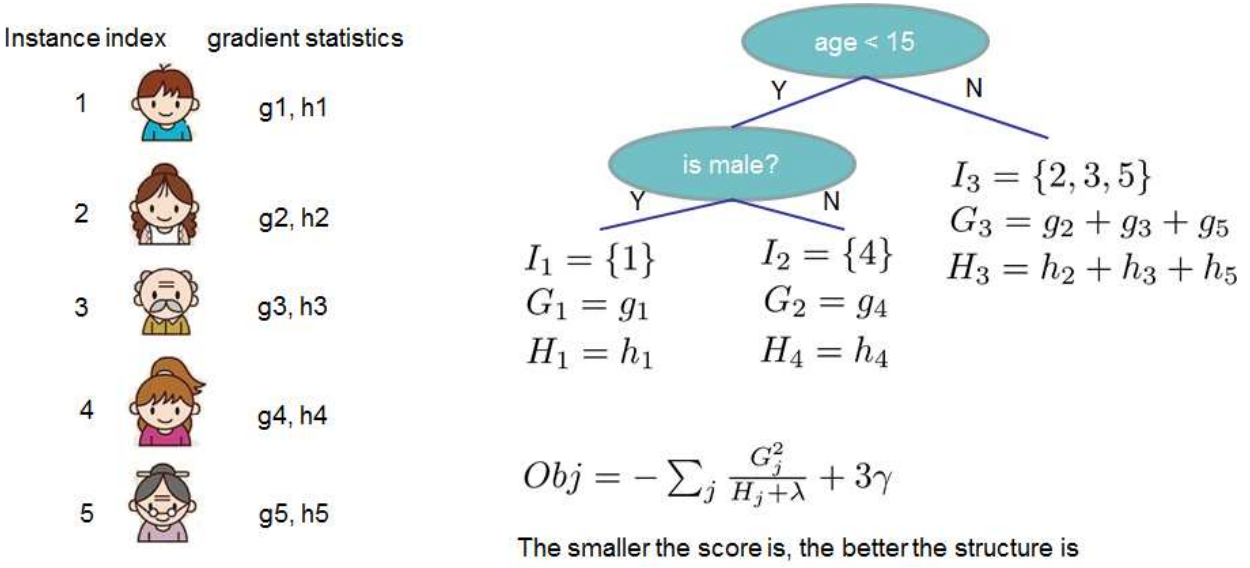


Figure 2 of the paper "XGBoost: A Scalable Tree Boosting System", https://arxiv.org/abs/1603.02754

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

Furthermore, gradient boosted trees frequently use:

- data subsampling: either bagging, or use only a fraction of the original training;
- 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.

Implementations

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

- XGBoost
- LightGBM

Playground

You can explore the Gradient Boosted Trees playground.