Decision Trees, Random Forests

Jindřich Libovický (reusing materials by Milan Straka)

November 28, 2023

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

Today's Lecture Objectives

After this lecture you should be able to

- Implement Decision Trees and Random Forests for classification and regression
- Explain how the splitting criterion depend on optimized loss function
- Tell how Random Forests differ from Gradient Boosted Decision Trees

Decision Trees

The idea of decision trees is to partition the input space into 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, …

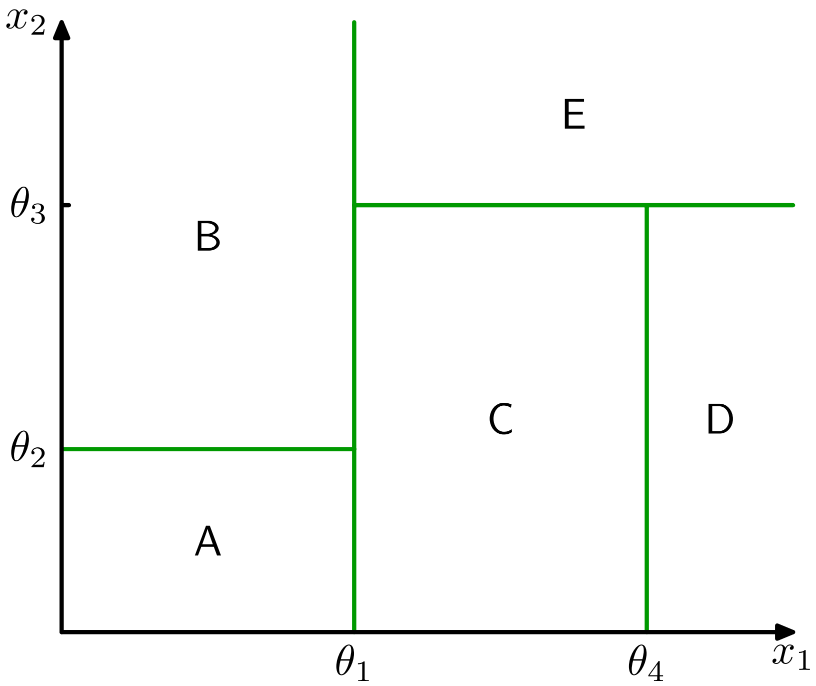


Figure 14.6 of Pattern Recognition and Machine Learning.

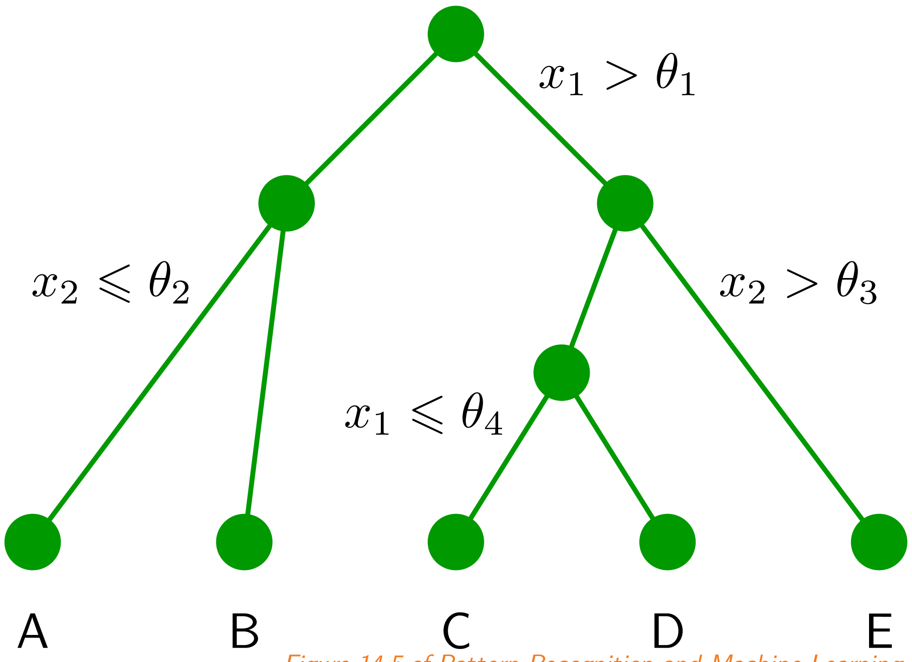


Figure 14.5 of Pattern Recognition and Machine Learning.

Inference

- Just follow the branching rules until you reach a leaf.
- Output a prediction (real value/distribution/predicted class) based on the leaf.

Training

- Training data is stored in tree leaves -- the leaf prediction is based on what is data items are in the leaf.
- At the beginning the tree is a single leaf node.
- Adding a node = leaf $\rightarrow$ decision node + 2 leafs
- The goal of training = finding the most consistent leafs for the prediction

https://medium.com/analytics-vidhya/decision-trees-explained-in-simple-steps-39ee1a6b00a2

Later, we will show that the consistency measures follow from the loss function, we are optimizing.

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 node $\mathcal{T}$.

For each leaf (a node without children), our model predicts 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 use a **criterion** $c_\mathcal{T}$ telling us how *uniform* or *homogeneous* the training examples of a node $\mathcal{T}$ are – for regression, we employ the sum of squares error between the examples belonging to the node and the predicted value in that node; this is proportional to the variance of the training examples belonging to the node $\mathcal{T}$, multiplied by the number of the examples. Note that even if it is not *mean* squared error, it is sometimes denoted as MSE.

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

To split a node, the goal is to find

1. A feature and (i.e., a for loop over all features)
2. Its value (i.e., a for loop over all unique feature values)

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.

Tree Construction: Heuristics

We usually employ several constraints, the most common ones are:

- **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**: we split until we reach the given number of leaves.

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 one of the above constraints is invalidated;
- if the maximum number of leaf nodes is given, 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.

Terminological note: Decision tree with unlimited size can be considered a non-parametric model: it is a way of building an index. With a limited size, it has a fixed number of parameters to be learned and it can be considered a parametric model.

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

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

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

- **Gini index**, also called **Gini impurity**, measuring how often a randomly chosen element would be incorrectly labeled if it was randomly labeled according to $\boldsymbol{p}_{\mathcal{T}}$:

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

- **Entropy Criterion**

$$c_{\mathrm{entropy}}(\mathcal{T}) \stackrel{\mathrm{def}}{=} |I_{\mathcal{T}}| \cdot H(\boldsymbol{p}_{\mathcal{T}}) = -|I_{\mathcal{T}}| \sum_{\substack{k \\ p_{\mathcal{T}}(k) \neq 0}} p_{\mathcal{T}}(k) \log p_{\mathcal{T}}(k).$$

From Loss Function to Splitting Criterion

- Training GLMs and MLPs is formulated as optimizing a loss function.
- For an already constructed decision tree, we can do it the same way. For each leaf, do the optimization and find the best parameter.
- So far, we were always interested in $\arg\min$, i.e., parameters that minimize the loss.
- If we plug the $\arg\min$ value in the loss function, we get the minimum reachable loss for the given tree structure.
- By splitting a leaf, we want to decrease the minimum reachable loss $\Rightarrow$ the **minimum node loss is the splitting criterion**.

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 $p_\mathcal{T} = \frac{1}{|I_\mathcal{T}|}\sum_{i \in I_\mathcal{T}} t_i = \frac{n_\mathcal{T}(1)}{n_\mathcal{T}(0)+n_\mathcal{T}(1)}$.

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

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

The value of the loss is then

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

Entropy as NLL Loss

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

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

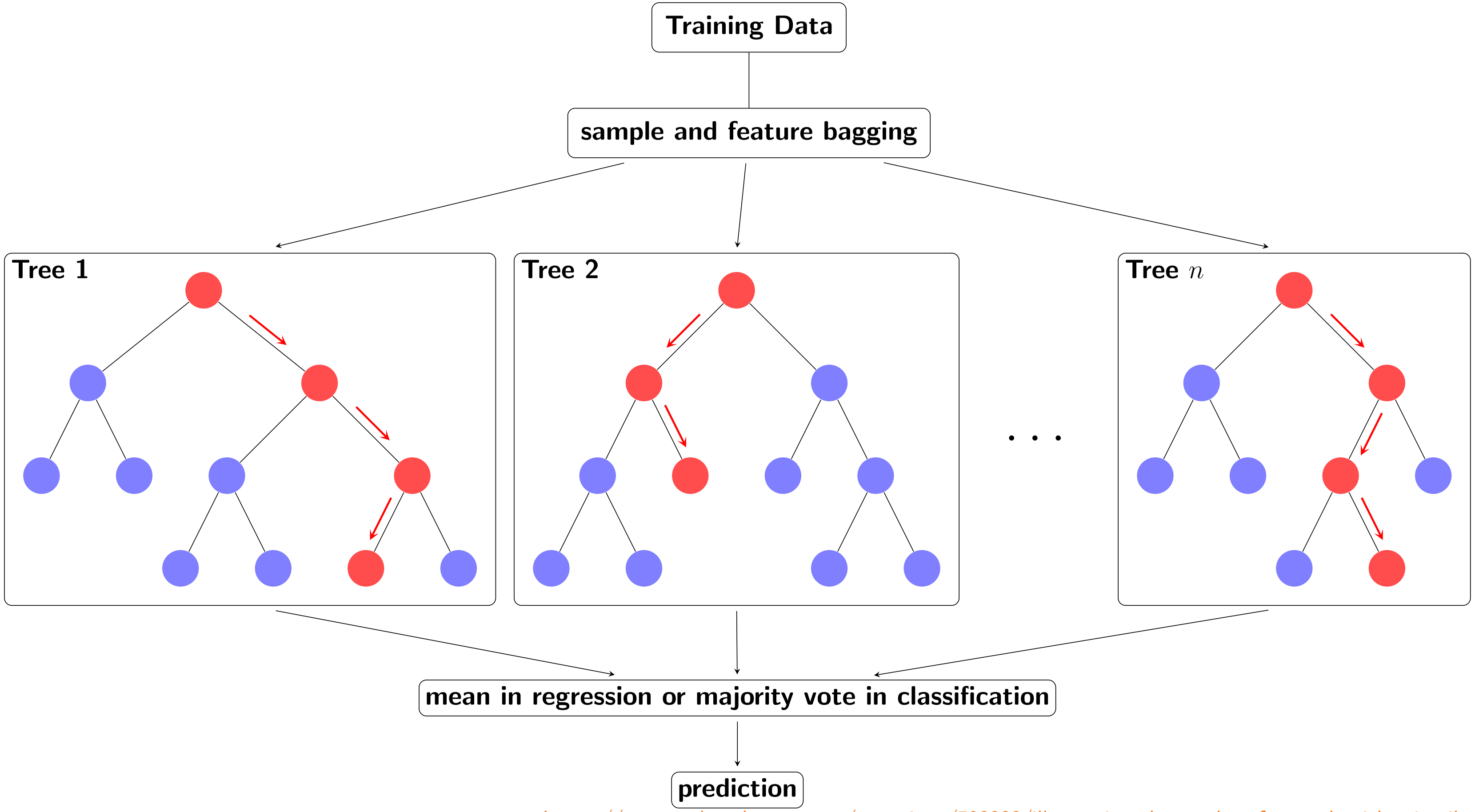
By setting the derivative of the loss with respect to p_k to zero (using a Lagrangian with constraint $\sum_k p_k = 1$), we get that the $\boldsymbol{p}$ minimizing the loss fulfills $p_k = p_\mathcal{T}(k)$.

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

$$\begin{aligned} L(\boldsymbol{p}_\mathcal{T}) &= \sum_{i \in I_\mathcal{T}} -\log p_{t_i} \\ &= - \sum_{\substack{k \\ p_\mathcal{T}(k) \neq 0}} n_\mathcal{T}(k) \log p_\mathcal{T}(k) \\ &= -|I_\mathcal{T}| \sum_{\substack{k \\ p_\mathcal{T}(k) \neq 0}} p_\mathcal{T}(k) \log p_\mathcal{T}(k) = |I_\mathcal{T}| \cdot H(\boldsymbol{p}_\mathcal{T}). \end{aligned}$$

Random Forests

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



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

Bagging

Every decision tree is trained using bagging (on a bootstrapped dataset).

Random Subset of Features

During each node split, only a random subset of features is considered when finding the 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 uniformly random samples from a feature's empirical range (minimum and maximum in the training data).

Demo

https://cs.stanford.edu/~karpathy/svmjs/demo/demoforest.html

- A collection of decision trees: each tree fixes the error of the previous trees
- Decreasing error = step in the direction of the error/loss gradient
- GBDT repeat SGD computation at the inference time (start with a stupid estimate and do gradient steps that improve the estimate)
- One tree is one gradient descent step

Theoretical steps to be taken

- Derive what exactly we want the intermediate steps to predict, so we do more clever steps than SGD that just moves with gradient
- Based on that, express the per-node loss and splitting criterion as the minimum reachable loss value

For regression, binary classification and multiclass classification.

GBDT 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.

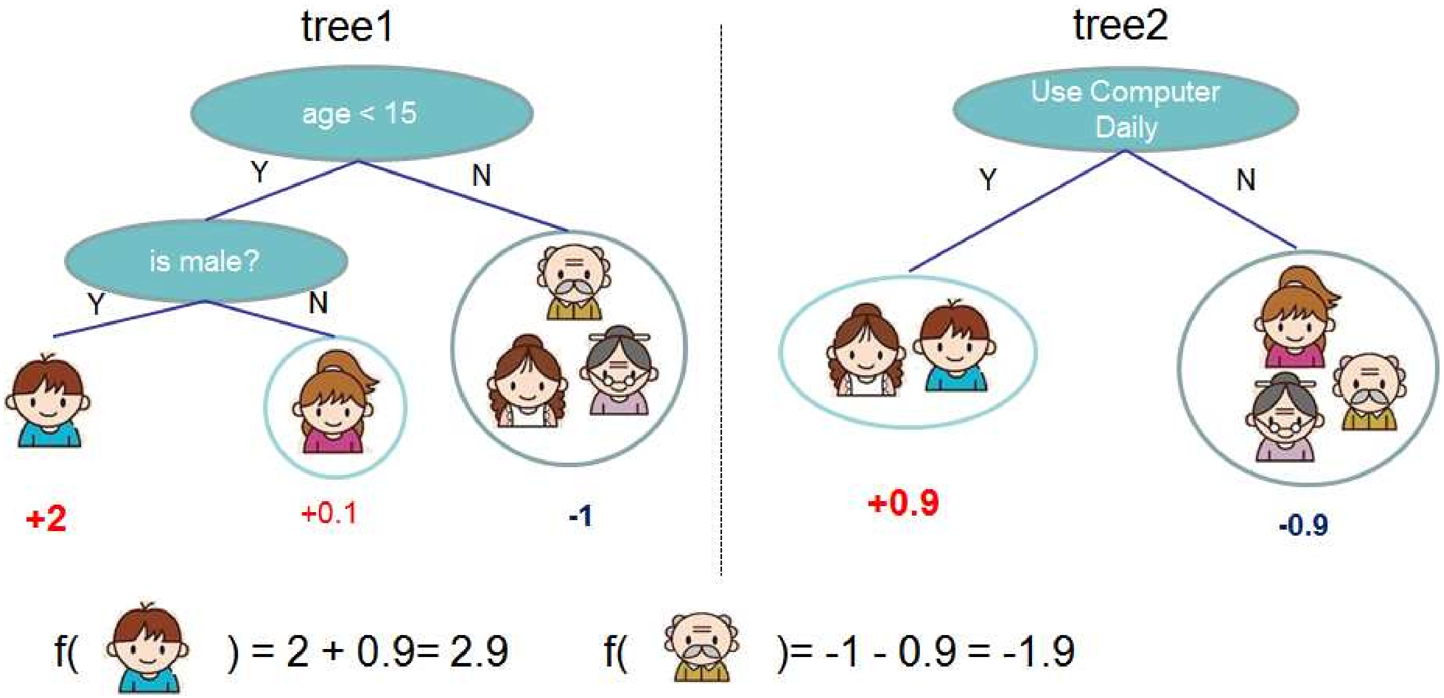


Figure 1 of "XGBoost: A Scalable Tree Boosting System", https://arxiv.org/abs/1603.02754

Gradient Boosting for Regression

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

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

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

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

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

$$y_t(\boldsymbol{x}_i) \leftarrow -\frac{\partial\ell\big(t_i, y^{(t-1)}(\boldsymbol{x}_i)\big)}{\partial y^{(t-1)}(\boldsymbol{x}_i)} = -\frac{\partial\ell\big(t_i, y\big)}{\partial y}\bigg|_{y=y^{(t-1)}(\boldsymbol{x}_i)}$$

as a direction minimizing the residual loss and then finding a suitable constant γ_t, which would minimize the loss

$$\sum_i \Big[\ell\big(t_i, y^{(t-1)}(\boldsymbol{x}_i) + \gamma_t y_t(\boldsymbol{x}_i)\big)\Big] + \frac{1}{2}\lambda\big\|\boldsymbol{w}_t\big\|^2.$$

Today's Lecture Objectives

After this lecture you should be able to

- Implement Decision Trees and Random Forests for classification and regression
- Explain how the splitting criterion depend on optimized loss function
- Tell how Random Forests differ from Gradient Boosted Decision Trees