# UNIT 4

## Random Forests and Gradient Boosting

**Johannes Kofler**

JOHANNES KEPLER
UNIVERSITY LINZ

Institute for
Machine Learning

# Copyright statement:

# Lecture Supervised Techniques: Planned Topics

- UNIT 1: Overview of Supervised Machine Learning
- UNIT 2: Basics of Supervised Machine Learning
- UNIT 3: Support Vector Machines
- **UNIT 4: Random Forests and Gradient Boosting**
- UNIT 5: Logistic Regression
- UNIT 6: Artificial Neural Networks
- UNIT 7: Special Network Architectures

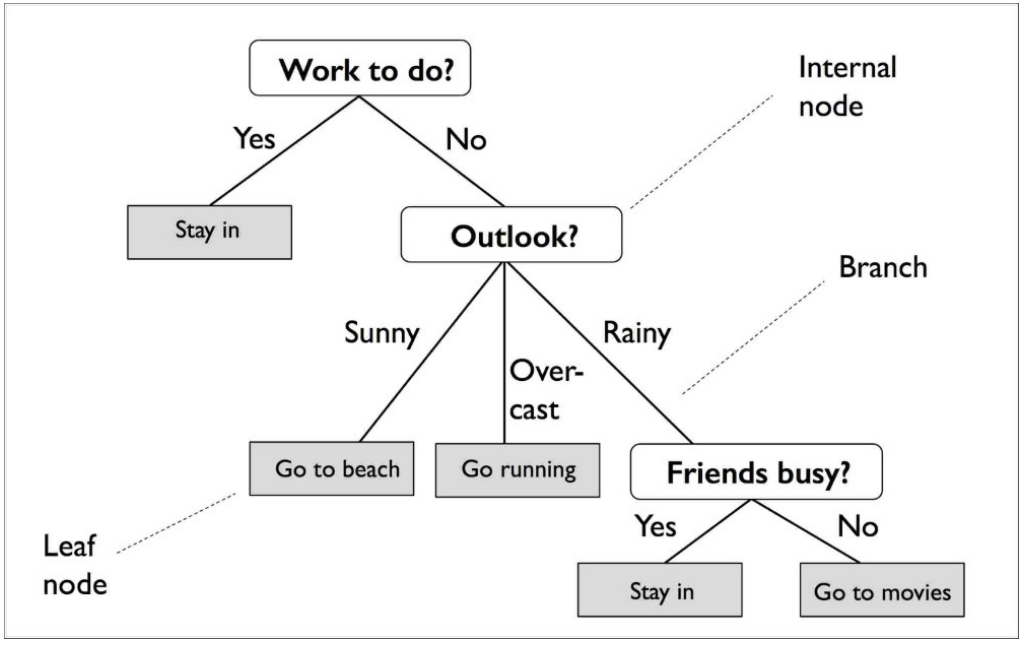
# Planned topics for Unit 4

- Basics of classification and regression trees: Definitions and intuition
- Random forests
- Gradient boosting and tree boosting

# Main steps of historical development

- 1984 resp. 1993: Decision trees: modern approaches by Breiman et al. and Quinlan: original idea way earlier
- 2001: Random forests: Breiman
- 1999-2001: Gradient boosting methods: mostly by Friedman

# Example of Decision Tree




Picture source: Towards Data Science

- Root node on top
- Branches
- Internal nodes
- Leaf nodes

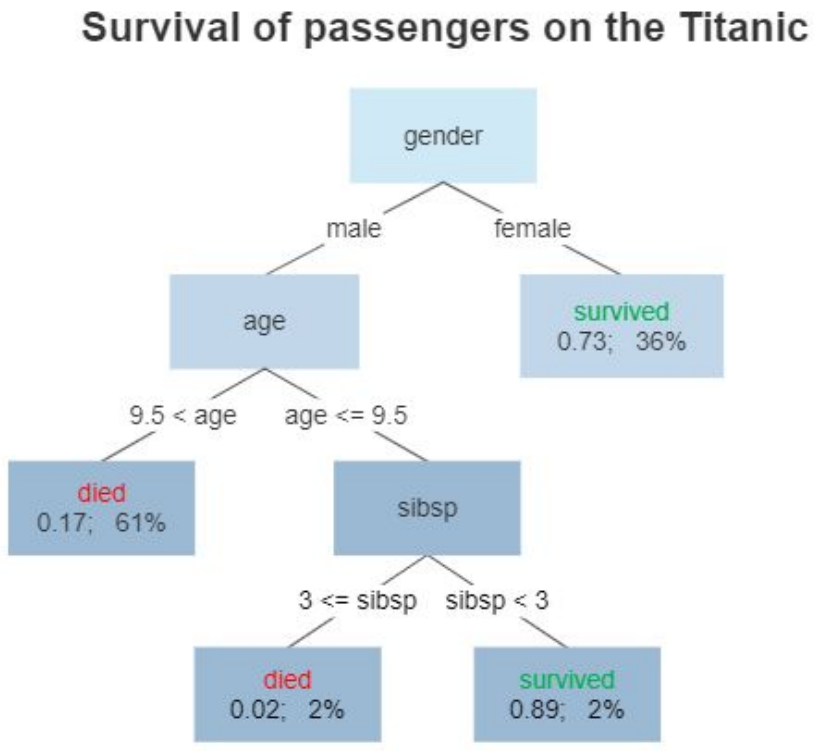
# Decision Trees: Introduction

- Decision tree: predictive model to go from observations about an item (represented in the branches) to conclusions about the item's target value (represented in the leaves).
- Decision tree classifies samples "by asking questions successively"; each non-leaf node corresponds to a question, each leaf corresponds to a final prediction.
- Decision tree learning partitions training data hierarchically such that leaf nodes are hopefully homogeneous in terms of the target class.
- Mainly designed for categorical data: classification trees (binary and multi-class)
- Can also be applied to numerical features: regression trees.

# Decision tree learning: Part 1

- All decision tree learning algorithms are recursive, depth-first search algorithms that perform hierarchical splits.
- There are three main design issues:
  - Splitting criterion: which splits to choose
  - Stopping criterion: when to stop further growing of the tree
  - Pruning: whether/how to collapse unnecessarily deep sub-trees
- The two latter: relevant for adjusting the complexity of decision trees (underfitting vs. overfitting).

# Example




Picture source: Wikipedia

- The figures under the leaves show the probability of survival and the percentage of observations in the leaf. Class label “died”/“survived” determined by sample majority in that leaf.
- “sibsp” is the number of spouses or siblings aboard.
- Summarizing: Chances of survival were good if you were (i) a female or (ii) a male younger than 9.5 years with strictly less than 3 siblings.

# Decision tree learning: Part 2: Recursive procedure

- Given:
  - Training set $\mathbf{Z} = \{(\mathbf{x}_i, y_i) \mid i = 1, \ldots, l\}$
  - Stopping criterion
  - Splitting criterion
- Call DecTree($\mathbf{Z}$, Root node, $S = \{$all possible splits$\}$)
- DecTree($\mathbf{Z}$, $N$, $S$)
  - If stopping criterion is fulfilled, exit.
  - Determine split $s \in S$ such that splitting criterion is maximal.
  - Divide $\mathbf{Z}$ into disjoint subsets $\mathbf{Z}_{s,t}$ according to split $s$ and result $t$.
  - For all $t$ such that $\mathbf{Z}_{s,t} \neq \emptyset$
    - Generate new node $N_t$
    - Call DecTree($\mathbf{Z}_{s,t}$, $N_t$, $S \backslash \{s\}$)

# Splitting: categorical vs. numerical features

- Categorical features:
  - Binary split:
    $\mathbf{Z}_L = \{(\mathbf{x}, y) \in \mathbf{Z} \mid x_i = c\}, \mathbf{Z}_R = \{(\mathbf{x}, y) \in \mathbf{Z} \mid x_i \neq c\}$
  - Split according to entire feature: $\mathbf{Z}_j = \{(\mathbf{x}, y) \in \mathbf{Z} \mid x_i = c_j\}$
  - Typically, all possible (binary or entire feature) splits w.r.t. all features are considered.
- Numerical features:
  - Apply threshold $c$ to $i$-th feature, i.e.
    $$\mathbf{Z}_L = \{(\mathbf{x}, y) \mid x_i < c\}, \mathbf{Z}_R = \{(\mathbf{x}, y) \mid x_i \geq c\}.$$
  - Typically, all possible splits w.r.t. all features are considered
  - Thresholds are chosen as mean values of “neighboring” values occurring in the data.

# Common splitting criteria

- Classification:
  - Information gain
  - Gini impurity
- Regression:
  - Variance reduction
- Nowadays, mostly Gini impurity and variance reduction are used.

# Information gain

- For any (sub)set of data $\mathbf{Z}$, the relative proportions of samples belonging to the $k$-th class (of classes $1, \ldots, M$) are defined as:

$$p_k(\mathbf{Z}) = \frac{|\{(\mathbf{x},y) \in \mathbf{Z} | y = k\}|}{|\mathbf{Z}|}.$$

- The entropy of $\mathbf{Z}$ w.r.t. the target is defined as

$$H(\mathbf{Z}) = -\sum_{k=1}^{M} p_k(\mathbf{Z}) \ln p_k(\mathbf{Z}).$$

  - Maximal ($\ln M$) if classes are uniformly distrib. in the set $\mathbf{Z}$.
  - Minimal ($0$) if all samples of $\mathbf{Z}$ belong to one single class.
  - The smaller/larger the entropy the larger/smaller the information.

- Information gain (Kullback-Leibler divergence) of employing the $s$-th split for partitioning $\mathbf{Z}$ into sets $\mathbf{Z}_{s,t=1}, \ldots, \mathbf{Z}_{s,t=K_s}$ is then defined as

$$g_E(\mathbf{Z}, s) = H(\mathbf{Z}) - \sum_{t=1}^{K_s} \frac{|\mathbf{Z}_{s,t}|}{|\mathbf{Z}|} H(\mathbf{Z}_{s,t}).$$

- $g_E$ is maximized if subset entropies $H(\mathbf{Z}_{s,t})$ are minimized, i.e. subsets should be as homogeneous as possible (limiting case: only one class).
- Typically, in each step, all possible splits are considered and the one with highest information gain is selected.

# Gini impurity

- With the notation from above: Gini impurity of $\mathbf{Z}$ is defined as

$$I_G(\mathbf{Z}) = \sum_{k=1}^{M} p_k(\mathbf{Z})\big(1 - p_k(\mathbf{Z})\big) = 1 - \sum_{k=1}^{M} p_k^2(\mathbf{Z})$$

- Interpretation: gives probability of incorrectly classifying randomly chosen element in dataset if it were randomly labeled according to the class distribution in the dataset.
- Value is:
  - Maximal $(1 - 1/M)$ if classes are uniformly distributed in the set $\mathbf{Z}$.
  - Minimal $(0)$ if all samples of $\mathbf{Z}$ belong to one single class.
- Gini purity gain (impurity decrease) of employing the $s$-th split for partitioning $\mathbf{Z}$ into sets $\mathbf{Z}_{s,t=1}, \ldots, \mathbf{Z}_{s,t=K_s}$ is then defined as:

$$g_G(\mathbf{Z}, s) = I_G(\mathbf{Z}) - \sum_{t=1}^{K_s} \frac{|\mathbf{Z}_{s,t}|}{|\mathbf{Z}|} I_G(\mathbf{Z}_{s,t}).$$

- $g_V$ is maximized if subset Gini impurities $I_G(\mathbf{Z}_{s,t})$ are minimized, i.e. subsets should be as homogeneous as possible.
- Standard splitting criterion employed by the decision tree algorithm CART for classification.

# Variance reduction

- For regression, we need an impurity metric that is suitable for continuous variables.
- For any (sub)set of data $\mathbf{Z} = \{(\mathbf{x}_i, y_i) \mid i = 1, \ldots, l\}$, let us denote the vector of target values as $\mathbf{y}(\mathbf{Z}) = (y_1, \ldots, y_l)$.
- The reduction of variance of employing the $s$-th split for partitioning $\mathbf{Z}$ into sets $\mathbf{Z}_{s,t=1}, \ldots, \mathbf{Z}_{s,t=K_s}$ is then defined as

$$g_V(\mathbf{Z}, s) = \mathrm{Var}\big[\mathbf{y}(\mathbf{Z})\big] - \sum_{t=1}^{K_s} \mathrm{Var}\big[\mathbf{y}(\mathbf{Z}_{s,t})\big].$$

- $g_V$ is maximized if subset variances $\mathrm{Var}\big[\mathbf{y}(\mathbf{Z}_{s,t})\big]$ are minimized, i.e. subsets should be as homogeneous as possible.
- Standard splitting criterion employed by the decision tree algorithm CART for regression.

# Stopping criteria and pruning

- When to stop:
  - Maximum depth reached
  - Minimum number of samples in current node under limit
  - Splitting gain below limit
- Pruning:
  - Reduced error pruning:
    - consider each node for pruning
    - pruning = removing the subtree at that node, make it a leaf and assign the most common class at that node
    - a node is removed if the resulting tree performs no worse than the original on the validation set
    - pruning continues until further pruning is harmful
    - uses training, validation and test sets; effective approach if a large amount of data is available

# Stopping criteria and pruning (cont.)

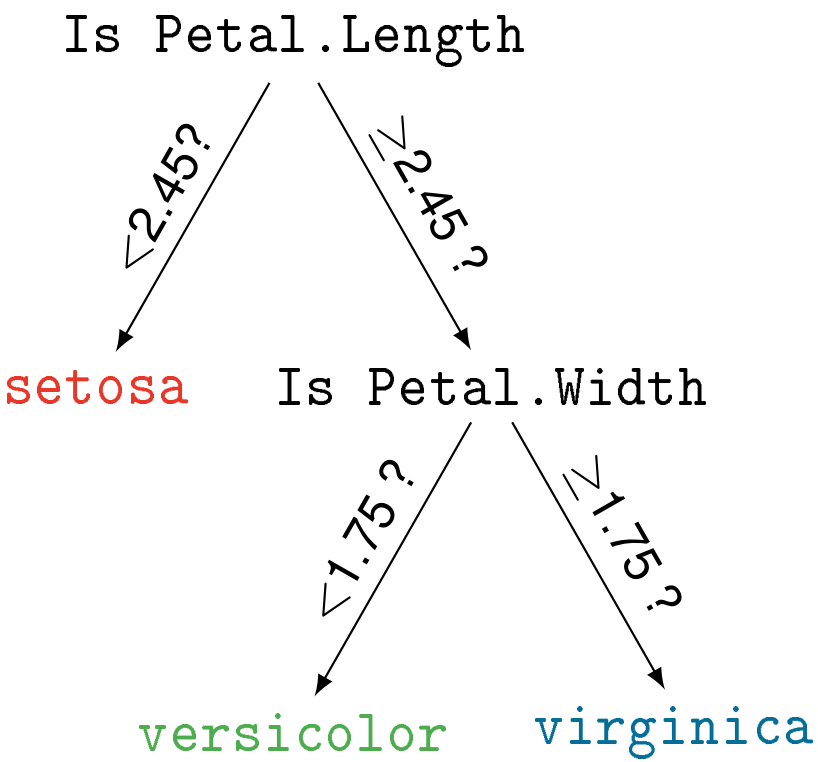
- Pruning:
  - Cost complexity pruning:
    - add complexity penalty (e.g. the number of terminal leaf nodes)
    - recursively remove sub-tree whose removal leads to smallest increase of error per removed leaf
    - repeat until root tree remains
    - of entire sequence of trees, that tree is used as final model which gives the best prediction performance (on training or validation set), i.e. best compromise between accuracy and complexity

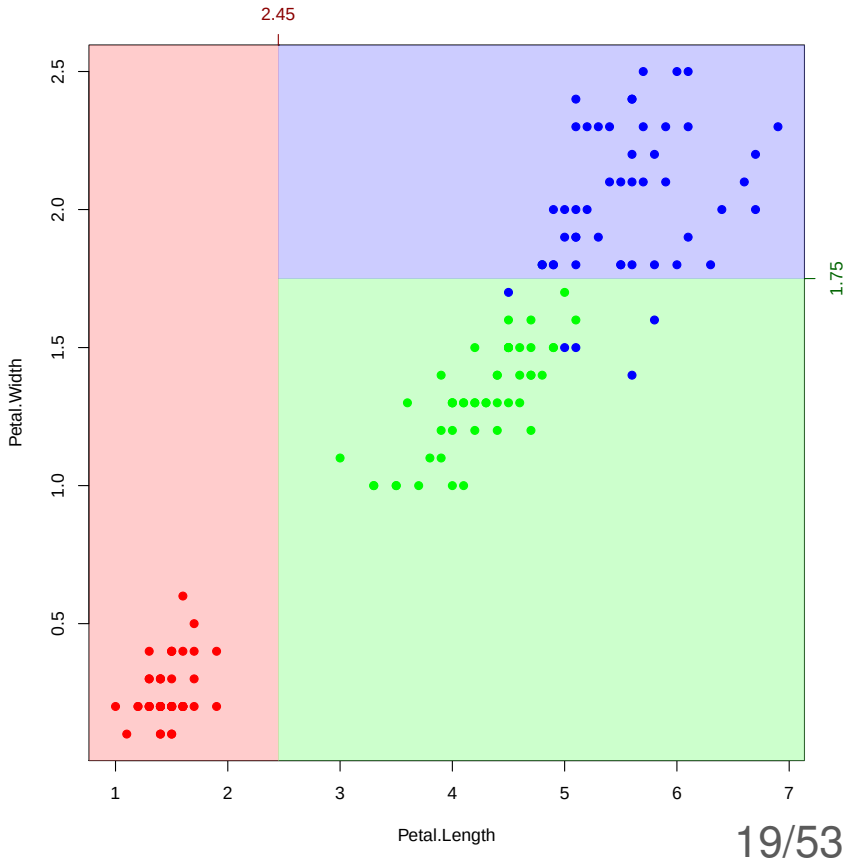
# Computing predictions

- Tree recursively partitions/splits training set into subsets, each of which is associated with leaf node.
- "Assignment" of samples to leaf nodes is the basis for making predictions with decision trees.
- For new input $\mathbf{x}$ traverse through tree by answering questions associated with each node until leaf node is reached to which sample $\mathbf{x}$ is assigned.
- Classification:
  - Assign class to leaf node that appears most prominently (relative majority) among training samples associated with this leaf node
  - Alternatively: compute relative frequencies of classes in leaf node and use frequencies as estimates of conditional probabilities $p(y = k \mid \mathbf{x})$.
- Regression: use mean target value of samples associated with leaf node.

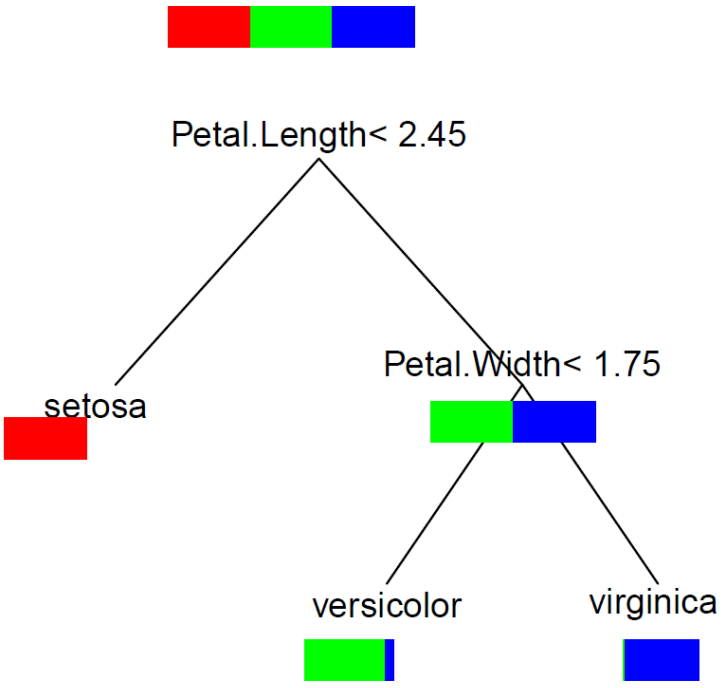
# Example: Iris Data Set: classification

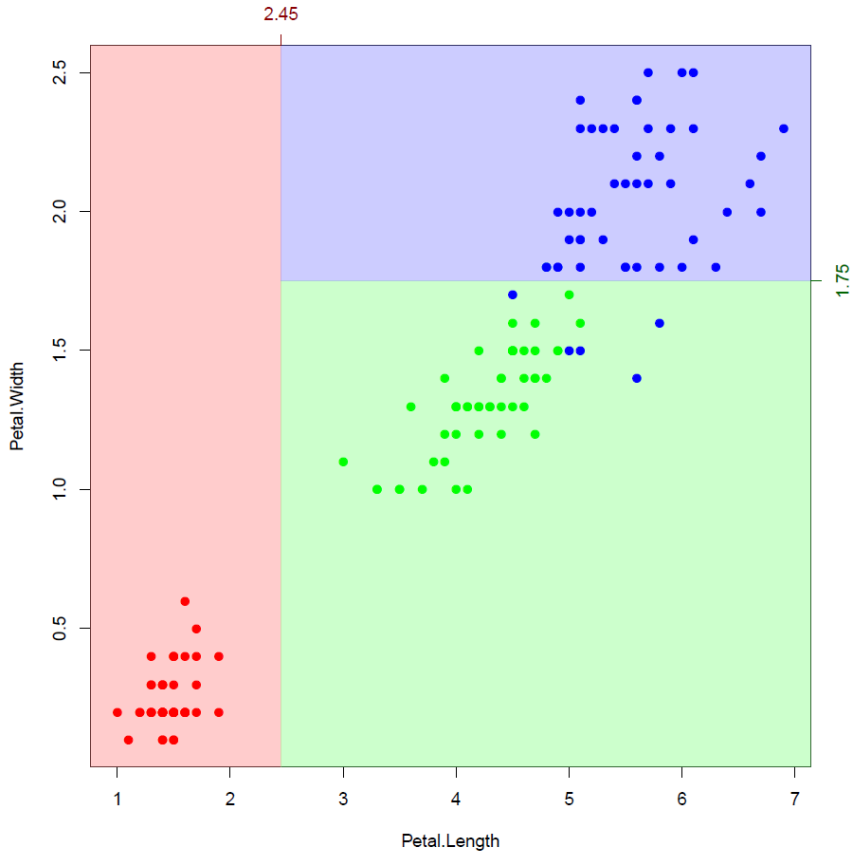
- Genus of three species of flowering plants with showy flowers:
  - Iris setosa (Beachhead Iris)
  - Iris versicolor (Larger Blue Flag, Harlequin Blueflag)
  - Iris virginica (Virginia Iris).
- Four features: length and width of sepals and petals.



# Example: Iris Data Set: classification



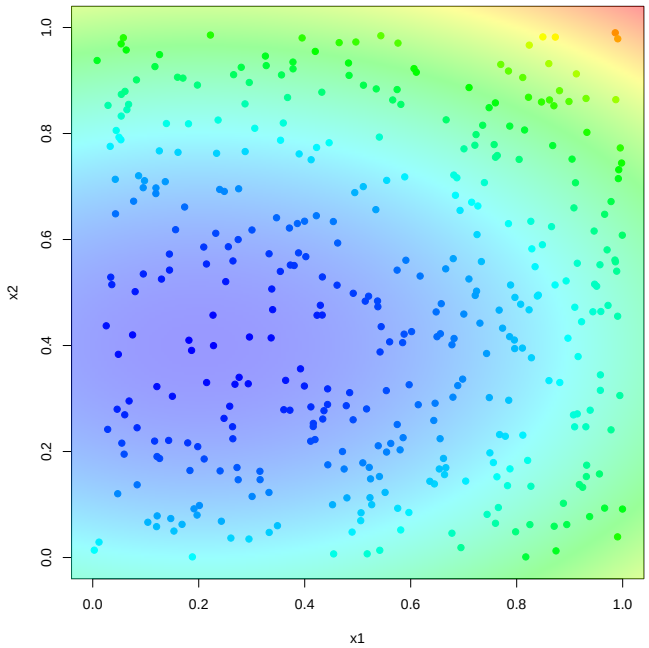



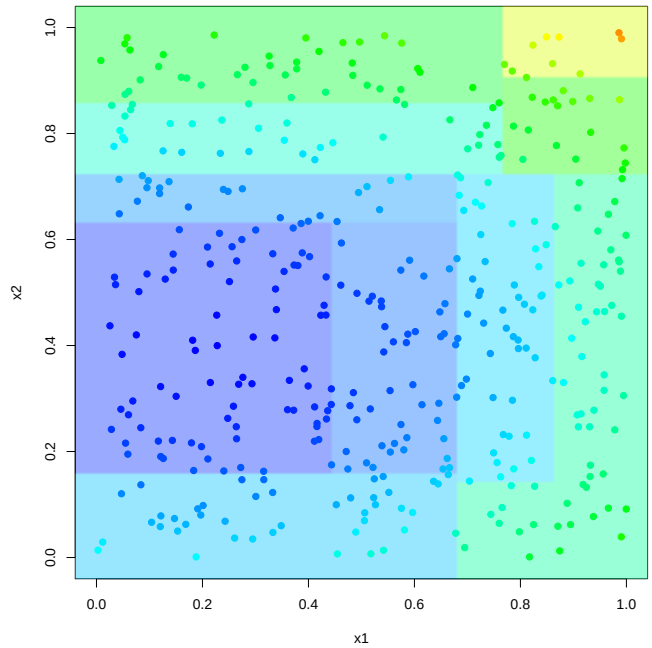
# Decision trees as rule base

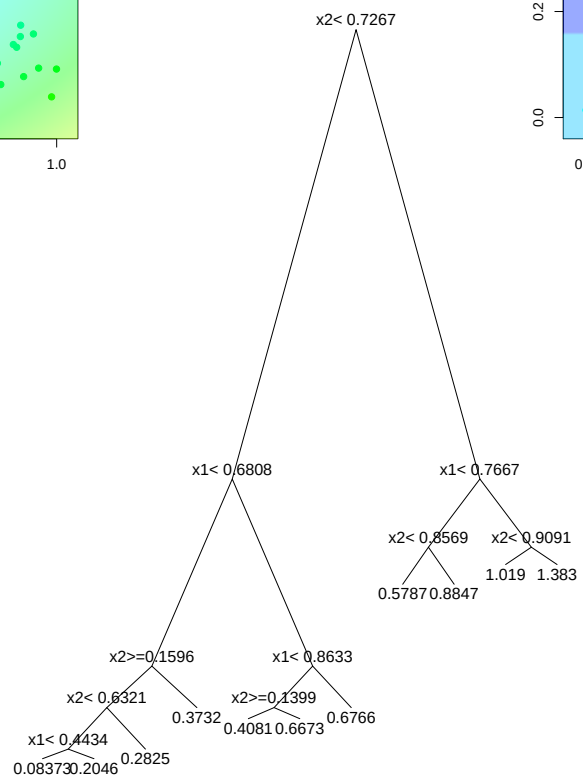
- Every path from root to leaf can be interpreted as rule, where each split corresponds to fulfillment/non-fulfillment of condition/predicate.
- Example: Decision tree from previous slide can be interpreted as follows:

| | | | |
|---|---|---|---|
| **IF** | Petal.Length $<$ 2.45 | **THEN** | Class $=$ setosa |
| **IF** | Petal.Length $\geq$ 2.45 & Petal.Width $<$ 1.75 | **THEN** | Class $=$ versicolor |
| **IF** | Petal.Length $\geq$ 2.45 & Petal.Width $\geq$ 1.75 | **THEN** | Class $=$ virginica |

# Example: regression data set



# Pros and Cons of decision trees

- Pros:
  - Simple and interpretable
  - Computationally efficient, i.e. they perform well with large data sets
  - Built-in feature selection
  - Can be applied to categorical and numerical attributes, i.e. can be used for classification and regression
  - Scaling-invariant for numerical features
- Cons:
  - Greedy splitting may lead to sub-optimal solutions
  - Only axis-parallel splits of numerical features
  - Shallow trees are not accurate (high bias), deep trees overfit (high variance); difficult to address for decision trees (stopping criteria, pruning)
  - Trees can be non-robust: a small change in training data can result in a large change in the tree and thus the final predictions

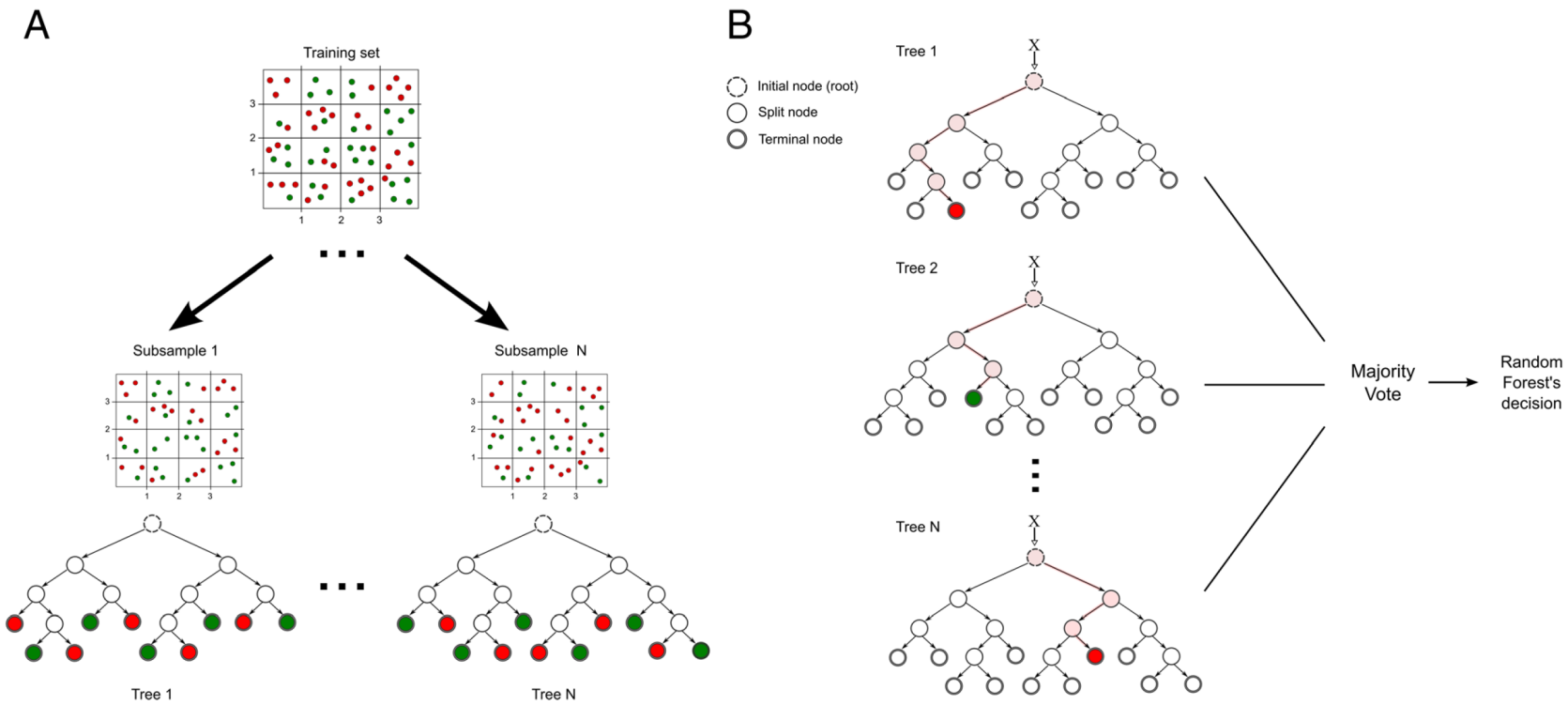
# Ensemble methods: basic ideas

- Instead of a single model, multiple models are trained.
- When making predictions, the results of these models are aggregated (e.g. averaged, voting, etc).
- Aim: Reduce
  - Variance
  - Bias
  - Both (if possible)
- We discuss two most commonly used ensemble methods for decision trees, namely:
  - Random forests (RFs); aim: reduce variance
  - Gradient tree boosting; aim: reduce bias
- We start with random forests, as they are conceptually easier.

# Random forests: most common variant

- Use CART (Classification and Regression Trees) for training single trees, i.e. binary splits with Gini purity gain (for classification) / variance reduction (for regression) as splitting criterion.
- For each tree, samples are chosen randomly from the training set (typically with replacement).
- For each split, only a sub-sample of randomly chosen features is considered.
- Trees are grown to full size and not pruned.
- Ensemble of trees = random forest. Classifcation by majority vote.

# Random forests (RFs): Part 1: Illustration




Picture source: DOI:10.1186/s13567-015-0219-7

- A: Each decision tree is built upon a random bootstrap subsample of the original data with positive (green) and negative (red) examples.
- B: Class prediction for new input using a random forest model is based on a majority voting procedure among all individual trees.

# Random forests (RFs): Part 2: Algorithm

- Training set $\mathbf{Z}$ where each element has $d$ features.
- Hyperparameters: $N$, $k$, $m$, $s_{\min}$, splitting criterion
- For $n = 1$ to $N$:
  - Draw bootstrap subsample $\mathbf{Z}^*$ of size $k$ from training data.
  - Grow a RF tree $T_n$ to bootstrapped data by recursively repeating the following steps for terminal node until minimum node size $s_{\min}$ is reached:
    1. From $d$ features select $m \leq d$
    2. Pick best variable/split among the $m$ features
    3. Split node into two daughter nodes
- Output ensemble of trees = random forest: $\{T_n\}_1^N$

To make prediction for new input $\mathbf{x}$:

- Regression: $g_{\mathrm{RF}}(\mathbf{x}) = \frac{1}{N}\sum_{n=1}^{N} T_n(\mathbf{x})$
- Classification: $g_{\mathrm{RF}}(\mathbf{x}) =$ majority vote $\{T_n(\mathbf{x})\}_1^N$.

# Random forests (RFs): Part 3: Comments

- Essential idea: Average many noisy but approximately unbiased models to reduce variance.
- Since trees are noisy, they benefit from averaging.
- Each tree is identically distributed $\rightarrow$ expectation of average of $N$ trees is same as expectation of any of them $\rightarrow$ no bias improvement.
- However, the variance of the forest can be computed as

$$\sigma^2_{\mathrm{RF}} = \rho\,\sigma^2 + \frac{1-\rho}{N}\,\sigma^2$$

  for pairwise correlation coefficient $\rho$ (between trees) and single tree variance $\sigma^2$ (see exercises).
- If $N$ increases, second term disappears, but first remains; size of correlation $\rho$ limits benefits of averaging.
- Basic idea of RFs: Reduce variance by reducing correlation between trees, without increasing individual variance too much. Achieved through random selection of features.

# More details about RFs: Part 1: Overfitting

- It can be proved (cf. Breiman) that random forests do not tend to overfit if the number of trees increases.
- This, however, does not mean that random forests cannot overfit generally. Like for any other machine learning method, training errors can be much smaller than the real generalization error (test error), especially if the number of trees is not large.

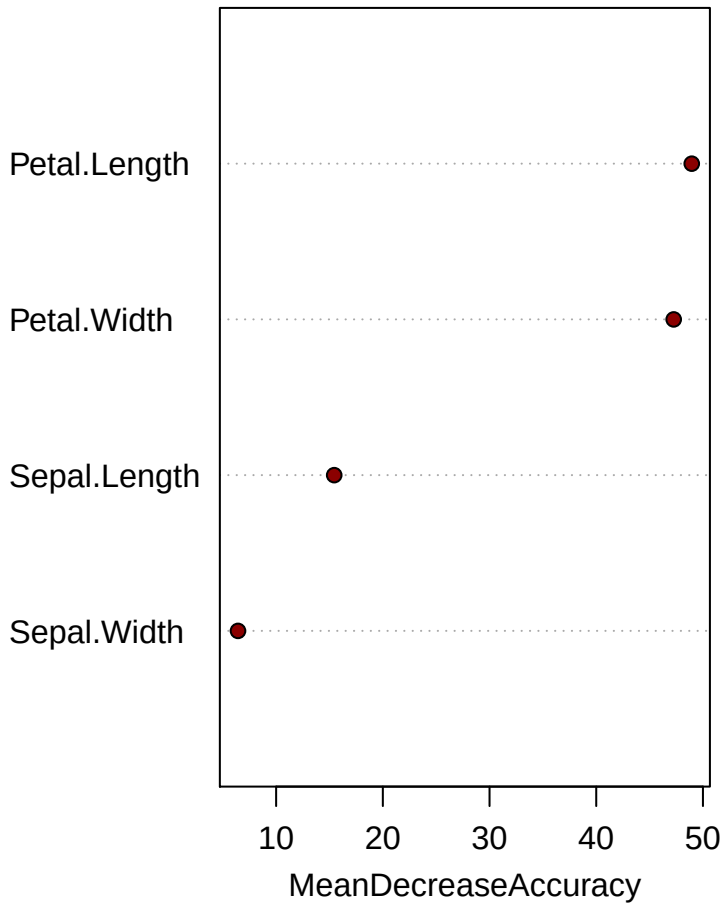
# More details about RFs: Part 2: OOB estimates

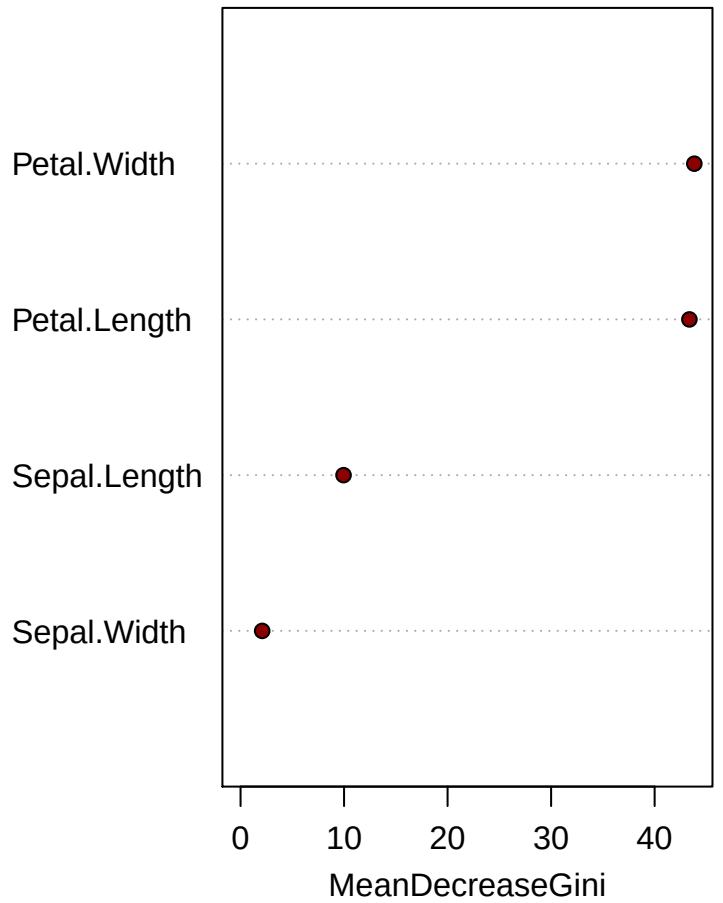
- RFs allow for assessing generalization performance on basis of training data.
- Out of bag (OOB) estimates: For each observation $\mathbf{z}_i = (\mathbf{x}_i, y_i)$ construct its RF predictor by averaging only trees corresponding to bootstrap samples in which $\mathbf{z}_i$ did not appear.
- Overall OOB error: averaging OOB errors of all samples.
- OOB error estimate is almost identical to that obtained by $N$-fold cross validation for large $N$.

# More details about RFs: Part 3: Variable importance

- Mean Gini purity gain (impurity decrease):
  - For each feature, average the Gini purity gains of all splits in all trees that involve this feature.
- Mean accuracy decrease:
  - Compute OOB error for each sample.
  - For each feature separately, consider random permutations among the data and compute the OOB errors for the data set with the permuted feature.
  - Then the importance score is computed by averaging the OOB error differences before and after permuting the feature (upon normalization by the standard deviation of the differences).

# Example: Variable importances for Iris data set (1000 trees)

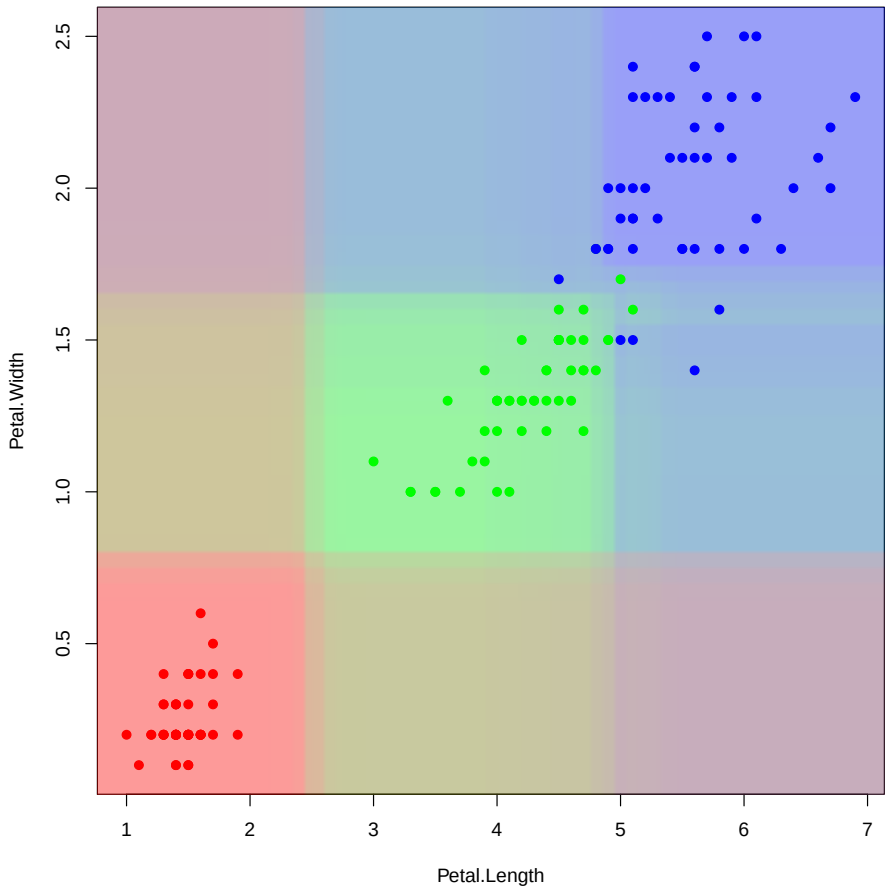


# More details about RFs: Part 4: Stratification for unbalanced class distributions

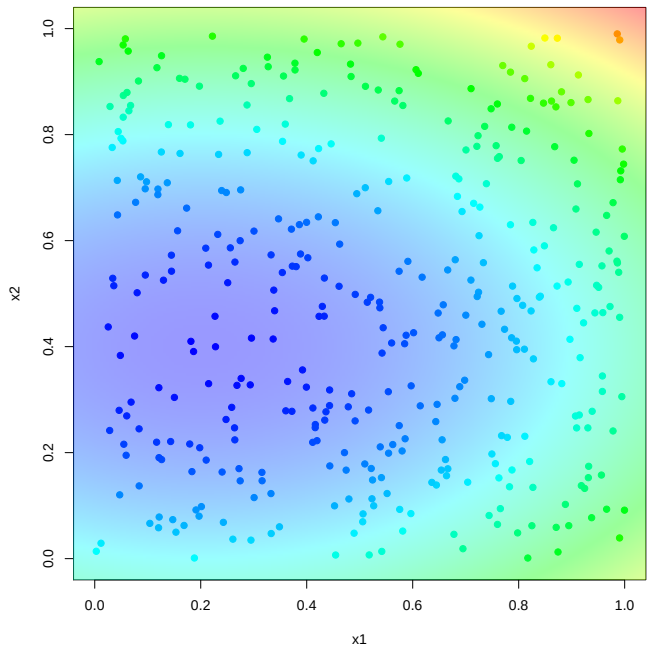
- The subsamples used for training the individual trees are as (un)balanced as the original data set.
- $\rightarrow$ Larger classes will receive more emphasis by final random forest.
- A simple strategy to equalize the importance of classes and, thereby, to improve balanced accuracy instead of standard accuracy is to stratify classes when sampling.
- Example: Suppose we have a data set with 1000 samples, 900 negative and 100 positive. Then we can always draw 50 negative and 50 positive samples for training trees.

# Example: Iris data set (1000 trees)

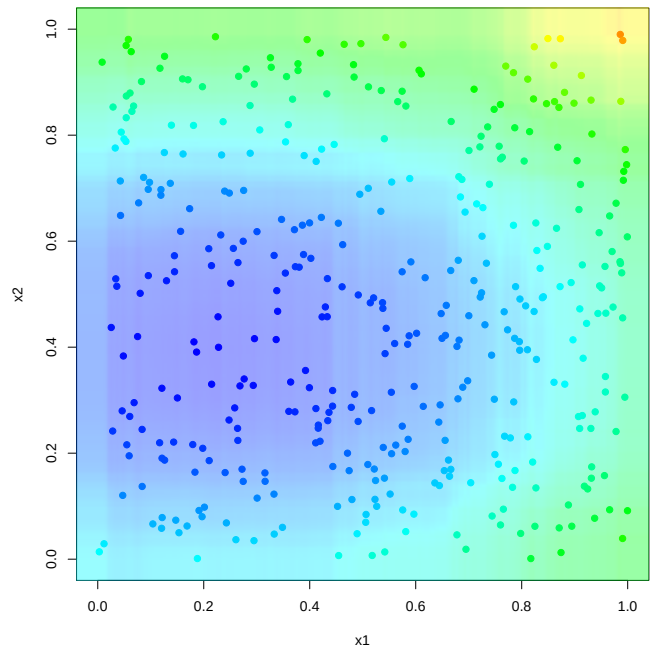


# Example: Regression data set

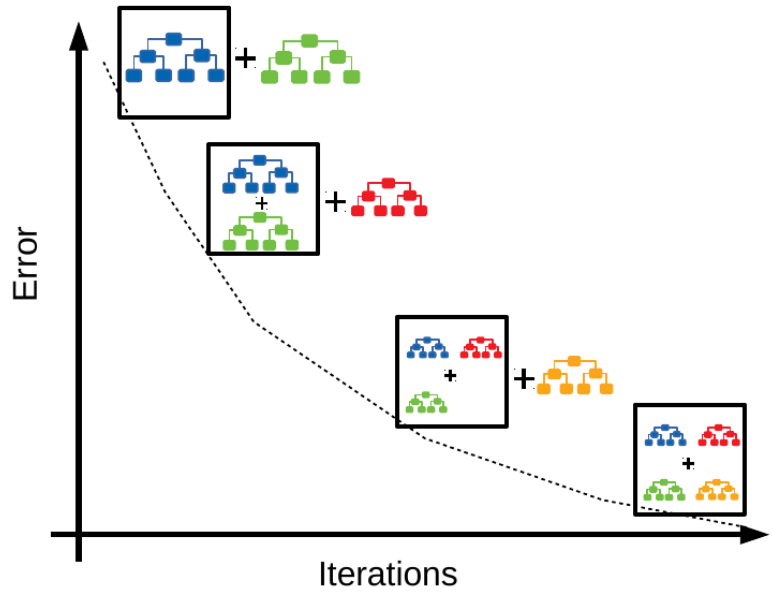
True function:




1000 trees:



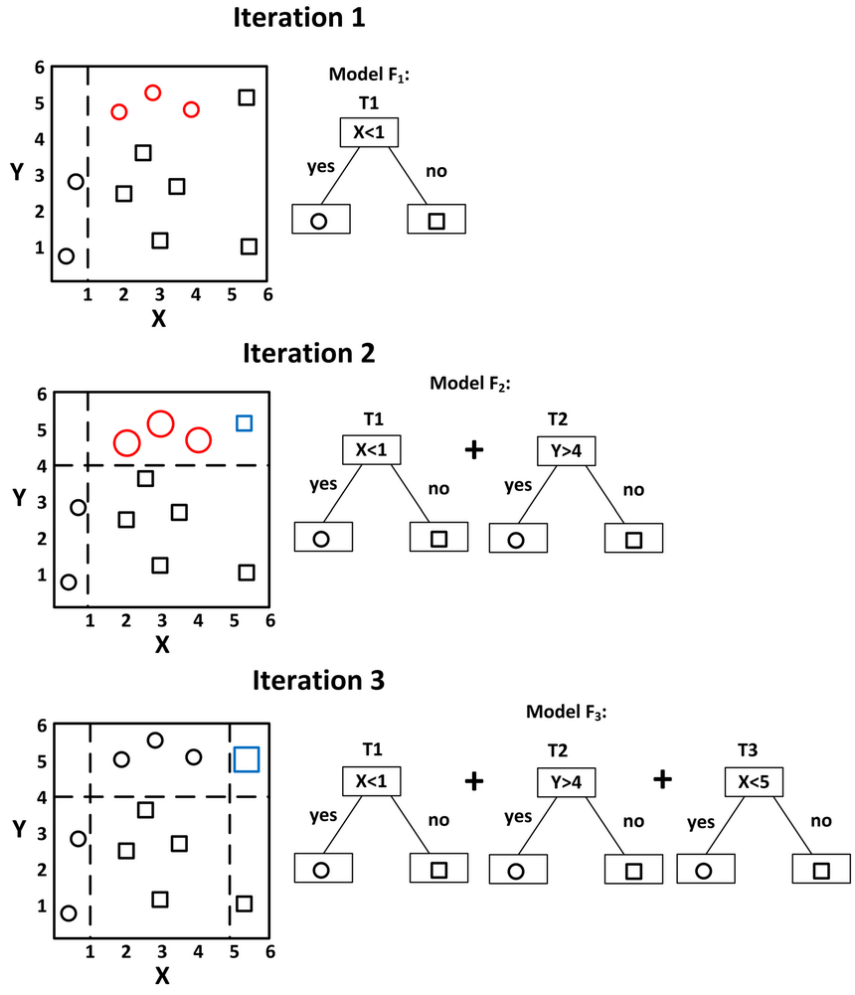
# Boosting methods: Main idea




Picture source: http://tvas.me/articles/2019/08/26/Block-Distributed-Gradient-Boosted-Trees.html

- Boosted trees are ensembles of decision trees.
- A new tree is added at each iteration of the algorithm, with emphasis on those labels that led to errors in the previous iteration.
- Special variant of boosting: Gradient boosting: train on the gradients of the errors, as determined by the employed loss function.

# Boosting methods: Illustration




Picture source: DOI:10.1038/s41598-018-28840-w

- Higher weights are assigned to those samples that led to errors in the previous iteration.

# Basic ideas of boosting methods: Part 1

- Consider two-class problem with outputs $y \in \{-1, 1\}$ and a classifier $g(\mathbf{x}) \in \{-1, 1\}$
- The error rate for $l$ samples is given by $\frac{1}{l}\sum_{i=1}^{l} I(y_i \neq g(\mathbf{x}_i))$. $I$ denotes indicator function.
- Weak classifier: only slightly better than random guessing
- Aim of boosting: sequentially apply weak classification algorithm to repeatedly modified versions of data, producing sequence of weak classifiers $g_n(\mathbf{x})$, $n = 1, ..., N$.
- Predictions from all of them are combined through weighted majority vote to produce final prediction:

$$g(\mathbf{x}) = \text{sign}(\sum_{n=1}^{N} \alpha_n \, g_n(\mathbf{x}))$$

- $\alpha_1, ..., \alpha_N$: computed by boosting algorithm. Effect: give higher influence to more accurate classifiers.

# Basic ideas of boosting methods: Part 2

- Data modifications at each boosting step $n$ consists of applying weights $w_1^{(n)}, ..., w_l^{(n)}$ to each of training observations $(\mathbf{x}_i, y_i)$, $i = 1, ..., l$
- Start: Initialize all weights by $w_i^{(n=1)} = \frac{1}{l}$
- Successive iterations: weights individually modified, algorithm reapplied to weighted observations.
- At step $n$: Observations misclassified by $g_{n-1}(\mathbf{x})$ have weights increased, if correctly classified they stay the same.
- As iterations proceed: observations that are difficult to classify correctly receive ever-increasing influence.

# Basic ideas of boosting methods: Part 3

AdaBoost.M1 algorithm (Friedman et al. 2000):

1. Initialize observation weights $w_i^{(n=1)} = \frac{1}{l}$, $i = 1, ..., l$.
2. For $n = 1$ to $N$:
   - Find optimal classifier $g_n(\mathbf{x})$ to training data with weights $w_i^{(n)}$.
   - Compute:
   $$\text{err}_n = \frac{\sum_{i=1}^{l} w_i^{(n)} \, I(y_i \neq g_n(\mathbf{x}_i))}{\sum_{i=1}^{l} w_i^{(n)}}$$
   - Compute $\alpha_n = \ln(\frac{1-\text{err}_n}{\text{err}_n})$
   - Set $w_i^{(n+1)} = w_i^{(n)} \cdot \exp(\alpha_n \, I(y_i \neq g_n(\mathbf{x}_i))), \;\; i = 1, 2, ..., l$
3. Output $g(\mathbf{x}) = \text{sign}(\sum_{n=1}^{N} \alpha_n \, g_n(\mathbf{x}))$

# Basic ideas of boosting methods: Part 4

- In some sense: boosting is way of fitting an additive expansion in set of elementary basis functions, i.e.

$$g(\mathbf{x}) = \sum_{n=1}^{N} \beta_n \, b(\mathbf{x}; \gamma_n)$$

  - $\beta_n$ for $n = 1, ..., N$: expansion coefficients
  - $b(\mathbf{x}; \gamma)$: simple functions with argument $\mathbf{x}$ and parameter $\gamma$
- Expansions like this play important role in many situations, e.g.:
  - Signal processing (wavelets)
  - Regression splines
  - Most importantly (for our purposes): trees: $\gamma$ parametrizes split variables, split points at internal nodes and predictions at terminal nodes
- To fit these models, we minimize a loss function $L$ (e.g. mean squared-error or likelihood-based) averaged over the training data. The choice of $L$ is crucial.

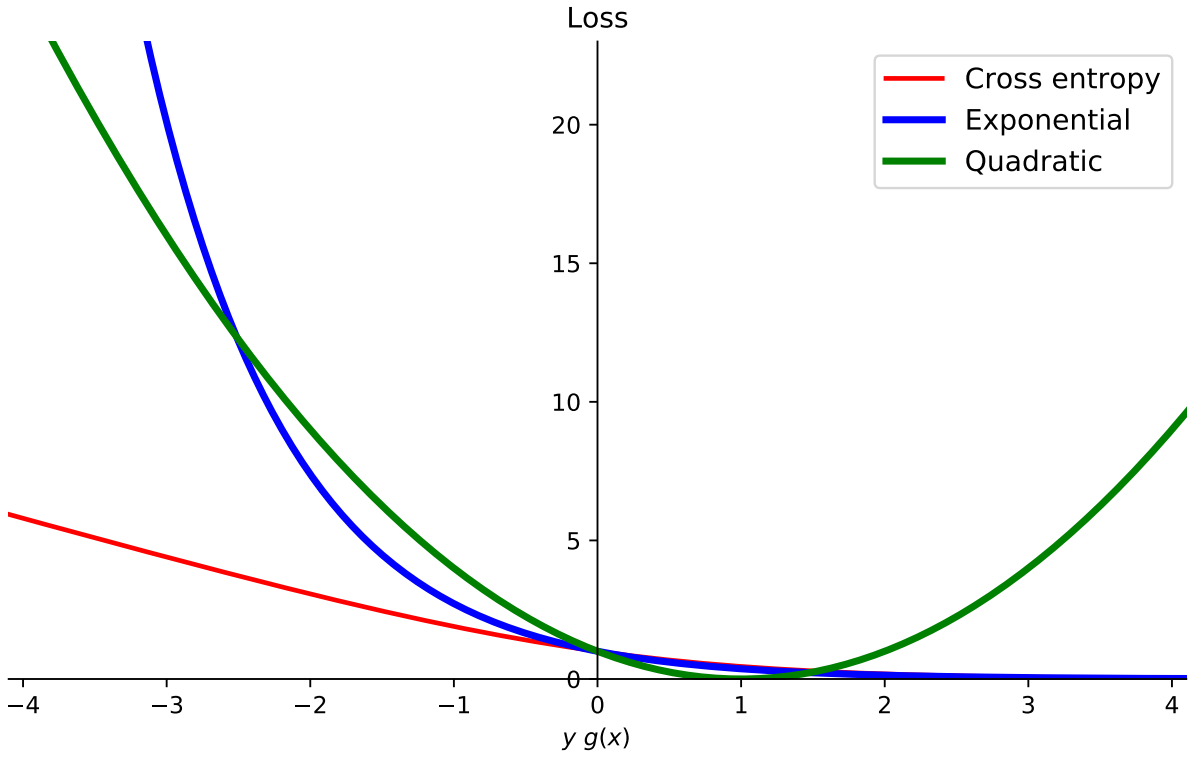
# Basic ideas of boosting methods: Part 5

- Need to solve the following optimization problem:

$$\min_{\{\beta_n,\gamma_n\}_{n=1}^N} \sum_{i=1}^l L(y_i, \sum_{n=1}^N \beta_n b(\mathbf{x}_i;\gamma_n))$$

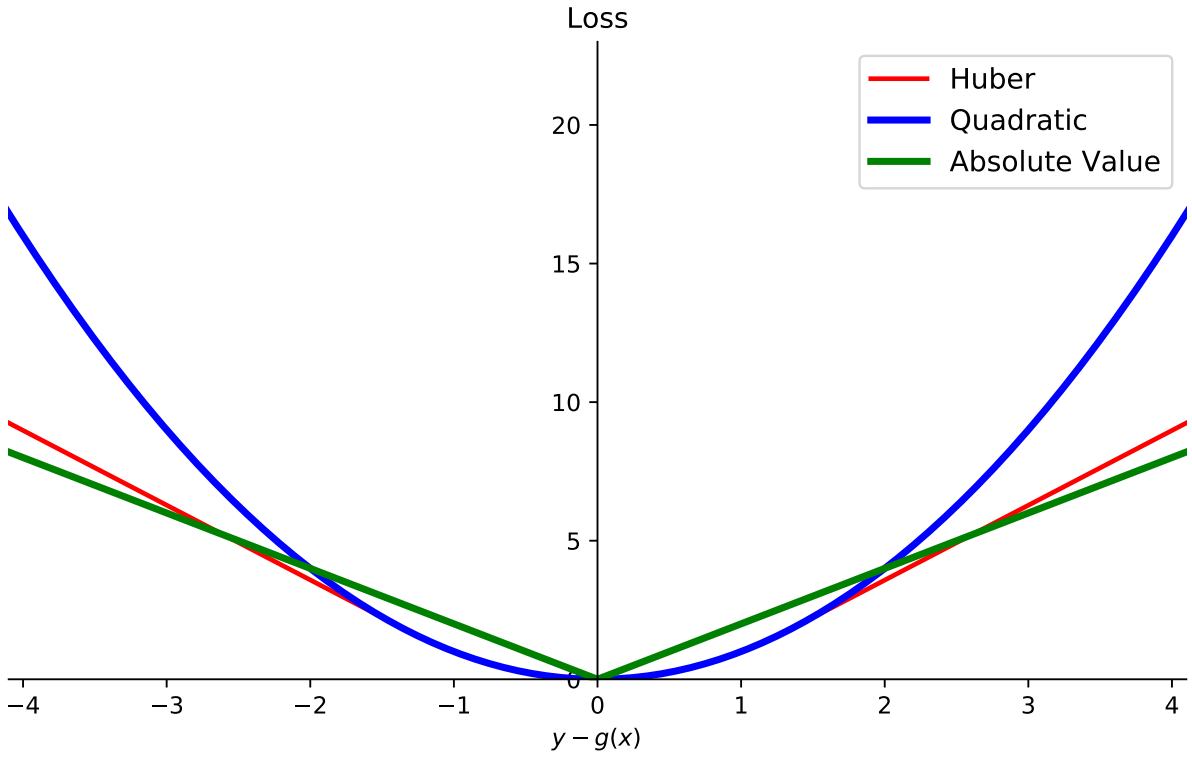
- Forward stagewise additive modelling:
  1. Initialize $g_0(\mathbf{x}) = 0$
  2. For $n = 1$ to $N$:
     - Compute
       $(\beta_n, \gamma_n) = \arg\min_{\beta,\gamma} \sum_{i=1}^l L(y_i, g_{n-1}(\mathbf{x}_i) + \beta b(\mathbf{x}_i;\gamma))$
     - Set $g_n(\mathbf{x}) = g_{n-1}(\mathbf{x}) + \beta_n b(\mathbf{x};\gamma_n)$
- At each step $n$: solve for optimal basis function $b(\mathbf{x};\gamma_n)$ and coefficient $\beta_n$ to add to current expression $g_{n-1}(\mathbf{x})$. Coefficients from previous steps not changed.
- Can be shown: AdaBoost.M1 is equivalent to forward stagewise additive modelling for $L(y, g(\mathbf{x})) = \exp(-y\, g(\mathbf{x}))$ (see exercises).
- Forward stagewise additive modelling is more robust, can be applied to various loss functions and settings.

# Robust losses for classification




- Loss functions for two-class classification. The response is $y = \pm 1$; the prediction is $g(x)$, with class prediction $\mathrm{sign}(g(x))$.
- Exponential and quadratic loss tend to punish outliers too strongly.

# Robust losses for regression




- Comparison of three loss functions for regression, plotted as a function of the margin $y - g(x)$.
- The Huber loss function combines the good properties of squared-error loss near zero and absolute error loss when $|y - g(x)|$ is large.

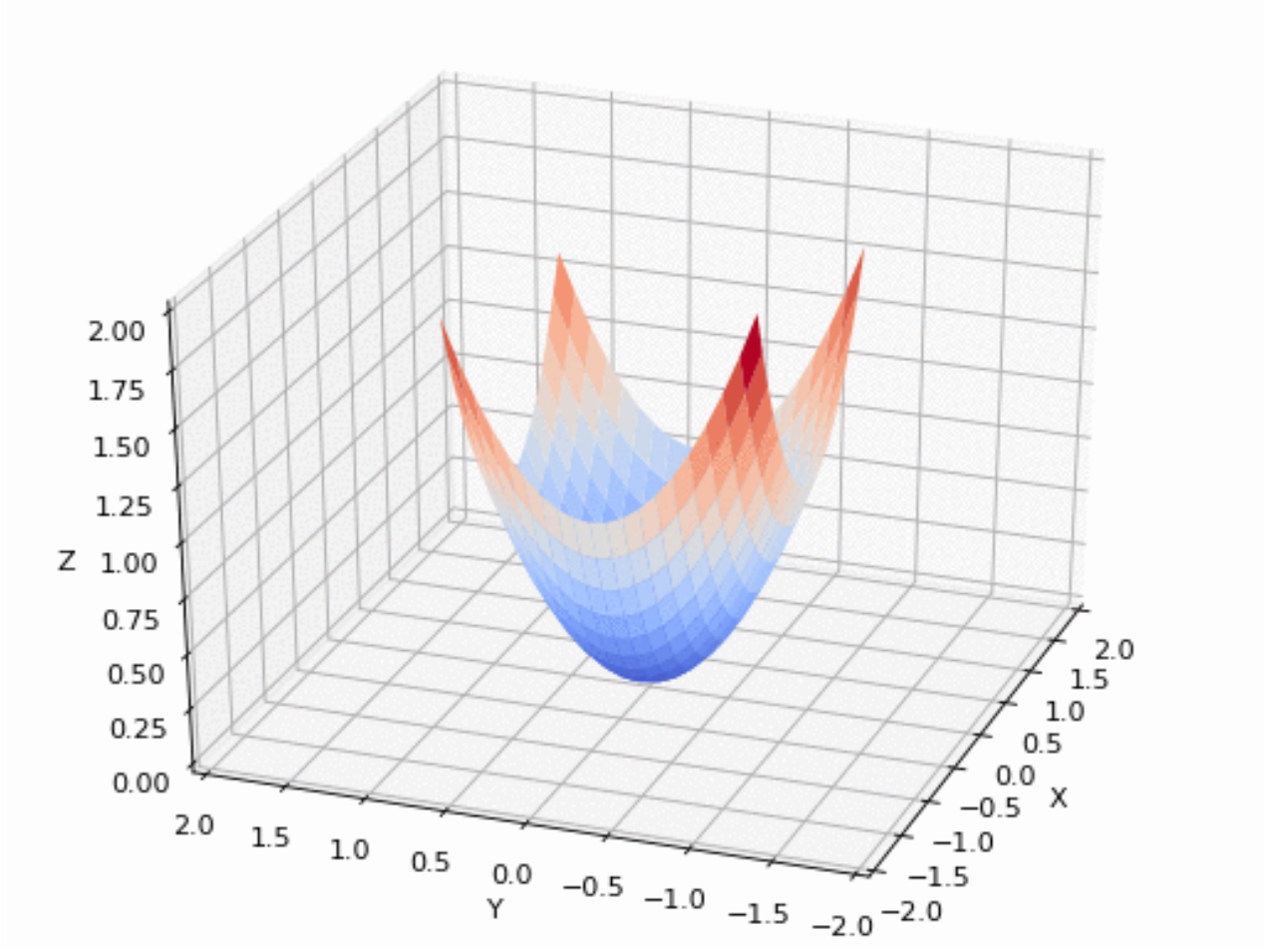
# Basic ideas of tree boosting: Part 1

- How does this look for trees in particular?
- Recall: Tree can be expressed as $T(\mathbf{x}; \Theta) = \sum_{j=1}^{J} \gamma_j I(\mathbf{x} \in R_j)$ with parameters $\Theta = \{R_j, \gamma_j\}$.
- Corresponding optimization problem:
  $\hat{\Theta} = \arg\min_{\Theta} \sum_{j=1}^{J} \sum_{\mathbf{x}_i \in R_j} L(y_i, \gamma_j)$
- Difficult part: find regions $R_j$. If they are given, estimating $\gamma_i$ is easy:
  - Regression: $\gamma_j =$ mean of the $y_i$ falling in region $R_j$
  - Classification: $\gamma_j =$ modal class of the observations falling in region $R_j$ (majority vote)
- Boosted tree model: $g_N(\mathbf{x}) = \sum_{n=1}^{N} T(\mathbf{x}; \Theta_n)$.
- Forward stagewise modelling: at each step need to solve:
  $\hat{\Theta}_n = \arg\min_{\Theta_n} \sum_{i=1}^{l} L(y_i, g_{n-1}(\mathbf{x}_i) + T(\mathbf{x}_i; \Theta_n))$ for region set $\Theta_n = \{R_{j,n}, \gamma_{j,n}\}$ of next tree given current model $g_{n-1}(\mathbf{x})$.

# Basic ideas of tree boosting: Part 2

- In specific situations: problem simplifies, e.g. mean square loss in regression tasks or exponential loss for two-class classification ($\rightarrow$ AdaBoost.M1).
- However to create more robust trees, need more robust loss functions without similar simple boosting algorithms.
- In general: need to solve $\arg\min_{\mathbf{g}} L(\mathbf{g})$, where $\mathbf{g} = (g(\mathbf{x}_1), ..., g(\mathbf{x}_l)) \in \mathbb{R}^l$ and $L(\mathbf{g}) = \sum_{i=1}^{l} L(y_i, g(\mathbf{x}_i))$.
- Use $\mathbf{g}_N = \sum_{n=1}^{N} \mathbf{h}_n$, where $\mathbf{h}_n$ are computed recursively. Different methods available, most popular:
- Steepest descent: $\mathbf{h}_n = -\rho_n \mathbf{G}_n$, where:
  - $\mathbf{G}_n \in \mathbb{R}^l$: gradient of $L(\mathbf{g})$ evaluated at $\mathbf{g} = \mathbf{g}_{n-1}$, i.e. $G_{i,n} = \left[\frac{\partial L(y_i, g(\mathbf{x}_i))}{\partial g(\mathbf{x}_i)}\right]_{g(\mathbf{x}_i)=g_{n-1}(\mathbf{x}_i)}$
  - $\rho_n = \arg\min_{\rho} L(\mathbf{g}_{n-1} - \rho\, \mathbf{G}_n)$ (optimal step size)
  - Greedy algorithm, since $-\mathbf{G}_n$ is local direction in $\mathbb{R}^l$, where $L(\mathbf{g})$ is most rapidly decreasing at $\mathbf{g} = \mathbf{g}_{n-1}$

# Basic ideas of tree boosting: Part 3



# Basic ideas of tree boosting: Part 4

- Differences between forward stagewise modelling with trees and steepest descent: model outputs are constrained to be tree predictions, whereas negative gradient is unconstrained maximal descent direction.
- If goal were minimizing loss on training data: prefer steepest descent only!
- However: our goal is to minimize generalization error!
- $\rightarrow$ Induce tree $T(\mathbf{x}; \Theta_n))$ at $n$-th iteration, whose predictions $\mathbf{t}_n$ are as close as possible to $-\mathbf{G}_n$.
- Using squared error for measuring closeness:

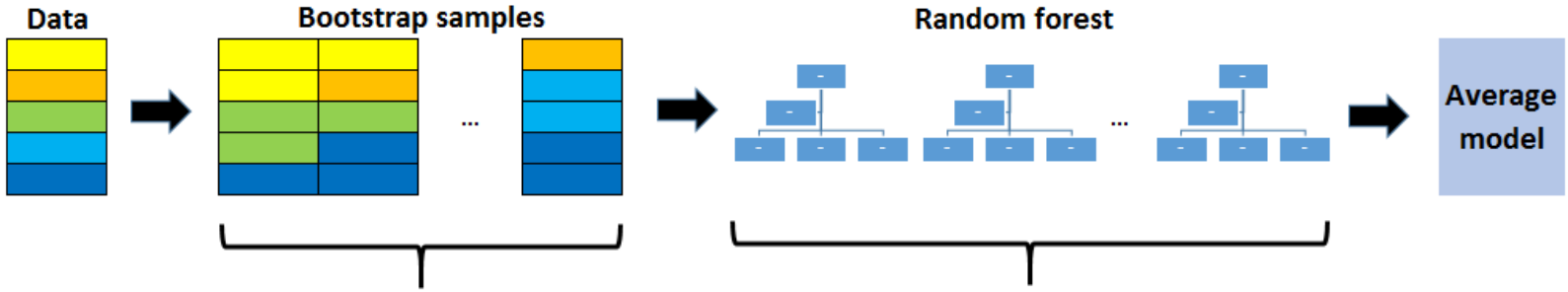
$$\hat{\Theta}_n = \arg\min_{\Theta_n} \sum_{i=1}^{l} (-G_{i,n} - T(\mathbf{x}_i; \Theta_n))^2$$

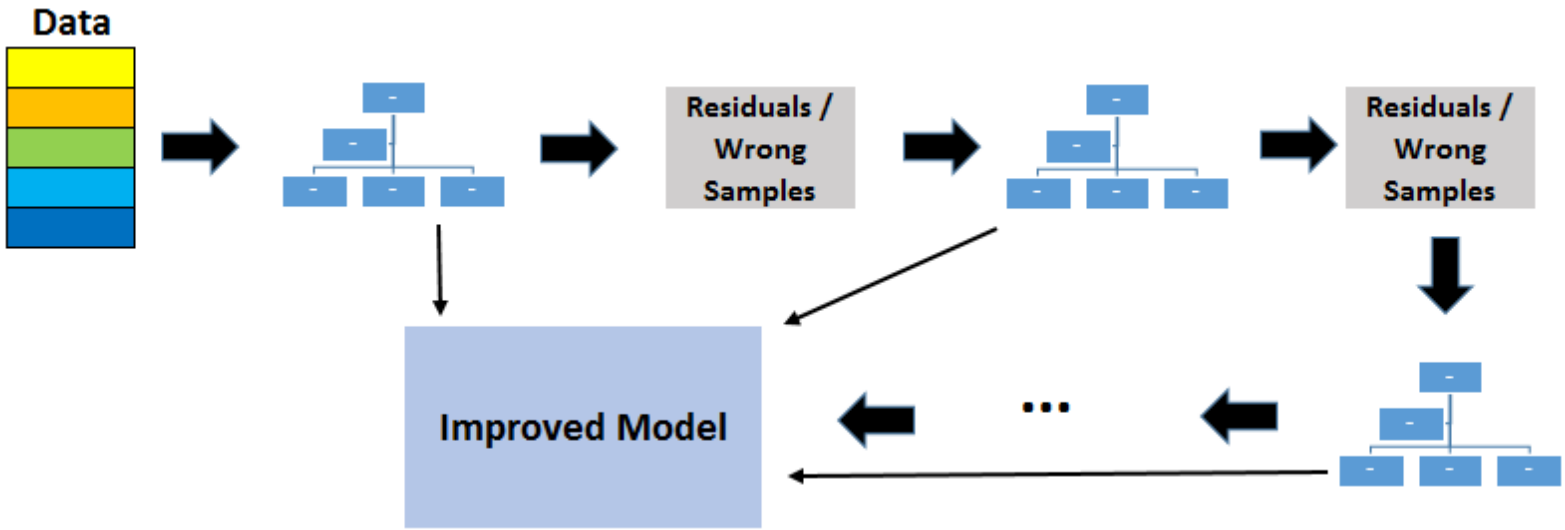
- For this task: fast algorithms exist. Computed regions $\tilde{R}_{j,n}$ may be different to $R_{j,n}$ solving forward stagewise procedure, however: similar enough for practical purposes!

# Basic ideas of tree boosting: Part 5

- Gradient boosted trees (GBT) algorithm for regression:
  1. Initialize $g_0(\mathbf{x}) = \arg\min_\gamma \sum_{i=1}^{l} L(y_i, \gamma)$
  2. For $n = 1, ..., N$:
     - For $i = 1, ..., l$ compute:
       $$G_{i,n} = \left[\frac{\partial L(y_i, g(\mathbf{x}_i))}{\partial g(\mathbf{x}_i)}\right]_{g(\mathbf{x}_i)=g_{n-1}(\mathbf{x}_i)}$$
     - Fit regression tree to targets $G_{i,n}$ giving terminal regions $R_{j,n}$, $j = 1, ..., J_n$.
     - For $j = 1, ..., J_n$ compute:
       $\gamma_{j,n} = \arg\min_\gamma \sum_{\mathbf{x}_i \in R_{j,n}} L(y_i, g_{n-1}(\mathbf{x}_i) + \gamma)$
     - Set $g_n(\mathbf{x}) = g_{n-1}(\mathbf{x}) + \sum_{j=1}^{J_n} \gamma_{j,n} I(\mathbf{x} \in R_{j,n})$
  3. Output $\hat{g}(\mathbf{x}) = g_N(\mathbf{x})$
- For classification: specific loss has to be used (multiclass log-loss, see UNIT 5), sub-steps of step 2 have to be repeated $M$ times, if there are $M$ classes, i.e. $M$ trees are trained. No further details here.

# Comparison of RF (top) and GBT (bottom)




Picture source: https://towardsdatascience.com/gradient-boosted-decision-trees-explained-9259bd8205af

# General discussion: Pros, Cons and typical applications of GBTs

- Important application of GBTs:
  - Anomaly detection: unbalanced data such as DNA sequences, credit card transactions or cybersecurity.
- Advantage of GBTs over RFs:
  - GBTs can be used to solve almost all differentiable objective functions. Includes ranking and Poisson regression, which is harder for RFs.
- Disadvantages of GBTs compared to RFs:
  - More sensitive to overfitting if the data is noisy.
  - Training generally takes longer because of the fact that trees are built sequentially.
  - Harder to tune than RFs. There are typically three parameters: number of trees, depth of trees and learning rate, and each built tree is generally shallow.

# General discussion: Pros, Cons and typical applications of RFs: Part 1

- Important applications of RFs:
  - Multi-class object detection in large-scale real-world computer vision problems
  - Bioinformatics, especially if:
    - Data are noisy and contain missing values.
    - Different data sources with need to be weighted.
    - Large accuracy for high-dimensional problem with highly correlated features.
- Advantages of RFs over GBTs:
  - Much easier to tune than GBTs. Typically two parameters: number of trees and number of features to be selected at each node.
  - Harder to overfit than GBTs.

# General discussion: Pros, Cons and typical applications of RFs: Part 2

- Disadvantages of RFs compared to GBTs:
  - Main limitation: large number of trees may make the algorithm slow for real-time prediction.
  - For data including categorical variables with different number of levels, random forests are biased in favor of those attributes with more levels. Importance scores not reliable.
  - If data contain groups of correlated features of similar relevance for output, smaller groups are favored over larger groups.
- We only scratched the surface. For much more details, consider the particularly nice book "The Elements of Statistical Learning" by Hastie, Tibshirani, and Friedman. Also available online here.