Machine Learning Crash Course: 4. Tree-based Methods

Carlo Zanella

September 12, 2018

This module

In this module we look at tree-based methods for both regression and classification. Trees divide the feature space into partitions and apply simple local models to the partitititions of the data. We introduce methods such as bagging, random forests, and boosting which combine multiple trees in order to get better results.

A Decision Tree

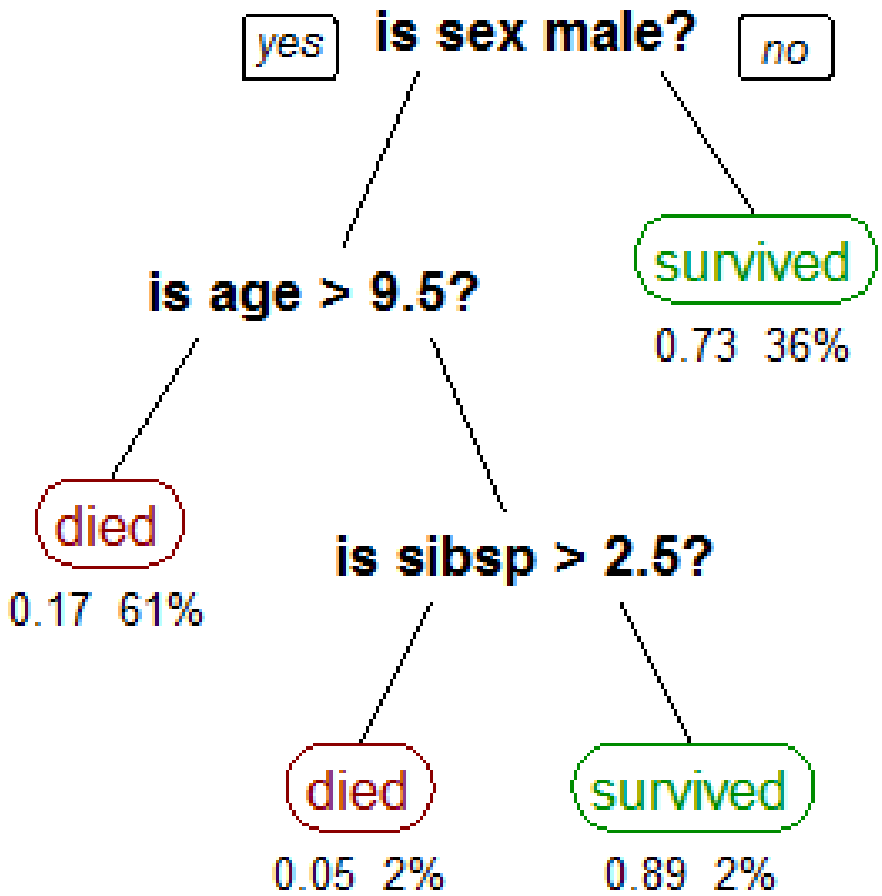


Figure: A Decision Tree for Titanic Survivors (Source: Wikipedia)

Classification and Regression Trees (CART)

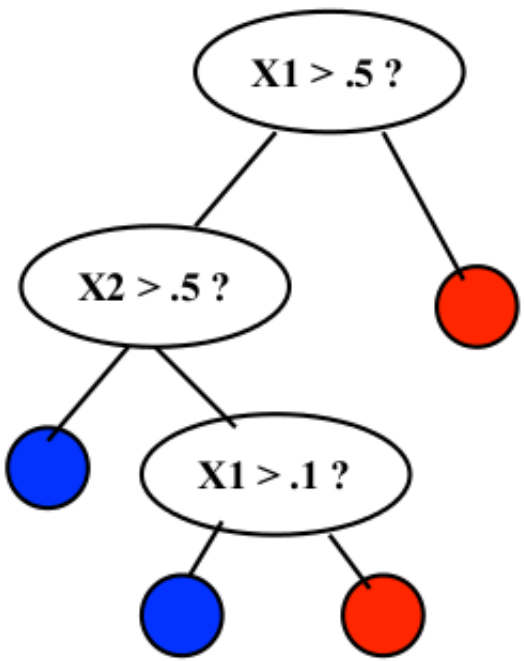
- Such decision trees can be learnt and can be used for classification and regression problems.

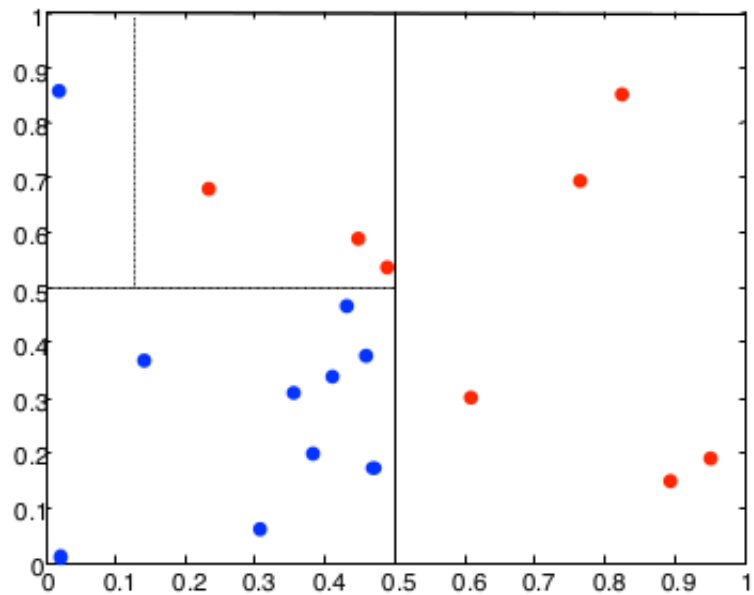
Classification and Regression Trees (CART)

- Such decision trees can be learnt and can be used for classification and regression problems.
- Results in a very interpretable model

Classification and Regression Trees (CART)

- Such decision trees can be learnt and can be used for classification and regression problems.
- Results in a very interpretable model
- Trees partition the feature space $X_1, X_2, \ldots$ into J distinct and non-overlapping regions $R_1, \ldots, R_J$:





Classification and Regression Trees (CART) (2)

- A classification tree will assign each observation in one region R_j the same class.

Classification and Regression Trees (CART) (2)

- A classification tree will assign each observation in one region R_j the same class.
- In case of a regression tree it will usually assign each region the mean value of Y.

Classification and Regression Trees (CART) (2)

- A classification tree will assign each observation in one region R_j the same class.
- In case of a regression tree it will usually assign each region the mean value of Y.
- How are trees built?

Classification and Regression Trees (CART) (2)

- A classification tree will assign each observation in one region R_j the same class.
- In case of a regression tree it will usually assign each region the mean value of Y.
- How are trees built?
- In a regression problem, the goal is to minimize the RSS

$$\sum_{j=1}^{J} \sum_{i:x_i \in R_j} (y_i - \hat{y}_{R_j})^2,$$

where $\hat{y}_{R_j}$ is the mean response for all observations in region R_j.

Classification and Regression Trees (CART) (2)

- A classification tree will assign each observation in one region R_j the same class.
- In case of a regression tree it will usually assign each region the mean value of Y.
- How are trees built?
- In a regression problem, the goal is to minimize the RSS

$$\sum_{j=1}^{J} \sum_{i:x_i \in R_j} (y_i - \hat{y}_{R_j})^2,$$

  where $\hat{y}_{R_j}$ is the mean response for all observations in region R_j.
- In classification trees, instead of RSS a measure of classification error is used.

Classification and Regression Trees (CART) (2)

- A classification tree will assign each observation in one region R_j the same class.
- In case of a regression tree it will usually assign each region the mean value of Y.
- How are trees built?
- In a regression problem, the goal is to minimize the RSS

$$\sum_{j=1}^{J} \sum_{i:x_i \in R_j} (y_i - \hat{y}_{R_j})^2,$$

  where $\hat{y}_{R_j}$ is the mean response for all observations in region R_j.
- In classification trees, instead of RSS a measure of classification error is used.
- Let's look at regression trees for now.

Growing trees

- It is computationally not feasible to search over all trees $R_1, \ldots, R_J$ for the minimal RSS.

Growing trees

- It is computationally not feasible to search over all trees $R_1, \ldots, R_J$ for the minimal RSS.
- Instead CART employs a greedy top-down algorithm.

Growing trees

- It is computationally not feasible to search over all trees $R_1, \ldots, R_J$ for the minimal RSS.
- Instead CART employs a greedy top-down algorithm.
- We start out by considering all possible predictors $X_1, \ldots, X_m$.

Growing trees

- It is computationally not feasible to search over all trees $R_1, \ldots, R_J$ for the minimal RSS.
- Instead CART employs a greedy top-down algorithm.
- We start out by considering all possible predictors $X_1, \ldots, X_m$.
- For each predictor X_j and given a cut-off value s define the two half-planes

$$R_1(j, s) = \{X | X_j < s\} \text{ and } R_2(j, s) = \{X | X_J \geq s\}.$$

Growing trees

- It is computationally not feasible to search over all trees $R_1, \ldots, R_J$ for the minimal RSS.
- Instead CART employs a greedy top-down algorithm.
- We start out by considering all possible predictors $X_1, \ldots, X_m$.
- For each predictor X_j and given a cut-off value s define the two half-planes

$$R_1(j, s) = \{X | X_j < s\} \text{ and } R_2(j, s) = \{X | X_J \geq s\}.$$

- Now we look for the predictor X_j and the value of s that minimizes the sum of the RSS in the two regions R_1 and R_2:

$$\sum_{i:x_i \in R_1} (y_i - \hat{y}_{R_1})^2 + \sum_{i:x_i \in R_2} (y_i - \hat{y}_{R_2})^2.$$

Growing trees (2)

- After having found j and s we have split the feature space into two regions R_1 and R_2.

Growing trees (2)

- After having found j and s we have split the feature space into two regions R_1 and R_2.
- This creates us our very first decision node: "if $X_j < s$ predict $\hat{y}_{R_1}$ otherwise predict $\hat{y}_{R_2}$".

Growing trees (2)

- After having found j and s we have split the feature space into two regions R_1 and R_2.
- This creates us our very first decision node: "if $X_j < s$ predict $\hat{y}_{R_1}$ otherwise predict $\hat{y}_{R_2}$".
- Now we can continue do the same procedure again for R_1 and R_2

Growing trees (2)

- After having found j and s we have split the feature space into two regions R_1 and R_2.
- This creates us our very first decision node: "if $X_j < s$ predict $\hat{y}_{R_1}$ otherwise predict $\hat{y}_{R_2}$".
- Now we can continue do the same procedure again for R_1 and R_2
- We can in turn search for the optimal predictor and threshold s which splits region 1 and region 2.

Growing trees (2)

- After having found j and s we have split the feature space into two regions R_1 and R_2.
- This creates us our very first decision node: "if $X_j < s$ predict $\hat{y}_{R_1}$ otherwise predict $\hat{y}_{R_2}$".
- Now we can continue do the same procedure again for R_1 and R_2
- We can in turn search for the optimal predictor and threshold s which splits region 1 and region 2.
- This will give as further tree nodes.

Growing trees (2)

- After having found j and s we have split the feature space into two regions R_1 and R_2.
- This creates us our very first decision node: "if $X_j < s$ predict $\hat{y}_{R_1}$ otherwise predict $\hat{y}_{R_2}$".
- Now we can continue do the same procedure again for R_1 and R_2
- We can in turn search for the optimal predictor and threshold s which splits region 1 and region 2.
- This will give as further tree nodes.
- We may continue like this until there is only a couple of observations left in every region.

Growing trees (2)

- After having found j and s we have split the feature space into two regions R_1 and R_2.
- This creates us our very first decision node: "if $X_j < s$ predict $\hat{y}_{R_1}$ otherwise predict $\hat{y}_{R_2}$".
- Now we can continue do the same procedure again for R_1 and R_2
- We can in turn search for the optimal predictor and threshold s which splits region 1 and region 2.
- This will give as further tree nodes.
- We may continue like this until there is only a couple of observations left in every region.
- This algorithm is called recursive binary splitting.

Tree pruning

- The trees grown using recursive binary splitting may give good prediction on the training sample but is likely to overfit the data

Tree pruning

- The trees grown using recursive binary splitting may give good prediction on the training sample but is likely to overfit the data
- Why?

Tree pruning

- The trees grown using recursive binary splitting may give good prediction on the training sample but is likely to overfit the data
- Why?
- Hence the resulting tree may be too complex and we may want to regularize it.

Tree pruning

- The trees grown using recursive binary splitting may give good prediction on the training sample but is likely to overfit the data
- Why?
- Hence the resulting tree may be too complex and we may want to regularize it.
- Tree pruning is a set of methods to reduce the complexity of a tree by choosing a subtree T of a fully-grown tree T_0.

Tree pruning

- The trees grown using recursive binary splitting may give good prediction on the training sample but is likely to overfit the data
- Why?
- Hence the resulting tree may be too complex and we may want to regularize it.
- Tree pruning is a set of methods to reduce the complexity of a tree by choosing a subtree T of a fully-grown tree T_0.
- Popular technique is cost complexity pruning: given a hyperparameter $\alpha > 0$ chooses the subtree $T \subset T_0$ that minimizes

$$\sum_{m=1}^{|T|} \sum_{i:x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha |T|,$$

where $|T|$ is the number of terminal nodes in the tree (which is equal to the number of regions J of T).

Tree pruning

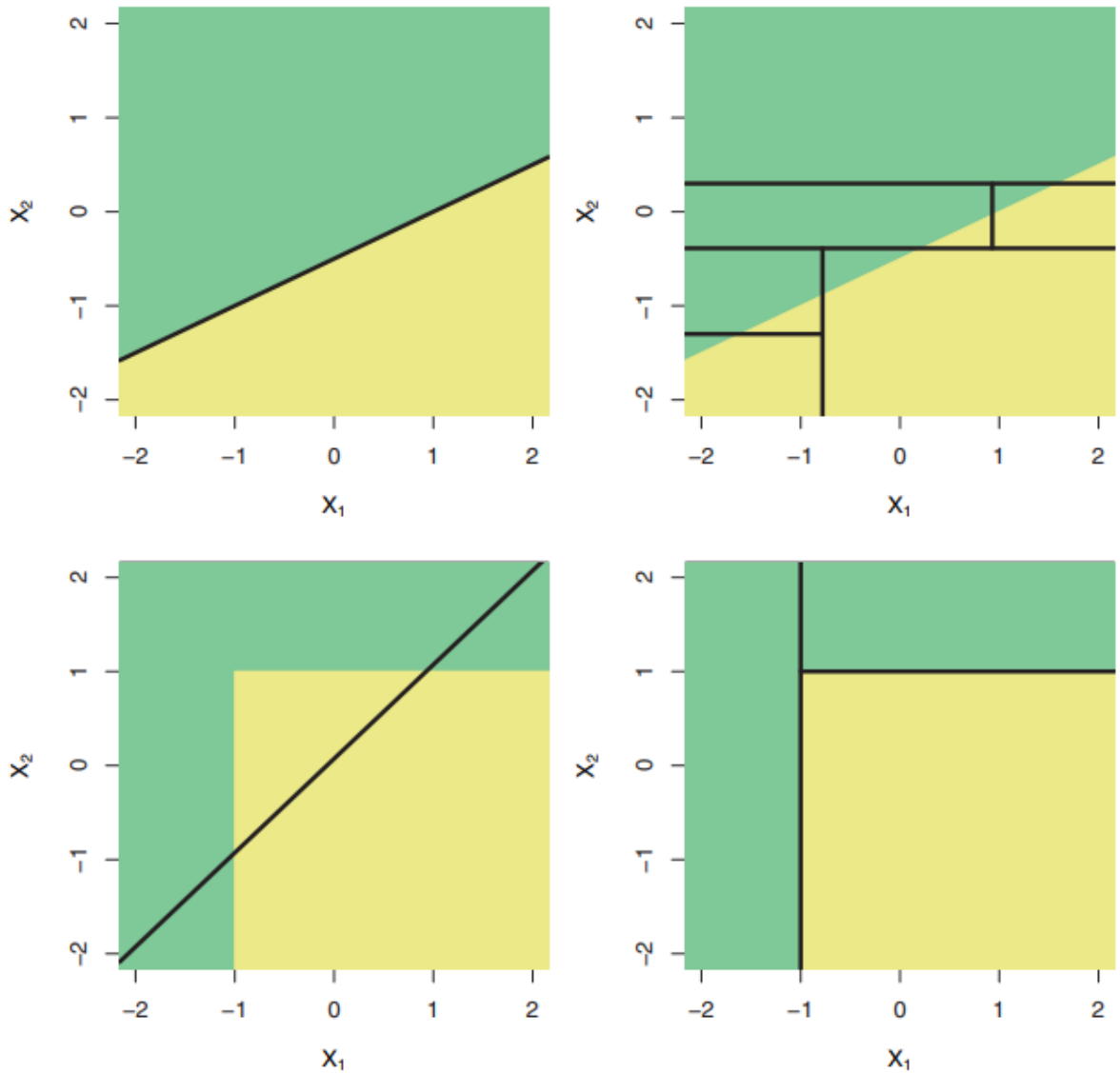
- The trees grown using recursive binary splitting may give good prediction on the training sample but is likely to overfit the data
- Why?
- Hence the resulting tree may be too complex and we may want to regularize it.
- Tree pruning is a set of methods to reduce the complexity of a tree by choosing a subtree T of a fully-grown tree T_0.
- Popular technique is cost complexity pruning: given a hyperparameter $\alpha > 0$ chooses the subtree $T \subset T_0$ that minimizes

$$\sum_{m=1}^{|T|} \sum_{i:x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha |T|,$$

where $|T|$ is the number of terminal nodes in the tree (which is equal to the number of regions J of T).

- How to determine α?

Trees vs Linear Methods



When are trees useful?

- Trees are easy to interpret

When are trees useful?

- Trees are easy to interpret
- They can capture non-linear relationships between regressors

When are trees useful?

- Trees are easy to interpret
- They can capture non-linear relationships between regressors
- Allow for regularization to present overfitting

When are trees useful?

- Trees are easy to interpret
- They can capture non-linear relationships between regressors
- Allow for regularization to present overfitting
- Bad news: overall quite bad performance, not competitive with most methods

When are trees useful?

- Trees are easy to interpret
- They can capture non-linear relationships between regressors
- Allow for regularization to present overfitting
- Bad news: overall quite bad performance, not competitive with most methods
- Also not very robust to single observations

When are trees useful?

- Trees are easy to interpret
- They can capture non-linear relationships between regressors
- Allow for regularization to present overfitting
- Bad news: overall quite bad performance, not competitive with most methods
- Also not very robust to single observations
- Next we explore methods that combine multiple trees to give better performance

Bagging

- Bagging, short for Bootstrap aggregating is a general ensemble algorithms that works as follows:

Bagging

- Bagging, short for Bootstrap aggregating is a general ensemble algorithms that works as follows:
- Bootstrap B samples from your training data by sampling uniformly and with replacement

Bagging

- Bagging, short for Bootstrap aggregating is a general ensemble algorithms that works as follows:
- Bootstrap B samples from your training data by sampling uniformly and with replacement
- Grow a full tree on each of the B bootstrap samples (don't prune it!)

Bagging

- Bagging, short for Bootstrap aggregating is a general ensemble algorithms that works as follows:
- Bootstrap B samples from your training data by sampling uniformly and with replacement
- Grow a full tree on each of the B bootstrap samples (don't prune it!)
- For a new observation x let $f_i(x)$ be the prediction of the ith tree

Bagging

- Bagging, short for Bootstrap aggregating is a general ensemble algorithms that works as follows:
- Bootstrap B samples from your training data by sampling uniformly and with replacement
- Grow a full tree on each of the B bootstrap samples (don't prune it!)
- For a new observation x let $f_i(x)$ be the prediction of the ith tree
- The bagging prediction f is then

$$f(x) = \sum_{i=1}^{B} \frac{1}{B} f_i(x),$$

that is take the average prediction of your B trees.

Bagging

- Bagging, short for Bootstrap aggregating is a general ensemble algorithms that works as follows:
- Bootstrap B samples from your training data by sampling uniformly and with replacement
- Grow a full tree on each of the B bootstrap samples (don't prune it!)
- For a new observation x let $f_i(x)$ be the prediction of the ith tree
- The bagging prediction f is then

$$f(x) = \sum_{i=1}^{B} \frac{1}{B} f_i(x),$$

  that is take the average prediction of your B trees.
- Intuition: a single tree has high variance but low bias. Averaging a set of observations reduces variance.

Bagging

- Bagging, short for Bootstrap aggregating is a general ensemble algorithms that works as follows:
- Bootstrap B samples from your training data by sampling uniformly and with replacement
- Grow a full tree on each of the B bootstrap samples (don't prune it!)
- For a new observation x let $f_i(x)$ be the prediction of the ith tree
- The bagging prediction f is then

$$f(x) = \sum_{i=1}^{B} \frac{1}{B} f_i(x),$$

  that is take the average prediction of your B trees.
- Intuition: a single tree has high variance but low bias. Averaging a set of observations reduces variance.
- Method is general but works especially well with trees

Bagging: Out-of-bag error

- With bootstrapping each bagged tree on average uses 2/3 of the observations

Bagging: Out-of-bag error

- With bootstrapping each bagged tree on average uses 2/3 of the observations
- On average 1/3 of observations are unused

Bagging: Out-of-bag error

- With bootstrapping each bagged tree on average uses 2/3 of the observations
- On average 1/3 of observations are unused
- We can use them as our "validation sample"

Bagging: Out-of-bag error

- With bootstrapping each bagged tree on average uses 2/3 of the observations
- On average 1/3 of observations are unused
- We can use them as our "validation sample"
- These observations are called out-of-bag

Bagging: Out-of-bag error

- With bootstrapping each bagged tree on average uses 2/3 of the observations
- On average 1/3 of observations are unused
- We can use them as our "validation sample"
- These observations are called out-of-bag
- And the error bagging makes on these observations is called the out-of-bag error

Bagging: Out-of-bag error

- With bootstrapping each bagged tree on average uses 2/3 of the observations
- On average 1/3 of observations are unused
- We can use them as our "validation sample"
- These observations are called out-of-bag
- And the error bagging makes on these observations is called the out-of-bag error
- This means we don't need to use cross-validation to estimate the test error because bootstrapping is already "kind of cross-validation"

Random Forests

- An even better method of aggregating trees is called Random Forests.

Random Forests

- An even better method of aggregating trees is called Random Forests.
- Problem with bagged trees is that trees are still very correlated

Random Forests

- An even better method of aggregating trees is called Random Forests.
- Problem with bagged trees is that trees are still very correlated
- Random forests use a simple small tweak to decorrelate the trees.

Random Forests

- An even better method of aggregating trees is called Random Forests.
- Problem with bagged trees is that trees are still very correlated
- Random forests use a simple small tweak to decorrelate the trees.
- Recall that when we grow the full tree at each step we consider all predictors X_j and all thresholds s.

Random Forests

- An even better method of aggregating trees is called Random Forests.
- Problem with bagged trees is that trees are still very correlated
- Random forests use a simple small tweak to decorrelate the trees.
- Recall that when we grow the full tree at each step we consider all predictors X_j and all thresholds s.
- Rather than considering all m predictors, Random Forests randomly restrict the choice to $p < m$ predictors

Random Forests

- An even better method of aggregating trees is called Random Forests.
- Problem with bagged trees is that trees are still very correlated
- Random forests use a simple small tweak to decorrelate the trees.
- Recall that when we grow the full tree at each step we consider all predictors X_j and all thresholds s.
- Rather than considering all m predictors, Random Forests randomly restrict the choice to $p < m$ predictors
- Different set of predictors at every step, for every tree.

Random Forests

- An even better method of aggregating trees is called Random Forests.
- Problem with bagged trees is that trees are still very correlated
- Random forests use a simple small tweak to decorrelate the trees.
- Recall that when we grow the full tree at each step we consider all predictors X_j and all thresholds s.
- Rather than considering all m predictors, Random Forests randomly restrict the choice to $p < m$ predictors
- Different set of predictors at every step, for every tree.
- Usually $p \approx \sqrt{m}$ (but in principle this is a tuning parameter).

Random Forests: Intuition

- Suppose there is an important variable and all trees will always choose this variable as the first split.

Random Forests: Intuition

- Suppose there is an important variable and all trees will always choose this variable as the first split.
- Bagging then aggregates over very similar trees

Random Forests: Intuition

- Suppose there is an important variable and all trees will always choose this variable as the first split.
- Bagging then aggregates over very similar trees
- Random Forests may by chance not consider this important predictor in the first split.

Random Forests: Intuition

- Suppose there is an important variable and all trees will always choose this variable as the first split.
- Bagging then aggregates over very similar trees
- Random Forests may by chance not consider this important predictor in the first split.
- Hence it is able to build a truly different tree

Boosting

- Random Forests is a method to reduce the variance of trees

Boosting

- Random Forests is a method to reduce the variance of trees
- An alternative approach is called Boosting.

Boosting

- Random Forests is a method to reduce the variance of trees
- An alternative approach is called Boosting.
- It uses weak learners i.e. learners with high bias but low variance

Boosting

- Random Forests is a method to reduce the variance of trees
- An alternative approach is called Boosting.
- It uses weak learners i.e. learners with high bias but low variance
- Tries to reduce bias

Boosting

- Random Forests is a method to reduce the variance of trees
- An alternative approach is called Boosting.
- It uses weak learners i.e. learners with high bias but low variance
- Tries to reduce bias
- In terms of decision trees, a weak learner is a shallow tree (i.e. only one or two splits)

Boosting: How does it work?

- Boosting sequentially grows B trees each with at most d splits

Boosting: How does it work?

- Boosting sequentially grows B trees each with at most d splits
- The first tree is grown by fitting the residuals y_i.

Boosting: How does it work?

- Boosting sequentially grows B trees each with at most d splits
- The first tree is grown by fitting the residuals y_i.
- We then shrink the tree by a small parameter λ and compute the residuals

$$r_i^1 = y_i - \underbrace{\lambda \tilde{f}^b(x_i)}_{\text{shrunken tree}} .$$

Boosting: How does it work?

- Boosting sequentially grows B trees each with at most d splits
- The first tree is grown by fitting the residuals y_i.
- We then shrink the tree by a small parameter λ and compute the residuals

$$r_i^1 = y_i - \underbrace{\lambda \tilde{f}^b(x_i)}_{\text{shrunken tree}} .$$

- The second tree is grown to fit the residuals of the shrunken first tree and then shrunken by λ.

Boosting: How does it work?

- Boosting sequentially grows B trees each with at most d splits
- The first tree is grown by fitting the residuals y_i.
- We then shrink the tree by a small parameter λ and compute the residuals

$$r_i^1 = y_i - \underbrace{\lambda \tilde{f}^b(x_i)}_{\text{shrunken tree}} .$$

- The second tree is grown to fit the residuals of the shrunken first tree and then shrunken by λ.
- This is continued B times, each time trees are grown on the residuals of the previous shrunken tree

Boosting: How does it work?

- Boosting sequentially grows B trees each with at most d splits
- The first tree is grown by fitting the residuals y_i.
- We then shrink the tree by a small parameter λ and compute the residuals

$$r_i^1 = y_i - \underbrace{\lambda \tilde{f}^b(x_i)}_{\text{shrunken tree}} .$$

- The second tree is grown to fit the residuals of the shrunken first tree and then shrunken by λ.
- This is continued B times, each time trees are grown on the residuals of the previous shrunken tree
- If λ is very small, then this learning algorithm is very slow

Boosting: How does it work?

- Boosting sequentially grows B trees each with at most d splits
- The first tree is grown by fitting the residuals y_i.
- We then shrink the tree by a small parameter λ and compute the residuals

$$r_i^1 = y_i - \underbrace{\lambda \tilde{f}^b(x_i)}_{\text{shrunken tree}} .$$

- The second tree is grown to fit the residuals of the shrunken first tree and then shrunken by λ.
- This is continued B times, each time trees are grown on the residuals of the previous shrunken tree
- If λ is very small, then this learning algorithm is very slow
- This is called a slow learner

Boosting: How does it work? (2)

- Intuition of slow learning: in each iteration we try to capture a small part of the signal

Boosting: How does it work? (2)

- Intuition of slow learning: in each iteration we try to capture a small part of the signal
- After learning is complete, the boosting prediction is

$$\tilde{f}(x) = \sum_{b=1}^{B} \frac{1}{B} \lambda f^b(x).$$

Boosting: How does it work? (2)

- Intuition of slow learning: in each iteration we try to capture a small part of the signal
- After learning is complete, the boosting prediction is

$$\tilde{f}(x) = \sum_{b=1}^{B} \frac{1}{B} \lambda f^b(x).$$

- Boosting has three hyperparameters:

Boosting: How does it work? (2)

- Intuition of slow learning: in each iteration we try to capture a small part of the signal
- After learning is complete, the boosting prediction is

$$\tilde{f}(x) = \sum_{b=1}^{B} \frac{1}{B} \lambda f^b(x).$$

- Boosting has three hyperparameters:
  1. number of trees B: Unlike Bagging and RF this algorithm can overfit

Boosting: How does it work? (2)

- Intuition of slow learning: in each iteration we try to capture a small part of the signal
- After learning is complete, the boosting prediction is

$$\tilde{f}(x) = \sum_{b=1}^{B} \frac{1}{B} \lambda f^b(x).$$

- Boosting has three hyperparameters:
  1. number of trees B: Unlike Bagging and RF this algorithm can overfit
  2. shrinkage parameter λ: this is the learning rate of the algorithm. Lower values mean longer training time

Boosting: How does it work? (2)

- Intuition of slow learning: in each iteration we try to capture a small part of the signal
- After learning is complete, the boosting prediction is

$$\tilde{f}(x) = \sum_{b=1}^{B} \frac{1}{B} \lambda f^b(x).$$

- Boosting has three hyperparameters:
  1. number of trees B: Unlike Bagging and RF this algorithm can overfit
  2. shrinkage parameter λ: this is the learning rate of the algorithm. Lower values mean longer training time
  3. Number of splits d: controls the complexity / depth of the trees. Usually $d = 1$ is used, i.e. only one split

Boosting: How does it work? (2)

- Intuition of slow learning: in each iteration we try to capture a small part of the signal
- After learning is complete, the boosting prediction is

$$\tilde{f}(x) = \sum_{b=1}^{B} \frac{1}{B} \lambda f^b(x).$$

- Boosting has three hyperparameters:
  1. number of trees B: Unlike Bagging and RF this algorithm can overfit
  2. shrinkage parameter λ: this is the learning rate of the algorithm. Lower values mean longer training time
  3. Number of splits d: controls the complexity / depth of the trees. Usually $d = 1$ is used, i.e. only one split
- Parameters are chosen by Cross-validation

Boosting vs Random Forests

- Boosting often has better predictive performance than random forests

Boosting vs Random Forests

- Boosting often has better predictive performance than random forests
- Drawback is that boosting need a great deal of tuning whereas Random Forests is almost tuning-free

Boosting vs Random Forests

- Boosting often has better predictive performance than random forests
- Drawback is that boosting need a great deal of tuning whereas Random Forests is almost tuning-free
- Take more time to compute because trees are grown sequentially