WIRTSCHAFTS
UNIVERSITÄT
WIEN VIENNA
UNIVERSITY OF
ECONOMICS
AND BUSINESS

Data Science and Machine Learning

Random Forests and Boosted Regression Trees

Katharina Fenz

November 10, 2020

Outline

The theory behind Random Forests and Boosted Regression Trees

- Random Forests and the method of bagging
- Boosted Regression Trees and the method of boosting

Random Forests and Boosted Regression Trees in R

- Random Forests with `ranger`
- Boosted Regression Trees with `xgboost`
- Cross validation with `caret` and `tuneRanger`

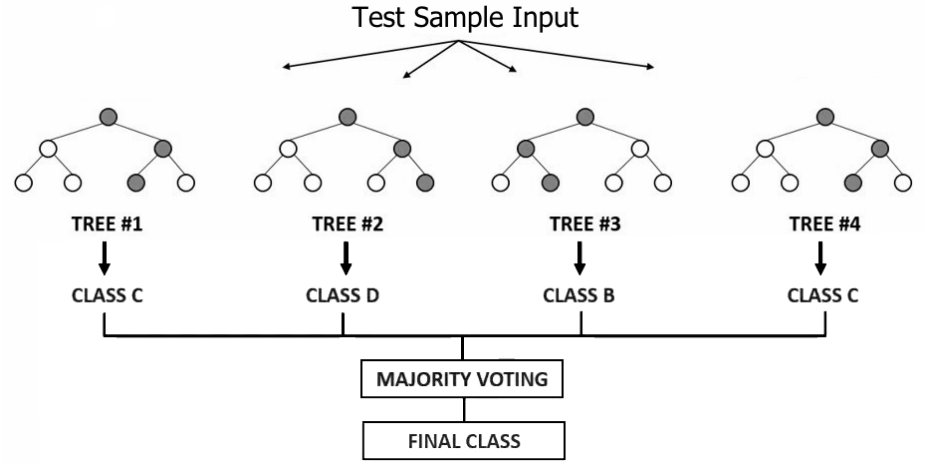
Random Forests and Boosted Regression Trees

- Random Forests and Boosted Regression Trees are ensemble methods
- Ensemble methods use multiple learning algorithms (e.g. decision trees) and combine their predictions based on bagging, boosting, or Bayesian model averaging
- The goal of ensemble methods is to improve upon the results of individual algorithms and make them more accurate and robust
- Random Forests combine decision trees with bagging
- Boosted Regression Trees combine decision trees with boosting

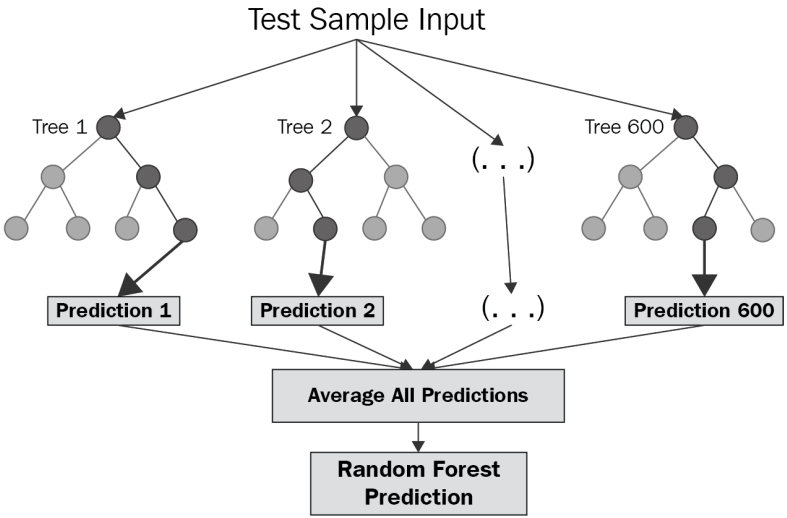
Creating a Random Forest

- Random Forests consist of sets of decision trees
- Random Forests can be grown from classification trees or regression trees
- Classification trees:
  - Dependent variable is discrete
  - Prediction is based on majority voting
- Regression trees:
  - Dependent variable is continuous
  - Prediction is based on averaging

Random Forest of Classification Trees



Random Forest of Regression Trees



Adding Randomness to the Forest

There are two sources of randomness in a Random Forest:

- Each individual tree only considers a random subset of the training data (= bagging)
- Each individual tree only considers a random subset of the explanatory variables
- This reduces the variance and the risk of overfitting

Bagging I

- Decision trees can tend to suffer from a high variance
- If we split the training data into two parts at random and fit a decision tree to each half, the results we get could be quite different
- However, we would prefer a method with low variance, i.e. one that yields similar results if applied repeatedly to different parts to the training data
- Bagging (= bootstrap aggregation) constitutes a way to reduce the variance of estimations based on decision trees

Bagging II

- Given a set of n independent observations $Z_1, ..., Z_n$, each with variance σ^2, the variance of the mean $\overline{Z}$ of the observations is given by $\frac{\sigma^2}{n}$
- Averaging a set of observations reduces its variance
- A way to reduce the variance and increase the prediction accuracy of a statistical learning method is to take many training sets, build a separate prediction model with each training set, and average the resulting predictions
- However, this is not really practical since we generally do not have access to multiple training sets

Bagging III

- To work around this limitation and simulate a situation with many training data sets, we can use bootstrapping
- This means that we repeatedly draw random subsets of our training data with replacement
- Generating B different bootstrapped training data sets, we can then create the individual estimates $\hat{f}^1(x), \hat{f}^2(x), ..., \hat{f}^B(x)$ and averaging these predictions obtain

$$\hat{f}_{bag}(x) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}^b(x)$$

Bagging IV

- This method of repeatedly drawing random subsets of the training data with replacement and averaging over the individual estimates is called bagging
- The individual trees are usually rather deep, leading to a low bias and a high variance
- Averaging over the predictions of the individual trees (or using majority voting for classification trees) then reduces the variance
- Hundreds or even thousands of individual trees can be combined with this method
- Bagging has been shown to drastically improve the accuracy of estimations based on decision trees

The Out-of-Bag Error

- There is a very straightforward way to estimate the test error of a bagged model
- The observations not used to fit a given bagged tree are referred to as the out-of-bag (OOB) observations
- For each observation in the training data we can make one prediction based on each of the trees in which that observation was OOB
- Averaging these predicted responses, we can obtain one OOB prediction for each observation
- Based on the OOB predictions for all observations, we can compute the overall OOB mean squared error (for a regression problem) or classification error (for a classification problem)

Variable Importance I

- A disadvantage of bagging is that it can complicate the interpretation of results - it is no longer clear which variables are most important
- To get a summary of the importance of each predictor we can use the sum of squared residuals (for bagging regression trees) or the Gini index (for bagging classification trees)
- The sum of squared residuals (SSR) is a measure for the amount of variance in a data set that cannot be explained by a model
- The Gini index is a measure for the variance across classes in a data set

Variable Importance II

- When bagging regression trees, we can record how much the SSR is decreased due to splits over a given predictor, averaged over all trees
- In the context of bagging classification trees, we can sum up how much the Gini index is decreased by splits over a given predictor, averaged over all trees
- In both cases a large value, i.e. a large decrease in the SSR or the Gini index, indicates an important predictor

Random Subsets of Predictors I

- In addition to only considering a random subset of the training data, the trees in Random Forests are also based on random subsets of the explanatory variables
- Each time a split in a tree is considered, a random sample of m predictors is chosen as split candidates from the full set of p predictors
- This technique helps to further reduce the variance of the final predictions

Random Subsets of Predictors II

- Consider a data set with one very strong predictor and several moderately strong predictors
- Most or all trees considering all covariates will use the strong predictor in the top split
- Then all bagged trees will look rather similar and their individual predictions will be highly correlated
- Since averaging over highly correlated quantities does not lead to a large reduction in the variance, bagging alone will not really help us in this case

Random Subsets of Predictors III

- Random Forests overcome this problem by forcing each split to consider only a random subset of the predictors
- On average, a share of $(p - m)/p$ of the splits will not even consider the strong predictor and give other covariates more of a chance
- This process of "decorrelating" the trees further improves the accuracy of Random Forests

Random Subsets of Predictors IV

- When applying this technique we have to take into account that the choice over the number m of randomly selected covariates to consider at each split can have a distinct effect on the results of a prediction
- Hence, it is important to find a suitable value for m
- We can tune m by computing OOB error rates for different levels of m and then choosing the value that minimizes the OOB error

Boosted Regression Trees

- Boosted Regression Trees are also based on the combination of a large number of individual decision trees
- As opposed to Random Forests, Boosted Regression Trees build their trees in sequence with each tree learning from and improving on the previous one
- The method of boosting does not include any bootstrap sampling, but rather fits the individual trees on a modified version of the original data set
- For Boosted Regression Trees we start off with one tree and then sequentially boost the model's performance by adding more trees
- In the course of this process each new tree slightly improves upon the performance of the model by correcting for the errors of the previous trees

Boosting I

- We start with the prediction that each observation of the dependent variable equals the mean of the observations of the dependent variable
- Then we compute the residuals of this prediction by subtracting our predicted values from the actual ones
- Now we fit a tree that tries to explain these residuals with the explanatory variables
- Next, we multiply the predicted values of the residuals with a learning rate and then add them to our prediction of the dependent variable (in the first round the mean of the observed values)

Boosting II

- By computing the difference between the actual values and our updated prediction we now obtain new residuals
- Again we fit a tree to these updated residuals and make a prediction
- We multiply the predicted values by the learning rate and again add them to our prediction
- This process is repeated until a predefined number of rounds is completed or until additional trees cannot reduce the size of the residuals any further
- The final prediction corresponds to the initial prediction plus the sum of the individual predictions multiplied by the learning rate

Boosting III

1. Start with $\hat{f}(x) = f_0(x) = \overline{y}$ and $r_i = y_i - \overline{y}$ for all observations i in the training data
2. Given a set number of rounds B, for $b = 1, 2, ..., B$, repeat the following:
   - 2.1 Fit a tree f_b to the training data (X, r)
   - 2.2 Update $\hat{f}$ by adding a shrunken version of the new tree: $\hat{f}(x) \leftarrow \hat{f}(x) + \eta f_b(x)$
   - 2.3 Update the residuals: $r_i \leftarrow r_i - \eta f_b(x_i)$
3. Output the boosted model: $\hat{f}(x) = f_0(x) + \sum_{b=1}^{B} \eta f_b(x)$

Loss Functions I

- We have started our boosting algorithm with $f_0(x) = \overline{y}$
- In general, boosting starts with a loss function $L(y_i, f(x_I))$ and is initialized with a constant value γ that minimizes the loss

$$f_0(x) = \underset{\gamma}{\text{argmin}} \sum_{i=1}^{n} L(y_i, \gamma)$$

- The loss function evaluates how close the initial prediction is to the data
- To find the predicted value that minimizes the loss we can take the first derivative of the loss with respect to the prediction and set the sum of the derivatives to zero

Loss Functions II

- The loss function most commonly used is $L(y_i, f(x_i)) = \frac{1}{2}(y_i - f(x_i))^2$
- Using a constant value $\hat{y}$ for $f(x_i)$ and taking the first derivative with respect to $\hat{y}$ we get

$$\frac{\partial}{\partial \hat{y}} \frac{1}{2}(y_i - \hat{y})^2 = \frac{2}{2}(y_i - \hat{y})(-1) = \hat{y} - y_i$$

- Setting the sum of the derivatives of all y_i, with $i = 1, 2, ..., n$, to zero, we obtain

$$(\hat{y} - y_1) + (\hat{y} - y_2) + ... + (\hat{y} - y_n) = 0$$

$$n\hat{y} = \sum_{i=1}^{n} y_i$$

$$\hat{y} = \frac{1}{n} \sum_{i=1}^{n} y_i = \overline{y}$$

- Hence, the overall loss is minimized by starting with $f_0(x) = \overline{y}$

Gradient Boosting

- The way we compute the residuals also depends on the selected loss function
- For gradient boosting, the residual is set to the negative gradient of the loss function
- This means that for each observation i, with $i = 1, 2, ..., n$, and each tree m, with $m = 1, 2, ..., M$

$$r_{im} = -\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}$$

- Hence, the residuals correspond to the derivative of the loss function with respect to our prediction, multiplied by minus one
- Plugging into our loss function, we obtain

$$r_{im} = -\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)} = \frac{\partial}{\partial f(x_i)} \frac{1}{2}(y_i - f(x_i))^2 = -(f(x_i) - y_i) = y_i - f(x_i)$$

Extreme Gradient Boosting I

- Extreme gradient boosting is a specific implementation of gradient boosting that uses a slightly different approach to building its trees
- Two additional parameters for extreme gradient boosting are λ and γ
- λ: regularization parameter to reduce the prediction's sensitivity to individual observations
- γ: minimum loss reduction required to make a further partition on a leaf node of the tree

Extreme Gradient Boosting II

- For extreme gradient boosting we also start with a prediction that assumes that each observation of the dependent variable equals a constant value
- Then we compute the residuals of this prediction by subtracting our predicted values from the actual ones and fit a tree that tries to explain the residuals with the explanatory variables
- To define how deep this tree should be we use post-pruning with a similarity score that is computed for each node in the tree
- The similarity score is given by

$$similarity = \frac{(sum\ of\ residuals)^2}{number\ of\ residuals + \lambda}$$

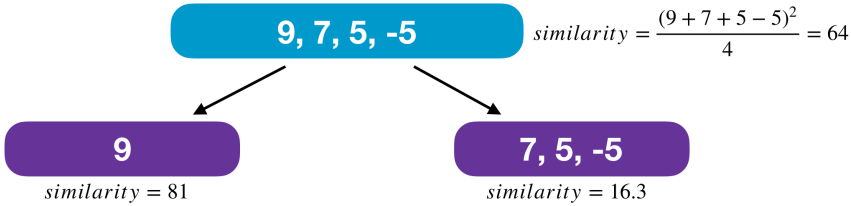
Extreme Gradient Boosting III

- Starting at the leaf nodes of the tree, we use the similarity scores to check how much we gained from the last split
- The gain from any specific split is given by

$$gain = similarity_{left} + similarity_{right} - similarity_{root}$$

- This gain is compared to the parameter γ
- If the gain is larger than γ, we keep the last split
- If the the gain is smaller then γ, we prune the tree by removing the last split and conduct the same comparison with the previous split

Extreme Gradient Boosting IV



$$gain = 81 + 16.3 - 64 = 33.3$$

Extreme Gradient Boosting V

- The same approach based on similarity and gains is also used to determine the split points when creating the trees
- Moreover, a very similar method is also applied to obtain the predictions of the individual trees
- Specifically, the output of any leaf node of a tree is given by

$$output = \frac{sum\ of\ residuals}{number\ of\ residuals + \lambda}$$

- This last equation shows that λ also reduces the weight of any individual observation in the prediction

Extreme Gradient Boosting VI

- The larger λ, the smaller the gains from splitting and the more we will prune our trees
- The larger λ, the lower the weight of an individual observation in the prediction
- The larger γ, the more conservative the algorithm will be and the more we will prune our trees

Shallow vs. Deep Trees

Shallow trees:

- Can only capture rather simple relationships
- Tend to have a high bias and a low variance

Deep trees:

- Can model more complex interactions
- Tend to have a low bias and a high variance
- Can be prone to overfitting

The Optimal Depth of Trees

Random Forests:

- Combine trees that are independent from each other
- That decreases the variance and speaks for deep trees

Boosted Regression Trees:

- Create trees that correct for errors in previous trees
- That decreases the bias and speaks for shallow trees

The Optimal Number of Trees

Random Forests:

- Increasing the number of trees will first increase the accuracy
- At some point the accuracy will stagnate and more trees will only lead to longer computation times

Boosted Regression Trees:

- Initially, increasing the number of trees will also increase the accuracy
- However, more trees will also make the model more prone to overfitting
- Hence, the number of trees is a more sensitive parameter in Boosted Regression Trees and might be set lower than in Random Forests

Parameters to Tune in Random Forests

- `mtry`: number of variables randomly sampled as candidates at each split
- `min.node.size`: minimum number of observations associated with each leaf node
- `splitrule`: criterion to find optimal split points
- `sample.fraction`: fraction of observations to sample in each tree
- `replace`: whether to sample with or without replacement
- `respect.unordered.factors`: handling of unordered factors in covariates

Parameters to Tune in Boosted Regression Trees

- `nrounds`: maximum number of boosting iterations
- `max_depth`: maximum depth of a tree
- `eta`: learning rate of `0<eta<1` (lower value creates model that is slower to compute, but more resistant to overfitting)
- `gamma`: minimum loss reduction required to make another partition on a node
- `colsample_bytree`: fraction of columns used to construct each tree
- `min_child_weight`: minimum number of observations associated with each leaf node
- `subsample`: fraction of observations used to construct each tree