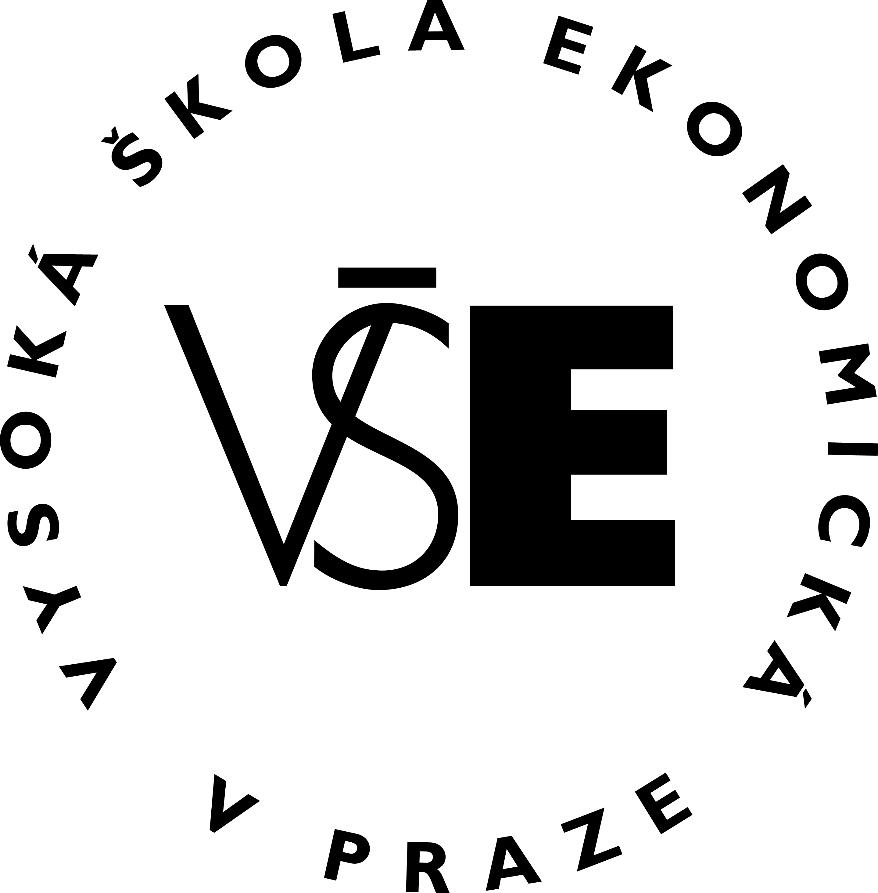
University of Economics in Prague



**Credit Scoring Using Ensemble Machine Learning Methods**

Field of study: Financial Engineering

Department of Banking and Insurance

Faculty of Finance and Accounting

Author of the Master Thesis: Bc. Jan Helcl

Supervisor of the Master Thesis: prof. RNDr. Jiří Witzany, PhD.

Year of Defence: 2018

The Declaration of Authorship

I hereby declare that I carried out the master thesis “Credit Scoring Using Ensemble Machine Learning Methods” independently, using only the resources and literature properly marked and included in the bibliography.

Prague, XXX, 2018 signature of the author

Acknowledgements

…

**Abstract:**

Blah, blah, blah

**Keywords:**

…

**Abstrakt:**

Bla, bla, bla

**Klíčová slova:**

…

Contents

[1 Introduction 1](#_Toc519548359)

[2 Introduction to machine learning 2](#_Toc519548360)

[3 Supervised machine learning 2](#_Toc519548361)

[4 Bias-variance tradeoff 3](#_Toc519548362)

[4.1 Intuition behind bias-variance tradeoff 3](#_Toc519548363)

[4.2 Bias–variance decomposition 6](#_Toc519548364)

[5 Decision trees 8](#_Toc519548365)

[5.1 Classification and Regression Trees 9](#_Toc519548366)

[5.1.1 Splitting criteria 10](#_Toc519548367)

[5.1.2 Stopping criteria 11](#_Toc519548368)

[5.1.3 Prediction 12](#_Toc519548369)

[5.2 Overview of other decision tree learning algorithms 12](#_Toc519548370)

[5.2.1 Iterative Dichotomiser 3 (ID3) 12](#_Toc519548371)

[5.2.2 C4.5 and C5.0 13](#_Toc519548372)

[5.2.3 Chi-square automatic interaction detection (CHAID) 13](#_Toc519548373)

[5.2.4 Multivariate adaptive regression splines (MARS) 13](#_Toc519548374)

[5.3 Decision trees: summary 13](#_Toc519548375)

[5.4 Geometrical interpretation of decision trees 14](#_Toc519548376)

[5.4.1 Different visual representations of decision trees 14](#_Toc519548377)

[5.4.2 Visual examples of decision tree models of different complexities 16](#_Toc519548378)

[6 Ensembles of decision trees 17](#_Toc519548379)

[6.1 Ensemble machine learning 17](#_Toc519548380)

[6.2 Decision trees as base learners 18](#_Toc519548381)

[6.3 Model averaging 18](#_Toc519548382)

[6.3.1 Variance reduction 19](#_Toc519548383)

[6.3.2 Random forest 19](#_Toc519548384)

[6.3.3 Extremely randomized trees 22](#_Toc519548385)

[6.4 Boosting 23](#_Toc519548386)

[6.4.1 Additive modelling 23](#_Toc519548387)

[6.4.2 AdaBoost 24](#_Toc519548388)

[6.4.3 Gradient boosting 27](#_Toc519548389)

[7 Credit scoring 30](#_Toc519548390)

[8 Application of Ensemble Machine Learning in Credit Scoring 31](#_Toc519548391)

[8.1 Dataset description 31](#_Toc519548392)

[8.2 Benchmark model: logistic regression 35](#_Toc519548393)

[8.2.1 Preselection: univariate scoring 35](#_Toc519548394)

[8.2.2 Univariate analysis and coarse classification 35](#_Toc519548395)

[8.2.3 Multivariate analysis: pairwise correlation 36](#_Toc519548396)

[8.2.4 Model estimation and evaluation 37](#_Toc519548397)

[8.3 Basic comparison of selected models 39](#_Toc519548398)

[8.3.1 Unrestricted experiment 39](#_Toc519548399)

[8.3.2 Experiment with preselected variables 41](#_Toc519548400)

[8.4 Variable selection with tree-based models 42](#_Toc519548401)

[8.5 Ensemble models with tuned hyper-parameters 44](#_Toc519548402)

[8.5.1 Random forest 45](#_Toc519548403)

[8.5.2 Extremely randomized trees 45](#_Toc519548404)

[8.5.3 AdaBoost 46](#_Toc519548405)

[8.5.4 Gradient boosting 46](#_Toc519548406)

[8.5.5 Performance comparison: summary 47](#_Toc519548407)

[9 Conclusions 48](#_Toc519548408)

# Introduction

Introduction…

# Introduction to machine learning

Machine learning is a very broad field with no clear and simple definition. Generally, it is a subfield of Computer Science, more specifically a subfield of artificial intelligence, concerned with algorithms that allow machines to learn. These algorithms can be classified into four wide categories by the nature of inputs they require during training.

* **Supervised learning** deals with labeled data. Each observation used during training is pre-classified. The aim is to infer a function that links an object to a specific output value.
* **Unsupervised learning** uses unlabeled data only. The goal is to find a structure that describes the data reasonably well.
* **Reinforcement learning** is inspired by behavioural psychology. There are no explicit input-output pairs during the training. The system learns through interaction with its environment based on rewards/punishments for its behaviour.
* **Apprenticeship learning** uses indirect hints derived from a teacher´s behaviour.

Credit scoring is, in principle, quite simple. We need a model, which can be used to discriminate between good and bad debtors. This model is built using data about past applications where we know, whether these credits turned out to be good or bad. This, in other words, resembles a classic supervised machine learning problem. In the next section we will investigate the principles behind supervised machine learning a bit further.

# Supervised machine learning

As mentioned above, supervised machine learning uses labeled data to extract patterns that can be used to predict an unknown label of a previously unseen observation. Typical use cases include among others image recognition, natural language processing (e.g. spam detection) or housing prices prediction.

Let’s first define the concept more formally. When talking about supervised learning, analogies to human reasoning or (especially in case of artificial neural networks) the way brain biologically works are often made. This phenomenon can, however, be understood mathematically as a function approximation problem[[1]](#footnote-1). Suppose is some underlying function that links input to output. Our goal is to find that approximates well enough for the given purpose.

We can further classify supervised learning problems as follows:

* For i.e. the output variable is a real number we speak about regression.
* For where is a finite set we speak about classification.

The input is a row vector of explanatory variables.[[2]](#footnote-2) During the training phase we use an by matrix (where is the number of observations and is the number of explanatory variables) and a vector of outputs to find.

There are numerous machine learning algorithms that can be used to find such a function. Before diving into details of some of these methods selected for the purpose of this thesis, we will first closely examine the error of this approximation.

# Bias-variance tradeoff

Ultimately, our goal is to build a model that can generalize well. This means we need a model that shows good prediction capability on previously unseen observations. The importance of testing on samples that were not used during training cannot be exaggerated. As illustrated bellow, it is actually easy to find a model that can label 100% of the training set correctly, given we can choose models that are arbitrarily complex.

## Intuition behind bias-variance tradeoff

To get some intuition first, let’s set up a simple simulation to demonstrate the relationship between generalization error and model complexity. To do so we need to introduce two sources of error (these will be more formally defined in the next section):

* **Bias** is an error arising from erroneous assumptions in the model. High bias can cause an algorithm to miss the relevant relations between explanatory variables and target output (this is often referred to as underfitting).
* **Variance** is an error coming from sensitivity to small fluctuations in the training set. High variance can cause an algorithm to fit the random noise in the training data, rather than the true relations (often referred to as overfitting).

Our data will come from a simple statistical model:

.

Where is normally distributed with 0 mean and 0.5 variance and is independent of . We will repeatedly draw samples of 25 observations from this population, 90% of the sample will be used as a training set to fit a polynomial of a given degree. The out-of-sample performance will be measured using the other 10%. Mean squared error (MSE) will be used as a metric for assessing the performance:

Where is the i-th observed value of the output variable, is the corresponding value predicted by the model and is the number of observations.

We will fit polynomials up to 12 degrees (50 iterations for each degree).

Figure 1 shows fitted curves for polynomials of selected degrees. In the upper left corner we can see the results of trying to fit a simple line to the generated data. What we observe is a clear example of underfitting. The model is not complex enough to capture the non-linear nature of the data. To put it in a different way, we can see that this model results in very high bias (due to the wrong assumption of linearity) but small variance as all the lines look very similar. The upper right figure shows fits of polynomials of degree 3. Intuitively, this seems to be the best model among the four. The blue line representing the average fit is quite close to the underlying function . Moreover, most of the green curves look very similar to each other. In the bottom of Figure 1 we can see two clear examples of overfitting. While the blue curve is close to the underlying function (i.e. we see low bias), the models are very sensitive to small variations in the data and each fitted curve is very different from the others.

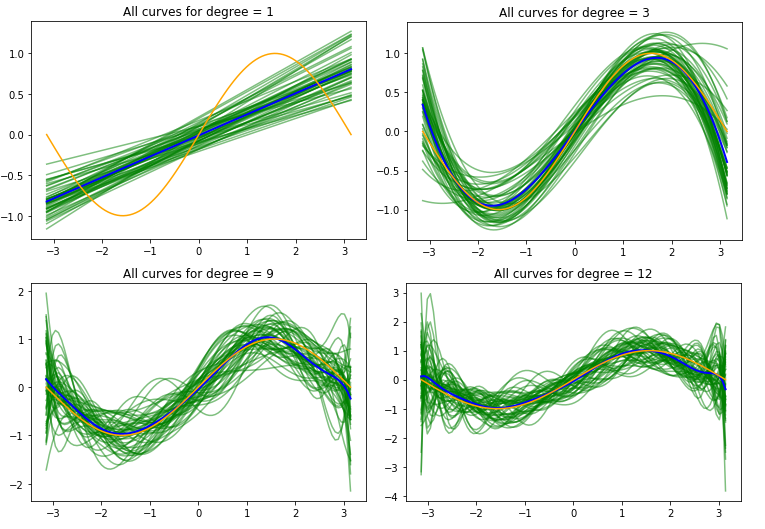


Figure 1: Curves for selected degrees of polynomials. Underlying function is shown in orange. Curves fitted during all 50 iterations are shown in green. The blue curve represents the average fit.

Next, we will have a look at the performance of the models. Figure 2 shows average train and test scores of polynomials of different degrees.

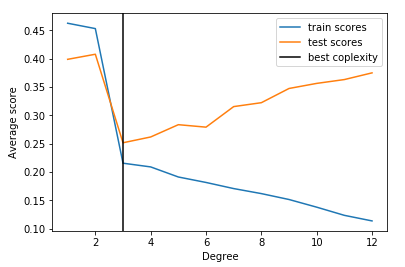


Figure 2: Average train and test performance of polynomials of different degrees (each point represents the average performance of all 50 models of the given degree).

As expected, we see that the polynomial of degree 3 provides the best out-of-sample performance. Note, how the training errors decrease with increasing model complexity while test errors start to go up after reaching certain degree of complexity.

We will get back to this simulation one more time after exploring the relation between bias and variance more formally in the next section.

## Bias–variance decomposition

Bias-variance decomposition was first introduced in the context of linear regression. If we assume , where and , we can derive the expected prediction error of . The derivation proceeds as follows[[3]](#footnote-3):

,

.

Since by definition and since we can rewrite:

,

.

As :

,

.

Finally, we can rewrite:

,

.

Irreducible error is inherent to the problem and therefore cannot be reduced. When searching for the right model, we are looking for optimal balance between bias and variance. Models with low level of complexity tend to have high bias and low variance, whereas largely complex models usually have low bias but suffer from very high variance. The relationship is demonstrated by Figure 3.

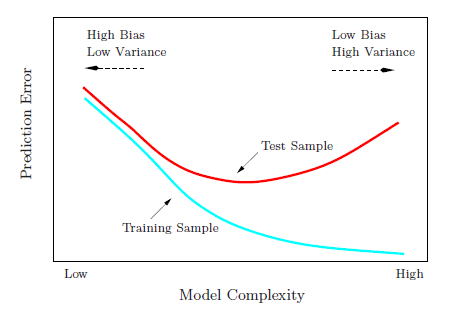


Figure 3: Bias-variance tradeoff, taken from Elements of statistical learning[[4]](#footnote-4)

Now, let´s return to our previous simulation and examine the test results more closely. In Figure 4 we can clearly see the variance (orange curve) increasing with the model complexity (degree of polynomial), bias (in blue) jumping to almost zero between degree 2 and 3 and the irreducible error as the difference between the green curve representing total error and the red curve (bias2 + variance).

**Note:** I introduced the bias-variance tradeoff concept in the context of regression as I find it more intuitive and easier to visualise. We could, however, derive it for classification problems as well. For derivation of a general decomposition of an arbitrary loss function see Domingos 2000[[5]](#footnote-5). Alternatively, in case of probabilistic classification, we could use the expected squared error of the predicted probabilities with respect to the true probabilities and decompose it as was shown above.[[6]](#footnote-6)

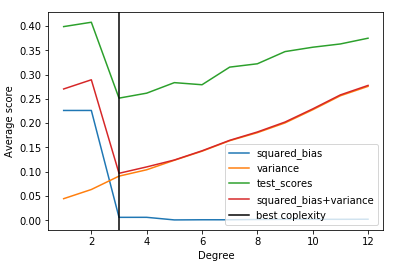


Figure 4: Bias-variance decomposition of the experiment’s results

# Decision trees

Representations of knowledge as tree-like structures can be found across many fields. It is a simple and very intuitive way to visualise what we know about some phenomena in the form of if-then statements resulting in a readable flowchart. Figure 5 shows an example of a very simple decision tree.

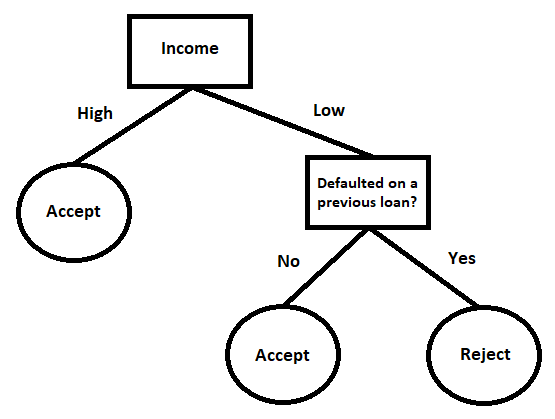


Figure 5: Very simplistic example of a tree that could be used during loan approval process

Decision trees are also among the most widely used supervised machine learning models. Building a decision tree can be seen as specialization in the space of hypotheses (trees) starting from an empty tree. The goal is to find a tree consistent with the data. The training data are iteratively partitioned into smaller and smaller subsets that are increasingly more homogenous. This procedure is known as *„top down induction of decision trees“*(TDIDT), sometimes also called “*divide and conquer”* method.[[7]](#footnote-7) Below is the general algorithm described in pseudocode.

Algorithm 1: General tree building procedure

1. Select one input variable as the root of a new sub-tree.
2. Partition the data according to values of this variable into subsets and create a node for each subset.
3. For each node:
   1. If stopping criterion is not met: go to point 1.
   2. Else: stop.
4. Output the trained model.

Various approaches differing in the way the input variable is selected, the partitions are made and stopping conditions has been proposed. Next, I will introduce one particular decision tree building algorithm that will later be used to construct more complex ensemble models. I will briefly introduce some of the other important approaches to decision tree building in the next section.

## Classification and Regression Trees

Classification and regression trees (CART) are a non-parametric decision tree learning technique that produces either classification or regression trees, depending on whether the output variable is categorical or numeric, respectively. It is probably the most commonly used decision tree algorithm.

CART use numeric inputs only, all categorical variables have to be converted to numerical. Most common approach is to use dummy variables (creating a set of binary variables indicating which category is present). At each node a binary split is made.

### Splitting criteria

At each node the first task is to select the best input variable and the cutting point to base the split on. This is achieved in the following way.

Algorithm 2: CART splitting criteria

1. For each input variable:
   1. Find all potential cutting points by sorting values of the input variable in ascending order, leaving out duplicate values and the last value.
   2. Evaluate all potential cutting points by splitting the data accordingly and measuring impurity of resulting subsets (impurity in output variable is measured).
   3. Remember the best split (cutting point and value of the metric used to measure impurity).
2. Choose the input variable that resulted in the best split, create nodes for both resulting subsets.

For classification tasks two measures of impurity are commonly used: Gini index and Information gain.

Gini index is defined as:

Where: C is the number of subsets (C = 2 in case of CART), K is the number of categories of the output variable (K = 2 for binary classification), is the probability of k-th value of input variable occurring in c-th subset, is the Gini index of c-th subset, is the weight computed as number of observations belonging to c-th subset divided by total number of observations in the original set. The lower the impurity (Gini coefficient) the better the split.

Information gain is a concept based on information theory. It is the difference between Information entropy of the original set and the weighted sum of entropies of the resulting subsets. Entropy is defined as:

Where: K is the number of categories of the output variable (K = 2 for binary classification) and is the percentage of the given class of the input variable in the set. Information gain is then simply:

Where C is again the number of subsets (C = 2 in case of CART). Naturally, the higher the Information gain the better the split.

For regression problems Mean squared error (MSE) defined earlier is often used. Split that resulted in the lowest weighted sum of MSE is considered the best.

Note, how the algorithm proceeds in a greedy fashion. At each node a locally optimal split is made. This selection is conditional on all previous splits and doesn’t take into account any later splits.

### Stopping criteria

Various stopping criteria can be specified by the user to control the complexity of resulting models.

* **Maximum depth** refers to the number of levels on which splits are made. (Figure 5 shows a tree with depth = 2). The higher the maximum depth the more complex the final model.
* **Minimum number of samples in node** is the number of samples required to be in a node to perform a split. The higher the number the simpler tree we get.
* **Minimum number of samples in leaf nodes.** Leaves are the final nodes that are not being split any further (the tree in Figure 5 has 3 leaves). Same logic applies, the higher the number of samples required to be in leaves the simpler the model.
* **Minimum impurity decrease** is a user defined threshold. All splits are required to decrease the impurity at least by this amount. This is a way to avoid overfitting as only splits with significant gain in accuracy are being made.

There is one natural case when the algorithm stops. It is when a node achieves the lowest possible impurity. For classification tasks this means that all observations in the node are of the same class, for regression the value of output variable in the node is constant. Now it is easy to see that in case none of the above mentioned stopping criteria is defined the model is trained to label all training data with 100% accuracy. Such models, however, suffer from high generalization errors as they have very high variance (but low bias).

Several ways to simplify such fully grown trees has been developed. These methods are commonly called tree pruning. I won’t describe these as I won’t need them for building the ensembles. Interested reader can see for example Hastie, Tibshirani, Friedman 2008[[8]](#footnote-8).

### Prediction

Making prediction using a built decision tree is quite straightforward. At each node the value of respective variable is tested and sent to the next node accordingly until a leaf node is reached. In case of classification, majority vote is performed in the leaves. In other words, the prediction represents the most frequent class among the training observations that fell into the given leaf node. For regression problems average value of the output variable of these observations is used.

**Note:** When doing classification, we are often more interested in the probability that the given observation falls into the respective class. Credit scoring (estimation of probability of default) represents such a case. Proportions of classes in the leaf node can be used as estimates of these probabilities.

## Overview of other decision tree learning algorithms

As mentioned earlier there are many ways in which decision trees can be built. What follows next is a brief overview of some of the alternatives to Classification and Regression Trees.

### Iterative Dichotomiser 3 (ID3)

ID3 is an algorithm invented by Ross Quinlan. It can be applied to discrete data only. At each node a variable resulting in the highest Information gain is selected and a new node for each category of the variable is created. Naturally, every variable can be used only once, when there are no variables left the algorithm stops. This procedure results in relatively simple decision trees. For more information refer to the Quinlan’s 1986 book.[[9]](#footnote-9)

### C4.5 and C5.0

C4.5 is an algorithm developed by the same author, it is an extension to ID3. It can handle continuous inputs by performing online discretization, procedure similar to that used in CART. It also handles missing values by ignoring them when computing the Information gain. C4.5 uses a pruning method to simplify the final tree.[[10]](#footnote-10) C5.0 is an improvement to C4.5 that Quinlan is commercially selling.[[11]](#footnote-11)

### Chi-square automatic interaction detection (CHAID)

CHAID is a technique that uses Person’s Chi-square tests of independence when performing the splits. Important advantage of this method and similar approaches is that it results in simpler trees as only statistically significant split are made (there is no need for pruning). As in ID3 and C4.5/C5.0 the splits are multiway. For more details see Magidson 1994[[12]](#footnote-12).

### Multivariate adaptive regression splines (MARS)

MARS can be viewed as generalization of stepwise linear regression or as a modification of CART. It produces complex binary trees (need for pruning) with linear function at each leaf node (instead of a constant as in CART).For detailed discussion on MARS refer to Hastie T., Tibshirani R., and Friedman J.H. (2009)[[13]](#footnote-13).

Other less commonly used techniques are available as well: evolutionary programing-based methods[[14]](#footnote-14), Markov chain Monte Carlo[[15]](#footnote-15) or bottom-up search[[16]](#footnote-16).

## Decision trees: summary

Decision trees comes with several important advantages.

* They are **simple and easy to interpret**. Decision trees can be easily visualized and can be interpreted intuitively.
* Decision trees are **computationally cheap**. Even large amounts of data can be analysed using standard resources in reasonable time.
* Most approaches represent **non-parametric methods**. There are no distributional assumptions on training data or prediction errors. There are also no independence or constant variance assumptions.
* There is **in-built feature selection**. Trees with limited depth (or limited using another above mentioned stopping criteria) are relatively robust to presence of irrelevant input variables, these are not used in any split.

Naturally, it also comes with certain limitations.

* Standalone decision trees tend to be **less accurate** than some of the other machine learning models[[17]](#footnote-17).
* Fully grown trees are **very sensitive to small variations in training data**. This often leads to the need for pruning.
* Some decision tree learning algorithms require **discretization of input variables**.

Later chapters on ensemble machine learning will show how some of these advantages can be leveraged to overcome the limitations by combining more decision trees into a single predictive model.

## Geometrical interpretation of decision trees

### Different visual representations of decision trees

Finally, we will have a closer look at the geometrical interpretation of decision trees. From now on, when I am discussing decision trees I refer to CART unless stated otherwise.

By performing binary splits the feature space is partitioned by lines parallel to axis. In two dimensions this results in creating areas of rectangular shape as shown in Figure 6. In more dimensions we speak of hyperrectangles.

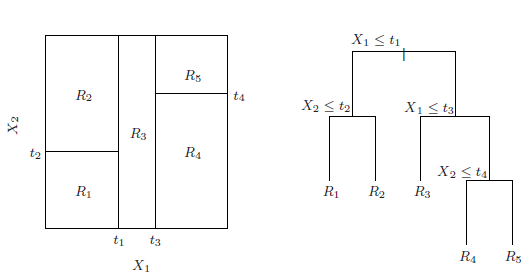


Figure 6: Partitions of two dimensional feature space by a decision tree. Taken from Elements of Statistical Learning.[[18]](#footnote-18)

In Figure 6 we can see two graphical representations of the same decision tree trained using two input variables and . On the right, we see the now familiar tree-like structure. On the left, the partitions the model makes are visualized.

In case of classification the most frequent class within each region is being predicted. When performing regression, prediction is constant within the given region. Figure 7 shows prediction surface of the same model. This is done by extending the space by a third dimension representing the output variable.

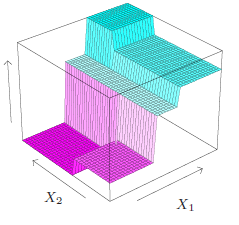


Figure 7: Prediction surface of the same regression tree. Taken from Elements of Statistical Learning.[[19]](#footnote-19)

### Visual examples of decision tree models of different complexities

I will end the discussion on standalone decision trees by showing visual examples of decision tree models of different complexities.

The first problem is a regression one. The data are generated by the same statistical model as the one used in Chapter 2 on bias-variance tradeoff ( + normally distributed noise), so there is only one input variable. Figure 8 shows prediction curves for regression trees with different maximum depths.

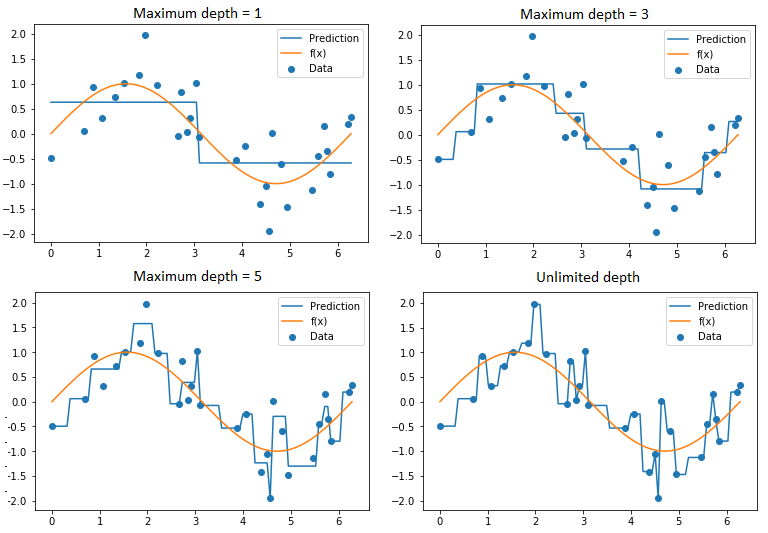


Figure 8: Predictions of regression decision tree models of different depths

The smooth curve is approximated by a stepwise constant function. Note, how with increasing depth the model starts to overfit the data.

Second, we will have a quick look at a simple, artificially created 2D binary classification problem. The data are generated as two interleaving half circles with some normally distributed noise added. This would be a difficult problem for some types of classifiers.

Figure 9 shows the results, two classes are represented by red and blue dots. The areas in which given classes are predicted are filled with respective colours.

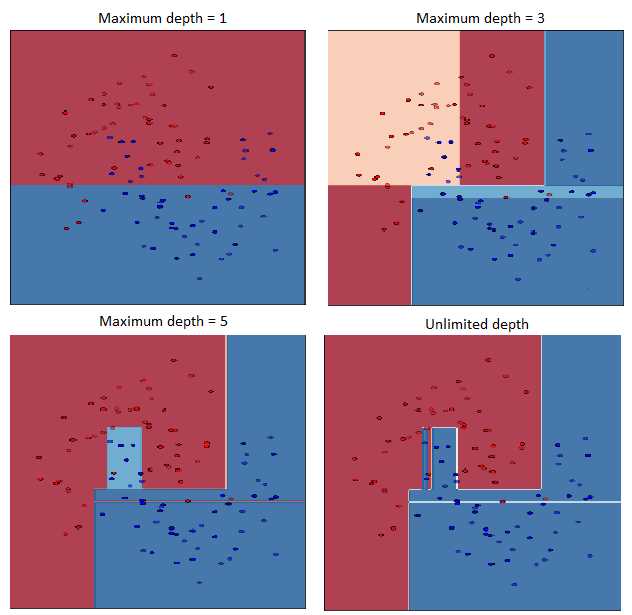


Figure 9: Partitions made by decision tree classifiers of different depths

Again we see the clear underfitting of the tree with maximum depth limited to 1 and signs of overfitting of the unrestricted model.

# Ensembles of decision trees

## Ensemble machine learning

The goal of ensemble machine learning is to enhance the predictive capability by combining more models into one more powerful one. The underlying models are called “base learners”, the ensemble is often referred to as a “strong learner”. Generally, the base learners within one ensemble can be of different types. Most common approaches, however, combine more models of the same type. Linear models and decision trees are often used as base learners. There are several ways in which the models can be combined. What comes next is a brief overview of the most common approaches.

* The principle behind **model averaging** is to combine models trained independently and average their predictions (or perform voting in case of classification) to achieve better accuracy and/or robustness.
* In **boosting**, on the other hand, the base learners are trained sequentially. Each successive model is trying to correct the errors made by previously built models.
* **Stacking** presents yet another common approach. Predictions made by the base learners are used as input variables to a meta-learner (sometimes in combination with some of the variables used to train the base learners), predictions of the meta-learner are used as predictions of the ensemble.

For the purpose of this thesis two examples of model averaging and two examples of boosting have been selected. In all four cases, the base learners are of the same type - CART.

## Decision trees as base learners

CART and some other decision tree learning algorithms are often used as base learners. They are not computationally expensive, so even large ensembles can be trained in reasonable time. The base learners are required to be either simple (low variance, high bias) or complex (low bias, high variance) depending on the ensemble technique used. By controlling the depth of the trees we can easily adjust the properties of decision trees to fit the given ensemble learning method. Decision trees can also be further adjusted easily to fit the specific needs of the ensembles. Decision trees can be used to both classification and regression, so the same ensemble techniques built upon them can also be used for both of these tasks. By combining trees into ensembles we lose their easy interpretation, other visualization techniques has been proposed to, at least partially, overcome this difficulty.

## Model averaging

Two ensemble learning methods based on model averaging will now be introduced: random forest and extremely randomized trees. Before diving into details of these particular methods let’s first have a look on how model averaging methods work in the context of bias variance tradeoff.

### Variance reduction

The following models use averaging as means of reducing the variance. Recall that if we have independent and identically distributed variables with variance the variance of their average equals to:

In case the variables are not independent we get:

Where is pairwise correlation. In the context of model averaging, the random variables represent the predictions of base learners. These are identically distributed as they are of the same type (CART) trained using the same dataset. We can see that by adding more learners we can eliminate the second term. The correlation among the base learners limits the benefits of averaging.

### Random forest

Random forest is one of the most widely used supervised machine learning algorithms. It shows very high predictive power in many areas. It also scores high in many studies comparing the performance of different learning methods[[20]](#footnote-20). The algorithm proceeds as follows[[21]](#footnote-21).

Algorithm 3: Random forest

1. For b = 1, 2, …, :
   1. Draw observations from the original dataset with replacement.
   2. Train a randomized CART using the samples drawn in previous step.
   3. Store the trained tree.
2. Output the trees.

Where is the number of base learners to be trained and is the number of samples in the training dataset.

#### Randomized CART

Random forests use a special version of decision trees. These differ from those introduced in Chapter 5 in only one aspect. At each node when selecting the next split only a subset of input variables is considered. Typical values of are or , where is the number of input variables in the training set.

#### Hyper-parameters of Random Forest

The user needs to specify the number of base learners to be trained (), the value of (in absolute terms or, more often, as a function of ) and the parameters of the trees: the stopping conditions and the metric used to evaluate the potential cutting points when making splits.

As explained earlier, the idea behind model averaging is to reduce the variance of the resulting ensemble as compared to single base learners. Therefore, we need an ensemble of very deep decision trees with low bias and high variance (which is lowered by averaging). To de-correlate the trees Random Forests use two sources of randomness. Each tree uses a different input data (drawn from the same empirical distribution) and a different subset of input variables is used on each split.

Generally, the higher the the better the model’s performance as increasing does not lead to overfitting. When choosing we want to find a value that results in good performance of the model while still being computationally feasible (naturally, the more decision trees we have in the ensemble the more computationally expensive the task becomes).

To avoid overfitting the complexity of base learners can be bound. In spite of that, unlimited depth is used in most cases, on same occasions limited (but still relatively deep) trees show better performance[[22]](#footnote-22).

#### Prediction

When doing classification majority vote among the trees in the ensemble is performed. In regression settings the final prediction is simply the average of predictions of the base learners. In case of probabilistic classification the probabilities predicted by the base learners can be averaged.

#### Special cases

Certain settings of parameters lead to special cases that are described in literature as distinct machine learning models.

##### ***Bagging***

When we get the bootstrap aggregating algorithm (commonly shortened to bagging). It is an ensemble machine learning method that can be used to combine base learners of any type. Decision trees are a natural choice thanks to their properties. It usually shows lower predictive capability as the trees are more similar to each other so there is lower reduction in variance.

##### ***Random subspace method***

If we, on the other hand, disable the random resampling (the first step before training the randomized CART) we get to the (local) random subspace method (also called attribute bagging or feature bagging), one of the predecessors of random forests. As with bagging random subspace method usually underperforms random forest because of higher correlation among the underlying trees.

#### Random Forest: Summary

Random forests present probably the best known ensemble machine learning method. This is because they come with many advantages.

* They tend to be very **accurate**.
* They are still more **computationally feasible** than some other advanced machine learning models (such as neural networks).
* They are relatively **robust to overfitting** and **outliers**.
* They are a **standard part of** many **data analytics tools** (both commercial and open source).

Although complex in nature, random forests are relatively easy to use. There are not as many hyper-parameters to be tuned as when using some other supervised learning methods. They are quite robust to outliers as these gets isolated in small “boxes” in the feature space[[23]](#footnote-23) and only some of the base learners are trained using these observations. Random forests are also robust to overfitting, when necessary overfitting can be controlled by tuning a single parameter – the depth of the trees. As always, there are some limitations as well.

* On some problems, random forests **get outperformed by other approaches**.
* They are **more computationally demanding** then standalone models.
* They **lose the natural visual interpretation** of single decision trees.

### Extremely randomized trees

Extremely randomized trees (ExtraTrees) are based on the same principles as random forest. They also belong to the model averaging family of ensemble machine learning models and use decision trees as weak learners. They differ in the sources of randomness used to de-correlate the individual trees in the ensemble.

The ensemble is composed of extremely randomized CART fitted to the same data (no resampling in this case).

#### Extremely randomized CART

As with randomized trees used in random forest, only a subset of input variables is considered at each node. Moreover, for each of these variables a cutting point is selected at random. These potential splits are then evaluated using a specified metric (e.g. Gini index, Information gain, MSE).

The rationale behind this approach, according to the authors, is to take full advantage of model averaging by further randomization of the base learners[[24]](#footnote-24).

#### Hyper-parameters of Extra Trees

In general, ExtraTrees have the same hyper-parameters as random forest. The authors, however, propose certain settings as default and claim that these perform well in most cases. The depth of the trees is expected to be unlimited. To avoid overfitting the authors suggest setting a minimum number of samples in node. The parameter determines the strength of the variable selection process (the degree of randomization), recommended values are for classification and for regression[[25]](#footnote-25). The number of trees in the ensemble regulates the strength of the variance reduction of the ensemble model aggregation. Same logic as with random forest applies, the higher the the stronger the reduction in variance. Although, after certain number of trees the marginal gain can become negligible. The goal is, again, to find a value that results in good model while maintaining computational feasibility.

#### Special case: Totally random trees

When we set we get an ensemble of decision trees in which all splits are completely independent of the output variable. The predictive power of such an ensemble would, naturally, be poor unless there are very many trees in the ensemble. Totally random trees can, however, be used in anomaly detection. A technique called isolation forest (mentioned earlier in a footnote) uses the average number of splits needed to separate an observation from all the others as a measure of normality[[26]](#footnote-26).

#### Extra trees: summary

Basically, extremely randomized trees can be thought of as a variant of random forest. Therefore, it has very similar advantages and limitations. The authors claim that ExtraTrees perform better than random forest on many problems[[27]](#footnote-27). ExtraTrees are computationally less demanding as there are less splits to be evaluated. It is a newer and less known machine learning method and is not implemented in many analytics tools.

## Boosting

Boosting algorithms are a family of ensemble machine learning methods that primarily focuses on reducing the bias (it has been shown that boosting also reduces variance[[28]](#footnote-28)). Unlike in model averaging, the base learners within these ensembles are very simple. They are often referred to as weak learners and, in themselves, perform only slightly better than random guessing. The goal is to combine such weak learners to produce a strong ensemble model.

### Additive modelling

Boosting is a way of fitting an additive model in the form:

Where is a simple basis function of multivariate input and parameters and is the weight of m-th basis function in the additive model. These models are usually fit by minimizing a loss function averaged over training data points[[29]](#footnote-29),

Solving such problems can be a computationally intensive task. Forward stagewise additive modelling approximates the solution by sequentially adding new basis functions to the model without adjusting the parameters and weights of those that have already been added[[30]](#footnote-30).

Both of the boosting algorithms used in this thesis are examples of forward stagewise modelling and use simple CART models as basis functions (parameter refers to the splitting variables and cutting points).

### AdaBoost

Adaptive boosting, commonly shortened to AdaBoost, is an ensemble machine learning algorithm developed by Yoav Freund and Robert Schapire, who won the 2003 Gödel Prize for their work. It was first introduced in the context of binary classification, but it can be generalized to multiple classification and regression as well. It can be used with weak learners of any type, decision trees are the most common choice.

AdaBoost is adaptive in the sense that at each stage it focuses on samples that has been mislabelled by previous weak learners. This is done by reweighting the samples at each iteration. The algorithm proceeds as follows[[31]](#footnote-31), assuming binary classification where

The predictions are simply computed as the sign of the weighted sum of the weak learners’ predictions,

Algorithm 4: AdaBoost

1. Initialize the observation weights:

1. For m = 1, 2, …, M:
   1. Train a weak learner using
   2. Compute the in-sample weighted error

.

* 1. Set the coefficient to:
  2. Update the model:

.

* 1. For update the weights:

Where is a normalization factor chosen so that is a valid probability distribution.

1. Output the model

In case of probabilistic prediction, the class probabilities of an input sample are computed as the weighted mean predicted class probabilities of the classifiers in the ensemble.

#### Exponential loss function

Many machine learning and statistical methods can be viewed as procedures for minimizing a loss function (also called cost function or objective function). Although not originally derived this way, AdaBoost can be shown to minimize the exponential loss function,

This is done in a greedy fashion as outlined in the section on additive modelling. It is easy to see that by minimizing this function we strongly penalize the misclassified cases (cases where the sings of and does not match). AdaBoost can be viewed as a form of functional gradient descent, as observed by Mason et al.[[32]](#footnote-32). This understanding has led to the generalization of boosting to a wide range of other loss functions and learning problems, such as multiclass classification or regression[[33]](#footnote-33).

#### Hyper-parameters of AdaBoost

As usual in ensemble learning, the user has to specify the maximum number of base learners to be trained. In case of prefect classification the algorithm stops prematurely. Unlike in previous cases with increasing number of base learners in the ensemble the model starts to overfit the training data. One way of avoiding overfitting is to specify a learning rate. The contribution of each base learner is then shrank by this proportion. There is a natural trade-off between learning rate and the number of base learners.

Parameters of the base learners can also be specified. When using CART as base learners, maximum depth is usually set to 1. Such trees are sometimes called decision stumps and represent the simplest form of decision trees with only one split.

#### AdaBoost: summary

AdaBoost is a classic boosting algorithm. It has several major advantages.

* AdaBoost tend to be **very accurate** on many problems.
* It is **easy to use** as there are not many hyper-parameters to be tuned.
* AdaBoost is **relatively fast** as the base learners are usually very simple.

As always there are certain limitations.

* AdaBoost **is not very robust to overfitting** and, unlike in random forest, the number of weak learners needs to be chosen carefully.
* AdaBoost is **sensitive to outliers** as it focuses primarily on hard-to-label samples.

Many alternative boosting methods has been proposed since the introduction of AdaBoost. These methods are developed to minimize other loss functions (e.g. LogitBoost) or to be more robust (e.g. BrownBoost).

### Gradient boosting

Gradient boosting is a very popular ensemble machine learning method. It proved to preform extremely well on many different data sets. In recent years gradient boosting based solutions have won many machine learning competitions[[34]](#footnote-34).

Gradient boosting is based on the idea that boosting can be interpreted as a loss function minimization problem. It can be seen as a generalization of AdaBoost that can be used to minimize an arbitrary differentiable loss function. Generally, gradient boosting can be performed using weak learners of any type, decision trees are, again, the most common choice.

Similarly to AdaBoost gradient boosting updates the model iteratively in a greedy fashion. The way each weak learner is constructed is, however, very different. The logic behind gradient boosting is best explained in the context of least-squares regression. At each iteration what we have is an imperfect model and want to improve this model by adding a new weak learner such that:

Gradient boosting solution to this problem is based on the observation that perfect choice of would mean:

Therefore at each stage gradient boosting fits a regression model with the vector of residuals as the output variable. We can generalize this idea to loss functions other than and to classification problems as well. This follows from the fact residuals are the negative gradients of the loss function with respect to . When using different loss functions at each stage a regression model is fit to the so called pseudo-residuals computed as the negative gradient of the respective loss function. The general algorithm proceeds as follows[[35]](#footnote-35).

Algorithm 5: Gradient boosting

1. Initialize a model with constant value:
2. For m = 1, 2, …, M:
   1. Compute the pseudo-residuals:

for

* 1. Fit a base learner to the pseudo-residuals (using original input variables as inputs and vector of pseudo-residuals as the output variable)
  2. Compute the multiplier :
  3. Update the model:

1. Output the model

Predictions are, again, the weighted sum of base learners’ outputs.

#### TreeBoost

As mentioned above gradient boosting is typically used with fixed size decision trees as weak learners. In this case Friedman proposes a modification to the general algorithm. At each stage a regression tree is fit to the pseudo-residuals giving hyperrectangular terminal regions (as explained in the section on geometric interpretation of decision trees). An optimal value of is chosen for each terminal region instead of having one single for the whole tree. This is done by solving:

And so the model is updated as:

In his paper Friedman also derives specific gradient boosting algorithms for several different loss functions for regression as well as for both binary and multiclass classification[[36]](#footnote-36).

#### Hyper-parameters of gradient boosting

When training a gradient boosting model we need to specify the loss function suitable for the problem at hand. As usual in ensemble learning we need to choose the number of base learners to be trained. Again a learning rate can be set to shrink the contributions of each weak learner.

We also have to specify the parameters of the base learners. Maximum depth (or, alternatively, maximum number of leaves) controls the level of interaction allowed among the input variables. Generally, we only want to model dominant interactions so we usually use very simple decision trees. According to Hastie et al.[[37]](#footnote-37) experience indicates that works well for most problems. Moreover, the results are usually fairly insensitive to particular choices within this range.

There is also another modification of the general algorithm which is inspired by the bagging method already mentioned. At each iteration only a subset of the original training data (drawn at random without replacement) is supplied to the weak learner. The subsample size is defined as constant fraction of the original training set. When we speak of stochastic gradient boosting. This modification often leads to significant improvement of the results. Friedman observed that leads to good results for most data sets[[38]](#footnote-38). This modification also improves computational efficiency.

#### Gradient boosting: summary

Gradient boosting has become very popular among machine learning practitioners and is used in many different areas. Here are some of its main advantages.

* Gradient boosting **performs very well on many data sets**.
* It is **very flexible** as it can be used to minimize an arbitrary differentiable loss function. By selecting a proper loss function it can be made more robust than AdaBoost.
* It is **still easier to use** than some of the most complex machine learning models (e.g. artificial neural networks).

Naturally, gradient boosting has its own limitations.

* It is **more difficult to use** than other before mentioned ensemble methods as it has more hyper-parameters to tune.
* Gradient boosting is **more computationally demanding**.
* It is **less prone to overfitting** than random forest or ExtraTrees.

Gradient boosting gained its popularity thanks to a good balance of its complexity, computational efficiency and accuracy. It is much easier to use and faster to train than some of the modern (deep) learning techniques while often being competitive in terms of predictive capability.

# Credit scoring

# Application of Ensemble Machine Learning in Credit Scoring

Finally, we get to the most important part of this work which is the application of the selected ensemble machine learning models to a real-world credit dataset. The results of these models will be compared to those of the standard approach represented by a logistic regression model.

While the main goal is to compare the selected approaches in terms of out-of-sample predictive capability, my aim is to answer other important questions as well. I am interested in how difficult and time consuming it is to train the models, especially how hard is it to find the right hyper-parameter settings. I also want to demonstrate how input variables selection affects the performance of the models. Along the way I will also make comments on the practicality of these approaches and suggestions for further research.

The whole development was executed within the Python environment. Python is an interpreted high-level general purpose programming language[[39]](#footnote-39). It is known for its design philosophy that emphasise code readability and availability of many specialized high quality open-source libraries. Libraries that were most important during the development are: Pandas[[40]](#footnote-40) and NumPy[[41]](#footnote-41) for data handling, scikit-learn[[42]](#footnote-42), StatsModels[[43]](#footnote-43) and XGBoost[[44]](#footnote-44) for modelling and Matplotlib[[45]](#footnote-45) and Seaborn[[46]](#footnote-46) for visualizations.

## Dataset description

As mentioned earlier the dataset used for this thesis comes from an Estonian peer-to-peer lending platform Bondora. The dataset was obtained from the company’s website[[47]](#footnote-47). It is the company’s policy to present the (potential) investors with detailed data so they can make informed decisions while investing their money.

As of June 2018, the data set contains information on more than 58 000 loans provided from 2009 to 2018. Since my goal is to predict default over one year horizon, only loans granted before 1st June 2017 were used during modelling. There were total of 36 565 of such loans.

The binary target variable “1Y\_default” was created using the provided columns “DefaultDate” and “LoanDate”. When the difference between these two dates is lower than 365 days “1Y\_default” equals 1, it equals 0 otherwise. Bondora define defaulted loans as those where the borrower missed the scheduled payment by more than 60 days.

The loans provided through Bondora platform show a high-risk profile. Of the loans used in modelling 29.732% defaulted by the end of the first year. We can also clearly see the rapid increase in riskiness during recent years. Number of credits provided also showed sharp increase during this period. In Figure 10 we see the number of loans granted during given years and average one year default rate of loans that originated during that year.

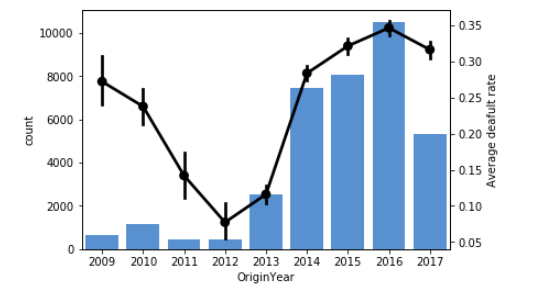


Figure 10: Number of loans granted and average default rate

Between years 2014 to 2017 we observe average default rate that was higher than that of post-crisis years. Note, that only loans granted before 1st June 2017 are shown in the figure.

Information provided by Bondora is quite throughout. The original dataset contains total of 112 columns. As explained above, my aim is to develop score cards that can be used to score newly granted loans. Therefore many columns providing information on what happened during the credit’s lifetime (collection, recovery, restructuring, secondary market info etc.) were not used during modelling. The dataset contains several scores estimated and provided by Bondora platform, these were not used as predictors as my goal is to assess whether the selected approaches can be used to develop standalone scoring functions. Other columns were dropped as well for different reasons (identifiers, various dates etc.). The full dataset description can be found on the company’s website[[48]](#footnote-48).

Total of 35 variables were identified as potential predictors. Out of these three more (“County”, “City” and “EmploymentPosition”) were dropped because of extremely high cardinality. The following table shows all the input variables selected for further analysis. Most names are self-explanatory, brief descriptions can be found in Appendix X. or, again, on the website.

| **Feature** | **Type** | **#categories** |
| --- | --- | --- |
| Interest | Num | N/A |
| LanguageCode | Cat | 13 |
| Country | Cat | 4 |
| MonthlyPayment | Cat | 3 |
| HomeOwnershipType | Cat | 11 |
| MaritalStatus | Cat | 5 |
| NoOfPreviousLoansBeforeLoan | Num | N/A |
| Gender | Cat | 3 |
| AmountOfPreviousLoansBeforeLoan | Num | N/A |
| NewCreditCustomer | Cat | 2 |
| VerificationType | Cat | 4 |
| Education | Cat | 5 |
| ExistingLiabilities | Num | N/A |
| OccupationArea | Cat | 20 |
| UseOfLoan | Cat | 16 |
| NrOfDependants | Cat | 4 |
| EmploymentStatus | Cat | 6 |
| AppliedAmount | Num | N/A |
| EmploymentDurationCurrentEmployer | Cat | 7 |
| Age | Num | N/A |
| WorkExperience | Cat | 6 |
| FreeCash | Num | N/A |
| IncomeOther | Num | N/A |
| DebtToIncome | Num | N/A |
| Amount | Num | N/A |
| IncomeFromLeavePay | Num | N/A |
| IncomeFromChildSupport | Num | N/A |
| IncomeFromPension | Num | N/A |
| IncomeFromSocialWelfare | Num | N/A |
| IncomeFromPrincipalEmployer | Num | N/A |
| IncomeFromFamilyAllowance | Num | N/A |
| IncomeTotal | Num | N/A |

Table 1: Variables selected for further analysis

The data contains variables with some missing values. These were treated in several different ways depending on the variable type (categorical/numeric) and the number of observations were the value is missing.

|  |  |
| --- | --- |
| **Feature** | **#missing** |
| DebtToIncome | 45 |
| Education | 53 |
| EmploymentDurationCurrentEmployer | 875 |
| EmploymentStatus | 197 |
| FreeCash | 45 |
| Gender | 45 |
| HomeOwnershipType | 1652 |
| MaritalStatus | 53 |
| MonthlyPayment | 6685 |
| NrOfDependants | 983 |
| OccupationArea | 137 |
| VerificationType | 53 |
| WorkExperience | 61 |

Table 2: Missing values

Monthly payment, originally continuous variable, was discretized into three categories “100<”, “<100” and “N/A” indicating missing values. Number of dependants was treated in a similar way. In “HomeOwnershipType” missing values were replaced by new category “N/A”. All other missing values were replaced by median in case of numerical variables and mode in case of categorical variables. All categorical features were encoded into dummy variables.

For the purpose of performance testing, the dataset was split into training and testing sets. Of all observations 70% was used during development and 30% was used for measuring out-of-sample performance. The split was stratified over the target variable to ensure the same proportion of defaulted loans in both sets.

## Benchmark model: logistic regression

Logistic regression (LR) represents the industry standard and was therefore used to build a benchmark model. Generally accepted development process was performed: preselection, univariate analysis, coarse classification, correlation analysis, forward selection from the short list and in- and out-of-sample performance measurement.

### Preselection: univariate scoring

The preselection was performed by measuring in-sample Gini coefficient of a logistic regression model trained using one input variable at a time. All decisions regarding variable selection and later transformations were based on the training set only.

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **Gini** | **Type** | **#categories** |
| Interest | 0.420 | Num | N/A |
| LanguageCode | 0.412 | Cat | 12 |
| Country | 0.396 | Cat | 4 |
| MonthlyPayment | 0.208 | Cat | 3 |
| HomeOwnershipType | 0.195 | Cat | 11 |
| Gender | 0.178 | Cat | 3 |
| NoOfPreviousLoansBeforeLoan | 0.176 | Num | N/A |
| VerificationType | 0.173 | Cat | 4 |
| NewCreditCustomer | 0.172 | Cat | 2 |
| AmountOfPreviousLoansBeforeLoan | 0.171 | Num | N/A |
| Education | 0.171 | Cat | 5 |
| MaritalStatus | 0.171 | Cat | 5 |
| ExistingLiabilities | 0.149 | Num | N/A |
| OccupationArea | 0.142 | Cat | 20 |
| UseOfLoan | 0.109 | Cat | 16 |

Table 3: Variables with in-sample Gini higher than 10%

Table 3 shows top 15 variables sorted by in-sample Gini coefficient, these are all the variables that achieved in-sample Gini of more than 10%.

### Univariate analysis and coarse classification

Variables preselected this way were inspected graphically. Special treatment was given to categorical variables with high cardinality. Categories with similar logarithmic bad/good odds ratio were merged into one category to make the model estimation more stable as explained in section XXX. For example “HomeOwnershipType” originally had 11 categories many of them being similar in terms of log B/G odds.

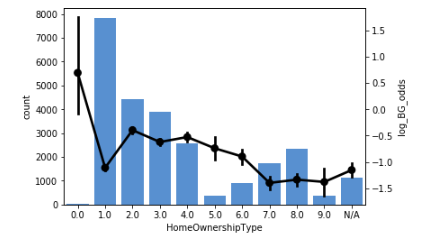


Figure 11: Original HomeOwnershipType

Categories showing similar riskiness were combined together. Category “0” (meaning the applicant is homeless) has only a few observations in it and was therefore merged into the most risky category despite having significantly higher log B/G odds. The resulting variable is shown in the following figure.

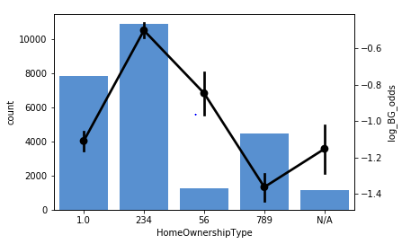


Figure 12: HomeOwnershipType coarse classified

Other categorical variables were treated the same way. In-sample Gini coefficient was measured after the transformation to ensure that not much information has been lost.

### Multivariate analysis: pairwise correlation

One of the assumptions of a logistic regression model is independence of input variables. High multicollinearity can cause severe problems. Pairwise correlation was measured using Pearson correlation coefficient and high multicollinearity (corr. coeff. > 60%) was eliminated. For the purpose of this analysis values of categorical variables were replaced by the corresponding weight of evidence values.

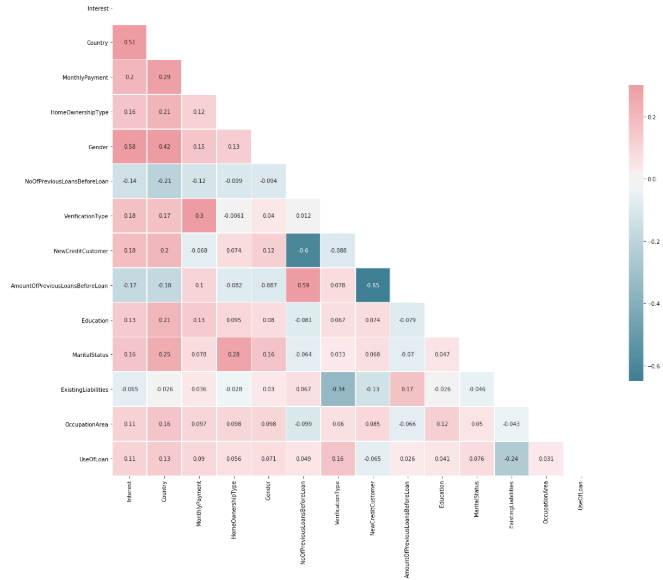


Figure 13: Correlation matrix

Variables “NoOfPreviousLoansBeforeLoan”, “AmountOfPreviousLoansBeforeLoan” and “NewCreditCustomer” show pairwise correlations about |60%|. These bear basically the same information, binary variable “NewCreditCustomer” was kept as it is the simplest one and the Gini coefficients are at about the same for all of them.

### Model estimation and evaluation

Forward selection with Bayesian information criterion (BIC) as the selection criterion was used to select the variables used in the final model. The following table shows the final set of variables used during training.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Feature** | **Gini** | **Type** | **#categories** | **Final model** |
| Interest | 0.420 | Num | N/A | Yes |
| Country | 0.396 | Cat | 3 | Yes |
| MonthlyPayment | 0.208 | Cat | 3 | Yes |
| HomeOwnershipType | 0.184 | Cat | 5 | Yes |
| Gender | 0.178 | Cat | 3 | Yes |
| VerificationType | 0.173 | Cat | 4 | Yes |
| NewCreditCustomer | 0.172 | Cat | 2 | Yes |
| Education | 0.171 | Cat | 5 | Yes |
| MaritalStatus | 0.171 | Cat | 4 | Yes |
| ExistingLiabilities | 0.149 | Num | N/A | No |
| OccupationArea | 0.129 | Cat | 4 | No |
| UseOfLoan | 0.102 | Cat | 3 | No |

Table 4: Final set of input variables

The complete model summary including model coefficients, p-values and other statistics can be found in Appendix X. The model showed solid and stable performance with Gini coefficient above 50%.

|  |  |  |
| --- | --- | --- |
| **Set** | **Gini** | **AUC** |
| Train | 0.5342 | 0.7671 |
| Test | 0.5287 | 0.7644 |

Table 5: LR model performance

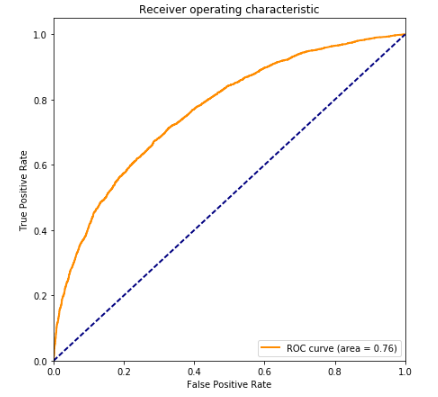


Figure 14: LR ROC curve

This model will serve as a baseline for comparison with all the other approaches developed in coming sections.

## Basic comparison of selected models

Before diving deep into different hyper-parameter settings of given ensemble models, I will run two experiments first to demonstrate some properties of the models. These experiments will help us answer some of the questions outlined in the introduction to this chapter and to get some more intuition for these methods.

The experiments consist of running all the models with parameter settings that can be considered usual or generally acceptable. Of course, choosing such parameters is to some extend arbitrary. My aim here is not primarily to compere the models in terms of predictive capability (this will be done after parameter tuning in coming sections), but to show how these models behave in different conditions. The parameter settings used are listed in Appendix X.

### Unrestricted experiment

In the first version of the experiment all models were trained using all the variables available (as listed in Table 1). So there is multicollinearity present in the dataset as well as several largely irrelevant variables. CARTs of three different depths (3, 7 and unlimited) were included to demonstrate how the base learners behave when used as standalone models.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model** | **test Gini** | **train Gini** | **test AUC** | **train AUC** |
| ExtraTrees | 0.5895 | 1.0000 | 0.7947 | 1.0000 |
| RandomForest | 0.5853 | 1.0000 | 0.7927 | 1.0000 |
| XGBoost | 0.5583 | 0.5981 | 0.7792 | 0.7990 |
| AdaBoost | 0.5446 | 0.5725 | 0.7723 | 0.7863 |
| DecisionTree\_7 | 0.5147 | 0.5786 | 0.7573 | 0.7893 |
| LogisticRegression | 0.4853 | 0.5053 | 0.7427 | 0.7527 |
| DecisionTree\_3 | 0.4684 | 0.4764 | 0.7342 | 0.7382 |
| DecisionTree\_unl | 0.2413 | 1.0000 | 0.6207 | 1.0000 |

Table 6: Results of the unrestricted experiment

In conditions described above, the assumptions of logistic regression are severely violated. It should come as no surprise that the model performed relatively poorly. All four ensembles, on the other hand, showed very good results, all of them outperforming the benchmark model developed in the previous section. Methods based on model averaging performed better than boosting. We can also see how the unlimited CART severely overfits the data. Also note how model averaging did its job in combining the overfit models into a strong learner. Figure 15 compares the models graphically.

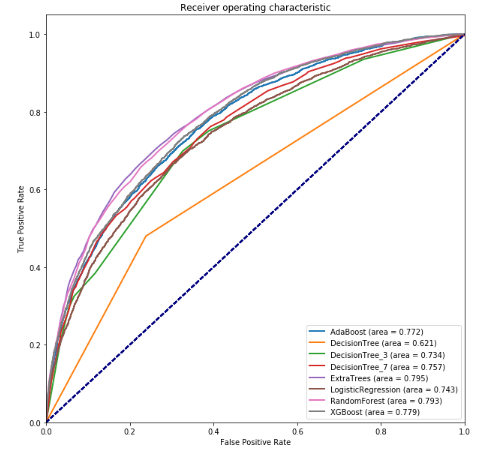


Figure 15: ROC curves of models in unrestricted experiment

Next we will have a look on how computationally expensive it is to train the models. This plays especially big role when searching for the right hyper-parameters via grid search where we have to fit each model many times.

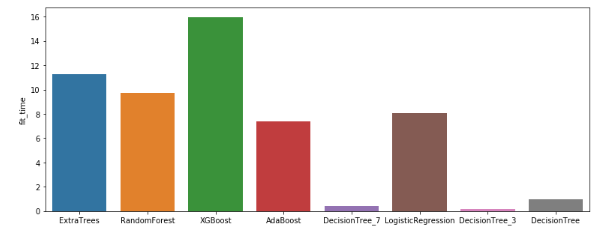


Figure 16: Training times (in seconds)

The time needed to fit the models is obviously determined by the machine used so only relative comparison is meaningful in this context. Different implementations of the same algorithm can also show different efficiency. Note for example that ExtraTrees which should theoretically be faster to train than random forest took more time in this experiment. Also the time needed to fit the ensemble model is largely determined by the number of base learners (all ensembles for now consist of 100 decision trees).

### Experiment with preselected variables

In the second version of the experiment the same models were trained using the variables preselected during the development of the benchmark model (as listed in Table 4). There are no irrelevant variables and no very high correlation. Some information was, however, lost due to coarse classification. Variables with strongly non-linear relationship with the target variable were also lost as simple logit model was responsible for the selection.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Model** | **test Gini** | **train Gini** | **test AUC** | | **train AUC** |
| XGBoost | 0.5485 | 0.5723 | 0.7743 | 0.7862 | |
| AdaBoost | 0.5390 | 0.5472 | 0.7695 | 0.7736 | |
| LogisticRegression | 0.5313 | 0.5362 | 0.7656 | 0.7681 | |
| RandomForest | 0.5219 | 0.9994 | 0.7609 | 0.9997 | |
| DecisionTree\_7 | 0.5144 | 0.5594 | 0.7572 | 0.7797 | |
| DecisionTree\_3 | 0.4563 | 0.4658 | 0.7282 | 0.7329 | |
| ExtraTrees | 0.4406 | 0.9998 | 0.7203 | 0.9999 | |
| DecisionTree\_unl | 0.2360 | 0.9998 | 0.6180 | 0.9999 | |

Table 7: Results of experiment with preselected variables

Performance of logistic regression model improved significantly, it is actually slightly better than the performance of benchmark model, selection based on BIC doesn’t seem to help in this case (from practical standpoint of view one could, however, argue that selecting simpler model giving comparable results is in fact beneficial). Performance of all ensemble models not only didn’t improve but dropped rapidly, standalone CARTs performed about the same as previously. Especially interesting are the poor results of extremely randomized trees, these got outperformed even by two of the standalone decision trees. When introducing ExtraTrees the authors argue that the splitting points tend to be very unstable and so making them random doesn’t hurt the performance but can actually improve it (recall that extremely randomized trees use random splitting points instead of resampling as means of decorrelation of the base learners)[[49]](#footnote-49). This simply didn’t work in this case, even after rerunning the experiment with different random seeds ExtraTrees achieved Gini coefficients about 0.44. When using resampling (as in random forest) the performance grew to Gini coefficient of 0.48, which is still relatively poor.

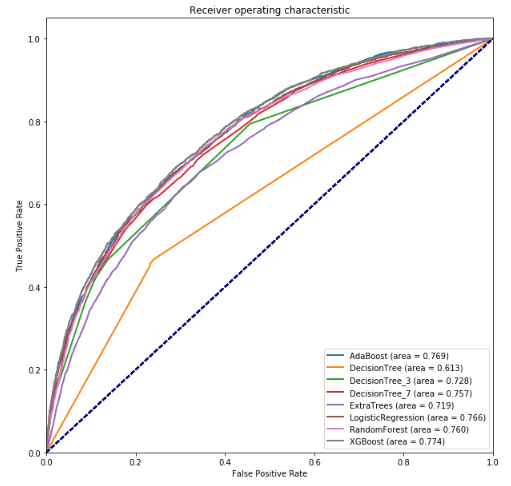


Figure 17: ROC curves of model in experiment with preselected predictors

All of this reminds us about the fundamental differences between linear and tree-based models. It is now also obvious that the whole data preparation and variable selection process needs to be re-though.

## Variable selection with tree-based models

When using ensemble models based on decision trees we need to approach variable selection differently. Univariate selection does not make much sense as there are no independence assumptions. When dealing with high-dimensional datasets one would probably need to discard some potential inputs based on business logic (variables where there is no reasonable hypothesis why this could predict default, variables that are expensive to acquire or otherwise hard to get, legal issues etc.) or some other criteria (very low variance, extremely high cardinality etc.), but to get most of the data one has to take into account the interactions among inputs when selecting variables for the final model. Coarse classification also seems unnecessary as decision trees perform implicit variable selection (not all the inputs need to be used for prediction) so dummy variables representing irrelevant categories are simply ignored. Categories with low number of observations (or irrelevant categories chosen by randomized trees) can cause problems. Unlike linear models decision trees are local models, therefore only certain areas of the feature space are influenced by such cases. Discretization of continuous variables is unnecessary as well as CART performs discretization online.

Naïve approach would be to try all possible combinations of input variables, test them out-of-sample (or even cross-validate them) and select the best combination. This brute force solution would be extremely computationally demanding and therefore totally impractical.

More sensible solution is to use variable importance measure introduced in chapter X. We can simply select best variables based on their importance, select variables with importance above some predefined threshold or recursively eliminate variables with lowest importance. Cross-validation can be used to select the best value of .[[50]](#footnote-50)

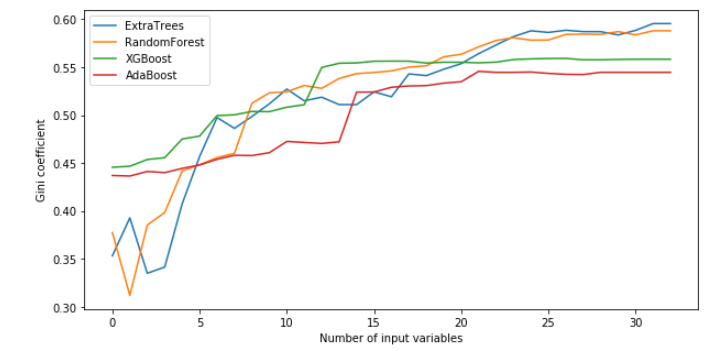


Figure 18: Performance of models with different number of input variables

In Figure 18 we see out-of-sample performance of models trained using different number of inputs. These were selected as best variables based on variable importance of a model trained using all available variables. All models use the same parameters as in previous section (see Appendix X). Variable selection does not improve performance of the models. After reaching certain number of inputs, very different for different models, the out-of-sample Gini coefficient (almost) stops increasing.

From all of the above it seems that the best practical approach would be to fine tune the model using all predictors available and then use feature importance to prune some variables from the original set of input variables. Naturally, we want the model to be as simple as possible. Models with lower number of predictors are easier to deploy and eventually debug. The lower the number of inputs the easier is to monitor the model while being used in production. The behaviour of some variables can change over time and can make the model’s performance to drop. Such cases are hard to recognize if the number of predictors is high. In practice, we should therefore eliminate all the features that does not improve the model’s predictive power, we may even be willing to sacrifice some of the predictive capability to have the model simpler and easier to manage.

As my main goal is to compare the models by their out-of-sample performance and because the variables selection process does not seem to improve the performance I will use all the available predictors in the coming sections.

## Ensemble models with tuned hyper-parameters

Finally, we can compare the models in terms of their out-of-sample predictive capability. To find optimal hyper-parameter settings exhaustive grid search was performed for each model. This was done in two steps, during the first run rougher grid was used to find promising hyper-parameters, these were further refined using another run of grind search with finer grid.

Such a task is computationally very demanding and given my resources I was forced to use relatively rough grids. I also focused on the most important parameters only. For example I only used the depth of the trees as means of limiting their complexity and used default values of other stopping criteria. Three-fold cross-validation stratified over the output variable was used during the search.

### Random forest

When searching for optimal settings of the random forest model three hyper-parameters were tuned: number of base learners, maximum depth of the trees and (the number of input considered at each split.

|  |  |  |
| --- | --- | --- |
| **Random forest** | |  |
| n\_estimators | 300 | |
| max\_depth | Unlimited | |
| *p* | log2*d* | |
|  | **Benchmark** | **Final** |
| train Gini | 1.0000 | 1.0000 |
| test Gini | 0.5853 | 0.5960 |

Table 8: Random forest – final model

Two values of were considered: and . Number of base learners tried during the search ranged from 20 to 300. As expected, increasing number of trees in the ensemble meant better performance. Several values higher than 300 were tested to see whether the performance can be further enhanced. Having more base learners in the model did not significantly improve the performance. Limiting the maximum depth of the trees did not lead to any improvement (values between 5 and 20 were considered). Random forest proved to be robust to different hyper-parameter settings.

### Extremely randomized trees

The same procedure was performed with the extremely randomized trees model.

|  |  |  |
| --- | --- | --- |
| **Extremely randomized trees** | | |
| n\_estimators | 300 | |
| max\_depth | Unlimited | |
| *p* | log2*d* | |
|  | **Benchmark** | **Final** |
| train Gini | 1.0000 | 1.0000 |
| test Gini | 0.5895 | 0.6004 |

Table 9: Extremely randomized trees – final model

The results are almost identical to those of random forest. Both models are based on the same principles and extremely randomized trees as a newer variant of this model did not proved to be notably better than the traditional one.

### AdaBoost

Only two hyper-parameters were played with while searching for the best adaptive boosting model. These were the number of base learners and learning rate. The selected model look as follows.

|  |  |  |
| --- | --- | --- |
| **AdaBoost** |  |  |
| n\_estimators | 480 | |
| learning rate | 0.2 | |
|  | **Benchmark** | **Final** |
| train Gini | 0.5725 | 0.5773 |
| test Gini | 0.5446 | 0.5497 |

Table 10: AdaBoost – final model

As originally recommended base learners with only one split were used, number of trees in the ensemble up to 600 and learning rate ranging from 0.1 to 1 were tried during the search (it is feasible to try larger ensembles as the base learners are very simple and computationally cheap). Similarly to previous models AdaBoost also shows robustness to hyper-parameter settings, results of both versions of the model are virtually identical.

### Gradient boosting

Four hyper-parameters were tuned during the search for the best gradient boosting model: number of base learners, learning rate, maximum depth and subsample size.

|  |  |  |
| --- | --- | --- |
| **Gradient boosting** | |  |
| n\_estimators | 220 | |
| learning rate | 0.05 | |
| max\_depth | 6 | |
| subsample | 0.7 | |
|  | **Benchmark** | **Final** |
| train Gini | 0.5981 | 0.7665 |
| test Gini | 0.5583 | 0.5826 |

Table 11: Gradient boosting – final model

Maximum depth up to 6, number of base learners up to 500, learning rate ranging between 0.025 and 0.3 and subsample between 0.5 and 1 were tried during the search. Lower learning rates were chosen as more complex models are being combined in this case. For the first time we see noticeable difference in performance between benchmark and tuned models. What is quite surprising is the relatively high maximum depth (larger than generally recommended).

### Performance comparison: summary

All four models showed strong out-of-sample performance, all outperforming the benchmark model developed in chapter X. Random forest, extremely randomized trees and gradient boosting showed very similar performance, adaptive boosting’s performance was closer to the logistic regression benchmark.

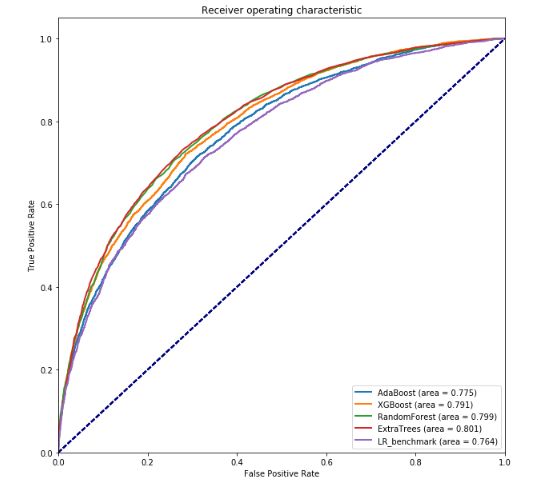


Figure 19: ROC curve of final models

**One final note:** although the models’ performances were often almost identical, they differ in how they use the input variables. When looking at variable importance, we see that, for example, extremely randomized trees model has these much more evenly distributed than random forest. The same is true for both boosting models before and after hyper-parameter optimization. This might have practical impact as we want the models to be as simple as possible (for reasons explained earlier). It seems reasonable to assume that models with more evenly distributed importance would be harder to prune (by leaving out variables with lowest variable importance). This hypothesis would, however, need to be further researched. Variable importance of final models can be found in appendix X.

# Conclusions

1. Elements (doplnit) [↑](#footnote-ref-1)
2. Output can also be a vector, consider for example a case of predicting a point in two dimensions (described by latitude and longitude). We won’t deal with this kind of problems in this thesis. [↑](#footnote-ref-2)
3. Vijayakumar, Sethu (2007). "The Bias–Variance Tradeoff" . University Edinburgh. Retrieved 19 August 2014. [↑](#footnote-ref-3)
4. Elements (doplnit) [↑](#footnote-ref-4)
5. Domingos, Pedro (2000). [*A unified bias-variance decomposition*](http://homes.cs.washington.edu/~pedrod/bvd.pdf) . ICML. [↑](#footnote-ref-5)
6. Manning, Christopher D.; Raghavan, Prabhakar; Schütze, Hinrich (2008). Introduction to Information Retrieval. Cambridge University Press. pp. 308–314. [↑](#footnote-ref-6)
7. |  |  |  |  |  |  |
   | --- | --- | --- | --- | --- | --- |
   | BERKA, P. | Dobývání znalostí z databází | Praha | Academia | 2003 | 80-200-1062-9 |

   [↑](#footnote-ref-7)
8. Elements [↑](#footnote-ref-8)
9. Quinlan, J. R. 1986. Induction of Decision Trees. Mach. Learn. 1, 1 (Mar. 1986), 81–106 [↑](#footnote-ref-9)
10. Quinlan, J. R. (1986). Induction of decision trees. Machine Learning, 1(1):81-106. [↑](#footnote-ref-10)
11. Pandya R., Pandya J. (2015). C5.0 Algorithm to Improved Decision Tree with Feature Selection and Reduced Error Pruning [↑](#footnote-ref-11)
12. Magidson, Jay; The CHAID approach to segmentation modeling: chi-squared automatic interaction detection, in Bagozzi, Richard P. (ed); Advanced Methods of Marketing Research, Blackwell, Oxford, GB, 1994, pp. 118–159 [↑](#footnote-ref-12)
13. Hastie T., Tibshirani R., and Friedman J.H. (2009) The Elements of Statistical Learning, 2nd edition. Springer, ISBN 978-0-387-84857-0 [↑](#footnote-ref-13)
14. Papagelis A., Kalles D.(2001). Breeding Decision Trees Using Evolutionary Techniques, Proceedings of the Eighteenth International Conference on Machine Learning, p.393-400, June 28-July 01, 2001 [↑](#footnote-ref-14)
15. Chipman, Hugh A., Edward I. George, and Robert E. McCulloch. "Bayesian CART model search." Journal of the American Statistical Association 93.443 (1998): 935-948. [↑](#footnote-ref-15)
16. Barros R. C., Cerri R., Jaskowiak P. A., Carvalho, A. C. P. L. F., A bottom-up oblique decision tree induction algorithm. Proceedings of the 11th International Conference on Intelligent Systems Design and Applications (ISDA 2011). [↑](#footnote-ref-16)
17. Gareth, James; Witten, Daniela; Hastie, Trevor; Tibshirani, Robert (2015). An Introduction to Statistical Learning. New York: Springer. p. 315. ISBN 978-1-4614-7137-0. [↑](#footnote-ref-17)
18. , 19 Elements [↑](#footnote-ref-18)
19. [↑](#footnote-ref-19)
20. Lessmann et al.,2008 (doplnit) [↑](#footnote-ref-20)
21. Breiman L (2001). "Random Forests". Machine Learning. 45 (1): 5–32. doi:10.1023/A:1010933404324. [↑](#footnote-ref-21)
22. Segal (2004) (doplnit) [↑](#footnote-ref-22)
23. In fact, there is a random tree-based unsupervised learning algorithm used to detect anomalies in the data based on this idea. It is called isolation forest. For more details see: Liu, Fei Tony, Ting, Kai Ming and Zhou, Zhi-Hua. “Isolation forest.” Data Mining, 2008. ICDM‘08. Eighth IEEE International Conference on. [↑](#footnote-ref-23)
24. , 25 P. Geurts, D. Ernst., and L. Wehenkel, “Extremely randomized trees”, Machine Learning, 63(1), 3-42, 2006. [↑](#footnote-ref-24)
25. [↑](#footnote-ref-25)
26. Liu, Fei Tony, Ting, Kai Ming and Zhou, Zhi-Hua. “Isolation forest.” Data Mining, 2008. ICDM‘08. Eighth IEEE International Conference on. [↑](#footnote-ref-26)
27. P. Geurts, D. Ernst., and L. Wehenkel, “Extremely randomized trees”, Machine Learning, 63(1), 3-42, 2006. [↑](#footnote-ref-27)
28. Leo Breiman (1996). "BIAS, VARIANCE, AND ARCING CLASSIFIERS", TECHNICAL REPORT. [↑](#footnote-ref-28)
29. Element (doplnit) [↑](#footnote-ref-29)
30. Elements [↑](#footnote-ref-30)
31. Robert E. Schapire, Explaining AdaBoost (doplnit) [↑](#footnote-ref-31)
32. Mason, L., Baxter, J., Bartlett, P., Frean, M.: Functional gradient techniques for combining

    hypotheses. In: Advances in Large Margin Classifiers. MIT Press (2000) [↑](#footnote-ref-32)
33. Robert E. Schapire, Explaining AdaBoost (doplnit) [↑](#footnote-ref-33)
34. <https://github.com/dmlc/xgboost/tree/master/demo#machine-learning-challenge-winning-solutions> (udělat citaci) [↑](#footnote-ref-34)
35. Friedman, J. H. "Greedy Function Approximation: A Gradient Boosting Machine." (February 1999) [↑](#footnote-ref-35)
36. Friedman, J. H. "Greedy Function Approximation: A Gradient Boosting Machine." (February 1999) [↑](#footnote-ref-36)
37. Elements [↑](#footnote-ref-37)
38. Friedman, J. H. "Stochastic Gradient Boosting." (March 1999) [↑](#footnote-ref-38)
39. https://www.python.org [↑](#footnote-ref-39)
40. https://pandas.pydata.org [↑](#footnote-ref-40)
41. http://www.numpy.org [↑](#footnote-ref-41)
42. scikit-learn.org [↑](#footnote-ref-42)
43. http://www.statsmodels.org [↑](#footnote-ref-43)
44. https://xgboost.readthedocs.io [↑](#footnote-ref-44)
45. https://matplotlib.org [↑](#footnote-ref-45)
46. https://seaborn.pydata.org [↑](#footnote-ref-46)
47. https://www.bondora.com/public-reports [↑](#footnote-ref-47)
48. https://www.bondora.com/public-reports [↑](#footnote-ref-48)
49. Citovat ExtraTrees [↑](#footnote-ref-49)
50. <http://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.RFECV.html#sklearn.feature_selection.RFECV> (checknout zdroj) [↑](#footnote-ref-50)