House Price Analysis with R

Creating Analysis and Building Regression Models for Ames, Iowa Dataset

Frederik Elischberger, Marco Niemann

January 11th, 2017

1. Introduction

This documentation aims at the provision of information about the R package KaggleHouse. It has been developed in the course of the Master level seminar *Applied Machine Learning in R* in the winter term 2016/2017 at the University of Münster (WWU). The seminar participants were given a list of Kaggle[1]competitions with diverse machine learning tasks. . For each group the task has been to apply machine learning methods to solve the provided challenges. The implementation should be done in R. Foundation for this seminar work has been the *Houses Prices: Advanced Regression Techniques*[2]challenge. In the second section the background and the objectives of the chosen challenge will be clarified and the mode of participation will be explained. The next part of this thesis serves an instruction on how to install the created package and about the components and packages that it makes use of. This should provide the reader with the ability to follow the further course of this paper with the option to experiment with the described functionality. Information about the core of the seminar, the implementation, usage and methodology behind the KaggleHouse package are then to be found in the sections 4-7. First, section 4 will be used to introduce the used data set further. It further contains an exploratory analysis of the data provided by Kaggle, including hints wherever problematic patterns were found. The proposed and applied fixes are presented in the reminder of this part. Section 5 will present the used feature selection methodology as well as a short insight into the generated results. In the remaining sections the focus will be on the used learning algorithms and the tuning applied on them. Section 6 will be used to introduce the applied tuning algorithm `irace`, a non-standard heuristic approach to find the optimal parameter set for a machine learner. The final section 7 then introduces the used learners `xgboost`, `glmnet` and `deeplearning`, including an overview over their underlying ideas as well as the actual performance.

[1]Kaggle is an online platform for predictive modelling and analytics competitions (Wikipedia 2016). It has received a lot of attention, both in academia as well as in practice. As a result universities as well as companies host competitions on Kaggle to get them solved by freelance data miners or (statistics) researchers (Metz 2013; Taieb and Hyndman 2013; Athanasopoulos and Hyndman 2011)

[2]The competition can be found here: https://www.kaggle.com/c/house-prices-advanced-regression-techniques

2. House Prices: Advanced Regression Techniques - Kaggle Challenge

As already shortly mentioned in the introduction, the `KaggleHouse` package and the accompanying documentation material were created in the course of a participation in the *House Prices: Advanced Regression Techniques* challenge.

This challenge is hosted as a so called '*Playground*' competition on Kaggle and is open for submissions from August 30th, 2016 till March 1st, 2017 (Kaggle Inc 2016a). The following section will provide an introduction in the topic of the challenge as well as the intended objectives of its creators. It will be rounded up by a short introduction in how this papers authors contributed to the challenge.

2.1. Background

The Kaggle challenge on *House Prices: Advanced Regression Techniques* bases on the work of De Cock (2011). Dean De Cock is a professor for statistics at the Truman State University in Missouri. For his statistics courses he tended to use the popular *Boston Housing Data Set* (Harrison and Rubinfeld 1978) which is still a commonly used dataset for regression tasks (e.g. still used in software systems like R) (Venables and Ripley 2002). He used the data for course projects where students should apply their regression knowledge to predict houses prices based on a given dataset.

Unsatisfied by the age of the *Boston Housing Data Set* ($\sim$ 25 years), as well as the limited numbers of observations and variables he began to search for a new datset. By serendipitous discovery he found out that the city of Ames in Iowa was updating their house price assessment model. After negotiations with the city's Assessor's Office Prof. De Cock got access to a data dump from Ames's records systems, which now underlies this seminar project.

The original dataset of 3970 observations with each 113 variables related to real estate sales in the city of Ames in the time frame between 2006 and 2010. To make the data usable for eductational purposes De Cock removed 33 categories that would require non-general knowledge. He left 80 variables describing the general properties of each sale (incl. *lot area*, *neighbourhood*, *diverse quality criteria...*). Furthermore 1040 observations were removed that dealt with non-residential sales and for each object sold multiple times only the most recent sale was kept. This was intended to reduce the complexity of dealing with the dataset, as such special sales according to De Dock introduced unusual conditions.

On top De Cock claims that he eliminated all apparent errors he could find within the dataset, while he points out that potential outliers have not been removed. Thus he explicitly warns that some of these items should potentially be removed for lecture use.

2.2. Objectives

While the challenge fully bases on the dataset prepared by De Cock (2011), some of the objectives differ from the originally intended ones. In his introductory paper to the dataset De Cock explains that he intends his students to create regression models of different complexity. In these regression models the students were supposed to find the right features and the right regression method to create a satisfying model for the prediction of sales prices based on the given house features. While he mentions the use of software for the model creation, one of his explicit requirements is that for testing purpose the generated models have to be computed by hand (see e.g. Equation 1).

$$\hat{Y} = -27810.661 + 9322.460 \cdot X_1 - 4798.086 \cdot \sqrt{X_5} + 5119.678 \cdot \ln X_{10} - 110.087 \cdot X_1 \cdot X_{10} \quad (1)$$

The Kaggle challenge now aims to complement the application of these rather simple techniques with advanced regression techniques commonly used in Machine Learning. KAGGLE INC (2016A) proposes methods such as *random forest* (Breiman 2001) and ensemble learning via boosting (Schapire et al. 1998) - or more specificially: gradient boosting via *XGBoost* (Chen and Guestrin 2016) to create these more sophisticated regression models. While these methods are recommended by Kaggle others are not excluded from consideration.

Apart from the application of regression models on the data which is prevalent in both task sets (De Cock 2011; Kaggle Inc 2016a), Kaggle encourages to fulfil further subtasks. Those include data exploration, feature engineering as well as the creation of insightful visualizations (Kaggle Inc 2016d). This collection of tasks serves as the basis for this seminar work. It will cover all major parts from the initial exploratory data analysis with extensive plotting (as e.g. recommended in Maindonald and Braun 2003) to the training of one or multiple suitable models.

2.3. Mode of Participation

One of the intentions for the seminar work was to actually participate in the Kaggle challenges assigned to each group. Accordingly this group did not only use the provided data to apply different machine learning techniques, but also uploaded result data to compare with other teams. Consequently both authors joined Kaggle and formed the *AML* team (see Table 1).

Team **AML**	
Frederik Elischberger	`frederik`
Marco Niemann	`marco_n`
Ranking	1157 (*out of 3322*)

Table 1: Team Setup and Leaderboard Ranking (as of January 11th, 2017)

Each team's ranking within the competition is based on the best submission uploaded to Kaggle[3]. A submission consists of the predicted sales prices for the test data set. For the leaderboard only 50% of the available test data is evaluated, so that the final scores reached might differ from those available online so far (Kaggle Inc 2016c).

[3]The *House Prices: Advanced Regression Techniques* is restricted to five submissions per team and day (Kaggle Inc 2016b).

3. Installation and Setup

Since the `KaggleHouse` package is not available on the Comprehensive R Archive Network (CRAN), the commonly known `install.packages('KaggleHouse')` will not work. However, since the package is currently hosted on GitHub it can be installed with a similar level of convenience via the `devtools` package (Wickham and Chang 2016).

```
devtools::install_github(
  "https://github.com/MarcoNiemann/kaggle_house","KaggleHouse"
)
```

To successfully install the `KaggleHouse` package a set of dependencies has to be fulfilled (see Table 2). While each package and it's use will be explained in more detailed in the respective sections, Table 2 also presents a short overview over each packages purpose.

To take away the efforts of installing all packages manually, a short script is provided in Appendix A.1. It will check whether all required packages are present and install those which are still missing. While this script will be able to install most dependencies in a very convenient fashion, it has to be noted that some packages need special treatment due to incompatibilities among different packages. The affected packages included `mlr`, `xgboost` and `h2o`. `mlr` is currently available in version `2.9` on CRAN, which, however causes severe issues with the current version of the `irace` package (as e.g. discussed in `mlr` GitHub Issue # 1282). Until `mlr` in version `2.10` will be available on CRAN the recommened (and applied) fix is to use the current GitHub release (see Appendix A.2.1 for instructions). Besides `mlr`, also the versions `0.4.4` and `0.6.0` of `xgboost`[4]caused issues when used for model training. Effects can be mitigated by changing to the most recent version `0.6.2`, which works stable in the presented dependency configuration for the `KaggleHouse` package (an instruction to update to the current version can be found in Appendix A.2.2). Again more similar to the `mlr` case, the most recent CRAN version of `h2o` (at the time of this writing: `3.10.0.8`) was also causing issues. Since the stable release was already two month old with nightly releases provided each day, the decision was made to use a current nightly build (version `3.11.0.3723`). Instructions on how to obtain an `h2o` nightly build instead of the CRAN version can be found in Appendix A.2.3.

Package	Citation	Usage/Description
`Amelia`	(Honaker et al. 2011)	*Package for multiple imputation.*
`Boruta`	(Kursa and Rudnicki 2010)	*All relevant feature selection wrapper algorithm.*
`Ckmeans.1d.dp`	(Wang and Song 2011)	*Fast, dynamic algorithm for optimal k-means computation.*
`compiler`	(R Core Team 2016)	*Allows to compile R code into Byte code. Helps to increase execution performance.*
`corrplot`	(Wei and Villiam 2016)	*Visualization of correlation matrices.*
`dbscan`	(Hahsler 2016)	*Package providing an implementation for density-based clustering.*
`FSelector`	(Romanski and Kotthoff 2016)	*Provision of attribute selection functions (from a dataset).*

Package	Citation	Usage/Description
`ggplot2`	(Wickham 2009)	*Package for declaratively creating graphics. Looks better than default plots.*
`glmnet`	(Friedman et al. 2010)	*Package providing efficient procedures for fitting the lasso/elastic-net regularization path for the diverse regression models (linear, logistic, ...).*
`gridExtra`	(Auguie 2016)	*Used to align multiple `ggplot2` graphics into a printable grid.*
`h2o`	(The H2O.ai team 2016; Candel et al. 2015)	*Package providing the R interface for the H2O Machine Learning engine.*
`irace`	(López-Ibáñez et al. 2016)	*Package for (offline) parameter tuning of optimization algorithms. Provides automated configuration based on the Iterated F-race method.*
`magrittr`	(Bache and Wickham 2014)	*Package to provide the new forward-pipe operator.*
`methods`	(R Core Team 2016)	*Provision of formal methods and classes for R objects (S4).*
`mlr`	(Bischl et al. 2016b; Bischl et al. 2016c)	*Generic, object-oriented, extensible framework for classification, regression, survival analysis and clustering in R.*
`mice`	(Buuren and Groothuis-Oudshoorn 2011)	*Incomplete multivariate data imputation via chained equations.*
`parallelMap`	(Bischl and Lang 2015)	*Parallelization framework allowing diverse parallelization options.*
`ParamHelpers`	(Bischl et al. 2016a)	*Provision of functions for parameter descriptions and operations in black-box optimization, tuning and machine learning.*
`Rcpp`	(Eddelbuettel and Francois 2011) (Eddelbuettel 2013)	*Used to integrate C++ code into R. Also provides semantic sugar like R-like datastructures for C++.*
`xgboost`	(Chen et al. 2016)	*Library providing classification and regression based on gradient (tree) boosting.*

Table 2: `KaggleHouse` Dependency Packages

Apart from these necessary components there exists a variety of packages (see Table 3) that are suggested to be installed to make use of the full functionality of the `KaggleHouse` package.

Package	Citation	Usage/Description
`kernlab`	(Karatzoglou et al. 2004)	*Allows to use Support Vector Machines for classification and regression.*
`randomForest`	(Liaw and Wiener 2002)	*Allows to use random forests for classification and regression.*

Table 3: `KaggleHouse` Recommended Packages

4. Data Analysis and Preparation

The original data for analysis is provided by Kaggle as part of the '*House Prices: Advanced Regression Techniques*'. It comes in two separate files (listed in 4), providing the data for training and testing the requested model. Since `R` packages allow convenient access to data when it is stored as an `.RData` file in the `/data` directory, one of the first steps conducted was the necessary conversion. The package will be delivered with the raw `.csv` datasets, which by a specific function (`KaggleHouse:::rda.conversion.convertData()`) will then be converted to `.RData` files upon installation. So simple loading of the package via `require` or `library` will make the two datasets `data_train` and `data_test` available in the current `R` environment.

On top an additional `data_description` file is provided including additional information regarding the used features. Those include a rough description of each attribute, as well as an explanation regarding the different `factor` levels for categorial and nominal variables.

File	Features	Observations	Purpose
`train`	80 + 1	1460	*Dataset including the target value $SalePrice$. Meant to be the basis for model creation.*
`test`	80	1459	*Dataset for which $SalePrice$ should be predicted based on the created models.*

Table 4: Competition Datasets

4.1. Exploratory Analysis

After loading and converting the required data sets the recommended first step to actually understand the task and the analysis to be conducted (Maindonald and Braun 2003). Furthermore it is intended to provide the basis for potentially necessary data cleaning, conversion or imputation steps. This way the issue of data dredging (blind application of data mining methods leading to the discovery of potentially meaningless patterns) should be avoided (Fayyad et al. 1996). The actual analysis should deliver insights about potential distributions, outliers, clusters, and correlations etc., which are usually captured in informative graphics (Maindonald and Braun 2003; Zaki and Meira 2014). For the current case interestingly two sources for the exploratory part exist: The first is the analysis already conducted by De Cock (2011), which can then be complemented by personal analysis results.

As a first step, the prepared data that accompanies the downloaded datasets was evaluated. It provides information about the different scales of measure used throughout the dataset. While the exact assignment of variables to scales can be looked up in the official data documentation, the overall scale picture is as follows (see Table 5):

	Continuous	**Discrete**	**Nominal**	**Ordinal**
Features	19	14	23	23

Table 5: Scales of Measures

In case someone is wondering why only 79 variables have been categorized, it has to be stated that two variables have been ignored for this analysis: the `Id` and the `SalePrice`. The reason is that both are not 'real' variables in terms of being influence factors for the price. While the `SalePrice` is the actual target value, the `Id` is merely an index to uniquely identify each sale and can be ignored for the analysis.

The following subsections describe in more detail the data preparation process in this package that is executed when calling the function `run_generate_data()`.

With regard to outliers Dean De Cock mentioned in his work De Cock (2011) to remove all tuples with a `GrLivArea` larger than 4000. To verify his suggestions Figure 1 has been created to ensure that the described points actually represent outliers. The dots marked with red triangles clearly indicate that the mentioned points are in fact outliers.

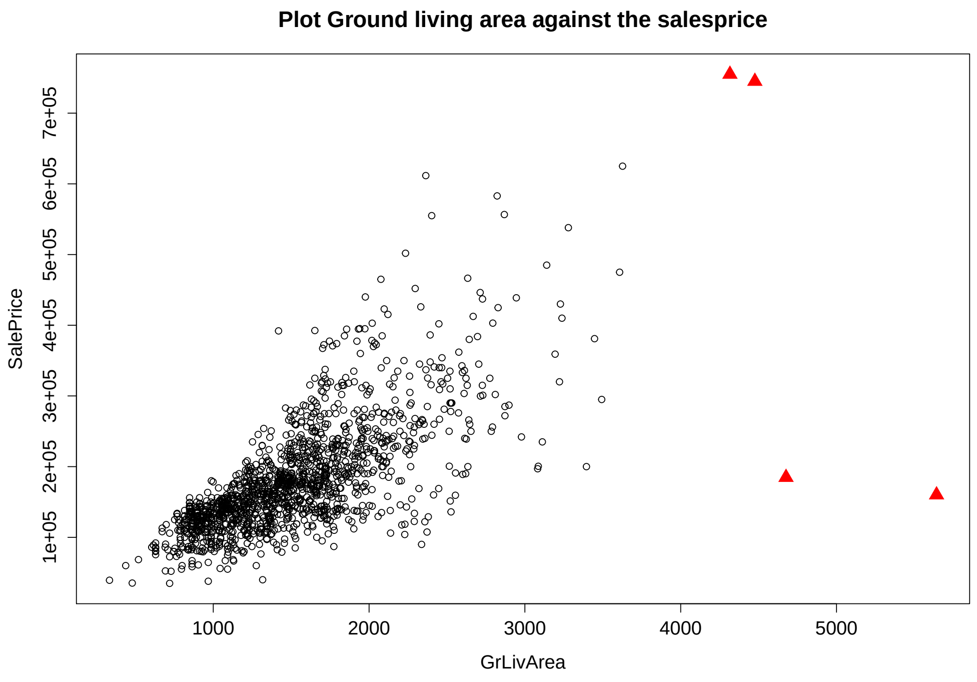


Figure 1: Outliers as Remarked by De Cock (2011)

Further search for outliers with dotplots and Local Outlier Factor (LOF) (Breunig et al. 2000) returned no further (apparent) outliers. However, while searching through the data distributions, the feature Utilities was found to have two potential characteristics of which one only occurs once (see Figure 2).

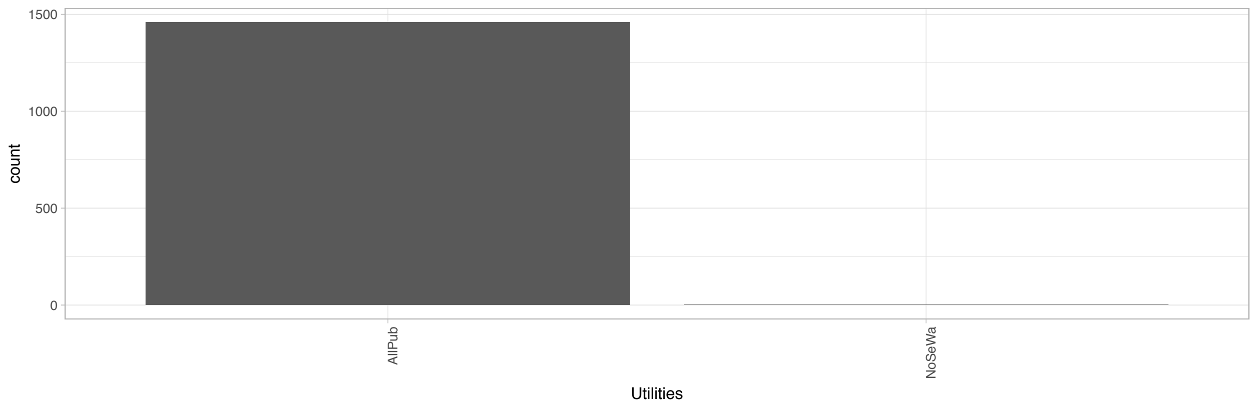


Figure 2: Value Distribution of the Utilities Column

Apart from finding suspicious values that apparently do not fit the analyzed data, it is further necessary to check for missing values as well. In difference to the outlier search this step returned many more results. Some features like Alley only have a value for 91 observations (missing values for 1369) whereas in other cases like Electrical only a single value was missing. To get an overview over the whole situation the following two plots (see Figures 3 and 4) were created.

This first plot indicates the values already pointed out above and apart from that shows one further problematic characteristic: Each line contains at least one NA value. So, a simple removal of lines containing missing values is plainly infeasible.

Analyzing the test data set reveals a very similar scenario. Again, each line contains NA values so that a simple removal is not feasible. The suitable response to this situation appears to be a thorough imputation of the missing value in order not to lose information. For the exact procedure refer to Section 4.3.

After the analysis of missing values and outliers, the analysis of existing and relevant data was focused. One objective was to understand the correlations between the different variables. Another one was to get a general understanding of the impact of features to the target value. To achieve this a so called corrplot (Wei and Villiam 2016) has been created. It represents a more compact and color-focused alternative to the frequently used scatterplot. Since mostly highly correlations are of interest, a minimum correlation of 0.7 with another feature is required for a variable to be not eliminated.

Looking at both corrplots (see Figures 5 and 6) there seems to be a high correlation of OverallQual, YearBuilt, GrLivArea and all garage related features to the SalePrice. These insights can later be used to evaluate the feature selection since variables that are highly correlated with the SalePrice should be relevant for the final model training.

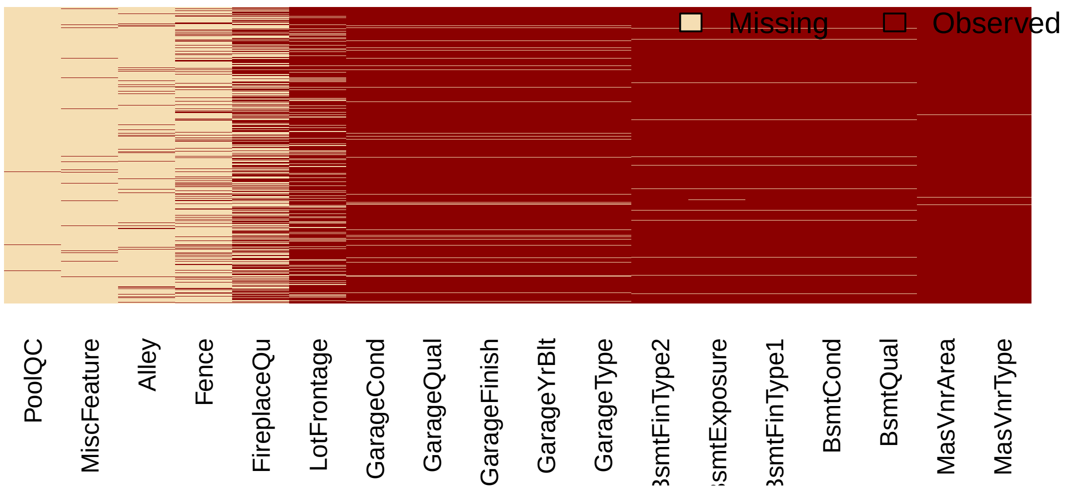


Figure 3: missmap of the Train Data Set

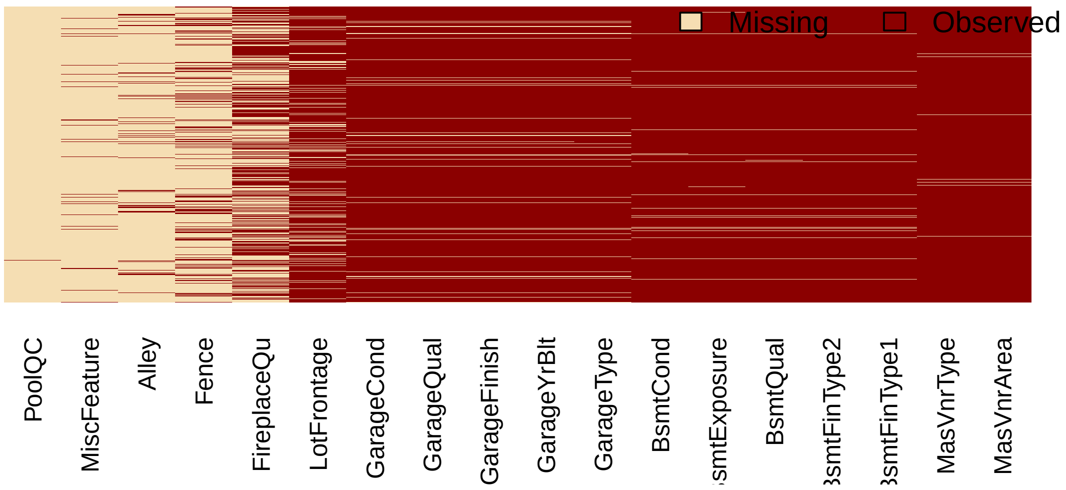


Figure 4: missmap of the Test Data Set

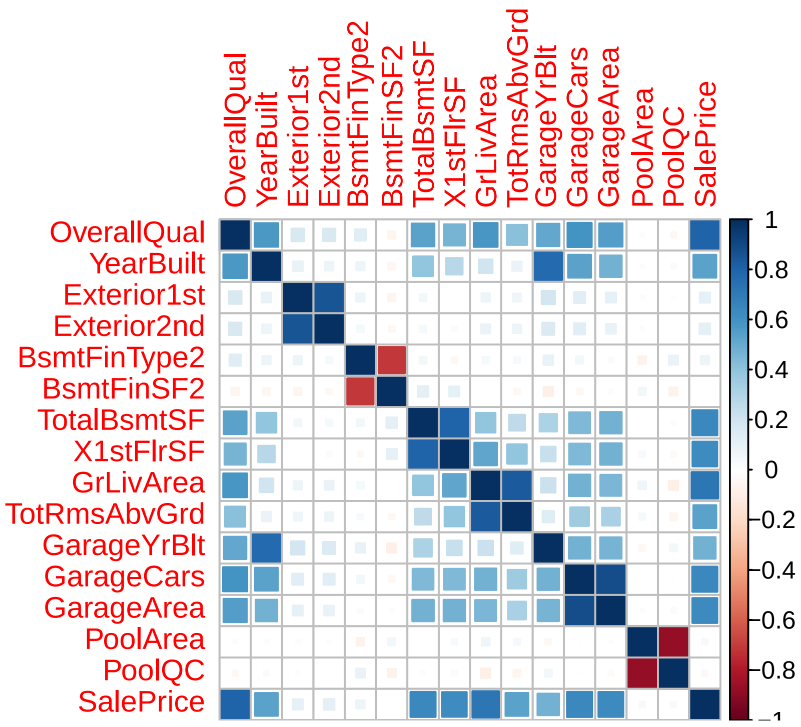


Figure 5: corrplot with Pearson Correlation

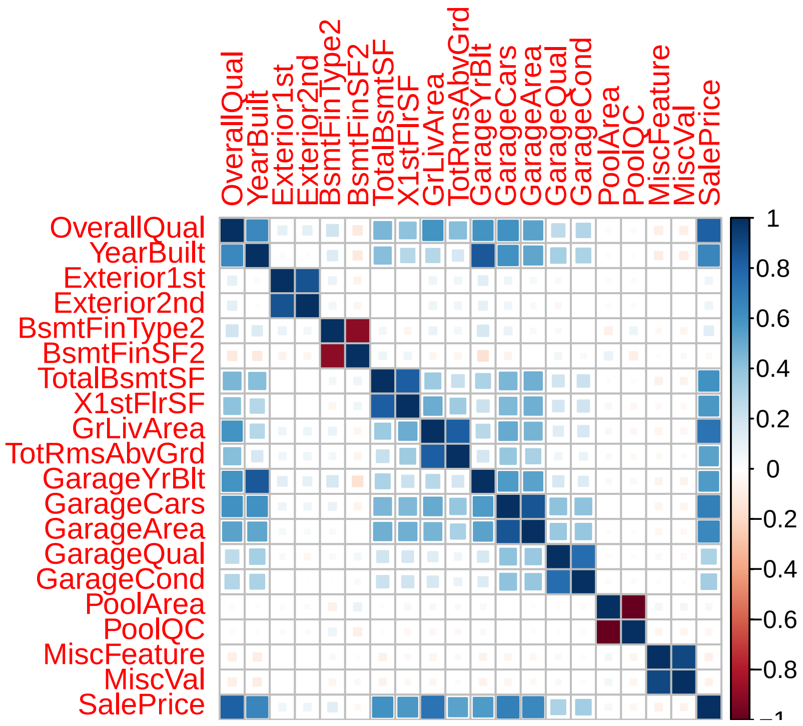


Figure 6: corrplot with Spearman Correlation

4.2. Data cleaning

Since the previous exploratory analysis (see Section 4.1) revealed that the dataset contains several outliers, the decision has been made to remove the ones identified. So the rows 524, 692, 1183 and 1299 have been removed. Furthermore, the `Utilities` column has been eliminated since it apparently provides not valuable input.

4.3. Imputation

This section deals with the issue of removing missing data. As this dataset contains relatively many rows with at least one missing value removing these rows was not possible. Many features as described in the `data_description` file had a class `'None'` or something similiar but others e.g. `Alley` stated that a missing value actually means that there is no alley access at all. To not lose this information when learning a model all missing values in those features are substituted with the character `'None'`. The transformation into numeric values then creates a new class for those missing values that can be utilized in the learning process. This is done for that training as well as the test dataset.

Next the imputation is splitted into another manual imputation and automatic imputation using the package `mice`(Buuren and Groothuis-Oudshoorn 2011).

The manual imputation for the training and test dataset only deals with missing values in the columns `LotFrontage`, `Electrical` and `GarageYrBlt`. For `LotFrontage` missing values are imputed with zero because in theory there is zero feet of street connected to the property. The missing value in the feature `Electrical` becomes a new class by adding one to the highest numerical class. At last missing values in `GarageYrBlt` are substituted with the median of all remaining rows. This is done because a missing value here is not missing at random since there actually is no garage for that property and the imputed value should not possibly not affect the target value too much.

The package `mice` combines different imputation methods e.g. Predictive mean matching(pmm), Bayesian linear regression. This package uses pmm for imputing the remaining features with at least one missing value. The following paragraph describes the general approach of pmm based on the work of (**vink2014predictive**). $Y = (Y_{obs}, Y_{mis})$ denotes a variable where Y_{obs} corresponds to the observable values and Y_{mis} to the missing values in Y. All other variables that can be used to predict Y are $X = (X_{obs}, X_{mis})$ where X_{obs} refers to covariates of Y_{obs} and X_{mis} to covariates of Y_{mis}. The imputation method now either uses logistic regression or linear regression. We assume that pmm uses linear regression one can write the prediction for the ith unit

$$Y_i = \mathbf{X_i^T}\beta + \epsilon_i, \tag{2}$$

w-here Y_i is the ith target value, X_i the vector of values, β the vector of coefficients for more dimensional linear regression and ϵ_i is a normally distributed random error with zero as expectation value and variance σ^2. The first step is to compute estimates given Y_{obs} and X_{obs} for $\hat{\beta}$, $\hat{\sigma}$

and $\hat{\epsilon}$ by means of ordinary least squares. Next use the estimates to calculate σ^{2*} and β^* to draw distributions of these parameter that can be used to calculate

$$\hat{Y}_{mis} = X_{mis}\beta^* \tag{3}$$

. To find the best matching $\hat{Y}_{obs}$ one has to calculate for each $\hat{Y}_{mis,i}$

$$\Delta = |\hat{Y}_{obs} - \hat{Y}_{mis,i}| \tag{4}$$

and pick the observation of Y_{obs} with the smallest Δ and use it as imputation. Mice typically performs multiple imputations and saves the whole dataset that is then presented to the user who has to decide which imputation set is to choose.

5. Feature Selection

At this point, already a lot of preprocessing steps have been conducted. So far the focus of the work has been on cleaning the data and improving the quality (imputing missing data etc.) - all steps increasing at the maximization and optimization of the usable observations. While a large amount of observations is desirable, having many features, like the 79 features of the *Ames Housing Data Set*, is associated with some limitations. Those 79 features might not be an excessive number of variables (especially considering that in industry applications often thousands of variables are considered), yet their number is still sufficiently large to have a high chance of having some variables that are redundant or only exercise a very minor influence.

The first limitation which is caused by high-dimensional data sets is a very technical one: *computational speed* (Guyon and Elisseeff 2003; Kohavi and John 1997; Kursa and Rudnicki 2010; Schiffner et al. 2016). While there have been huge improvements in the computing speed over the recent years, allowing ever larger and larger models, still the number of features is an important influence factor for the computing speed. Especially the presence of irrelevant features or correlated features is said to rapidly degrade the performance of many learner algorithms (Kohavi and John 1997).

Another limitation is the actual model *interpretability* (Hastie et al. 2009; Schiffner et al. 2016). While using all 79 features might theoretically be useful to not miss any potentially influential detail, it is quite likely to lose the big picture and fail to make useful recommendations based on created models (Hastie et al. 2009).

A third issue associated with the *model accuracy* that can be reached based on the given features. Typically, one might assume that a higher number of features should always provide better results. However, research shows that often an overly large feature set is rather decreasing the predictive performance (Kohavi and John 1997). The resulting high-dimensional feature spaces lead to an effect which is commonly called the Curse of Dimensionality (Bellman 1961; Verleysen and François 2005). This curse basically describes the problem that for the given large set of variables only relatively little observations are present to learn train a predictive model. E.g. Verleysen and François (2005) use the example that a feature might need 10 observations to be learned - so that a 2D space would already need 100, and a 3D space 1000.

5.1. Lasso Feature Selection

The first algorithm which is typically used as a regression method is called Lasso and Elastic-Net Regularized Generalized Linear Models (`glmnet`) (Friedman et al. 2010). This section should provide a basic theoretical understanding of `glmnet` and how it can be utilized for feature selection and is based on the work of (Friedman et al. 2010).

As already present in the name the basic learner for generating predictions $\hat{y}$ is based on linear regression. The complete objective function that has to be minimized looks as follows

$$\min_{(\beta_0,\beta)\in\mathcal{R}^{p+1}} \frac{1}{2N}\sum_{i=1}^{N}(y_i - \beta_0 - \mathbf{x_i}^T\beta)^2 + \lambda\left[\frac{(1-\alpha)||\beta||_2^2}{2} + \alpha||\beta||_1\right] \tag{5}$$

where the left summand is the loss function as quadratic error of the prediction $\hat{y} = \beta_0 + \mathbf{x_i^T}\beta$ to the real observation y_i and the right summand describes the penalization of the coefficients β.

To solve the optimization problem the general procedure is to compute the first derivative, set all the resulting function to zero and calculate the independent variable which in this case is β_j assuming that there already is an estimation about β_l where $l \neq j$

This regression is called elastic net because of the compromise between `ridge regression penalty` when α is chosen to be zero and `lasso penalty` when α is chosen to be one. Since α highly effects the calculation of β and therefore the prediction it is in the set of tuning parameters.

The other parameter λ describes the general penalty for the coefficients β. Being able to get a set of features as output of the glmnet execution one has to set λ to a specific value an take all features for which the β coefficient is unequal to zero. This means that for the given λ the feature is still important enough for the linear regression model that regression accuary overweights the penalty for the coefficient. In the implementation of this package we decided to choose the λ that minimizes the total deviation of the of the linear regression. The effect of λ on the value of the coefficient in this setup is shown in the Figure 7. When λ increases more and more β coefficients eventually become zero. But coefficients that later become zero do have a bigger impact on the linear regression model since the squared deviation of the prediction $\hat{y}_i$ is still smaller than the increase due to the higher penalty given by a larger λ.

5.2. Boruta Feature Selection

In difference to the Lasso feature selection algorithm (see Section 5.1), the Boruta algorithm (Kursa and Rudnicki 2010; Kursa et al. 2010) is relatively young with the first publications dating back to the year 2010. However, it is not only the age that sets this algorithm apart from the older Lasso, but also the general aim that this method follows. For this one must understand that in feature selection multiple problem scenarios are thinkable: The first candidate is about finding a minimal feature set that delivers optimal prediction results, the so called *Minimal-Optimal*

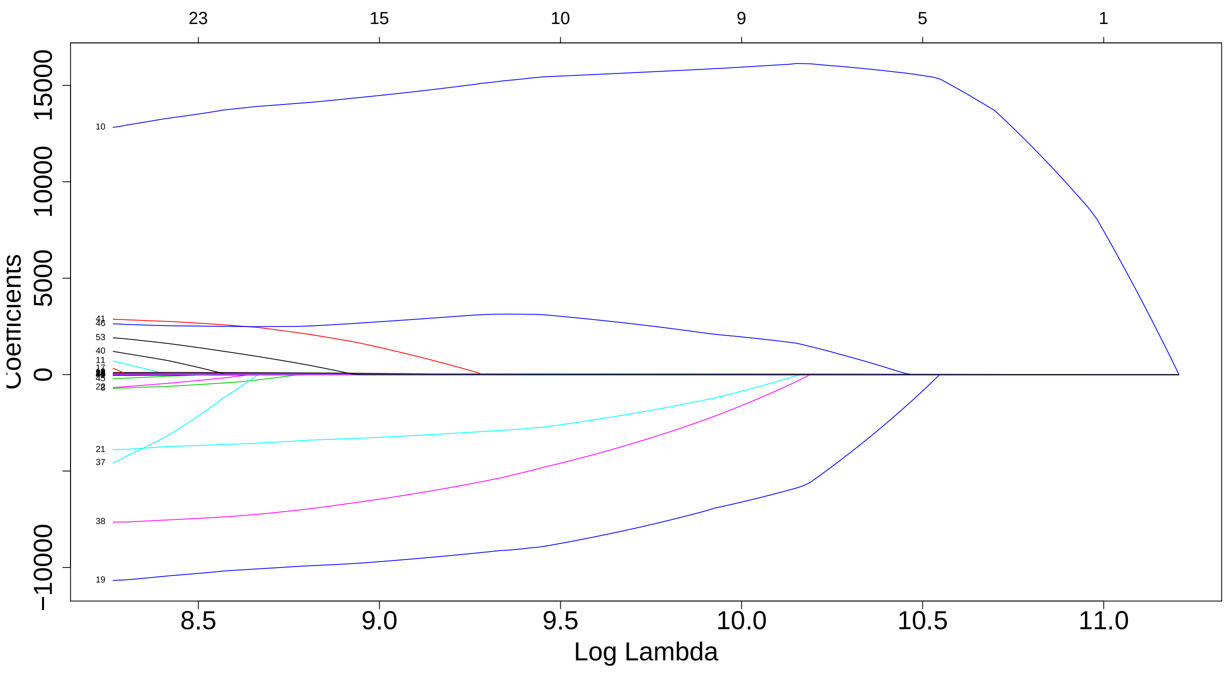


Figure 7: Effect of lambda on the value of the coefficients

problem (Nilsson et al. 2007). As a second option one can search for all possibly relevant variables leading to the so-called *All-Relevant* problem (Nilsson et al. 2007) (for visual comparison see Figure 12).

While most algorithms, like Lasso, focus on obtaining the minimal set of needed features, Boruta is an *All-Relevant* algorithm (which in itself is a relatively young research area (Rudnicki et al. 2015)). Intuitively the idea of selecting all relevant variables might sound strange given the introductory part stating that it is desirable to reduce the feature set as far as possible. Nevertheless Kursa and Rudnicki (2010) point out that at some point it can be beneficial to know all influential variables - even if they are correlated with another one - to ensure a full understanding of the predictive model. While they use medical research as one example, where identifying all cancer-causing genes can be important, similar use cases are also thinkable for the real estate market.

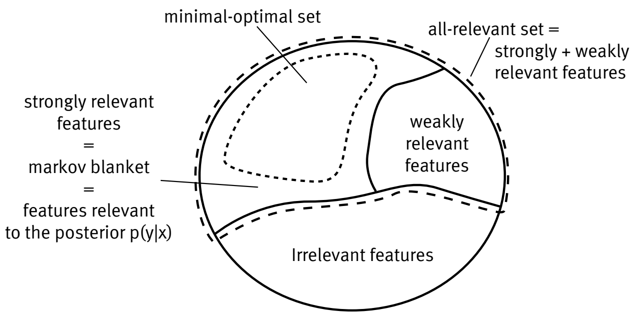


Figure 8: Feature Subsets - *All-Relevant* and *Minimal-Optimal* (Nilsson et al. 2007)

On the algorithmic part the Boruta algorithm can be seen as some kind of a wrapper around the `randomForest` (Liaw and Wiener 2002) package, respectively a random forest implementation (newer versions use `ranger` (Wright 2016) for performance reasons) (Kursa and Rudnicki 2010). The random forests are used to classify the given observations as they have the useful property of running without extensive parameter tuning (Kursa and Rudnicki 2010). Afterwards the loss of accuracy is computed for each tree (since all of them use different data samples) and then aggregated via the Z score[5]. Since the Z score of random forests is not within $\mathbb{N}(0, 1)$ (Rudnicki et al. 2006) KURSA AND RUDNICKI (2010) propose the use of so called *shadow attributes*.

F_1	F_2	F_3	F_4
1	1	2	3
3	0	2	1
2	1	3	2

(a) Original Data

S_1	S_2	S_3	S_4
3	1	2	1
1	1	3	2
2	0	2	3

(b) Shadow Features

	F_1	F_2	F_3	F_4	S_1	S_2	S_3	S_4
MDI	.2	.05	.01	.15	.001	.02	.009	.09
Hit	+1	0	0	+1	-	-	-	max

(c) Selection of Relevant Features

Table 6: Shadow Attribute Example (Homola 2015)

These shadow variables are basically copies of original variables that have been inserted to the dataset in randomized order. Due to the value shuffling the influence of the shadow attributes should be zero, however, via random fluctuations it can be greater than 0. So, the authors KURSA AND RUDNICKI (2010) propose to classify the data incl. the shadow variables. Afterwards the shadow variable with the largest influence factor (Z score) is taken as a reference. Each real feature having less influence is removed from the data set. Finally, the shadow variables are removed as well. This procedure will be iterated till a converged state is reached or a certain number of steps has been executed (whole process see Figure 9).

After the algorithm is finished the user will be given three sets of variables: *Confirmed* (all features which are confirmed to be relevant), *Tentative* (variables where the decision could not be made so far) and *Rejected* (features that have been found to be irrelevant) variables (Kursa and Rudnicki 2010). Since the original package was lacking certain functionality the `KaggleHouse` package required and because the usability sometimes was inconvenient, the `KaggleHouse` package provides a separate `Boruta` wrapper. It can either be called as a preconfigured function via

```
# Requires prior execution of run_generate_data().
boruta <- KaggleHouse:::feature.boruta()
```

or also as a function where each parameter can be set manually:

[5] The Z score is computed by dividing the average loss by its standard deviation.

```r
# Shorten real variable name for snippet brevity. Requires prior execution
# of run_generate_data.
data <- data_train_numeric_clean_imputed

# Creates a new Boruta object based on training data. Ensures that confirmed,
# rejected and tentative variables become object parameters and adds a
# formula object e.g. usable for linear models.
boruta <- KaggleHouse::feature.boruta.comp(
  target = data$SalePrice, predictors = data[-ncol(data)], variables = T,
  selected = T, formula = T, maxRuns = 1000,  doTrace = 2, verbose = T
)
```

Furthermore, every learner provides a switch that allows to conduct a prior Boruta feature selection run, before the actual learning process is started. Please note that the execution of a Boruta run allocates a lot of computational resources over a longer time span of at least 30-60 minutes[6].

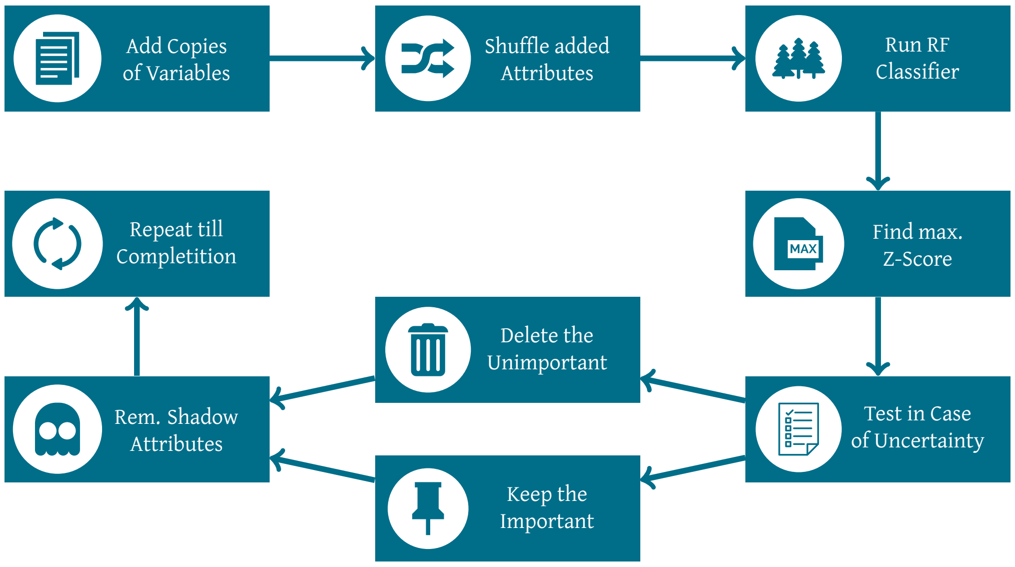


Figure 9: Boruta Algorithm Steps (Kursa and Rudnicki 2010)

To get information about the selected variables the `Boruta` package offers different functionalities. These have again been extended and wrapped for the `KaggleHouse` package to increase convenience of result analysis. Calling

```r
# To obtain an analysable Boruta object.
boruta <- KaggleHouse::feature.boruta()
KaggleHouse::feature.boruta.report(boruta)
```

[6] Computations have been executed on an AMD Phenom FX-8320 with eight cores with each 3.5 GHz, 16 GB RAM and a 128 GB SSD. The average CPU load is about 90-95%.

With a created Boruta object will return two graphs with information regarding selected features (see Figure 10) as well as information about the influence of features over time. Furthermore, a summary statistic will be prepared gathering most information in textual format.

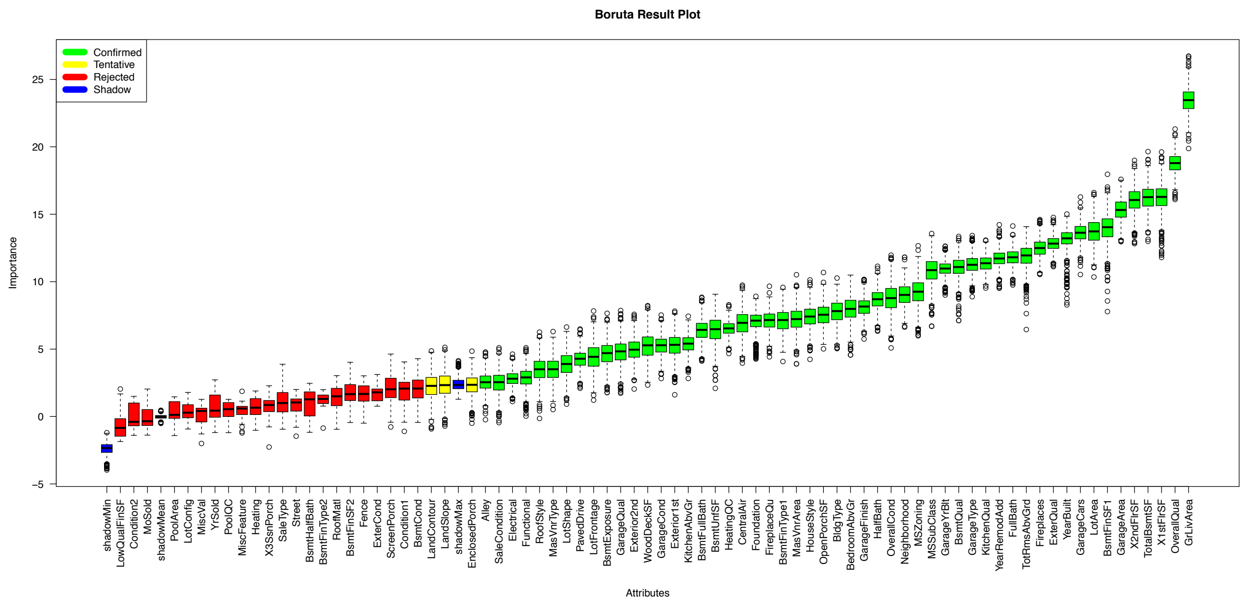


Figure 10: Boruta Result for the Training Data Set

In the example output Figure 10 one can see that in the run conducted in the course of the work on this thesis, 22 features were excluded, three were still tentative and 53 were confirmed. Furthermore, the plot also indicates that `OverallQual` and `GrLivArea` are the most influential variables present. As a rather technical side note it might be worth mentioning that some correlated variables (`TotalBsmtSF`, `BsmtFinSF1`, ...) are included as expected for an *All-Relevant* algorithm[7].

6. Iterated racing(irace)

For tuning the basic learner xgboost and glmnet this package uses the control structure irace (López-Ibáñez et al. 2016). Since the parameter set of the tuned learner is out of scope for grid search we decided to use irace which should outperform random search in the parameter space. While tuning with respect to time and computational resources the maximum number of operations in glmnet is set to 2000 and to 1000 in xgboost with threefold cross validation.

```
kagglehouse_task = mlr::makeRegrTask(id = "kh", data = data,
                                     target = "SalePrice")
basic_lrn = mlr::makeLearner("regr.glmnet")
ps = ParamHelpers::makeParamSet(...)
rdesc = mlr::makeResampleDesc("CV", iters = 3)
```

[7] Due to `TotalBsmtSF = BsmtFinSF1 + BsmtFinSF2 + BsmtUnfSF`.

```
ctrl =  mlr::makeTuneControlIrace(maxExperiments = 2000L)
mlr::tuneParams(basic_lrn, task = kagglehouse_task,
                resampling = rdesc, par.set = ps,
                control = ctrl, show.info = TRUE)
```

This chapter is based on the work of (López-Ibáñez et al. 2016) and roughly summarizes the idea behind irace.

Irace basically consists of the following three steps

1. Sampling new configurations according to a particular distribution
2. Selecting the best configurations from the newly sampled one by means of racing
   a) Start with the set of configurations from 2.
   b) Evaluate all configurations on one instance
   c) Discard configurations that perform statistically worse
   d) Continue racing with the remaining configurations until a certain stopping criteria
3. Updating the sampling distribution in order to bias the sampling towards the best configurations

until a termination criteria is met (López-Ibáñez et al. 2016). Configurations in the context of learner tuning is a set of parameter for that learner to enable execution and assessment of it on an instance.

7. Learner

The idea of using exactly these three basic learners is based on other publicly available solutions. We decided to add `deeplearning` as third basic learner without tuning since `deeplearning` with enough hidden layers and nodes per layer should be able to predict the `SalePrice`. Our initial approach was directed towards a comparison of state of the art learners in order to pick the one with the best score for which the mean-squared error (MSE) is used. But the MSE of all basic learners was not sufficiently low to only use one basic learner. Given this we decided to combine to prediction of these three learner with a stacked learner which is described in more detail in the section 7.3.

[7]https://www.kaggle.com/zenstat/house-prices-advanced-regression-techniques/xgboost-lasso-copied

7.1. Extreme gradient boosting(Xgboost)

We decided to incorporate xgboost since it is used in nearly all of the publicly available solution of the top 100 participants in this challenge. Another incentive for using xgboost is that "among the 29 challenge winning solutions 3 published at Kaggle's blog during 2015, 17 solutions used XG-Boost" (Chen and Guestrin 2016). The following paragraph should provide a basic understanding of xgboost based on the work of (Chen and Guestrin 2016).

Xgboost is a highly scaleable and distributable ensemble method based on tree boosting. The additive function for a prediction $\hat{y}$ can be written as

$$\hat{y}_i = \phi(x_i) = \sum_{k=1}^{K} f_k(x_i). \tag{6}$$

Each f_k represents one independent regression tree q with leaf weights w that is basically a score contributing to the total regression value. To compute a regression score for an instance one has to insert this instance into each f_k and sum up all scores to obtain the final regression score. This is also shown in the following graphic The learning of the set of functions takes places by

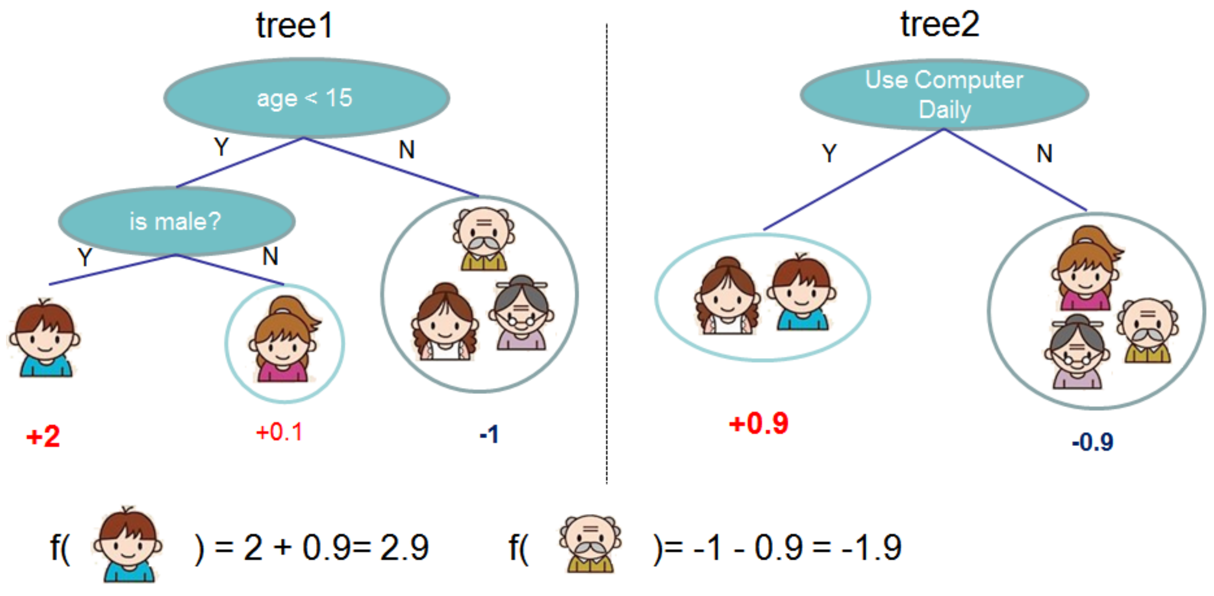


Figure 11: Tree ensemble method using two regression trees(Chen and Guestrin 2016)

minimizing the following optimization function

$$\mathcal{L}(\phi) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k) \tag{7}$$

7.2. Deeplearning

Most machine learning algorithms and techniques seldom bring it to public reception. Deep Learning, however, was very present in the media (Borowiec 2016; Sang-Hun 2016; Süddeutsche

Zeitung 2016) due to Google's Alpha Go beating the European and World champions in Go, one of the games that Artificial Intelligence (AI) was not able to master till now (Silver et al. 2016). While winning this competition is considered a major breakthrough for machine learning, deep learning has been able to win multiple other competitions over the last years, like handwriting recognition (Ciresan et al. 2010) or traffic sign recognition (Ciresan et al. 2011; Ciresan et al. 2012). The survey paper by SCHMIDHUBER (2015) even points out that deep networks begin to outperform other more traditional machine learning methods like Support Vector Machines. Here it is interesting to note that deep learning is basically an extension of a rather old concept, the Artificial Neural Network (ANN). While the first ideas date back to the 1950s/1960s, the methodology needed some time to be optimized as well as advances in computational power before becoming popular around the 1990s/2000s (LeCun et al. 2015; Schmidhuber 2015). The general concept of neural is to model the human brain, respectively the way it functions (Haykin 2008). It does so by combining a huge number of very simple decision nodes (so called neurons) that can learn by the adjustment of the weights of inputs and outputs (Haykin 2008).

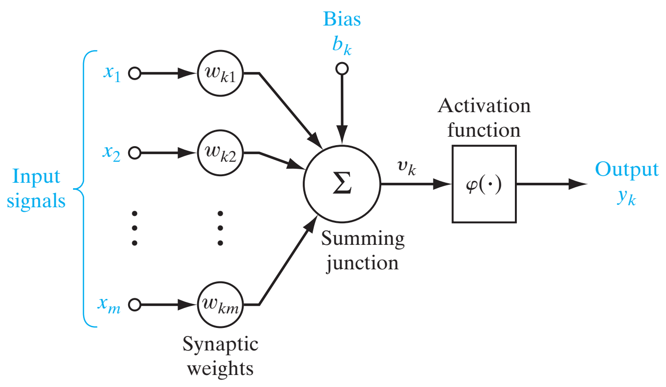
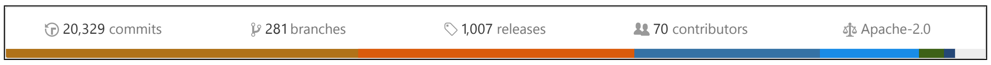


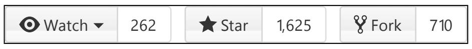
Figure 12: Schema of an ANN Neuron (Haykin 2008)

The major difference is that '*classical*' neural networks usually never use more than three layers (input - hidden - output) (Deng and Yu 2013), whereas deep neural nets use multiple hidden layers. Beside this neural nets and thus also deep learners exhibit further valuable properties: This includes neural networks being able to self-adjust their weights over time (Haykin 2008) as well as their ability to work without complex feature engineering before usage (LeCun et al. 2015). Considering all these facts it only appeared to be reasonable to try deep learning on the given regression problem.

For the implementation of the deep learning only one package was suitable. Since model training, tuning and execution were conducted with `mlr` (Bischl et al. 2016b) the only available deep learner was the `h2o` package (Aiello et al. 2016). This `R` package basically builds a wrapper around the `h2o` machine learning library. Yet even apart from being a somewhat forced decision the `h2o` library appears to be an appropriate choice as according to own claims the library has already 30.000 users in 2.000 cooperations (Candel et al. 2015). Looking at the GitHub repo of `h2o` indicates that this package is very active but also rather mature (given the huge number of commits so far). The actual implementation of `h2o` was then slightly more difficult than anticipated. While neural networks (and deep learners) can also profit from tuning, this was basically infeasible with the given computational resources. The `mlr` learner failed completely and the grid search functionality provided by `h2o` could only test a pretty limited parameter set before fail-



(a) Code Statistics



(b) Fork Statistics

Figure 13: GitHub Statistics of the h2o Package

ing due to performance issues. This way only a test with a very limited parameter set could be conducted:

```
hyper_params <- list(hidden = list(c(2,2), c(4,4),...,
  c(1024,1024,1024)), input_dropout_ratio = c(0, ..., 0.2),
  rate = c(0.01, 0.02), rate_annealing = c(1e-8, ..., 1e-5),
  epochs = c(seq(from = 10, to = 90, by = 10), ...,
    seq(from = 10000, to = 90000, by = 10000), 100000),
  activation = c("Tanh", "Rectifier", "Maxout"))
```

Some other combinations like larger networks had to be trained manually each time to evaluate their quality. Interestingly especially huge networks suffered from the issue that they later predicted each target for the test data set with the exact same value - even despite providing a good predictive performance on the training data set.

Thus, for our final model we decided to use a deep learner with 3000 nodes split equally across ten different layers:

```
# Start the h2o instance that has to conduct the deep learning process.
h2o_init = h2o::h2o.init(nthreads = -1)

# Train and predict with deeplearning
kagglehouse_task = mlr::makeRegrTask(
  id = "kh", data = input_train, target = "SalePrice"
)
base_lrn_deeplearning = mlr::makeLearner(
  "regr.h2o.deeplearning",
  par.vals = list(hidden=rep(300,10), epochs = 3000)
)
deeplearning_model = mlr::train(base_lrn_deeplearning, kagglehouse_task)
```

The results can be found in table 7. All in all, the numbers are slightly disappointing regarding the promise deep learners should have (see introductory part). Here the missing tuning might have had some impact as well as the pretty limited number of observations (only 1460).

7.3. Stacked Learner

After finishing the tuning process of the basic learner the results were not as good as expected. The resulting MSE with regard to the feature selection methods are shown in the table 7. It was

	whole	glmnet	boruta
xgboost	$5.50e+08$	$5.58e+08$	$7.61e+06$
glmnet	$7.74e+08$	$9.73e+08$	$7.39e+08$
deeplearning	$3.91e+08$	$3.79e+08$	$5.79e+06$

Table 7: Results of tuning the basic learner xgboost, glmnet and deeplearning using different feature selection strategies

clear that features provided by boruta outperforms all other feature selection methods regarding the mean of the MSE over the folds. So we decided to drop all features that are excluded by boruta to the stacked learner. Next the best configuration of the turning are manually inserted into each learner to compute the predictions on the boruta dataset.

When sketching the structure of the stacked learner we generally followed the proposed model from the work (Wolpert 1992). Wolpert first proposes to include a variety of classifiers at the level-0 network. The general idea behind that is that each basic learner on the lowest network layer is able to find one local minima on the error space and then to combine all predictions to improve the performance (Hatami and Ebrahimpour 2007). The level-1 network proposed by Wolpert should have a relatively simple structure. In addition many top participants of the kaggle competition used basic weightings for each set of predicitions which is basically linear regression The structure of the stacked learner used in this package is shown in the figure 14. In comparison to the serial learning of the stacked approach we decided to provide the boruta dataset to each learner as training dataset simply because the size of the dataset was to small for splitting it up into good and bad rows. Then each learner is trained on this dataset and the three sets of predictions are forwarded into the level-1 learner which is as mentioned before a simply linear regression. The resulting weightings of the linear regression are shown in the formula 8.

$$\hat{y} = -5.829 + 0.9947 \cdot p_1 - 0.0001 \cdot p_2 + 0.0054 \cdot p_3 \tag{8}$$

Another advantage of linear regression is the interpretabilty. In this scenario the final prediction of the stacked learner is with a weigthing around 0.99 nearly solely based on the results of xgboost although both basic learner do have a similar error rate. The resulting MSE of the stacked learner highly improved to around 80000.

	`xgboost`	`glmnet`	`deeplearning`	`stacked`
Kaggle Score	0.14966	0.17410	0.13240	0.14065

Table 8: Kaggle Scores for the used Learners

Since no seed has been set, each execution of the learners will provide slightly different results. Uploading the datasets predicted with the respective learners to Kaggle returned the scores displayed in Table 8.

[7]For comparison see e.g. BLUEDRAGON (2016) and CHOUDHARY (2016).

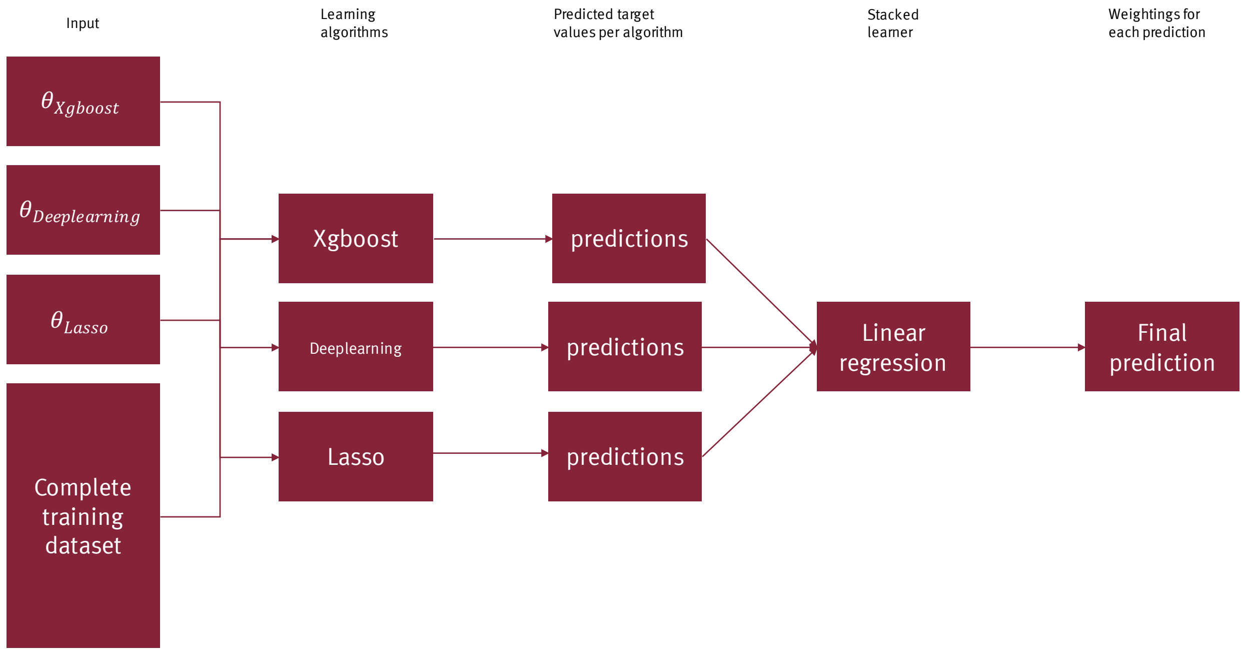


Figure 14: General implementation schema of the stacked learner

Looking at the results it becomes visible that in the last run conducted for the creation of this thesis, deep learning with Boruta feature selection outperformed all other learners – even including the stacked learner. Especially the last point has been surprising, since the stacked learner previously managed to predict best, delivering a Kaggle score of 0.12553. At this point it is hard to determine the exact reason, but it appears to be a reasonable assumption that a good fit on the training set might in some cases lead to overfitting on the test data set.

8. Outlook

The current state of the KaggleHouse package provides a solid basis for the analysis of the Ames Housing data set. It already allows its users to make a through data analysis and provides methods to clean the data set from `NA` values and other non-fitting values. Apart from this preparatory functionality KaggleHouse provides a well-performing ensemble of learners and feature selectors that were able to reach a way better than average result in the *Houses Price: Advanced Regression Techniques* challenge.

Nevertheless, it is apparent that there are still various paths for optimization for the presented package: One path would be to do more extensive tuning of the learner parameters and to add further suitable learners to the ensemble. Given sufficient computational resources it would be interesting to replace the `irace` heuristic with an actual grid search. This way it can be ensured that an optimal parameter set will be found in a given value range.

The second path would be to increase the number of layers of the stacked learner. So instead of only creating a simple linear model over the predictions of the three implemented learners, their output could be the input for another set of level-1 learners. From here they could be aggregated by a learner or the above step could be iterated multiple times. Thinking back to the procedure depicted in Figure 7.3 this would result in multiple predictions layers between the learning algorithms and the stacked learner.

The third option would be to invest more time into the imputation, engineering and selection of features since current results already indicate overfitting. Here the underlying rationale is to create stronger features by combining less influential variables so that a feature selection might result in a much smaller and easier to train set of features. Simultaneously the removal of information should prevent overfitting.

A fourth path to go would be to generalize the package further. Currently many of the methods are biased a lot towards the *Houses Prices* challenge and it would be desirable to create a package that can provide a good entry point to a larger set of regression competitions.

Finally, the performance would be open for optimization, too. For the most parts `R`'s internal options for tuning (compilation) have been used and smaller parts have already been implemented in `C++` to boost the run time. However, some of the packages used (`xgboost`, `Boruta`) might have more efficient implementations in other languages like `Python` or `C++` so that a successive replacement with `R` as an interface could realize a significant improvement.

9. Acknowledgements

We want to thank Matthias Carnein and Pascal Kersche for their support with questions arising in the course of the work. Furthermore we want to thank Dennis Assenmacher for providing interesting ideas as well as code templates for the work with the `Rcpp` package.

A. Setup

A.1. Installing the KaggleHouse Package

To ensure that all required dependencies are installed on your system, copy the following R snippet to your R environement and execute it. First it will check which of the obligatory packages are already installed, and in a subsequent step only add those which are still missing.

```
toInstall <- c("Amelia", "Boruta", "Ckmeans.1d.dp", "compiler", "corrplot",
               "dbscan", "FSelector", "ggplot2", "glmnet", "gridExtra",
               "h2o", "irace", "magrittr", "methods", "mice", "mlr",
               "parallelMap", "ParamHelpers", "Rcpp", "xgboost")
check <- toInstall %in% rownames(installed.packages())

if(any(!check)) {
  install.packages(toInstall[!check])
}
```

A.2. Installing Dependencies

A.2.1. Installing `mlr 2.10` Package

The current CRAN version of `mlr` (Version `2.9`) causes issues with other packages like the `irace` package. To obtain the most recent `mlr` version containing bug fixes for such problems the following code snippet can be executed to remove old version and to install the most recent GitHub release.

```
# The following two commands remove any previously installed mlr packages
# for R (e.g. the one which is installed by the RStudio package manager).
if ("package:mlr" %in% search()) { detach("package:mlr", unload=TRUE) }
if ("mlr" %in% rownames(installed.packages())) { remove.packages("mlr") }

# Now we download, install and initialize the mlr GitHub package for R.
devtools::install_github("mlr-org/mlr")
```

A.2.2. Installing `xgboost 0.6.2` Package

In order to update to the `xgboost` version `0.6.`, being the most recent version at the time of the writing of this documentation, you can either use the *Update* tab in RStudio's *Package* tab or the following code snippet:

```r
# The following two commands remove any previously installed mlr packages
# for R (e.g. the one which is installed by the RStudio package manager).
if ("package:xgboost" %in% search())
  { detach("package:xgboost", unload=TRUE) }
if ("xgboost" %in% rownames(installed.packages()))
  { remove.packages("xgboost") }

# Now we download, install and initialize the mlr GitHub package for R.
install.packages(xgboost)
```

A.2.3. Installing h2o Package

While the h2o version on CRAN is still 3.10.0.8, the current nightly releases on the official h2o website already reached version 3.11.x.y. Since using the official releases with mlr caused several problems, the following snippet helps to install a more up-to-date nightly build of h2o that has been found to work more stable in combination with mlr.

To get the most recent nightly release visit h2o-release and click on the '*Install in R*' tab. There you can find the most recent version of the install script below.

```r
# The following two commands remove any previously installed h2o packages
# for R (e.g. the one which is installed by the RStudio package manager).
if ("package:h2o" %in% search()) { detach("package:h2o", unload=TRUE) }
if ("h2o" %in% rownames(installed.packages())) { remove.packages("h2o") }

# Next, we download packages that H2O depends on.
if (! ("methods" %in% rownames(installed.packages())))
  { install.packages("methods") }
if (! ("statmod" %in% rownames(installed.packages())))
  { install.packages("statmod") }
if (! ("stats" %in% rownames(installed.packages())))
  { install.packages("stats") }
if (! ("graphics" %in% rownames(installed.packages())))
  { install.packages("graphics") }
if (! ("RCurl" %in% rownames(installed.packages())))
  { install.packages("RCurl") }
if (! ("jsonlite" %in% rownames(installed.packages())))
  { install.packages("jsonlite") }
if (! ("tools" %in% rownames(installed.packages())))
  { install.packages("tools") }
if (! ("utils" %in% rownames(installed.packages())))
  { install.packages("utils") }

# Now we download, install and initialize the H2O package for R.
```

```r
install.packages(
  "h2o",
  type="source",
  repos=(c("http://h2o-release.s3.amazonaws.com/h2o/master/3723/R"))
)
```

References

Aiello, Spencer, Tom Kraljevic, and Petr Maj (2016). *h2o: R Interface for H2O*. URL: http://www.h2o.ai/.

Athanasopoulos, George and Rob J. Hyndman (2011). "The value of feedback in forecasting competitions". In: *International Journal of Forecasting* 27.3, pp. 845–849. ISSN: 01692070. DOI: 10.1016/j.ijforecast.2011.03.002. URL: http://www.sciencedirect.com/science/article/pii/S0169207011000495.

Auguie, Baptiste (2016). *gridExtra: Miscellaneous Functions for "Grid" Graphics*. URL: https://cran.r-project.org/package=gridExtra.

Bache, Stefan Milton and Hadley Wickham (2014). *magrittr: A Forward-Pipe Operator for R*. URL: https://cran.r-project.org/package=magrittr.

Bellman, Richard E. (1961). *Adaptive Control Processes - A Guided Tour*. Princeton University Press. ISBN: 9780691625850. URL: http://press.princeton.edu/titles/101.html.

Bischl, Bernd and Michel Lang (2015). *parallelMap: Unified Interface to Parallelization Back-Ends*. URL: https://cran.r-project.org/package=parallelMap.

Bischl, Bernd et al. (2016a). *ParamHelpers: Helpers for Parameters in Black-Box Optimization, Tuning and Machine Learning*. URL: https://cran.r-project.org/package=ParamHelpers.

Bischl, Bernd et al. (2016b). *mlr: Machine Learning in R*. URL: https://cran.r-project.org/package=mlr.

Bischl, Bernd et al. (2016c). "mlr: Machine Learning in R". In: *Journal of Machine Learning Research* 17, pp. 1–5. URL: http://jmlr.org/papers/v17/15-066.html.

BlueDragon (2016). *XGBoost + Lasso - *Copied**. URL: https://www.kaggle.com/zenstat/house-prices-advanced-regression-techniques/xgboost-lasso-copied (visited on 01/10/2017).

Borowiec, Steven (2016). *AlphaGo seals 4-1 victory over Go grandmaster Lee Sedol*. URL: https://www.theguardian.com/technology/2016/mar/15/googles-alphago-seals-4-1-victory-over-grandmaster-lee-sedol (visited on 01/10/2017).

Breiman, Leo (2001). "Random Forests". In: *Machine Learning* 45.1, pp. 5–32. ISSN: 08856125. DOI: 10.1023/A:1010933404324. arXiv: /dx.doi.org/10.1023{\%}2FA{\%}3A1010933404324 [http:]. URL: http://link.springer.com/10.1023/A:1010933404324.

Breunig, Markus M. et al. (2000). "LOF: Identifying Density-Based Local Outliers". In: *Proceedings of the 2000 ACM Sigmod International Conference on Management of Data*, pp. 93–104. ISSN: 01635808. DOI: 10.1145/335191.335388. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.35.8948.

Buuren, S. and K. Groothuis-Oudshoorn (2011). "mice: Multivariate Imputation by Chained Equations in R". In: *Journal of Statistical Software* 45.3. ISSN: 1548-7660.

Candel, Arno et al. (2015). *Deep Learning with H2O*. Third Edit. August. H2O.ai, Inc. URL: http://h2o.gitbooks.io/deep-learning/.

Chen, Tianqi and Carlos Guestrin (2016). “XGBoost: A Scalable Tree Boosting System”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD ’16*. New York, New York, USA: ACM Press, pp. 785–794. ISBN: 9781450342322. DOI: 10.1145/2939672.2939785. URL: http://dl.acm.org/citation.cfm?id=2939672.2939785http://dl.acm.org/citation.cfm?doid=2939672.2939785.

Chen, Tianqi, Tong He, and Michael Benesty (2016). *xgboost: Extreme Gradient Boosting*. URL: https://cran.r-project.org/package=xgboost.

Choudhary, Amit (2016). *script_v6*. URL: https://www.kaggle.com/amitchoudhary/house-prices-advanced-regression-techniques/script-v6/run/512428/code (visited on 01/10/2017).

Ciresan, Dan Claudiu et al. (2010). “Deep Big Simple Neural Nets Excel on Handwritten Digit Recognition”. In: *Neural Computation* 22.12, pp. 3207–3220. ISSN: 1530-888X. DOI: 10.1162/NECO_a_00052. arXiv: 1003.0358. URL: http://arxiv.org/abs/1003.0358.

Ciresan, Dan et al. (2011). “A committee of neural networks for traffic sign classification”. In: *The 2011 International Joint Conference on Neural Networks*. Vol. 1. 1. San Jose, California, USA: IEEE, pp. 1918–1921. ISBN: 978-1-4244-9635-8. DOI: 10.1109/IJCNN.2011.6033458. URL: http://ieeexplore.ieee.org/document/6033458/.

Ciresan, Dan et al. (2012). “Multi-column deep neural network for traffic sign classification”. In: *Neural Networks* 32, pp. 333–338. ISSN: 08936080. DOI: 10.1016/j.neunet.2012.02.023. arXiv: arXiv:1202.2745v1. URL: http://dx.doi.org/10.1016/j.neunet.2012.02.023http://linkinghub.elsevier.com/retrieve/pii/S0893608012000524.

De Cock, Dean (2011). “Ames, Iowa: Alternative to the Boston Housing Data as an End of Semester Regression Project”. In: *Journal of Statistics Education* 19.3, pp. 1–15. ISSN: 10691898. URL: www.amstat.org/publications/jse/v19n3/decock.pdf.

Deng, Li and Dong Yu (2013). “Deep Learning: Methods and Applications”. In: *Foundations and Trends® in Signal Processing* 7.3-4, pp. 197–387. ISSN: 09598138. DOI: 10.1136/bmj.319.7209.0a. arXiv: 1309.1501.

Eddelbuettel, Dirk (2013). *Seamless R and C++ Integration with Rcpp*. Springer. ISBN: 978-1-4614-6867-7.

Eddelbuettel, Dirk and Romain Francois (2011). “Rcpp: Seamless R and C++ Integration”. In: *Journal of Statistical Software* 40.8, pp. 1–18. URL: http://www.jstatsoft.org/v40/i08/.

Fayyad, UM, G Piatetsky-Shapiro, and P Smyth (1996). “Knowledge Discovery and Data Mining: Towards a Unifying Framework.” In: *The Second International Conference of Knowledge Discovery and Data Mining*. Portland, Oregon, USA: AAAI Press, pp. 82–88. ISBN: 1-57735-004-9. DOI: 10.1.1.27.363. URL: http://www.aaai.org/Papers/KDD/1996/KDD96-014.

Friedman, Jerome, Trevor Hastie, and Robert Tibshirani (2010). “Regularization Paths for Generalized Linear Models via Coordinate Descent”. In: *Journal of Statistical Software* 33.1, pp. 1–22.

ISSN: 1548-7660. DOI: 10.18637/jss.v033.i01. arXiv: arXiv:0908.3817v2. URL: http://www.jstatsoft.org/v33/i01/https://www.jstatsoft.org/index.php/jss/article/view/v033i01.

Guyon, Isabelle and André Elisseeff (2003). “An Introduction to Variable and Feature Selection”. In: *Journal of Machine Learning Research* 3, pp. 1157–1182. URL: http://www.jmlr.org/papers/volume3/guyon03a/guyon03a.pdf.

Hahsler, Michael (2016). *dbscan: Density Based Clustering of Applications with Noise (DBSCAN) and Related Algorithms*. URL: https://cran.r-project.org/package=dbscan.

Harrison, David and Daniel L. Rubinfeld (1978). “Hedonic housing prices and the demand for clean air”. In: *Journal of Environmental Economics and Management* 5.1, pp. 81–102. ISSN: 00950696. DOI: 10.1016/0095-0696(78)90006-2. arXiv: arXiv:1011.1669v3. URL: http://linkinghub.elsevier.com/retrieve/pii/0095069678900062.

Hastie, Trevor, Robert Tibshirani, and Jerome Friedman (2009). *The Elements of Statistical Learning*. Second. Vol. 1. Springer Series in Statistics. New York, NY: Springer New York. ISBN: 978-0-387-84857-0. DOI: 10.1007/978-0-387-84858-7. arXiv: 1010.3003. URL: http://www.springerlink.com/index/10.1007/b94608.

Hatami, Nima and Reza Ebrahimpour (2007). “Combining Multiple Classifiers: Diversify with Boosting and Combining by Stacking”. In: *IJCSNS International Journal of Computer Science and Network Security* 7.1, pp. 127–131. URL: http://paper.ijcsns.org/07{_}book/200701/200701A18.pdf.

Haykin, Simon (2008). *Neural Networks and Learning Machines*. Third Ed. Vol. 3. Pearson Education, Inc. ISBN: 9780131471399. DOI: 978-0131471399.

Homola, Daniel (2015). *BorutaPy - an all relevant feature selection method*. URL: http://danielhomola.com/2015/05/08/borutapy-an-all-relevant-feature-selection-method/ (visited on 01/03/2017).

Honaker, James, Gary King, and Matthew Blackwell (2011). “Amelia II: A Program for Missing Data”. In: *Journal of Statistical Software* 45.7. ISSN: 1548-7660. DOI: 10.18637/jss.v045.i07. URL: http://www.jstatsoft.org/v45/i07/.

Kaggle Inc (2016b). *House Prices: Advanced Regression Techniques - Competition Rules*. URL: https://www.kaggle.com/c/house-prices-advanced-regression-techniques/rules (visited on 12/28/2016).

– (2016c). *House Prices: Advanced Regression Techniques - Leaderbord*. URL: https://www.kaggle.com/c/house-prices-advanced-regression-techniques/leaderboard (visited on 12/28/2016).

– (2016d). *House Prices: Advanced Regression Techniques - Prizes*. URL: https://www.kaggle.com/c/house-prices-advanced-regression-techniques/details/prizes (visited on 12/28/2016).

– (2016a). *House Prices: Advanced Regression Techniques*. URL: https://www.kaggle.com/c/house-prices-advanced-regression-techniques (visited on 12/26/2016).

Karatzoglou, Alexandros et al. (2004). “kernlab - An S4 Package for Kernel Methods in R”. In: *Journal of Statistical Software* 11.9, pp. 1–20. URL: http://www.jstatsoft.org/v11/i09/.

Kohavi, Ron and George H. John (1997). “Wrappers for feature subset selection”. In: *Artificial Intelligence* 97.1-2, pp. 273–324. ISSN: 00043702. DOI: 10.1016/S0004-3702(97)00043-X. URL: http://www.sciencedirect.com/science/article/pii/S000437029700043X.

Kursa, Miron B. and Witold R. Rudnicki (2010). “Feature Selection with the Boruta Package”. In: *Journal Of Statistical Software* 36.11, pp. 1–13. ISSN: 15487660. DOI: Vol.36,Issue11,Sep2010. URL: http://www.jstatsoft.org/v36/i11/paper.

Kursa, Miron B., Aleksander Jankowski, and Witold R. Rudnicki (2010). “Boruta - A system for feature selection”. In: *Fundamenta Informaticae* 101.4, pp. 271–285. ISSN: 01692968. DOI: 10.3233/FI-2010-288. URL: http://content.iospress.com/articles/fundamenta-informaticae/fi101-4-02.

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton (2015). “Deep learning”. In: *Nature* 521.7553, pp. 436–444. ISSN: 0028-0836. DOI: 10.1038/nature14539. arXiv: arXiv:1312.6184v5. URL: http://dx.doi.org/10.1038/nature14539.

Liaw, Andy and Matthew Wiener (2002). “Classification and Regression by randomForest”. In: *R News* 2.3, pp. 18–22. URL: http://cran.r-project.org/doc/Rnews/.

López-Ibáñez, Manuel et al. (2016). “The irace package: Iterated racing for automatic algorithm configuration”. In: *Operations Research Perspectives* 3, pp. 43–58. ISSN: 22147160. DOI: 10.1016/j.orp.2016.09.002. URL: http://linkinghub.elsevier.com/retrieve/pii/S2214716015300270.

Maindonald, John and W. John Braun (2003). *Data Analysis and Graphics Using R - an Example-Based Approach*. Third. Cambridge University Press. ISBN: 9780521762939. URL: http://www.cambridge.org/catalogue/catalogue.asp?isbn=9780521762939.

Metz, Rachel (2013). “A Startup Called Kaggle Tries to Bring Smart People to Knotty Problems”. In: *MIT Technology Review*, pp. 1–2. URL: https://www.technologyreview.com/lists/innovators-under-35/2013/entrepreneur/anthony-goldbloom/.

Nilsson, Roland et al. (2007). “Consistent Feature Selection for Pattern Recognition in Polynomial Time”. In: *The Journal of Machine Learning Research* 8, pp. 589–612. ISSN: 15324435. URL: http://www.jmlr.org/papers/volume8/nilsson07a/nilsson07a.pdf.

R Core Team (2016). *R: A Language and Environment for Statistical Computing*. Vienna, Austria. URL: https://www.r-project.org/.

Romanski, Piotr and Lars Kotthoff (2016). *FSelector: Selecting Attributes*. URL: https://cran.r-project.org/package=FSelector.

Rudnicki, Witold R. et al. (2006). “A Statistical Method for Determining Importance of Variables in an Information System”. In: *Rough Sets and Current Trends in Computing*. Ed. by Chien-Chung Chan, Jerzy W. Grzymala-Busse, and Wojciech P. Ziarko. Vol. 4259. Lecture Notes in Computer Science. Kobe, Japan: Springer Berlin Heidelberg, pp. 557–566. ISBN: 978-3-540-88423-1. DOI: 10.1007/11908029_58. URL: http://link.springer.com/10.1007/11908029{_}58.

Rudnicki, Witold R., Mariusz Wrzesień, and Wiesław Paja (2015). “All Relevant Feature Selection Methods and Applications”. In: *Studies in Computational Intelligence*. Ed. by Urszula Stańczyk and Lakhmi C. Jain. Vol. 584. Springer Berlin Heidelberg. Chap. Chapter 2, pp. 11–28. ISBN: 978-3-662-45619-4. DOI: 10.1007/978-3-662-45620-0_2. URL: http://www.scopus.com/inward/record.url?eid=2-s2.0-84921712130{\&}partnerID=tZOtx3y1http://link.springer.com/10.1007/978-3-662-45620-0{_}2.

Sang-Hun, Choe (2016). *Google’s Computer Program Beats Lee Se-dol in Go Tournament*. URL: http://www.nytimes.com/2016/03/16/world/asia/korea-alphago-vs-lee-sedol-go.html?{_}r=0 (visited on 01/10/2017).

Schapire, Robert E. et al. (1998). “Boosting the margin: A new explanation for the effectiveness of voting methods”. In: *The Annals of Statistics* 26.5, pp. 1651–1686. ISSN: 00905364. DOI: 10.1214/aos/1024691352. URL: http://projecteuclid.org:80/Dienst/getRecord?id=euclid.aos/1024691352/.

Schiffner, Julia et al. (2016). *mlr Tutorial*. Tech. rep. arXiv: 1609.06146. URL: http://arxiv.org/abs/1609.06146.

Schmidhuber, Jürgen (2015). “Deep learning in neural networks: An overview”. In: *Neural Networks* 61, pp. 85–117. ISSN: 08936080. DOI: 10.1016/j.neunet.2014.09.003. arXiv: 1404.7828. URL: http://linkinghub.elsevier.com/retrieve/pii/S0893608014002135.

Silver, David et al. (2016). “Mastering the game of Go with deep neural networks and tree search”. In: *Nature* 529.7587, pp. 484–489. ISSN: 0028-0836. DOI: 10.1038/nature16961. URL: http://www.nature.com/doifinder/10.1038/nature16961.

Süddeutsche Zeitung (2016). *Google schlägt Mensch - Go-Meisterspieler entschuldigt sich*. URL: http://www.sueddeutsche.de/news/wirtschaft/computer-google-schlaegt-mensch---go-meisterspieler-entschuldigt-sich-dpa.urn-newsml-dpa-com-20090101-160312-99-187694 (visited on 01/10/2017).

Taieb, Souhaib Ben and Rob J. Hyndman (2013). “A gradient boosting approach to the Kaggle load forecasting competition”. In: *International Journal of Forecasting* 30.2, pp. 382–394. ISSN: 01692070. DOI: 10.1016/j.ijforecast.2013.07.005. URL: http://www.sciencedirect.com/science/article/pii/S0169207013000812.

The H2O.ai team (2016). *h2o: R Interface for H2O*. URL: https://github.com/h2oai/h2o-3.

Venables, W. N. and B. D. Ripley (2002). *Modern Applied Statistics with S*. Fourth Edi. Statistics and Computing. New York, NY: Springer New York. ISBN: 978-1-4419-3008-8. DOI: 10.1007/978-0-387-21706-2. URL: http://link.springer.com/10.1007/978-0-387-21706-2.

Verleysen, Michel and Damien François (2005). “The Curse of Dimensionality in Data Mining”. In: *Computational Intelligence and Bioinspired Systems*. Ed. by Joan Cabestany, Alberto Prieto, and Francisco Sandoval. Vol. 3512. Vilanova i la Geltrú, Barcelona, Spain: Springer Berlin Heidelberg, pp. 758 –770. ISBN: 3540262083. DOI: 10.1007/11494669_93. URL: http://www.springerlink.com/index/n65tna6vwt3b1pw6.pdf.

Wang, Haizhou and Mingzhou Song (2011). "Ckmeans.1d.dp: Optimal k-means Clustering in One Dimension by Dynamic Programming". In: *The R Journal* 3/2, pp. 29–33. URL: https://journal.r-project.org/archive/2011-2/RJournal{_}2011-2{_}Wang+Song.pdf.

Wei, Taiyun and Simko Villiam (2016). *corrplot: Visualization of a Correlation Matrix*. URL: https://cran.r-project.org/package=corrplot.

Wickham, Hadley (2009). *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York. ISBN: 978-0-387-98140-6. URL: http://ggplot2.org.

Wickham, Hadley and Winston Chang (2016). *devtools: Tools to Make Developing R Packages Easier*. URL: https://cran.r-project.org/package=devtools.

Wikipedia (2016). *Kaggle*. URL: https://en.wikipedia.org/wiki/Kaggle (visited on 11/27/2016).

Wolpert, David H. (1992). "Stacked Generalization". In: *Neural Networks* 5.2, pp. 241–259. URL: http://www.sciencedirect.com/science/article/pii/S0893608005800231.

Wright, Marvin N. (2016). *ranger: A Fast Implementation of Random Forests*. URL: https://cran.r-project.org/package=ranger.

Zaki, Mohammed J and Meira Jr. Meira (2014). *Data Mining and Analysis: Fundamental Concepts and Algorithms*. 1st Edit. Cambridge University Press, p. 562. ISBN: 9780521766333. URL: http://www.dataminingbook.info/pmwiki.php.