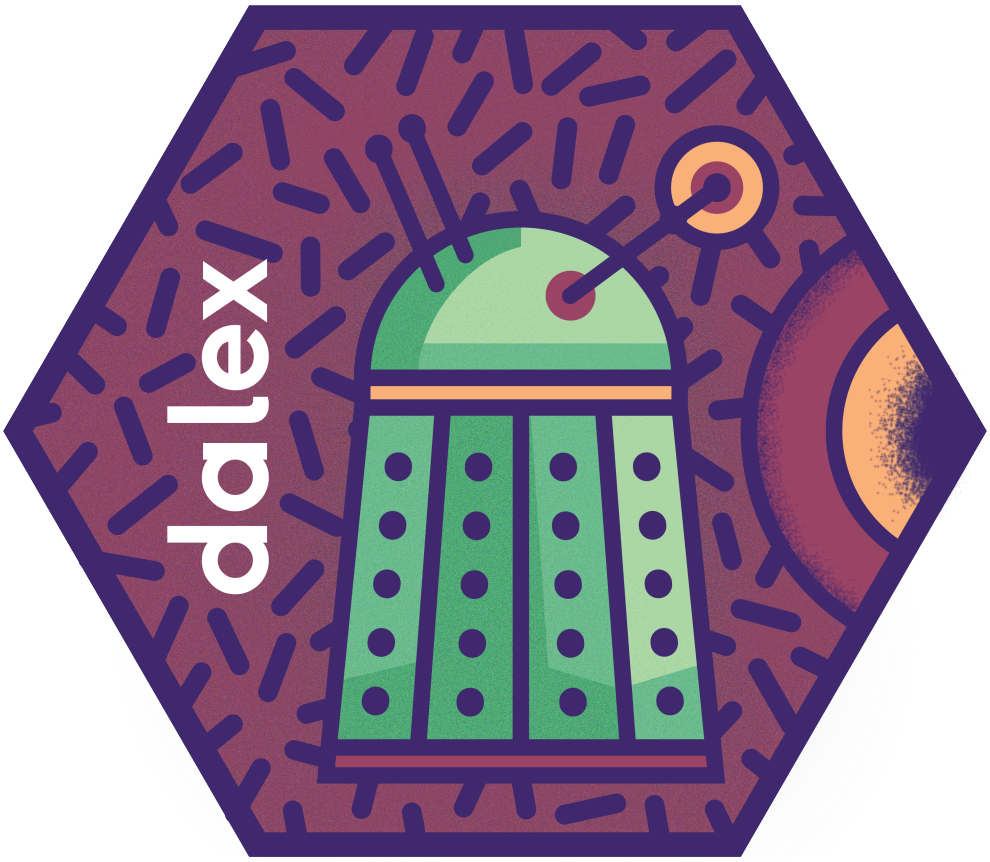
DALEX: Descriptive mAchine Learning EXplanations

Przemysław Biecek

2018-06-19



How to understand a black-box model?

Choose the right visual explainer in 2.875 simple steps

1. Want to understand a model or a single prediction?
- entire model
- prediction for a single observation

2. Is it *how to change it* or *why it happened*?
- interested in *what-if* scenarios
- how variables affected this single prediction

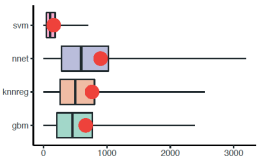
3. Variable attribution or importance?
- decompose prediction (breakDown, Shapley)
- identify key features (live, LIME)

3. Evaluate performance or validate fit?
- compare models performance
- audit residuals and goodness of fit

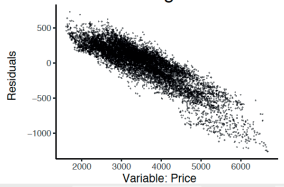
2. Interested in model performance or structure?
- how good is the model
- how does it work

3. Which variable are you interested in?
- all
- a categorical
- a continuous

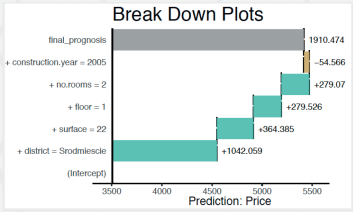
Model Performance Plots



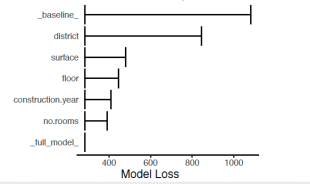
Residual Diagnostic Plots



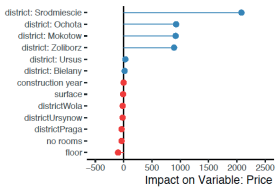
Break Down Plots



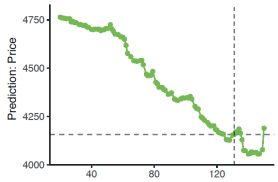
Variable Importance Plots



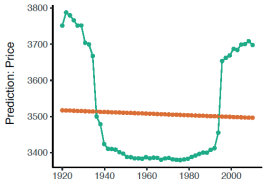
Local Variable Importance



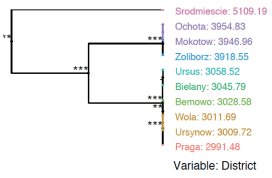
Ceteris Paribus Plots



Partial Dependency Plots



Merging Path Plots



Find more at:
https://github.com/pbiecek/DALEX

Contents

Chapter 1

Introduction

Machine Learning (ML) models have a wide range of applications in classification or regression problems. Due to the increasing computational power of computers and complexity of data sources, ML models are becoming more and more sophisticated. Models created with the use of techniques such as boosting or bagging of neural networks are parametrized by thousands of coefficients. They are obscure; it is hard to trace the link between input variables and model outcomes - in fact they are treated as black boxes. They are used because of their elasticity and high performance, but their deficiency in interpretability is one of their weakest sides.

In many applications we need to know, understand or prove how the input variables are used in the model. We need to know the impact of particular variables on the final model predictions. Thus we need tools that extract useful information from thousands of model parameters.

DALEX (see Biecek, 2018) is an R (R Core Team, 2018) library with such tools. DALEX helps to understand the way complex models work. In this document we show two typical use-cases for DALEX: one case will increase our understanding of a model, while the other will increase our understanding of predictions for particular data points.

Figure 1.1. Workflow of a typical machine learning modeling. A) Modeling is a process in which domain knowledge and data are turned into models. B) Models are used to generate predictions. C) Understanding of a model structure may increase our knowledge, and in consequence it may lead to a better model. DALEX helps here. D) Understanding of drivers behind a particular model's predictions may help to correct wrong decisions, and in consequence it leads to a better model. DALEX helps here.

1.1 Motivation

Machine Learning is a vague name. There is some *learning* and some *machines*, but what the heck is going on? What does it really mean? Is it possible that the meaning of this term evolves over time?

- A few years ago I would say that the term refers to *machines learning from humans*. In the supervised learning problems, a human being creates a labeled dataset and machines are tuned/trained to predict correct labels from data.
- Recently we have more and more examples of *machines that are learning from other machines*. Self-playing neural nets like AlphaGo Zero (Silver et al., 2017) learn from themselves with blazing speed. Humans are involved in designing the learning environment but the labeling turns out to be very expensive or not feasible, and we are looking for other ways to learn from partial labels, fuzzy labels, or no labels at all.
- I could imagine that in close future *humans will learn from machines*. Well trained black-boxes may teach us how to be better at playing Go, how to be better at reading PET images (Positron-Emission

Tomography images), or how to be better at diagnosing patients.

As the human supervision over learning is decreasing over time, the understanding of black-boxes is more important. To make this future possible, we need tools that extract useful information from black-box models.

DALEX is *the tool* for this.

1.1.1 Why DALEX?

In recent years we have been observing an increasing interest in tools for knowledge extraction from complex machine learning models, see (Štrumbelj and Kononenko, 2011), (Tzeng and Ma, 2005), (Puri et al., 2017), (Zeiler and Fergus, 2014).

There are some very useful R packages that may be used for knowledge extraction from R models, see for example `pdp` (Greenwell, 2017), `ALEPlot` (Apley, 2017), `randomForestExplainer` (Paluszynska and Biecek, 2017), `xgboostExplainer` (Foster, 2017), `live` (Staniak and Biecek, 2017) and others.

Do we need yet another R package to better understand ML models? I think so. There are some features available in the DALEX package which make it unique.

- Scope. DALEX is a wrapper for a large number of very good tools / model explainers. It offers a wide range of state-of-the-art techniques for model exploration. Some of these techniques are more useful for understanding model predictions; other techniques are more handy for understanding model structure.
- Consistency. DALEX offers a consistent grammar across various techniques for model explanation. It's a wrapper that smoothes differences across different R packages.
- Model agnostic. DALEX explainers are model agnostic. One can use them for linear models, tree ensembles, or other structures, hence we are not limited to any particular family of black-box models.
- Model comparisons. One can learn a lot from a single black-box model, but one can learn much more by contrasting models with different structures, like linear models with ensembles of trees. All DALEX explainers support model comparisons.
- Visual consistency. Each DALEX explainer can be plotted with the generic `plot()` function. These visual explanations are based on `ggplot2` (Wickham, 2009) package, which generates elegant, customizable, and consistent graphs.

Chapter 2 presents the overall architecture of the DALEX package. Chapter 3 presents explainers that explore global model performance and variable importance of feature effects. Chapter 4 presents explainers that explore feature attribution for single predictions of validation of a model prediction's reliability.

In this document we focus on three primary use-cases for DALEX explainers.

1.1.2 To validate

Explainers presented in Section 3.1 help in understanding model performance and comparing performance of different models.

Explainers presented in Section 4.1 help to identify outliers or observations with particularly large residuals.

Explainers presented in Section 4.2 help to understand which key features influence model predictions.

1.1.3 To understand

Explainers presented in Section 3.2 help to understand which variables are the most important in the model. Explainers presented in Section 4.2 help to understand which features influence single predictions. They are useful in identifying key influencers behind the black-box.

Explainers presented in Section 3.3 help to understand how particular features affect model prediction.

1.1.4 To improve

Explainers presented in Section 3.3 help to perform feature engineering based on model conditional responses.

Explainers presented in Section 4.2 help to understand which variables result in incorrect model decisions. These explainers are useful in identifying and correcting biases in the training data.

1.2 Trivia

The Daleks are a fictional extraterrestrial race portrayed in the Doctor Who BBC series. Rather dim aliens, known to repeat the phrase *Explain!* very often. Daleks were engineered. They consist of live bodies closed in tank-like robotic shells. They seem like nice mascots for explanations concerning Machine Learning models.

Chapter 2

Architecture of DALEX

DALEX's architecture is simple and consistent. Actually, there are only three rules that should be remembered while using this tool.

- First - use the `explain()` function to enrich a black-box model with additional metadata required by explainers. Various explainers require various metadata. You may find their list in Section 2.1.
- Second - use the explainer function that calculates required descriptions. Consecutive explainers are introduced in Chapters 3 and 4.
- Third - use generic `print()` or `plot()` function to see the explainer. Both functions work for one or more models.

These three steps are presented in Figure 2.1.

Figure 2.1. The overview of DALEX's architecture. *A)* Any predictive model with defined input x and output $y_{raw} \in \mathcal{R}$ may be used. *B)* Models are first enriched with additional metadata, such as a function that calculates predictions, validation data, model label or other components. The `explain()` function creates an object belonging to the `explainer` class that is used in further processing. *C)* Specialized explainers calculate numerical summaries that can be plotted with generic `plot()` function.

2.1 The `explain()` function

DALEX is designed to work with various black-box models like tree ensembles, linear models, neural networks etc. Unfortunately R packages that create such models are very inconsistent. Different tools use different interfaces to train, validate and use models. Two most popular frameworks for machine learning are `mlr` (Bischl et al., 2016) and `caret` (from Jed Wing et al., 2016). Apart from them, dozens of R packages may be used for modeling.

This is why as the first step DALEX wraps-up the black-box model with meta-data that unifies model interfacing.

Below is a list of arguments required by the `explain()` function.

```
explain(model, data, y, predict_function,
              link, ..., label)
```

- `model` - an R object, a model to be explained. *Required by*: all explainers.
- `data` - `data.frame` or `matrix`, a set that will be used for model validation. It should have the same structure as the dataset used for training. *Required by*: model performance, variable importance. *Default*: if possible, it should be extracted from the `model` object.
- `y` - a numeric vector with true labels paired with observations in `data`. *Required by*: variable importance. *Default*: no default.

- `predict_function` - a function that takes two arguments: model and data, and returns numeric vector with predictions. Predictions should be calculated in the same scale as the `y` labels. *Required by*: all explainers. *Default*: the generic `predict()` function.
- `link_function` - a transformation/link function that is applied to model predictions. *Required by*: variable effect. *Default*: the identity `I()` function.
- `label` - a character, a name of the model that will be used in plots. *Required by*: plots. *Default*: extracted from the `class` attribute of the `model`.

Figure 2.2. The `explain()` function embeds `model`, validation `data` and `y` labels in a container. Model is accessed via universal interface specified by `predict_function()` and `link_function()`. The `label` field contains a unique name of the model.

The next section introduces use cases of regression. It will help to understand how to use the `explain()` function and for what purposes. Same functions may be used for binary classification.

2.2 Use case: Regression. Apartment prices in Warsaw

To illustrate applications of DALEX to regression problems we will use an artificial dataset `apartments` available in the `DALEX` package. Our goal is to predict the price per square meter of an apartment based on selected features such as construction year, surface, floor, number of rooms, district. It should be noted that four of these variables are continuous while the fifth one is a categorical one. Prices are given in Euro.

```
library("DALEX")
head(apartments)
```

\begin{table}

\caption{(#tab:hr_data)Artificial dataset about apartment prices in Warsaw. The goal here is to predict the price per square meter for a new apartment.}

m2.price	construction.year	surface	floor	no.rooms	district
5897	1953	25	3	1	Srodmiescie
1818	1992	143	9	5	Bielany
3643	1937	56	1	2	Praga
3517	1995	93	7	3	Ochota
3013	1992	144	6	5	Mokotow
5795	1926	61	6	2	Srodmiescie

\end{table}

2.2.1 Model 1: Linear regression

The first model is based on linear regression. It will be a simple model without any feature engineering.

```
apartments_lm_model <- lm(m2.price ~ construction.year + surface + floor +
                          no.rooms + district, data = apartments)
summary(apartments_lm_model)
```

```
##
## Call:
## lm(formula = m2.price ~ construction.year + surface + floor +
##     no.rooms + district, data = apartments)
##
## Residuals:
```

```
##    Min     1Q Median     3Q    Max
## -247.5 -202.8 -172.8  381.4  469.0
##
## Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)         5020.1391   682.8721   7.352 4.11e-13 ***
## construction.year     -0.2290     0.3483  -0.657   0.5110
## surface              -10.2378     0.5778 -17.720  < 2e-16 ***
## floor                -99.4820     3.0874 -32.222  < 2e-16 ***
## no.rooms             -37.7299    15.8440  -2.381   0.0174 *
## districtBielany       17.2144    40.4502   0.426   0.6705
## districtMokotow      918.3802    39.4386  23.286  < 2e-16 ***
## districtOchota       926.2540    40.5279  22.855  < 2e-16 ***
## districtPraga        -37.1047    40.8930  -0.907   0.3644
## districtSrodmiescie 2080.6110    40.0149  51.996  < 2e-16 ***
## districtUrsus         29.9419    39.7249   0.754   0.4512
## districtUrsynow      -18.8651    39.7565  -0.475   0.6352
## districtWola         -16.8912    39.6283  -0.426   0.6700
## districtZoliborz     889.9735    40.4099  22.024  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 281.3 on 986 degrees of freedom
## Multiple R-squared:  0.905,  Adjusted R-squared:  0.9037
## F-statistic: 722.5 on 13 and 986 DF,  p-value: < 2.2e-16
```

We have also another `apartmentsTest` dataset that can be used for validation of the model. Below is presented the mean square error calculated on the basis of validation data.

```
predicted_mi2_lm <- predict(apartments_lm_model, apartmentsTest)
sqrt(mean((predicted_mi2_lm - apartmentsTest$m2.price)^2))
```

```
## [1] 283.0865
```

To create an explainer for the regression model it is enough to use `explain()` function with the `model`, `data` and `y` parameters. In the next chapter we will show how to use this explainer.

```
explainer_lm <- explain(apartments_lm_model,
                        data = apartmentsTest[,2:6], y = apartmentsTest$m2.price)
```

2.2.2 Model 2: Random forest

The second model is based on the random forest. It's a very elastic out-of-the-box model.

```
library("randomForest")
set.seed(59)

apartments_rf_model <- randomForest(m2.price ~ construction.year + surface + floor +
                      no.rooms + district, data = apartments)
apartments_rf_model
```

```
##
## Call:
##  randomForest(formula = m2.price ~ construction.year + surface +    floor + no.rooms + district, data = ap
##                Type of random forest: regression
##                      Number of trees: 500
```

```
## No. of variables tried at each split: 1
##
##           Mean of squared residuals: 82614.7
##                     % Var explained: 89.94
```

Below you may see the mean square error calculated for `apartmentsTest` dataset.

```
predicted_mi2_rf <- predict(apartments_rf_model, apartmentsTest)
sqrt(mean((predicted_mi2_rf - apartmentsTest$m2.price)^2))
```

```
## [1] 286.5357
```

We will create an explainer also for the random forest model. In the next chapter we will show how to use this explainer.

```
explainer_rf <- explain(apartments_rf_model,
                        data = apartmentsTest[,2:6], y = apartmentsTest$m2.price)
```

These two models have identical performance! Which one should be used?

Chapter 3

Model understanding

In this chapter we introduce three groups of explainers that can be used to boost our understanding of black-box models.

- Section 3.1 presents explainers for model performance. A single number may be misleading when we need to compare complex models. In this section you will also find plots that give more information about model performance in a consistent form.
- Section 3.2 presents explainers for variable importance. Knowing which variables are important allows us to validate the model and increase our understanding of the domain.
- Section 3.3 presents explainers for variable effect. You may find in it plots that summarize the relation between model response and particular variables.

All explainers are illustrated on the basis of two models fitted to the `apartments` data.

```
library("DALEX")
apartments_lm_model <- lm(m2.price ~ construction.year + surface + floor +
                      no.rooms + district, data = apartments)
library("randomForest")
set.seed(59)
apartments_rf_model <- randomForest(m2.price ~ construction.year + surface + floor +
                      no.rooms + district, data = apartments)
```

First we need to prepare wrappers for these models. They are in `explainer_lm` and `explainer_rf` objects.

```
explainer_lm <- explain(apartments_lm_model,
                        data = apartmentsTest[,2:6], y = apartmentsTest$m2.price)
explainer_rf <- explain(apartments_rf_model,
                        data = apartmentsTest[,2:6], y = apartmentsTest$m2.price)
```

3.1 Model performance

As you may remember from the previous chapter, the root mean square of residuals is identical for both considered models. Does it mean that these models are equally good?

```
predicted_mi2_lm <- predict(apartments_lm_model, apartmentsTest)
sqrt(mean((predicted_mi2_lm - apartmentsTest$m2.price)^2))
```

```
## [1] 283.0865
```

```
predicted_mi2_rf <- predict(apartments_rf_model, apartmentsTest)
sqrt(mean((predicted_mi2_rf - apartmentsTest$m2.price)^2))
```

```
## [1] 286.5357
```

Function `model_performance()` calculates predictions and residuals for validation dataset `apartmentsTest`.

Generic function `print()` returns quantiles for residuals.

```
mp_lm <- model_performance(explainer_lm)
mp_rf <- model_performance(explainer_rf)
mp_lm
```

```
##        0%       10%       20%       30%       40%       50%       60%
## -472.3560 -423.9131 -398.2811 -370.8841  161.2473  174.0677  184.1412
##       70%       80%       90%      100%
##  195.8834  209.2460  221.4659  257.2555
```

```
mp_rf
```

```
##           0%          10%          20%          30%          40%
## -1262.554308  -408.920183  -197.591180   -89.661883    -7.454146
##          50%          60%          70%          80%          90%
##    55.441061   108.398858   157.924244   218.241574   294.264602
##         100%
##   727.445065
```

The generic `plot()` function shows reversed empirical cumulative distribution function for absolute values from residuals. This function presents a fraction of residuals larger than `x`. The figure below shows that majority of residuals for the random forest is smaller than residuals for the linear model, yet the small fraction of very large residuals affects the root mean square.

```
plot(mp_lm, mp_rf)
```

Use the `geom = "boxplot"` parameter for the generic `plot()` function to get an alternative comparison of residuals. The red dot stands for the root mean square.

```
plot(mp_lm, mp_rf, geom = "boxplot")
```

3.2 Feature importance

Explainers presented in this section are designed to better understand which variables are important.

Some models, such as linear regression or random forest, have a build-in *model specific* methods to calculate and visualize variable importance. They will be presented in Section 3.2.2.

Section 3.2.1 presents a model agnostic approach on the basis of permutations. The advantage of this approach is that different models can be compared within a single setup.

3.2.1 Model agnostic

Model agnostic variable importance is calculated by means of permutations. We simply substract the loss function calculated for validation dataset with permuted values for a single variable from the loss function calculated for validation dataset. This concept and some extensions are described in (Fisher et al., 2018).

This method is implemented in the `variable_importance()` function. The loss function is calculated for:

Figure 3.1: (#fig:global_explain_ecdf)Comparison of residuals for linear model and random forest

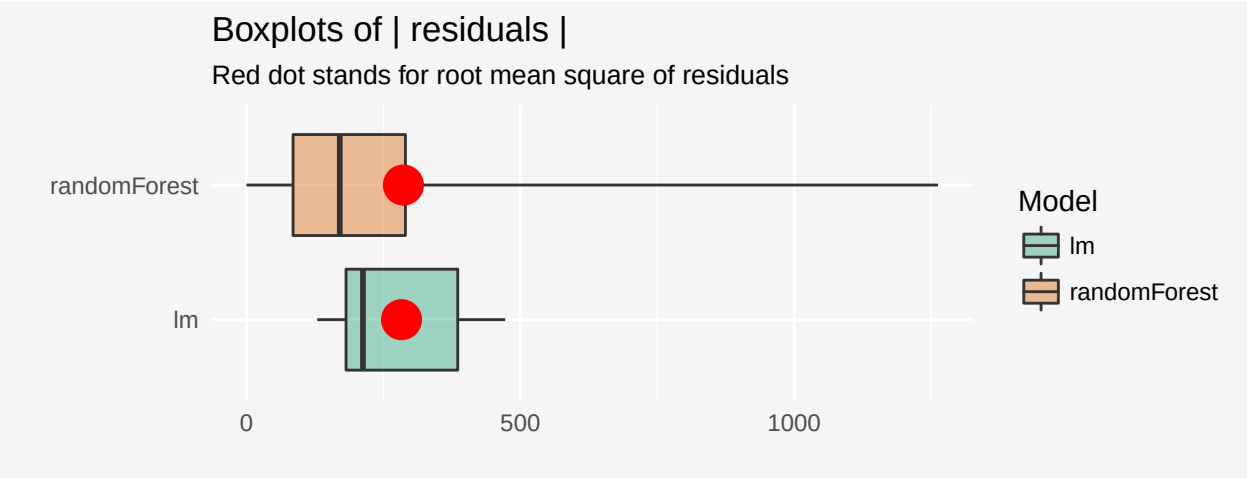


Figure 3.2: (#fig:global_explain_boxplot)Comparison of residuals for linear model and random forest

- the original validation `data`. It is an estimate of a model performance and will be denoted as `_full_model_`,
- validation data with resampled `y` labels. It is a kind of *worst case* loss when model are compared against random labels. It will be denoted as `_baseline_`,
- validation data with single variable being resampled. It tells us how much is gone from the model performance after the selected variable is blinded.

Let's see how this function works for a random forest model.

```
vi_rf <- variable_importance(explainer_rf, loss_function = loss_root_mean_square)
vi_rf
```

```
##            variable dropout_loss        label
## 1      _full_model_     285.1355 randomForest
## 2          no.rooms     391.0710 randomForest
## 3 construction.year     410.5866 randomForest
## 4             floor     445.2164 randomForest
## 5           surface     480.1431 randomForest
## 6          district     843.6519 randomForest
## 7        _baseline_    1081.3710 randomForest
```

Here the `loss_root_mean_square()` function is defined as square root from averaged squared differences between labels and model predictions. The same method may be applied to a linear model. Since we are using the same loss function and the same method for variable permutations, the losses calculated with both methods can be directly compared.

```
vi_lm <- variable_importance(explainer_lm, loss_function = loss_root_mean_square)
vi_lm
```

```
##            variable dropout_loss label
## 1      _full_model_     284.2788    lm
## 2 construction.year     284.2638    lm
## 3          no.rooms     295.5020    lm
## 4             floor     495.7685    lm
## 5           surface     600.4308    lm
## 6          district    1025.7208    lm
## 7        _baseline_    1232.6798    lm
```

It is much easier to compare both models when these values are plotted close to each other. The generic `plot()` function may handle both models.

```
plot(vi_lm, vi_rf)
```

What we can read out of this plot?

- left edges of intervals start in `_full_model_` for a given model. As we can see. the performances are similar for both models,
- length of the interval corresponds to variable importance. In both models the most important variables are `district` and `surface`,
- in the random forest model the `construction_year` variable has some importance, while its importance for linear model is almost equal to zero,
- the variable `no.rooms` (which is correlated with `surface`) has some importance in the random forest model but not in the linear model.

We may be interested in variables that behave differently between models (like `construction_year`) or variables that are important in both models (like `district` or `surface`). In the next section we introduce explainers for further investigation of these variables.

NOTE: If you want variable importance hooked at 0, just add `type = "difference"` parameter to

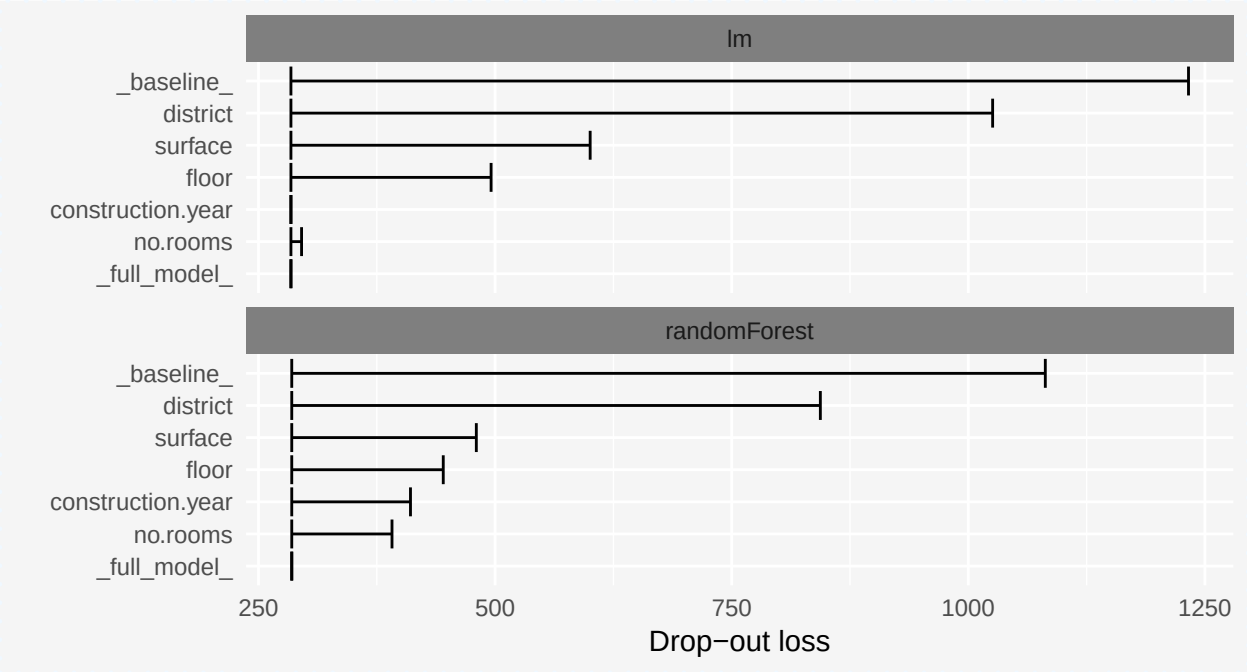


Figure 3.3: Model agnostic variable importance plot. Right edges correspond to loss function after permutation of a single variable. Left edges correspond to loss of a full model

`variable_importance()`.

```
vi_lm <- variable_importance(explainer_lm, loss_function = loss_root_mean_square, type = "difference")
vi_rf <- variable_importance(explainer_rf, loss_function = loss_root_mean_square, type = "difference")
plot(vi_lm, vi_rf)
```

3.2.2 Model specific

Some models have build-in tools for calculation of variable importance. Random forest uses two different measures - one based on out-of-bag data and second one based on gains in nodes. Read more about this approach in (Liaw and Wiener, 2002).

Below we show an example of a dot plot that summarizes default importance measure for a random forest. The `varImpPlot()` function is available in the `randomForest` package.

```
varImpPlot(apartments_rf_model)
```

It is easy to assess variable importance for linear models and generalized models, since model coefficients have direct interpretation.

Forest plots were initially used in the meta analysis to visualize effects in different studies. . At present, however, they are frequently used to present summary characteristics for models with linear structure / created with `lm` or `glm` functions.

There are various implementations of forest plots in R. In the package `forestmodel` (see (Kennedy, 2017)) one can use `forest_model()` function to draw a forest plot. This package is based on the `broom` package (see (Robinson, 2017)) and this is why it handles a large variety of different regression models.

```
library("forestmodel")
forest_model(apartments_lm_model)
```

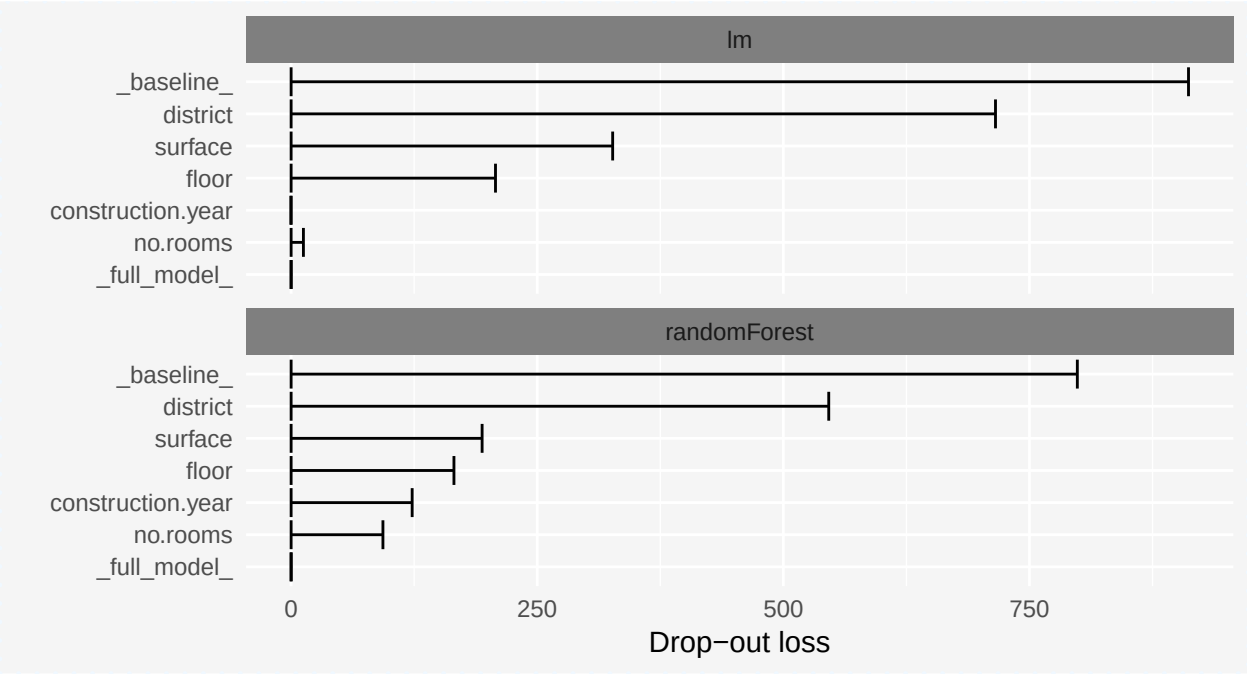


Figure 3.4: Model agnostic variable importance plot. Right edges correspond to difference between loss after permutation of a single variable and loss of a full model

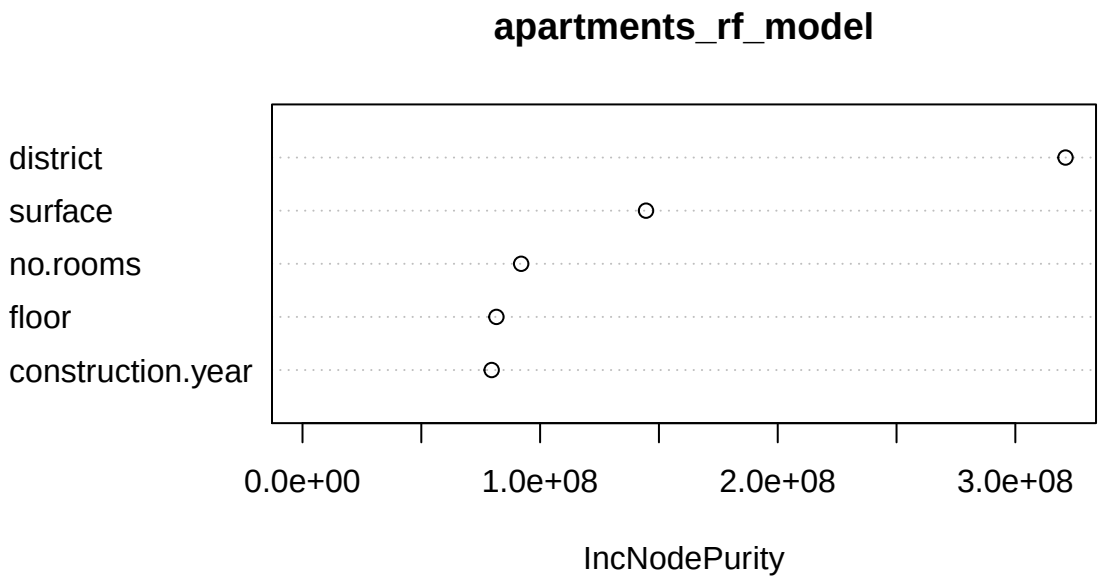


Figure 3.5: Built-in variable importance plot for random forest

Variable		N	Estimate		p
construction.year		1000		−0.23 (−0.91, 0.45)	0.51
surface		1000		−10.24 (−11.37, −9.10)	<0.001
floor		1000		−99.48 (−105.54, −93.42)	<0.001
no.rooms		1000		−37.73 (−68.82, −6.64)	0.02
district	Bemowo	98		Reference	
	Bielany	96		17.21 (−62.16, 96.59)	0.67
	Mokotow	107		918.38 (840.99, 995.77)	<0.001
	Ochota	96		926.25 (846.72, 1005.78)	<0.001
	Praga	92		−37.10 (−117.35, 43.14)	0.36
	Srodmiescie	100		2080.61 (2002.09, 2159.14)	<0.001
	Ursus	105		29.94 (−48.01, 107.90)	0.45
	Ursynow	103		−18.87 (−96.88, 59.15)	0.64
	Wola	106		−16.89 (−94.66, 60.87)	0.67
	Zoliborz	97		889.97 (810.67, 969.27)	<0.001
(Intercept)				5020.14 (3680.09, 6360.19)	<0.001

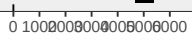


Figure 3.6: Forest plot created with `forestmodel` package

In the package `sjPlot` (see (Lüdecke, 2017)) one can find `sjp.xyz()` function to visualize coefficients of a `xyz` model (like `sjp.glm()` for `glm` models) or a generic wrapper `plot_model()`.

```
library("sjPlot")
plot_model(apartments_lm_model, type = "est", sort.est = TRUE)
```

Note!

The `forestmodel` package handles factor variables in a better way while the plots from `sjPlot` are easier to read.

3.3 Variable response

Explainers presented in this section are designed to better understand the relation between a variable and a model output.

Subsection 3.3.1 presents Partial Dependence Plots (PDP), one of the most popular methods for exploration of a relation between a continuous variable and a model outcome. Subsection 3.3.2 presents Accumulated Local Effects Plots (ALEP), an extension of PDP more suited for highly correlated variables.

Subsection 3.3.3 presents Merging Path Plots, a method for exploration of a relation between a categorical variable and a model outcome.

3.3.1 Partial Dependence Plot

Partial Dependence Plots (see `pdp` package (Greenwell, 2017)) for a black box $f(x;\theta)$ show the expected output condition on a selected variable.

$$p_i(x_i) = E_{x_{-i}}[f(x^i, x^{-i};\theta)].$$

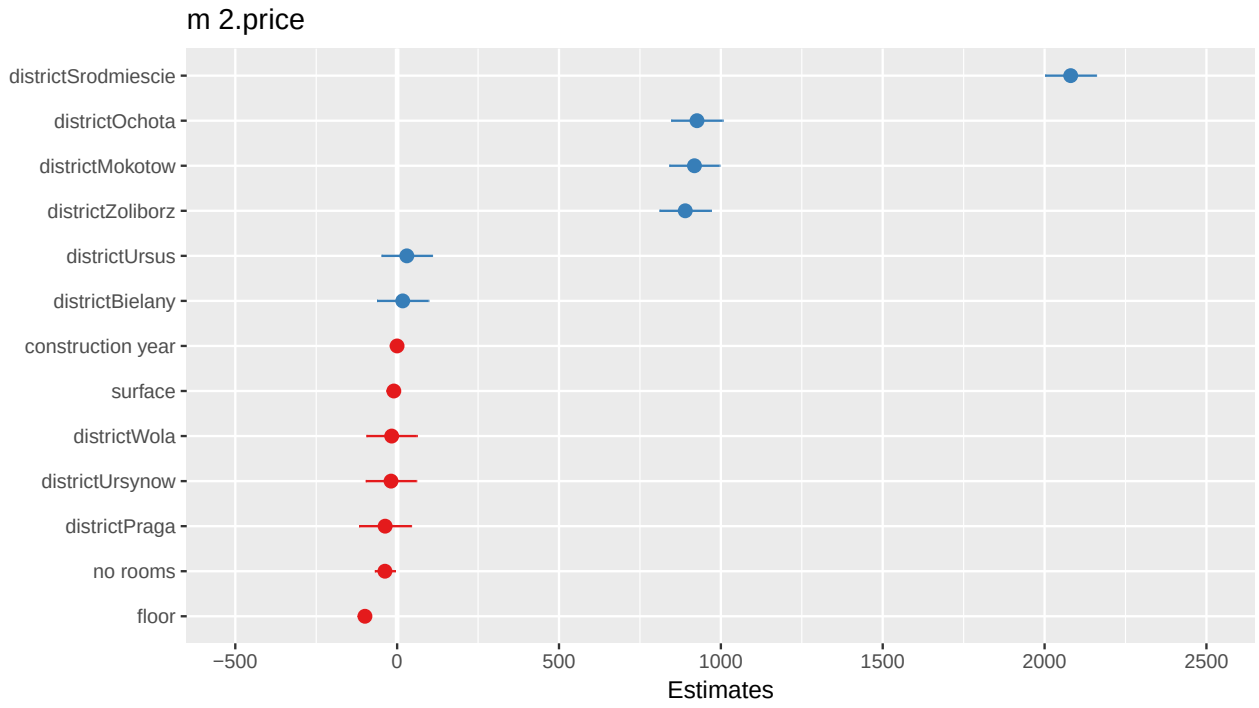


Figure 3.7: Model coefficients plotted with sjPlot package

Of course, this expectation cannot be calculated directly as we do not know fully neither the distribution of x_{-i} nor the $f()$. Yet this value may be estimated by

$$\hat{p}_i(x_i) = \frac{1}{n}\sum_{j=1}^{n} f(x_j^i, x_j^{-i}, \hat{\theta}).$$

Let's see an example for the model `apartments_rf_model`. Below we use `variable_response()` from `DALEX`, which calls `pdp::partial` function to calculate PDP response.

Section 3.2 shows variable importance plots for different models. The variable `construction.year` is interesting as it is important for the random forest model `apartments_rf_model` but not for the linear model `apartments_lm_model`. Let's see the relation between the variable and the model output.

```
sv_rf  <- single_variable(explainer_rf, variable =  "construction.year", type = "pdp")
plot(sv_rf)
```

We can use PDP plots to compare two or more models. Below we plot PDP for the linear model against the random forest model.

```
sv_lm  <- single_variable(explainer_lm, variable =  "construction.year", type = "pdp")

plot(sv_rf, sv_lm)
```

It looks like the random forest captures the non-linear relation that cannot be captured by linear models.

3.3.2 Accumulated Local Effects Plot

As demonstrated in section 3.3.1, the Partial Dependence Plot presents the expected model response with respect to marginal distribution of x_{-i}. In some cases, e.g. when repressors are highly correlated,

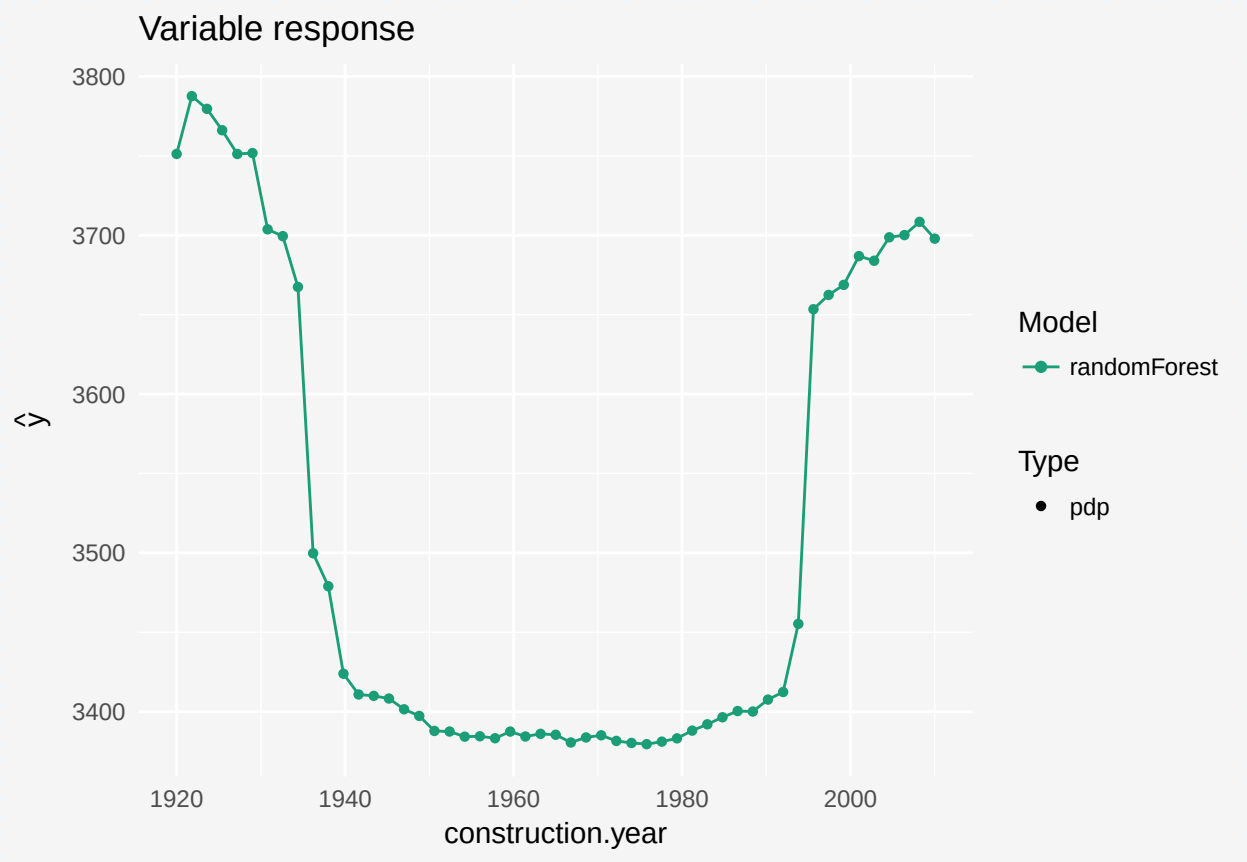


Figure 3.8: Relation between output from `apartments_rf_model` and variable `construction.year`

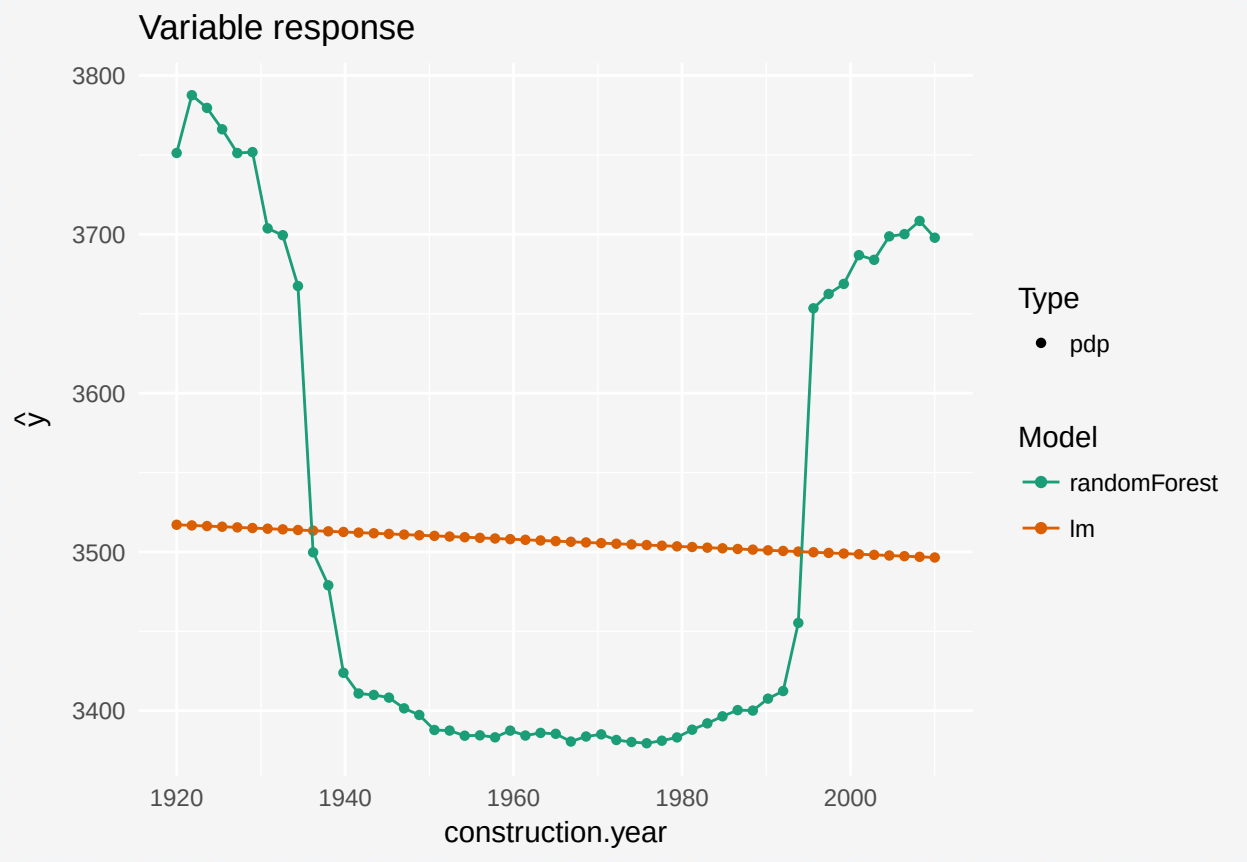


Figure 3.9: Relation between output from models `apartments_rf_model` and `apartments_lm_model` against the variable `construction.year`

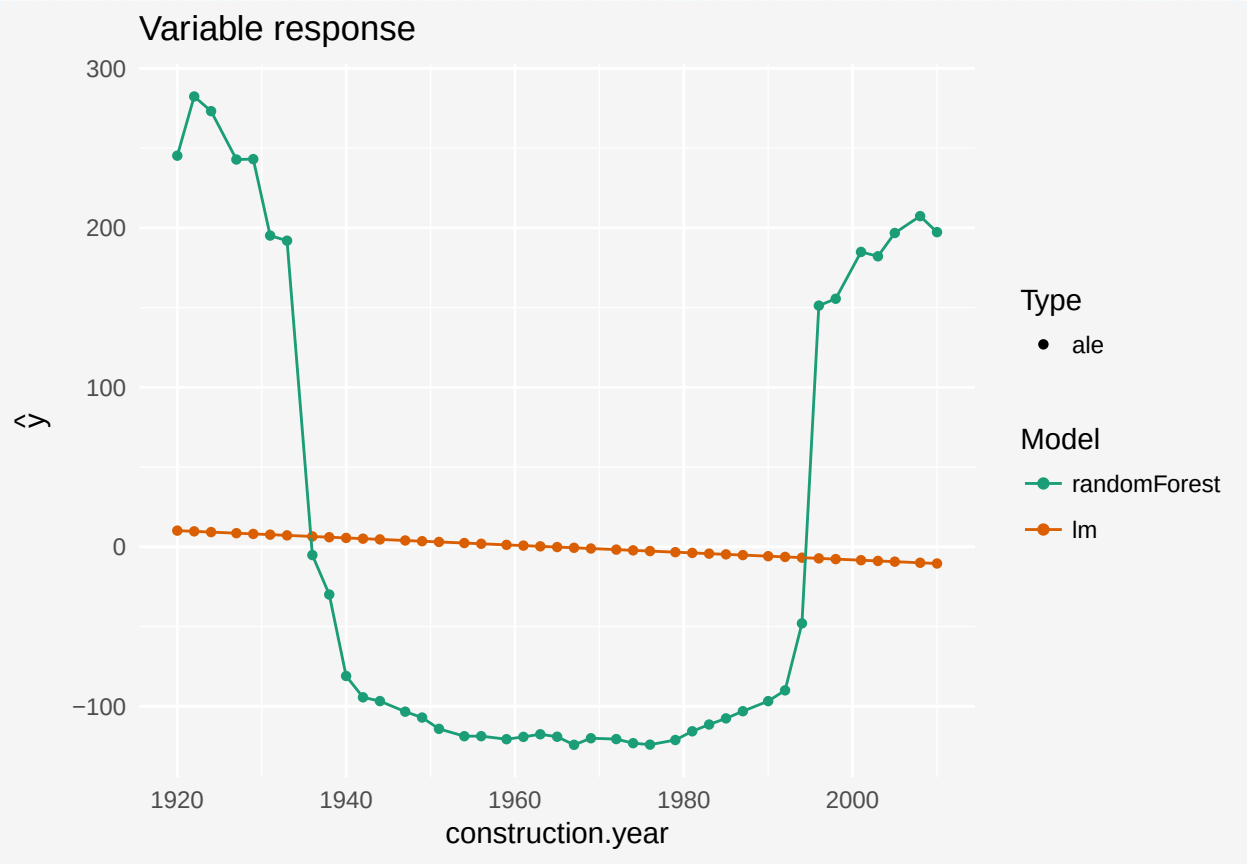


Figure 3.10: Relation between output from models `apartments_rf_model` and `apartments_lm_model` against the variable `construction.year` calculated with Accumulated local effects.

expectation towards the marginal distribution may lead to biases/poorly extrapolated model responses.

Accumulated local effects (ALE) plots (see `ALEPlot` package (Apley, 2017)) solve this problem by using conditional distribution $x_{-i}|x_i = x_i^*$. This solution leads to more stable and reliable estimates (at least when the predictors are highly correlated).

Estimation of the main effects for `construction.year` is similar to the PDP curves. We use here `DALEX::single_variable` function that calls `ALEPlot::ALEPlot` function to calculate the ALE curve for the variable `construction.year`.

```
sva_rf  <- single_variable(explainer_rf, variable = "construction.year", type = "ale")
sva_lm  <- single_variable(explainer_lm, variable = "construction.year", type = "ale")
```

```
plot(sva_rf, sva_lm)
```

Results for PDP and ALEP are very similar except that effects for ALEP are centered around 0.

3.3.3 Mering Path Plot

The package `ICEbox` does not work for factor variables, while the `pdp` package returns plots that are hard to interpret.

An interesting tool that helps to understand what happens with factor variables is the **factorMerger** package. See (Sitko and Biecek, 2017).

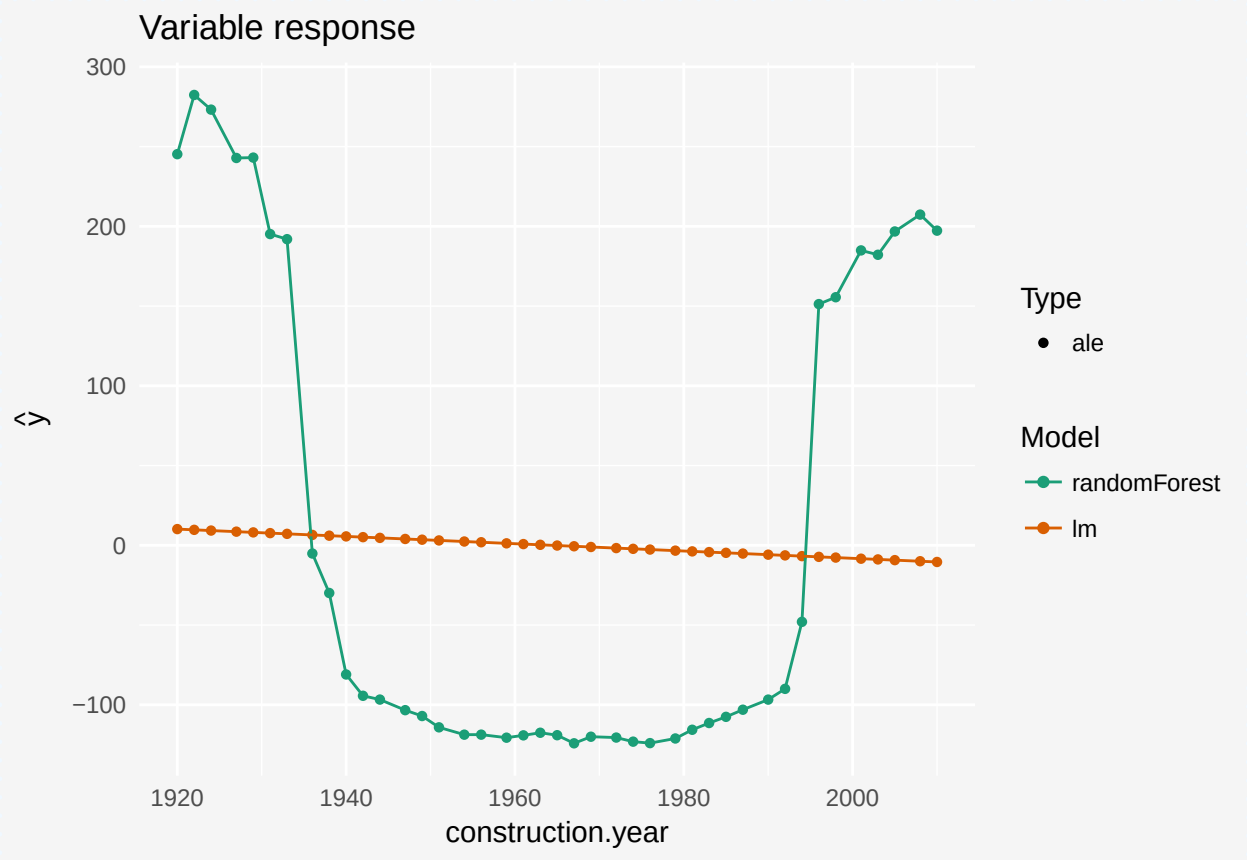


Figure 3.11: Relation between output from models `apartments_rf_model` and `apartments_lm_model` against the variable `construction.year` calculated with Accumulated local effects.

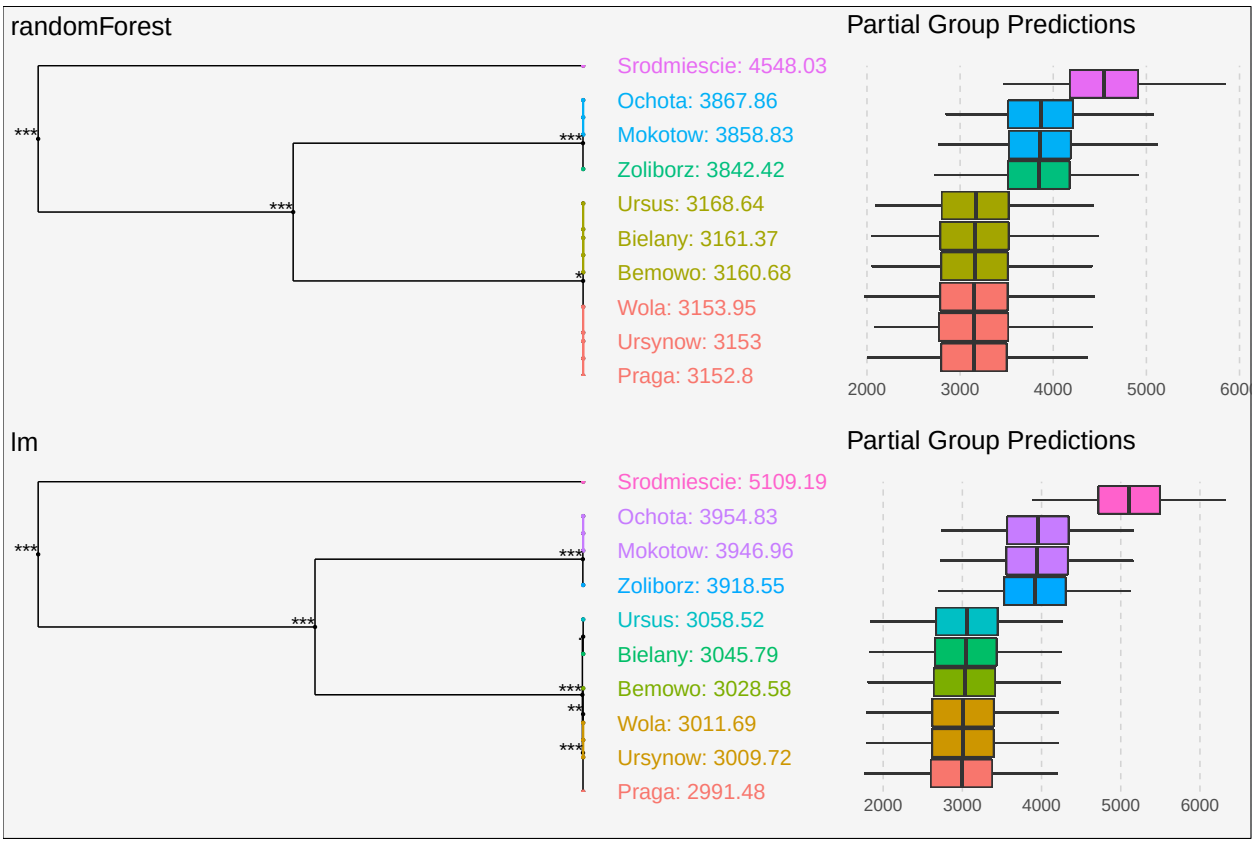


Figure 3.12: Merging Path Plot for `district` variable. Left panel shows the dendrogram for districts, here we have clearly three clusters. Right panel shows distribution of predictions for each district.

Below you may see a Merging Path Plot for a factor variable `district`.

```
svd_rf  <- single_variable(explainer_rf, variable = "district", type = "factor")
svd_lm  <- single_variable(explainer_lm, variable = "district", type = "factor")

plot(svd_rf, svd_lm)
```

The three clusters are: the city center (Srodmiescie), districts well communicated with city center (Ochota, Mokotow, Zoliborz) and other districts closer to city boundaries.

Factor variables are handled very differently by random forest and linear model, yet despite these differences both models result in very similar plots.

Chapter 4

Prediction understanding

In this chapter we introduce two groups of explainers that can be used to boost our understanding of model predictions.

- Section 4.1 presents explainers that helps to identify outliers.
- Section 4.2 presents explainers for model predictions. Each prediction can be split into parts attributed to particular variables. Having found out which variables are important and whether the prediction is accurate, one can validate the model.

Explainers presented here are illustrated based on two models fitted to the `apartments` data.

```
library("DALEX")
apartments_lm_model <- lm(m2.price ~ construction.year + surface + floor +
                       no.rooms + district, data = apartments)
library("randomForest")
set.seed(59)
apartments_rf_model <- randomForest(m2.price ~ construction.year + surface + floor +
                       no.rooms + district, data = apartments)
```

First we need to prepare wrappers for these models. They are in `explainer_lm` and `explainer_rf` objects.

```
explainer_lm <- explain(apartments_lm_model,
                        data = apartmentsTest[,2:6], y = apartmentsTest$m2.price)
explainer_rf <- explain(apartments_rf_model,
                        data = apartmentsTest[,2:6], y = apartmentsTest$m2.price)
```

4.1 Outlier detection

Function `model_performance()` may be used to identify outliers. This function was already introduced in section 3.1 but we will present here its other uses.

As you may remember, residuals for random forest were smaller in general, except for a small fraction of very high residuals.

Let's use the `model_performance()` function to extract and plot residuals against the observed true values.

```
mp_rf <- model_performance(explainer_rf)

library("ggplot2")
ggplot(mp_rf, aes(observed, diff)) + geom_point() +
```

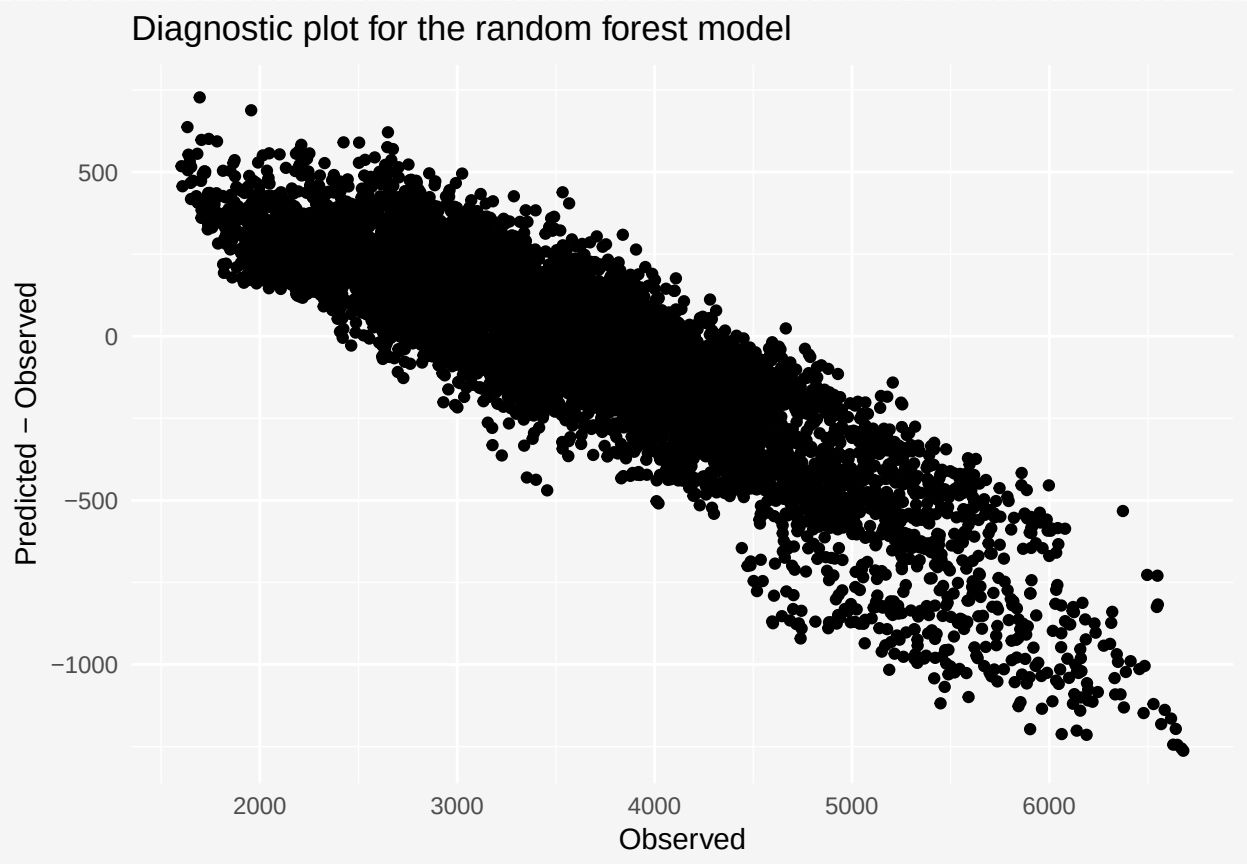


Figure 4.1: Diagnostic plot for the random forest model. Clearly the more expensive are apartments the more underestimated are model predictions

Table 4.1: Observation with the largest residual in the random forest model

	m2.price	construction.year	surface	floor	no.rooms	district
1161	6679	2005	22	1	2	Srodmiescie

```
        xlab("Observed") + ylab("Predicted - Observed") +
        ggtitle("Diagnostic plot for the random forest model") + theme_mi2()
```

Lets see which variables stand behind the model prediction for an apartment with largest residual.

```
which.min(mp_rf$diff)
## 1161
new_apartment <- apartmentsTest[which.min(mp_rf$diff), ]
new_apartment
```

4.2 Prediction breakDown

Does your ML algorithm learn from mistakes? Understanding what causes wrong model predictions will help to improve the model itself.

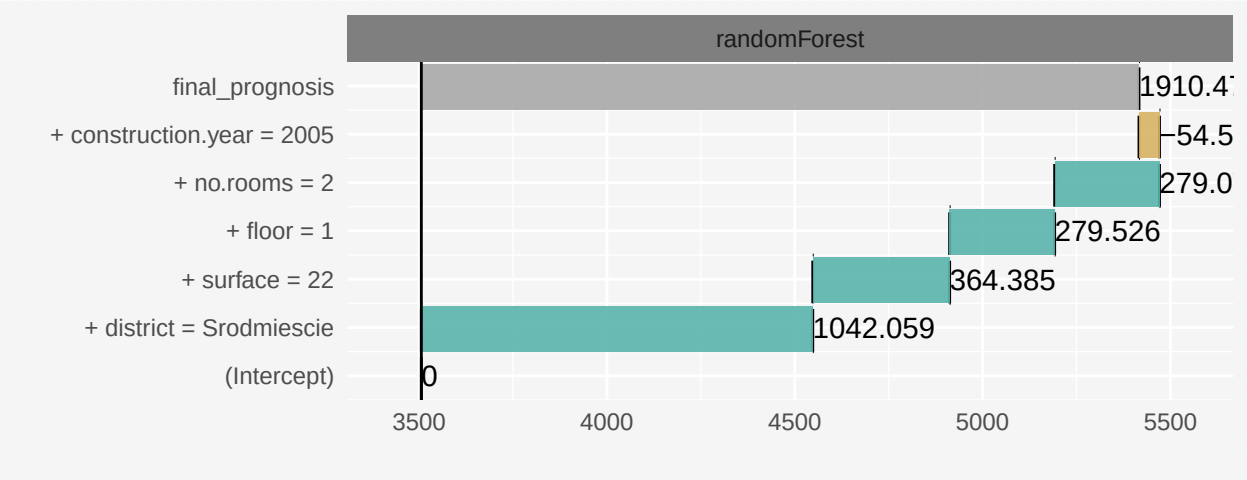


Figure 4.2: (#fig:single_prediction_break)Break Down Plot for prediction from the random forest model

Lots of arguments in favor of such explainers can be found in the (Ribeiro et al., 2016) article. This approach is implemented in the **live** package (see (Staniak and Biecek, 2017)) which may be seen as an extension of the LIME method.

In this section we present other method for explanations of model predictions, namely the one implemented in the `breakDown` package (Biecek, 2017). The function `single_prediction()` is a wrapper around this package.

Model prediction is visualized with Break Down Plots, which were inspired by waterfall plots as in `xgboostExplainer` package. Break Down Plots show the contribution of every variable present in the model.

Function `single_prediction()` generates variable attributions for selected prediction. The generic `plot()` function shows these attributions.

```
new_apartment_rf <- single_prediction(explainer_rf, observation = new_apartment)
breakDown:::print.broken(new_apartment_rf)
```

```
##                             contribution
## (Intercept)                        0.000
## + district = Srodmiescie        1042.059
## + surface = 22                   364.385
## + floor = 1                      279.526
## + no.rooms = 2                   279.070
## + construction.year = 2005       -54.566
## final_prognosis                 1910.474
## baseline:  3505.971
```

```
plot(new_apartment_rf)
```

Both the plot and the table confirm that all variables (`district`, `surface`, `floor`, `no.rooms`) have positive effects as expected. Still, these effects are too small while the final prediction - `3505 + 1881`- is much smaller than the real price of a square meter `6679`. Let's see how the linear model behaves for this observation.

```
new_apartment_lm <- single_prediction(explainer_lm, observation = new_apartment)
plot(new_apartment_lm, new_apartment_rf)
```

Prediction for linear model is much closer to the real price of square meter for this apartment.

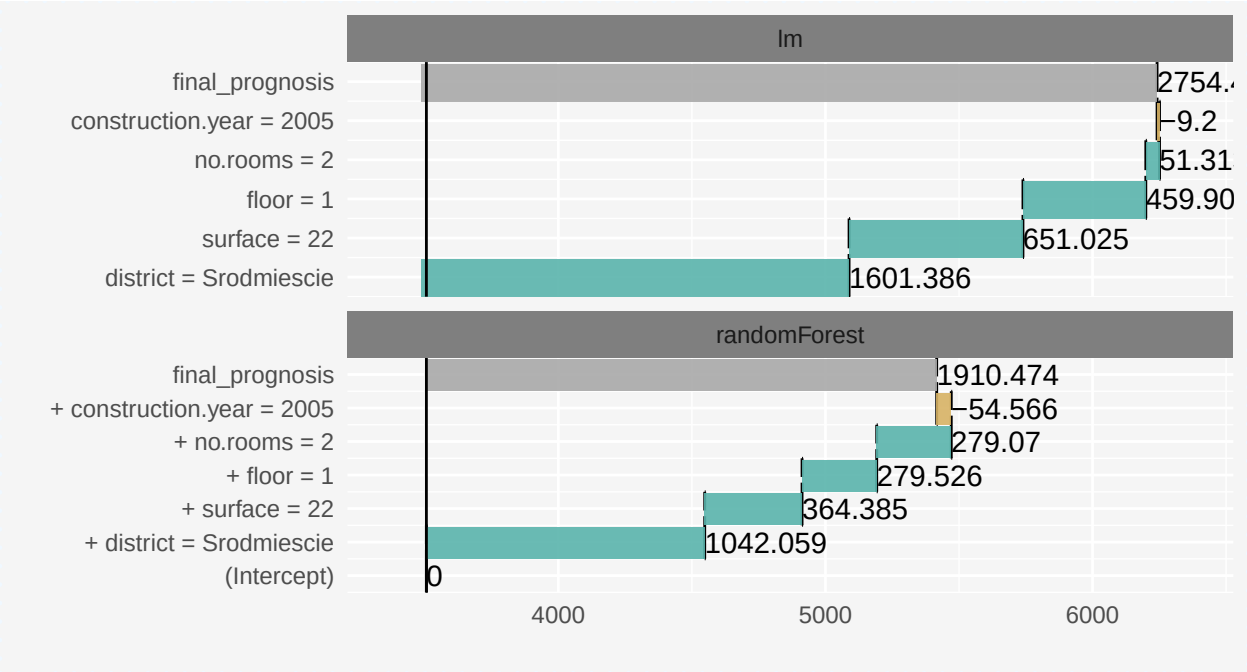


Figure 4.3: (#fig:single_prediction_break2)Break Down Plots that compare the linear model and the random forest model

Chapter 5

Epilogue

Let's summarize what has happened in the previous sections.

- Section 2.2 shows two models with equal performance for `apartments` dataset.
- Section 3.1 shows that in general the random forest model has smaller residuals than the linear model but there is a small fraction of very large residuals.
- Section 4.1 shows that the random forest model under-predicts expensive apartments. It is not a model that we would like to employ.
- Section 3.2 shows that `construction_year` is important for the random forest model.
- Section 3.3 shows that the relation between `construction_year` and the price of square meter is non linear.

In this section we showed how to improve the basic linear model by feature engineering of `construction_year`. Findings from the random forest models will help to create a new feature for the linear model.

```
library("DALEX")

apartments_lm_model_improved <- lm(m2.price ~ I(construction.year < 1935 | construction.year > 1995) + s
                          no.rooms + district, data = apartments)

explainer_lm_improved <- explain(apartments_lm_model_improved,
                           data = apartmentsTest[,2:6], y = apartmentsTest$m2.price)

mp_lm_improved <- model_performance(explainer_lm_improved)
plot(mp_lm_improved, geom = "boxplot")
```

In conclusion, the results presented above prove that the `apartments_lm_model_improved` model is much better than the two initial models introduced in Chapter 3.

In this use-case we showed that explainers implemented in DALEX help to better understand the model and that this knowledge may be used to create a better final model.

Find more examples, vignietts and cheatsheets at DALEX website https://github.com/pbiecek/DALEX.

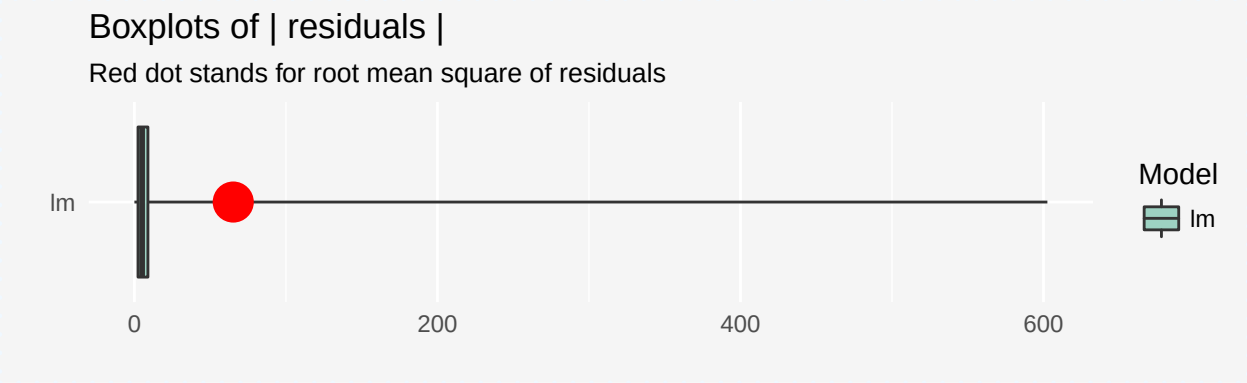


Figure 5.1: (#fig:final_model)Distribution of residuals for the new improved linear model

Chapter 6

Exercises

Examples in previous chapters were based on random forest model and linear model. But there are so many different approaches for regression modeling. Try `gbm` (Generalized Boosted Regression Models), `knn` (k-Nearest Neighbour), `svm` (Support Vector Machines), `nnet` (Neural Networks) or other models.

1. Prepare explainers for model performance,
2. Prepare explainers for variable importance,
3. Prepare explainers for variable response for `construction.year`,
4. Prepare explainers for model predictions.

Below you will find fits for few different models. Try them out.

```
library("DALEX")
library("gbm")

apartments_gbm_model <- gbm(m2.price ~ construction.year + surface + floor +
                         no.rooms + district, data = apartments, n.trees = 1000)
```

```
## Distribution not specified, assuming gaussian ...
```

```
explainer_gbm <- explain(apartments_gbm_model,
                          data = apartmentsTest[,2:6], y = apartmentsTest$m2.price,
                         predict_function = function(m, d) predict(m, d, n.trees = 1000))

library("nnet")
apartments_nnet_model <- nnet(m2.price ~ construction.year + surface + floor +
                         no.rooms + district, data = apartments,
                         linout=TRUE,
                         size = 50, maxit=100)
```

```
## # weights:  751
## initial  value 12982043148.246910
## final  value 821267660.638999
## converged
```

```
explainer_nnet <- explain(apartments_nnet_model,
                          data = apartmentsTest[,2:6], y = apartmentsTest$m2.price)

library("e1071")
apartments_svm_model <- svm(m2.price ~ construction.year + surface + floor +
                         no.rooms + district, data = apartments)
```

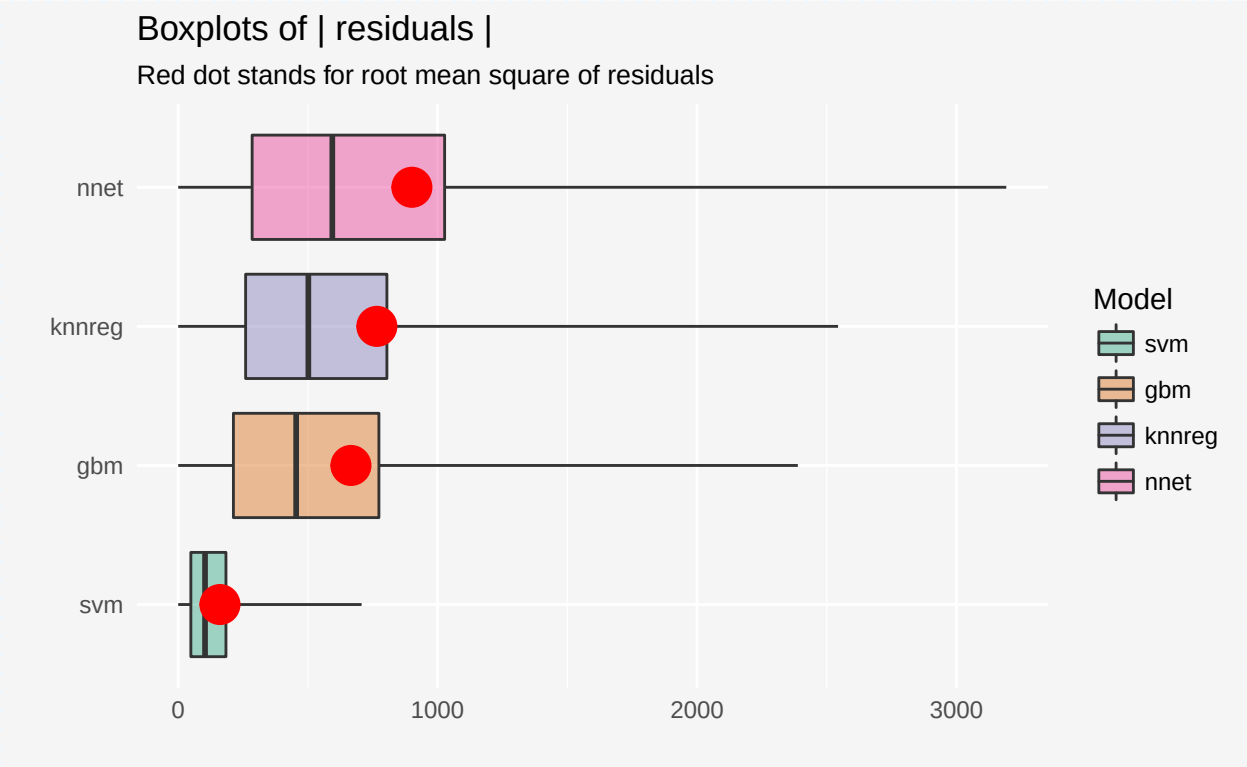


Figure 6.1: Distribution of residuals for fourn new models

```
explainer_svm <- explain(apartments_svm_model,
                         data = apartmentsTest[,2:6], y = apartmentsTest$m2.price)

library("caret")
mapartments <- model.matrix(m2.price ~ ., data = apartments)
mapartmentsTest <- model.matrix(m2.price ~ ., data = apartmentsTest)
apartments_knn_model <- knnreg(mapartments, apartments[,1], k = 5)

explainer_knn <- explain(apartments_knn_model,
                         data = mapartmentsTest, y = apartmentsTest$m2.price)

# Model performance

mp_knn <- model_performance(explainer_knn)
mp_svm <- model_performance(explainer_svm)
mp_gbm <- model_performance(explainer_gbm)
mp_nnet <- model_performance(explainer_nnet)
plot(mp_gbm, mp_nnet, mp_svm, mp_knn, geom = "boxplot")
```

Bibliography

Apley, D. (2017). *ALEPlot: Accumulated Local Effects (ALE) Plots and Partial Dependence (PD) Plots*. R package version 1.0.

Biecek, P. (2017). *breakDown: BreakDown Plots*. R package version 0.1.2.

Biecek, P. (2018). *DALEX: Descriptive mAchine Learning EXplanations*. R package version 0.2.

Bischl, B., Lang, M., Kotthoff, L., Schiffner, J., Richter, J., Studerus, E., Casalicchio, G., and Jones, Z. M. (2016). mlr: Machine learning in r. *Journal of Machine Learning Research*, 17(170):1–5.

Fisher, A., Rudin, C., and Dominici, F. (2018). Model class reliance: Variable importance measures for any machine learning model class, from the 'rashomon' perspective. *Journal of Computational and Graphical Statistics*.

Foster, D. (2017). *xgboostExplainer: An R package that makes xgboost models fully interpretable*. R package version 0.1.

from Jed Wing, M. K. C., Weston, S., Williams, A., Keefer, C., Engelhardt, A., Cooper, T., Mayer, Z., Kenkel, B., the R Core Team, Benesty, M., Lescarbeau, R., Ziem, A., Scrucca, L., Tang, Y., and Candan., C. (2016). *caret: Classification and Regression Training*. R package version 6.0-64.

Greenwell, B. M. (2017). pdp: An r package for constructing partial dependence plots. *The R Journal*, 9(1):421–436.

Kennedy, N. (2017). *forestmodel: Forest Plots from Regression Models*. R package version 0.4.3.

Liaw, A. and Wiener, M. (2002). Classification and regression by randomforest. *R News*, 2(3):18–22.

Lüdecke, D. (2017). *sjPlot: Data Visualization for Statistics in Social Science*. R package version 2.4.0.

Paluszynska, A. and Biecek, P. (2017). *randomForestExplainer: A set of tools to understand what is happening inside a Random Forest*. R package version 0.9.

Puri, N., Gupta, P., Agarwal, P., Verma, S., and Krishnamurthy, B. (2017). MAGIX: Model Agnostic Globally Interpretable Explanations. *ArXiv e-prints*.

R Core Team (2018). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Ribeiro, M. T., Singh, S., and Guestrin, C. (2016). *"Why Should I Trust You?": Explaining the Predictions of Any Classifier*, page 1135–1144. ACM Press.

Robinson, D. (2017). *broom: Convert Statistical Analysis Objects into Tidy Data Frames*. R package version 0.4.3.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., and et al. (2017). Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359.

Sitko, A. and Biecek, P. (2017). *factorMerger: Hierarchical Algorithm for Post-Hoc Testing*. R package version 0.3.4.

Staniak, M. and Biecek, P. (2017). *live: Local Interpretable (Model-agnostic) Visual Explanations*. R package version 1.4.0.

Tzeng, F. Y. and Ma, K. L. (2005). Opening the black box - data driven visualization of neural networks. In *VIS 05. IEEE Visualization, 2005.*, pages 383–390.

Štrumbelj, E. and Kononenko, I. (2011). A general method for visualizing and explaining black-box regression models. In *Proceedings of the 10th International Conference on Adaptive and Natural Computing Algorithms - Volume Part II*, ICANNGA'11, pages 21–30, Berlin, Heidelberg. Springer-Verlag.

Wickham, H. (2009). *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York.

Zeiler, M. D. and Fergus, R. (2014). *Visualizing and Understanding Convolutional Networks*, page 818–833. Springer International Publishing.