Predictive Models: Visualisation, Exploration and Explanation

Przemyslaw Biecek and Tomasz Burzykowski

2018-09-23

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.

This book is about

- We present techniques to examine particular predictions from ML models. In this book you will find theory and examples that explains model locally like break down, ceteris paribus, LIME or Shapley.
- We present techniques to examine fully trained ML models as a whole. In this book you will find theory and examples that explains model globally like Partial Dependency Plots, Variable Importance Plots and others.
- We present tools and methods for model comparison.

This book is NOT about.

- We do not focus on any specific model. Presented techniques are model agnostic and do not have any assumptions related to model structure.
- We do not focus on the data exploration. There are very good books and techniques related to this, like R for Data Science http://r4ds.had.co.nz/ or TODO
- We do not focus on the process of model building. There are also very good books about this, see An Introduction to Statistical Learning by Gareth James, Daniela Witten, Trevor Hastie and Robert Tibshirani http://www-bcf.usc.edu/~gareth/ISL/ or TODO
- We do not focus on particular tools for model building, see Applied Predictive Modeling By Max Kuhn and Kjell Johnson http://appliedpredictivemodeling.com/

1.1 Model Lifecycle

Variable importance

Model response as a function of a variable

Model performance / diagnostic / validation

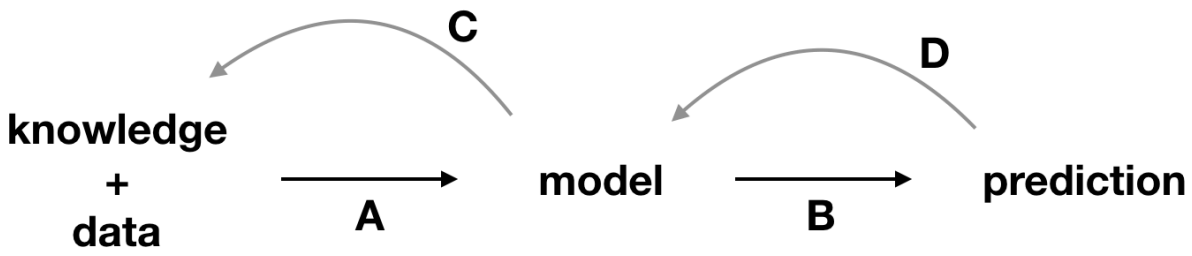


Figure 1.1: Workflow of a typical machine learning modeling. A) Modeling is a process in which domain knowledge and data are turned into models. B) Models are used to generate predictions. C) Understanding of a model structure may increase our knowledge, and in consequence it may lead to a better model. 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.

1.2 Why do we need model explainers?

Developer perspective:

AutoML

Feature Extraction

Model Improvement

Complex mthods are not being adopted quickly because people do not understand them. Model explainers can change this and open new applications

User perspective:

Justification of model decisions - civic rights

Model debudding / auditing

(**?**)

Cathy O'Neil: The era of blind faith in big data must end

"You don't see a lot of skepticism," she says. "The algorithms are like shiny new toys that we can't resist using. We trust them so much that we project meaning on to them." Ultimately algorithms, according to O'Neil, reinforce discrimination and widen inequality, "using people's fear and trust of mathematics to prevent them from asking questions".

But as Cathy O'Neil reveals in this urgent and necessary book, the opposite is true. The models being used today are opaque, unregulated, and uncontestable, even when they're wrong. Most troubling, they reinforce discrimination: If a poor student can't get a loan because a lending model deems him too risky (by virtue of his zip code), he's then cut off from the kind of education that could pull him out of poverty, and a vicious spiral ensues. Models are propping up the lucky and punishing the downtrodden, creating a "toxic cocktail for democracy." Welcome to the dark side of Big Data.

Bias w modelach:

(**?**)

Models may be biased. Examples for (**?**)

We propose a method to audit black-box risk models for potential bias by using model distillation. We demonstrate the methods on four public datasets: COMPAS, Lending Club, Stop-and-Frisk, and Chicago Police

1.3 How model exploration is different from data exploration?

Similarities:

- Models and data are related to some truth that we want to understand
- We use visuals to quicker understand complex relations

Differences:

- Data comes from some population; we treat them as a random sample. And since there is a sample there is also some randomness
- Models are just functions. Most models are not stochastic and we consider only deterministic models.
- We believe that data is noisy, but there is truth there. Data cannot be bad of misfitted, may be biased. Models can be bad, misfitted, unaccurate

1.4 Black-box models vs White-box models

(**?**)

1.5 Model agnostic vs Model specific

Tools designed to deal with specific models

- (**?**)
- (**?**)
- Specific for neural networks

Model agnostic

- DALEX, lime and all other presented in this book

```
library("xgboostExplainer")
```

1.6 Glossary / Notation

model healthcheck

Let $f_M(x) : \mathcal{R}^d \to \mathcal{R}$ denote a predictive model, i.e. function that takes d dimensional vector and calculate numerical score. In section in which we work with larger number of models we use subscript M to index models. But to simplify notation, this subscript is omitted if profiles for only one model are considered.

Symbol $x \in \mathcal{R}^d$ refers to a point in the feature space. We use subscript x_i to refer to a different data points and superscript x^j to refer to specific dimensions. Additionally, let x^{-j} denote all coordinates except j-th and let $x|^j = z$ denote a data point x^* with all coordinates equal to x except coordinate j equal to value z. I.e. $\forall_{i \neq j} x^i = x^{*,i}$ and $x^j = z$. In other words $x|^j = z$ denote a x with jth coordinate changed to z.

Now we can define Ceteris Paribus Profile for model f, variable j and point x as

$$CP^{f,j,x}(z) := f(x|^j = z).$$

I.e. CP profile is a model response obtained for observations created based on x with j coordinated changes and all other coordinates kept unchanged.

It is convenient to use an alternative name for this plot: What-If Plots. CP profiles show what would happen if only a single variable is changed.

Figure 5.1 shows an example of Ceteris Paribus profile. The black dot stands for prediction for a single observation. Grey line show how the model response would change if in this single observation coordinate `surface` will be changes to selected value. From this profile one may read that the model response is non monotonic. If `construction.year` for this observation would be below 1935 the model response would be higher, but if construction year were between 1935 and 1995 the model response would be lower.

Glossary:

- Black-box model
- White-box model
- Feature
- Variable
- Continuous variable
- Nominal variable
- Model-agnostic / Model specific

1.7 Thanks to

We are using the **bookdown** package (**?**) in this sample book

Chris Drake and Janusz Holyst

Prediction level explanations

Chapter 2

Introduction

Prediction level explainers help to understand how the model works for a single prediction. This is the main difference from the model level explainers that were focused on the model in general. Prediction level explainers are always in context of a single observation.

Think about following use-cases

- One wants to attribute effects of variables to a model predictions. Think about model for hart attack. Having a final score for a patient one wants to understand how much of this score come from smoking or age or gender.
- One wants to understand how the model response would change if some inputs are changed. Think about model for hart attack. How the model response would change if a patient cuts the number of smoked cigarettes by half.
- Model is not working correctly for a particular point and one wants to understand why predictions for this point are wrong.

2.1 Variable attribution vs What-if analysis

TODO: varaible attribution is not sparse, no reason for them to be like that

TODO: Three approaches: variable attribution, local model,

TODO: condvis (**?**)

Enslaving the Algorithm: From a 'Right to an Explanation' to a 'Right to Better Decisions'? (**?**)

TODO: Sparse model approximation / variable selection / feature ranking

There are many different tools that may be used to explore model around a single data point and in following sections we will describe the most popular approaches. They can be divided into two classes.

- Analysis of the model curvature. Here we treat the model as a function and we are interested in the curvature of this function around the point of interest (see Figure **??**). In the Section **??** we present the LIME method that approximates the black-box model in a point of interest while in section **??** we present Ceteris Paribus profiles that are more focused on conditional changes of model response given only one coordinate is modified.
- Analysis of the probabilistic behavior of the model. Here we are interested in decomposition of the model response to parts that can be attributed to particular features.

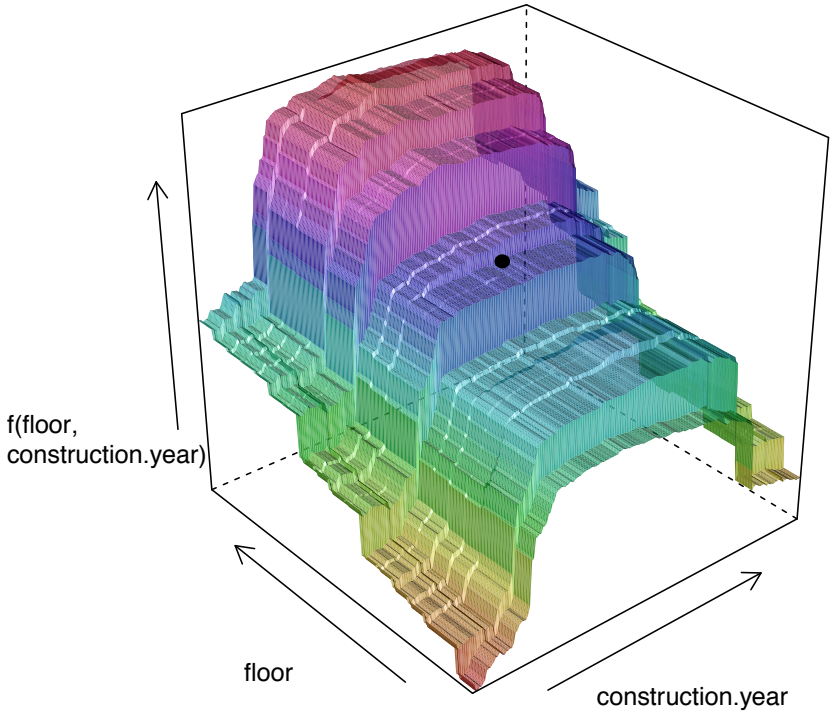


Figure 2.1: (fig:modelResponseCurve) Model response surface. We are interested in understanding the model behavior in a single point

2.2 When to use?

There are several use-cases for such explainers. Think about following.

- Model improvement. If model works particular bad for a selected observation (the residual is very high) then investigation of model responses for miss fitted points may give some hints how to improve the model. For individual predictions it is easier to notice that selected variable should have different a effect.
- Additional domain specific validation. Understanding which factors are important for model predictions helps to be critical about model response. If model contributions are against domain knowledge then we may be more skeptical and willing to try another model. On the other hand, if the model response is aligned with domain knowledge we may trust more in these responses. Such trust is important in decisions that may lead to serious consequences like predictive models in medicine.
- Model selection. Having multiple candidate models one may select the final response based on model explanations. Even if one model is better in terms of global model performance it may happen that locally other model is better fitted. This moves us towards model consultations that identify different options and allow human to select one of them.

2.3 A bit of philosophy: Three Laws for Prediction Level Explanations

76 years ago Isaac Asimov devised Three Laws of Robotics: 1) a robot may not injure a human being, 2) a robot must obey the orders given it by human beings and 3) A robot must protect its own existence. These laws impact discussion around Ethics of AI. Today's robots, like cleaning robots, robotic pets or autonomous cars are far from being conscious enough to be under Asimov's ethics.

Today we are surrounded by complex predictive algorithms used for decision making. Machine learning models are used in health care, politics, education, judiciary and many other areas. Black box predictive models have far larger influence on our lives than physical robots. Yet, applications of such models are left unregulated despite many examples of their potential harmfulness. See *Weapons of Math Destruction* by Cathy O'Neil for an excellent overview of potential problems.

It's clear that we need to control algorithms that may affect us. Such control is in our civic rights. Here we propose three requirements that any predictive model should fulfill.

- **Prediction's justifications**. For every prediction of a model one should be able to understand which variables affect the prediction and how strongly. Variable attribution to final prediction.
- **Prediction's speculations**. For every prediction of a model one should be able to understand how the model prediction would change if input variables were changed. Hypothesizing about what-if scenarios.
- **Prediction's validations** For every prediction of a model one should be able to verify how strong are evidences that confirm this particular prediction.

There are two ways to comply with these requirements. One is to use only models that fulfill these conditions by design. White-box models like linear regression or decision trees. In many cases the price for transparency is lower performance. The other way is to use approximated explainers – techniques that find only approximated answers, but work for any black box model. Here we present such techniques.

Chapter 3

Introduction to variable attribution methods

In this section we introduce method for additive decomposition of predictions. The main goal for these tools is to help understand how model output may be attributed to input variables or sets of variables.

Presented explainers are linked with the first law introduced in Section **??**, i.e. law for prediction's justifications. Note that there are more tools for variable attribution, some of them will be presented in next sections.

Think of following use cases:

- Think about a model for heart attack. A patient wants to know which factors have highest impact on the final heart risk score.
- Think about a model for apartment prices. An investor wants to know how much of the final price may be attributed to the location of an apartment.
- Think about a model for credit scoring. A customer wants to know if factors like gender, age or number of kids influence model decisions.

In the section **??** we will introduce key concepts and intuitions beyond variable attribution based on linear models. This approach may be easily applied to additive models and generalized linear models. In sections **??** and **??** we will present model agnostic extentions of this concept.

3.1 Variable attribution for linear models

Linear model with coefficients $\beta = (\beta_0, \beta_1, .., \beta_p)$ has following form.

$$f(x) = \beta_0 + x_1\beta_1 + \ldots + x_p\beta_p.$$

In other words, model response is the sum of weighted elements of $x = (x_1, x_2, \ldots, x_p)$.

From a global perspective of a model, we are usually interested in questions like, how good is the model, which variables are significant or how accurate are model predictions.

But in this chapter we are focues in a local perspective, i.e. for a single observation x^* how to measure the contribution of a variable x_i on model prediction $f(x^*)$.

Let $v(f, x^*, i)$ stands for the contribution of variable x_i on prediction of model $f()$ in point x^*. For linear models it is easy to define such contribution as

$$v(f, x^*, i) = f(x^*) - E[f(x)|x_{-1} = x^*_{-1}] = \beta_i x^*_i - E\beta_i X_i$$

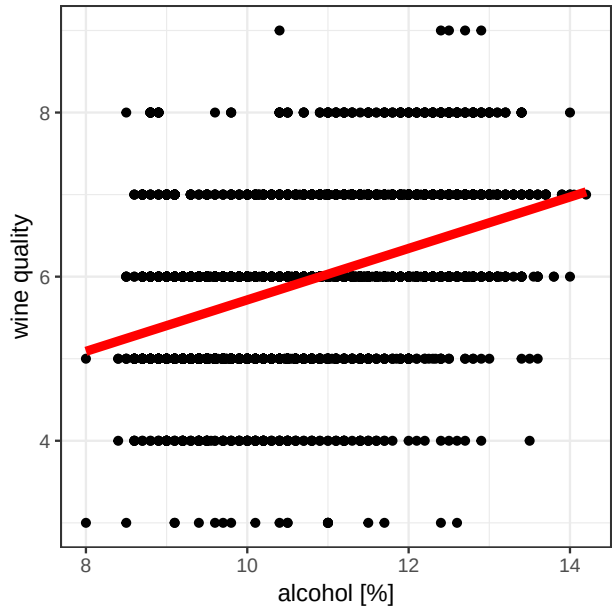


Figure 3.1: (fig:attribution1)Relation between wine quality and concentration of alcohol assessed with linear model

where the expected value can be estimated from the data

$$v(f, x^*, i) = \beta_i x_i^* - \beta_i \bar{x}_i = \beta_i (x_i^* - \bar{x}_i)$$

The logic behind the attribution is the following. Contribution of variable x_i is the difference between model response for value x_i^* minus the average model response.

3.1.1 Wine quality example

It may be a surprise, that the attribution for variable x_i is not the $\beta_i x_i$. But, please consider following example. Figure **??** shows the relation between alcohol and wine quality, based on the wine dataset (**?**). The corresponding linear model is

$$quality(alcohol) = 2.5820 + 0.3135 * alcohol$$

The weakest wine in this dataset has 8% of alcohol, average alcohol concentration is 10.51, so the contribution of alcohol to the model prediction is $0.3135 * (8 - 10.51) = -0.786885$. It means that low value of alcohol for this wine (8%) lower the prediction of quality by -0.786885.

Note, that it would be confusing to forget about normalisation and say, that for the alcohol contribution on quality is $0.3135 * 8 = 2.508$ as this is high positive value.

Note that the linear model ma be rewritten in a following way

$$f(x) = baseline + (x_1 - \bar{x}_1)\beta_1 + ... + (x_p - \bar{x}_p)\beta_p$$

where

$$baseline = \mu + \bar{x}_1\beta_1 + ... + \bar{x}_p\beta_p.$$

Here *baseline* is an average model response and variable contributions show how prediction for particular x^* is different from the average response.

** NOTE for careful readers **

There is a gap between expected value of X_i and average calculated on some dataset $\bar{x}_i$. The latter depends on the data used for calculation of averages. For the sake of simplicity we do not emphasise these differences. To live with this just assume that we have access to a very large validation data that allows us to calculate $\bar{x}_i$ very accurately.

3.2 Model agnostic approaches

In the Section **??** we introduced a method for calculation of variable attributions for linear models. This method is accurate, based directly on the structure of the model. But for most popular machine learning models we cannot assume that they are linear nor even additive.

In next sections we introduce a model agnostic approach. Note that even if the model itself is not additive, the model attribution will be additive.

Again, let $v(f, x^*, i)$ stands for the contribution of variable x_i on prediction of model $f()$ in point x^*.

We expect that such contribution will sum up to the model prediction in a given point (property called *local accuracy*), so

$$f(x^*) = baseline + \sum_{i=1}^{p} v(f, x^*, i)$$

where *baseline* stands for average model response.

Note that the equation above may be rewritten as

$$E[f(X)|X_1 = x_1^*, \ldots, X + p = x_p^*] = E[f(X)] + \sum_{i=1}^{p} v(f, x^*, i)$$

what leads to quite natural proposition for $v(f, x_i^*, i)$, such as

$$v(f, x_i^*, i) = E[f(X)|X_1 = x_1^*, \ldots, X_i = x_i^*] - E[f(X)|X_1 = x_1^*, \ldots, X_{i-1} = x_{i-1}^*]$$

In other words the contribution of variable i is the difference between expected model response conditioned on first i variables minus the model response conditioned on first $i-1$ variables.

Such proposition fulfills the *local accuracy* condition, but unfortunatelly variable contributions depends on the ordering of variables.

See for example Figure **??**. In the first ordering the contribution of variable `age` is calculated as 0.01, while in the second the contribution is calculated as 0.13. Such differences are related to the lack of additivness of the model $f()$. Propositions presented in next two sections present different solutions for this problem.

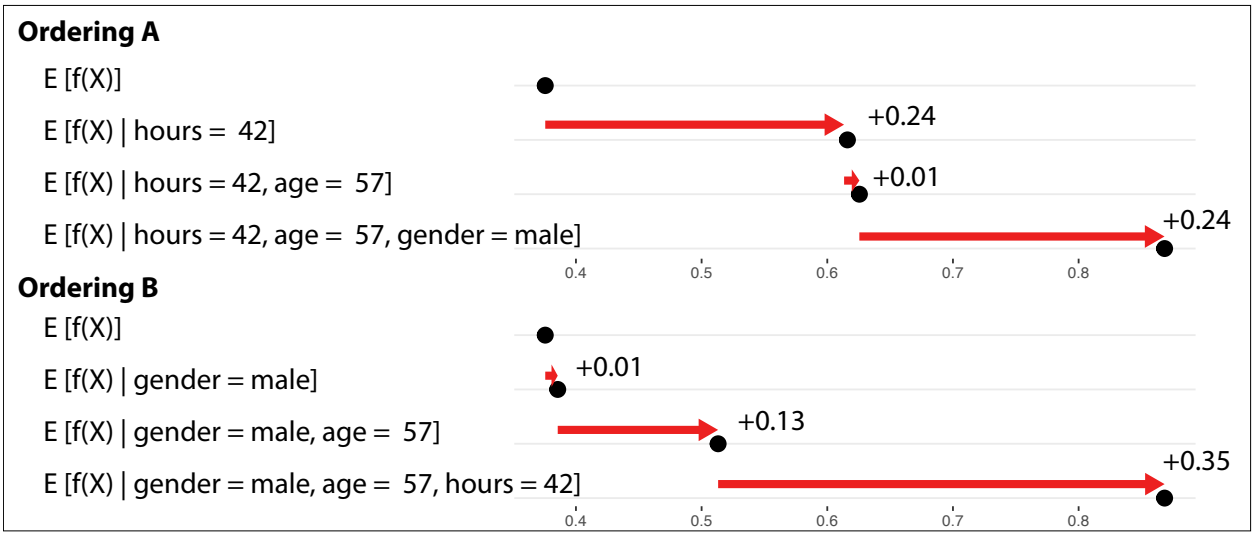


Figure 3.2: (fig:ordering) Two different paths between average model prediction and the model prediction for a selected observation. Black dots stand for conditional average, red arrows stands for changes between conditional averages.

Chapter 4

Break Down Attributions

The approach for variable attribution presented in the Section **??** has the property of *local accuracy*, but variable contributions depends on the variable ordering.

The Break Down method solves this problem by using two-step procedure. In the first step variables are ordered and in the second step the consecutive conditioning is applied to ordered variables.

4.1 The Algorithm

First step of this algorithm is to determine the order of variables for conditioning. It seems to be reasonable to include first variables that are likely to be most important, leaving the noise variables at the end. This leads to order based on following scores

$$score(f, x^*, i) = |E[f(X)] - E[f(X)|X_i = x_i^*]|$$

Note, that the absolute value is needed as variable contributions can be both positive and negative.

Once the ordering is determined in the second step variable contributions are calculated as

$$v(f, x_i^*, i) = E[f(X)|X_{I\cup\{i\}} = x^*_{I\cup\{i\}}] - E[f(X)|X_I = x_I^*]$$

where I is the set of variables that have scores smaller than score for variable i.

$$I = \{j : score(f, x^*, j) < score(f, x^*, i)\}$$

The time complexity of the first step id $O(p)$ where p is the number of variables and the time complexity of the second step is also $O(p)$.

4.2 HR dataset: Hire or Fire?

Let us consider a random forest model created for HR data. The average model response is $\bar{f}(x) = 0.385586$. For a selected observation x^* the table below presents scores for particular variables.

	Ei f(X)	scorei
hours	0.616200	0.230614
salary	0.225528	0.160058

	Ei f(X)	scorei
evaluation	0.430994	0.045408
age	0.364258	0.021328
gender	0.391060	0.005474

Once we determine the order we can calculate sequential contributions

variable	cumulative	contribution
(Intercept)	0.385586	0.385586
hours = 42	0.616200	0.230614
salary = 2	0.400206	-0.215994
evaluation = 2	0.405776	0.005570
age = 58	0.497314	0.091538
gender = male	0.778000	0.280686
final_prognosis	0.778000	0.778000

4.3 Break Down Plots

Once we calculated variable attributions we may plot them in an intuitive form. This intuition behind Break Down Plots is described in Figure **??**.

The variable ordering determined in the first step of Break Down Algorithm is reflected by the ordering of variables in rows of the plot.

The last row of a plot shows the *baseline*, i.e. an average model prediction. The next row corresponds to average model prediction for observations with variable `surface` fixed to value `35`. The next for corresponds to average model prediction with variables `surface` set to `35` and `floor` set to `1`, and so on. The first row corresponds to model response for x^*.

In panels A and B violines show distribution of model predictions for selected points, while red dots stands for averages.

The most minimal form that shows important information is presented in the panel C. Positive values are presented with green bars while negative differences are marked with yellow bar. They sum up to final model prediction, which is denoted by a grey bar in this example.

4.4 Pros and cons

Break Down approach is model agnostic, can be applied to any predictive model that returns a single number. It leads to additive variable attribution. Below we summarize key strengths and weaknesses of this approach.

Pros

- Break Down Plots are easy to understand and decipher.
- Break Down Plots are compact; many variables may be presented in a small space.
- Break Down Plots are model agnostic yet they reduce to intuitive interpretation for linear Gaussian and generalized models.
- Complexity of Break Down Algorithm is linear in respect to the number of variables.

Cons

- If the model is non-additive then showing only additive contributions may be misleading.

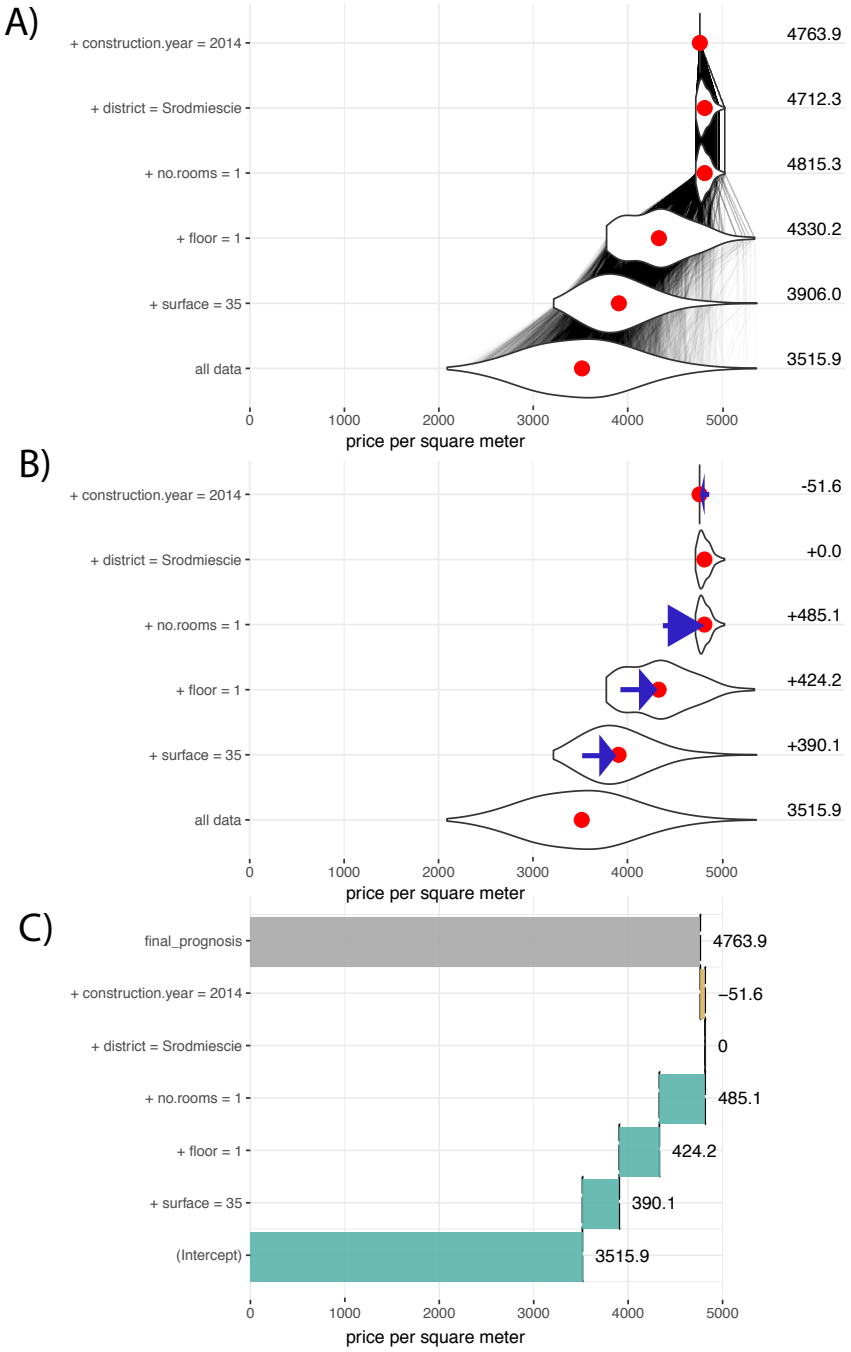


Figure 4.1: (fig:BDPrice4) Break Down Plots show how variables move the model prediction from population average to the model prognosis for a single observation. A) The last row shows distribution of model predictions. Next rows show conditional distributions, every row a new variable is added to conditioning. The first row shows model prediction for a single point. Red dots stand for averages. B) Blue arrows shows how the average conditional response change, these values are variables contributions. C) Only variable contributions are presented.

- Selection of the ordering based on scores is subjective. Different orderings may lead to different contributions.
- For large number of variables the Break Down Plot may be messy with many variables having small contributions.

4.5 Code snippets for R

In this section we present key features of the `breakDown` package for R (**?**). This package covers all features presented in this chapter. It is available on CRAN and GitHub. Find more examples at the website of this package `https://pbiecek.github.io/breakDown/`.

Model preparation

In this section we will present an example based on the `HR` dataset and Random Forest model (**?**). See the Section **??** for more details.

```
library("DALEX")
library("randomForest")
model <- randomForest(status ~ gender + age + hours + evaluation + salary, data = HR)
model
```

```
##
## Call:
##  randomForest(formula = status ~ gender + age + hours + evaluation +     salary, data = HR)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 27.56%
## Confusion matrix:
##          fired   ok promoted class.error
## fired     2263  386      206   0.2073555
## ok         538 1231      452   0.4457452
## promoted   196  385     2190   0.2096716
```

Model exploration with the `breakDown` package is performed in three steps.

1. Create an explainer - wrapper around model and validation data.

Since all other functions work in a model agnostic fashion, first we need to define a wrapper around the model. Here we are using the `explain()` function from `DALEX` package (**?**).

```
explainer_rf_fired <- explain(model,
                 data = HR,
                 y = HR$status == "fired",
                 predict_function = function(m,x) predict(m,x, type = "prob")[,1],
                 label = "fired")
```

2. Select an observation of interest.

Break Down Plots decompose model prediction around a single observation. Let's construct a data frame with corresponding values.

```
new_observation <- data.frame(gender = factor("male", levels = c("male", "female")),
                      age = 57.7,
                      hours = 42.3,
                      evaluation = 2,
                      salary = 2)
```

```
predict(model, new_observation, type = "prob")
```

```
##   fired    ok promoted
## 1  0.79 0.208    0.002
## attr(,"class")
## [1] "matrix" "votes"
```

3. Calculate Break Down decomposition

The `break_down()` function calculates Break Down contributions for a selected model around a selected observation.

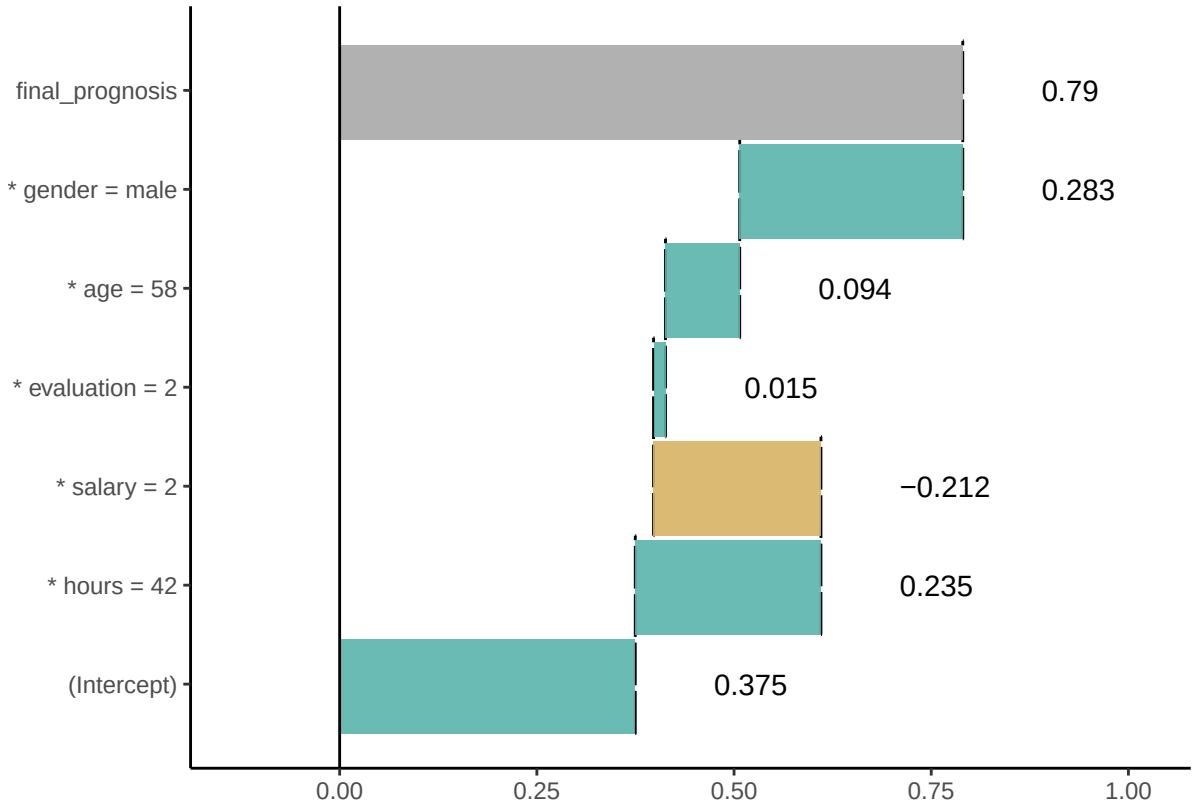
The result from `break_down()` function is a data frame with variable attributions.

```
library("breakDown")
bd_rf <- break_down(explainer_rf_fired,
                    new_observation,
                    check_interactions = FALSE,
                    keep_distributions = TRUE)

bd_rf
```

```
##                   contribution
## (Intercept)              0.375
## * hours = 42             0.235
## * salary = 2            -0.212
## * evaluation = 2         0.015
## * age = 58               0.094
## * gender = male          0.283
## final_prognosis          0.790
## baseline:  0
```

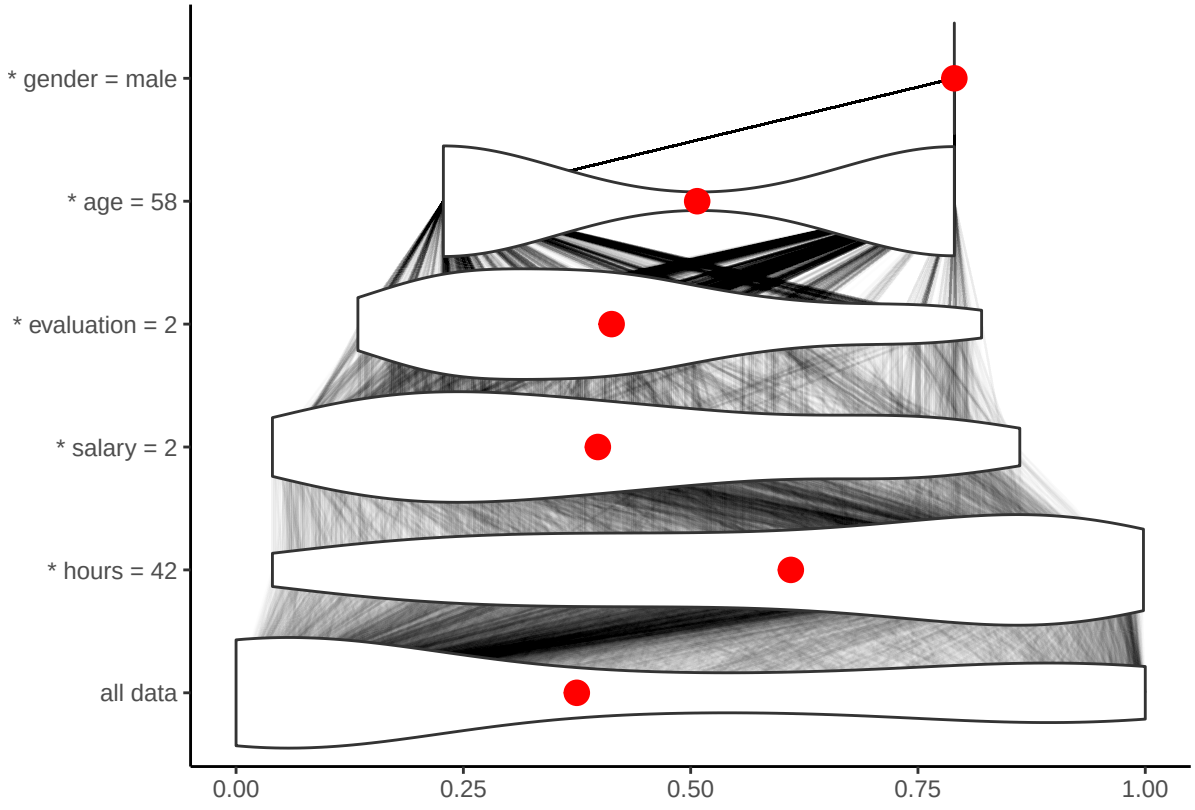
The generic `plot()` function creates a Break Down plots.

```
plot(bd_rf)
```



Add the `plot_distributions = TRUE` argument to enrich model response with additional information.

```
plot(bd_rf, plot_distributions = TRUE)
```



Chapter 5

Break Down for Interactions

In the Section **??** we presented model agnostic approach for additive decomposition of a model prediction for a single observation.

For non-additive models the variables contributions depend on values of other variables.

In this section we present an algorithm that identifies interactions between pairs of variables and include such interactions in variable decomposition plots. Here we present an algorithm for pairs of variables, but it can be easily generalized to larger number of variables.

5.1 The Algorithm

This algorithm is also composed out of two steps. In the first step variables and pairs of variables are ordered in terms of their importance, while in the second step the consecutive conditioning is applied to ordered variables.

To determine an importance of variables and pairs of variables following scores are being calculated.

For a single variable

$$score_1(f, x^*, i) = |E[f(X)|X_i = x_i^*] - E[f(X)]|$$

For pairs of variables

$$score_2(f, x^*, (i,j)) = \left|E[f(X)|X_i = x_i^*, X_j = x_j^*] - E[f(X)|X_i = x_i^*] - E[f(X)|X_j = x_j^*] + E[f(X)]\right|$$

Note that this is equivalent to

$$score_2(f, x^*, (i,j)) = \left|E[f(X)|X_i = x_i^*, X_j = x_j^*] - score_1(f, x^*, i) - score_1(f, x^*, j) + baseline\right|$$

In other words the $score_1(f, x^*, i)$ measures how much the average model response changes if variable x_i is set to x_i^*, which is some index of local variable importance. On the other hand the $score_2(f, x^*, (i,j))$ measures how much the change is different than additive composition of changes for x_i and x_j, which is some index of local interaction importance.

Note, that for additive models $score_2(f, x^*, (i,j))$ shall be close to zero. So the larger is this value the larger deviation from additivness.

The second step of the algorithm is the sequential conditioning. In this version in every new step we condition on a single variable of pair of variables in an order determined by $score_1$ and $score_2$.

The complexity of the first step id $O(p^2)$ where p stands for the number of variables. The complexity of the second step is $O(p)$.

5.2 HR dataset: Hire or Fire?

Again, let us consider a HR dataset. The table below shows $score_1$ and $score_2$ calculated for consecutive variables.

	Ei f(X)	score1	score2
hours	0.616200	0.230614	
salary	0.225528	-0.160058	
age:gender	0.516392		0.146660
salary:age	0.266226		0.062026
salary:hours	0.400206		-0.055936
evaluation	0.430994	0.045408	
hours:age	0.635662		0.040790
salary:evaluation	0.238126		-0.032810
age	0.364258	-0.021328	
evaluation:hours	0.677798		0.016190
salary:gender	0.223292		-0.007710
evaluation:age	0.415688		0.006022
gender	0.391060	0.005474	
hours:gender	0.626478		0.004804
evaluation:gender	0.433814		-0.002654

Once we determined the order, we can calculate sequential conditionings. In the first step we condition over variable `hours`, then over `salary`. The third position is occupied by interaction between `age:gender` thus we add both variables to the conditioning

variable	cumulative	contribution
(Intercept)	0.385586	0.385586
hours = 42	0.616200	0.230614
salary = 2	0.400206	-0.215994
age:gender = 58:male	0.796856	0.396650
evaluation = 2	0.778000	-0.018856
final_prognosis	0.778000	0.778000

5.3 Break Down Plots

Break Down Plots for interactions are similar in structure as plots for single variables. The only difference is that in some rows pair of variable is listed in a single row. See an example in Figure **??**.

5.4 Pros and cons

Break Down for interactions shares many features of Break Down for single variables. Below we summarize unique strengths and weaknesses of this approach.

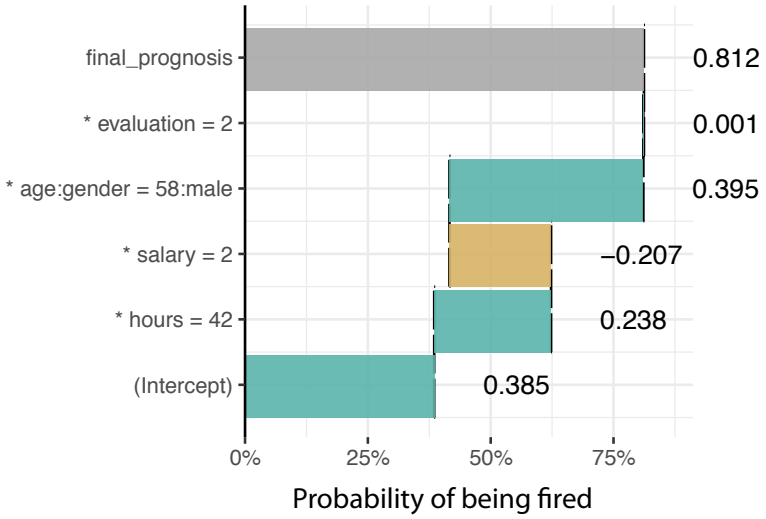


Figure 5.1: (fig:bdInter1) Break Down Plot for variable attrbution with interactions

Pros

- If interactions are present in the model, then additive contributions may be misleading. In such case the identification of interactions leads to better explanations.
- Complexity of Break Down Algorithm is quadratic, what is not that bad if number of features is small or moderate.

Cons

- For large number of variables, the consideration of all interactions is both time consuming and sensitive to noise as the number of $score_2$ scores grow faster than number of $score_1$.

5.5 Code snippets for R

The algorithm for Break Down for Interactions is also implemented in the `break_down` function from `breakDown` package.

It is enough to set argument `check_interactions = TRUE` to identify interactions.

Model preparation

First a model needs to be trained.

```
library("DALEX")
library("randomForest")
model <- randomForest(status ~ gender + age + hours + evaluation + salary, data = HR)
model
```

```
##
## Call:
##  randomForest(formula = status ~ gender + age + hours + evaluation +     salary, data = HR)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 27.25%
```

```
## Confusion matrix:
##          fired   ok promoted class.error
## fired     2280  376      199   0.2014011
## ok         541 1246      434   0.4389914
## promoted   201  387     2183   0.2121978
```

Model exploration with the `breakDown` package is performed in three steps.

1. Create an explainer - wrapper around model and validation data.

Since all other functions work in a model agnostic fashion, first we need to define a wrapper around the model. Here we are using the `explain()` function from `DALEX` package.

```
explainer_rf_fired <- explain(model,
                 data = HR,
                 y = HR$status == "fired",
                 predict_function = function(m,x) predict(m,x, type = "prob")[,1],
                 label = "fired")
```

2. Select an observation of interest.

Break Down Plots decompose model prediction around a single observation. Let's construct a data frame with corresponding values.

```
new_observation <- data.frame(gender = factor("male", levels = c("male", "female")),
                      age = 57.7,
                      hours = 42.3,
                      evaluation = 2,
                      salary = 2)

predict(model, new_observation, type = "prob")
```

```
##   fired    ok promoted
## 1 0.798 0.198    0.004
## attr(,"class")
## [1] "matrix" "votes"
```

3. Calculate Break Down decomposition

The `break_down()` function calculates Break Down contributions for a selected model around a selected observation.

Note that `check_interactions = TRUE` is needed to identify interactions.

The result from `break_down()` function is a data frame with variable attributions.

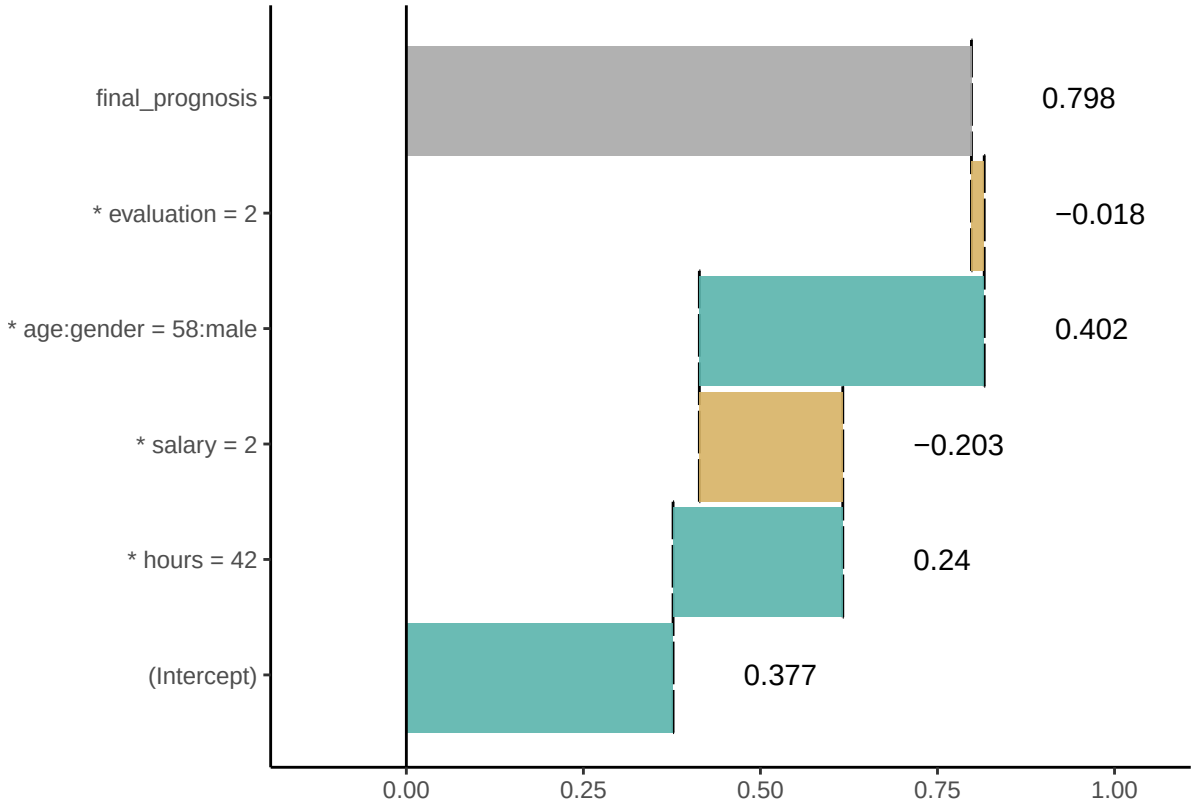
```
library("breakDown")
bd_rf <- break_down(explainer_rf_fired,
                 new_observation,
                 check_interactions = TRUE)

bd_rf
```

```
##                          contribution
## (Intercept)                     0.377
## * hours = 42                    0.240
## * salary = 2                   -0.203
## * age:gender = 58:male          0.402
## * evaluation = 2               -0.018
## final_prognosis                 0.798
```

```
## baseline:  0
```

The generic `plot()` function creates a Break Down plots.

```
plot(bd_rf)
```



Chapter 6

Shapley Attributions

In the Section **??** we show the problem related to the ordering of variables. In the Section **??** we show an approach in which the ordering was determined based on single step assessment of variable importance.

In this section we introduce other, very popular approach for additive variable attribution. The problem of contributions that depends on the variable ordering is solved by averaging over all possible orderings.

This method is motivated with results in cooperative game theory and was first introduced in (**?**). Wide adoption of this method comes with a NIPS 2017 paper (**?**) and python library SHAP `https://github.com/slundberg/shap`. Authors of the SHAP method introduced also efficient algorithm for tree-based models, see (**?**).

6.1 The Algorithm

The name *Shapley Values* comes from the solution in cooperative game theory attributed to Lloyd Shapley. The original problem was to assess how important is each player to the overall cooperation, and what payoff can he or she reasonably expect from the coalition? (**?**)

In the context of model interpretability the payoff is the average model response while the players are the variables in the conditioning. Then Formula for variable contributions is following.

$$v(f, x^*, i) = \frac{1}{|P|} \sum_{S \subseteq P \setminus \{i\}} \binom{|P|-1}{|S|}^{-1} \Big(E[f(X)|X_{S\cup\{i\}} = x^*_{S\cup\{i\}}] - E[f(X)|X_S = x^*_S] \Big)$$

where $P = \{1, \ldots, p\}$ is the set of all variables. The intuition beyond this contribution is following. We consider all possible orderings of variables (yes, there is 2^p of them) and calculate the contribution of variable i as an average from contributions calculated in particular orderings.

The part $E[f(X)|X_{S\cup\{i\}} = x^*_{S\cup\{i\}}] - E[f(X)|X_S = x^*_S]$ is the contribution of variable i which is introduces after variables from S.

Time complexity of this method is $O(2^p)$ where p stands for the number of variables. Such complexity makes this method impractical for most cases. Fortunately it is enough to assess this value. (**?**) proposed to use sampling. (**?**) proposed fast implementations for tree based ensembles.

6.2 Code snippets for R

In this section we will present an example based on the `HR` dataset and Random Forest model (**?**). See the Section **??** for more details.

```
library("DALEX")
library("randomForest")
model_rf <- randomForest(status ~ gender + age + hours + evaluation + salary, data = HR)
```

Here we use the `iml` package, see more examples in (**?**).

```
library("iml")
explainer_rf = Predictor$new(model_rf, data = HR, type="prob")
```

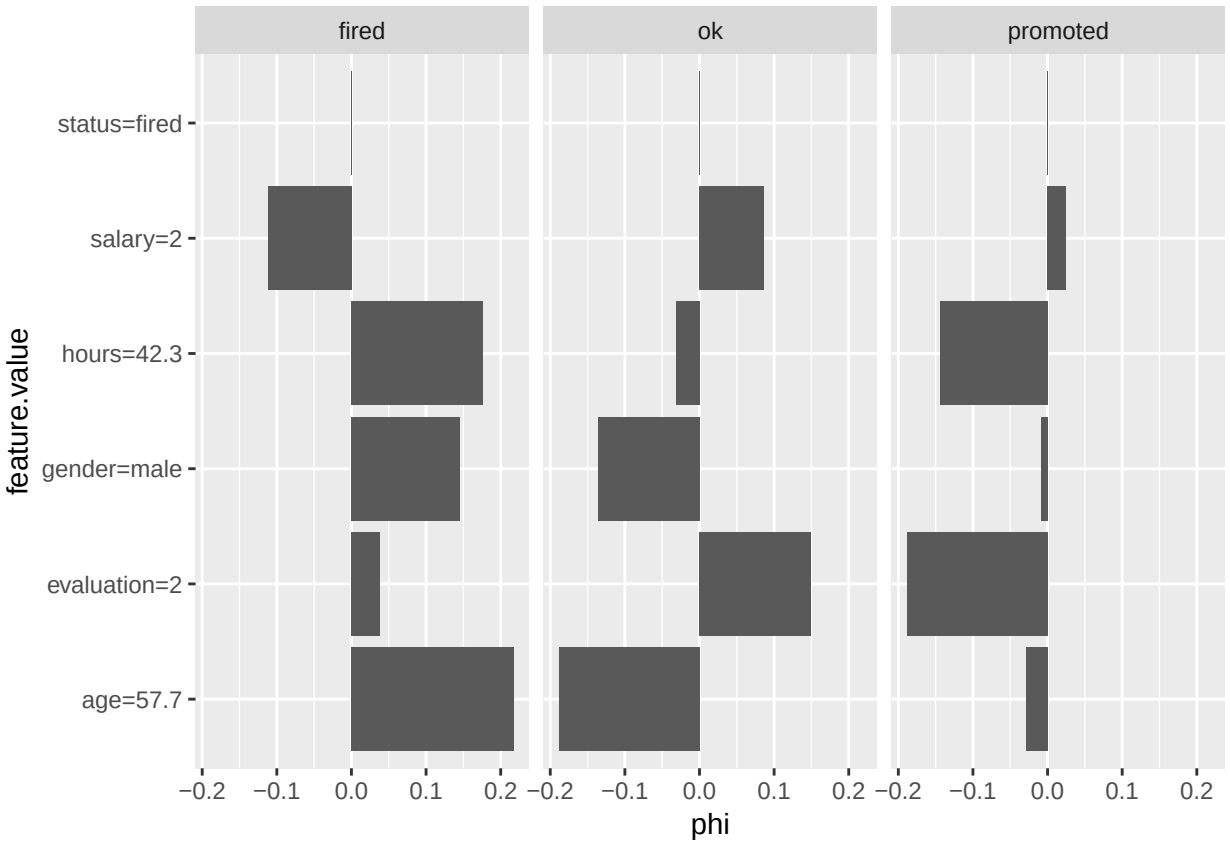
Explanations for a new observation.

```
new_observation <- data.frame(gender = factor("male", levels = c("male", "female")),
                      age = 57.7,
                      hours = 42.3,
                      evaluation = 2,
                      salary = 2,
                      status = factor("fired"))

shapley = Shapley$new(explainer_rf, x.interest = new_observation)
shapley
```

```
## Interpretation method:  Shapley
## Predicted value: 0.786000, Average prediction: 0.374997 (diff = 0.411003) Predicted value: 0.212000, Aver
##
## Analysed predictor:
## Prediction task: unknown
##
##
## Analysed data:
## Sampling from data.frame with 7847 rows and 6 columns.
##
## Head of results:
##      feature class      phi    phi.var feature.value
## 1     gender fired  0.14586 0.06538345   gender=male
## 2        age fired  0.21766 0.07000873      age=57.7
## 3      hours fired  0.17648 0.10503152    hours=42.3
## 4 evaluation fired  0.03776 0.01978063  evaluation=2
## 5     salary fired -0.11184 0.03369856      salary=2
## 6     status fired  0.00000 0.00000000  status=fired
```

And the plot with Shapley attributions.

```
plot(shapley)
```



See more examples for `iml` package in the (**?**) book.

6.3 Pros and cons

Shapley Values give a uniform approach to decompose model prediction into parts that can be attributed additively to variables. Below we summarize key strengths and weaknesses of this approach.

Pros

- There is a nice theory based on cooperative games.
- (**?**) shows that this method unifies different approaches to additive features attribution.
- There is efficient implementation available for Python.
- (**?**) shows more desired properties of this method, like symmetry or additivity.

Cons

- The exact calculation of Shapley values is time consuming.
- If the model is not additive, then the Shaply scores may be misleading. And there is no way to determine if model is far from additiveness.

Note that fully additive model solutions presented in sections **??**, **??** and **??** lead to same variable contributions.

Chapter 7

Local approximations with white-box model

A different approach to explanations of a single observations is through surrogate models. Models that easy to understand and are similar to black box model around the point of interest.

Variable attribution methods, that were presented in the Section **??** are not interested in the local curvature of the model. They rather compare model prediction against average model prediction and they use probability structure of the dataset.

The complementary approach would be to directly explore information about model curvature around point of interest. In the section **??** we introduced Ceteris Paribus tool for such what-if analysis. But the limitation of ceteris Paribus pltos is that they explore changes along single dimension or pairs of dimensions.

In this section we describe an another approach based on local approximations with white-box models. This approach will also investigate local curvature of the model but indirectly, through surrogate white-box models.

The most known method from this class if LIME (Local Interpretable Model-Agnostic Explanations), introduced in the paper *Why Should I Trust You?: Explaining the Predictions of Any Classifier* (**?**). This methods and it's clones are now implemented in various R and python packages, see for example (**?**), (**?**) or (**?**).

7.1 The Algorithm

The LIME method, and its clones, has following properties:

- *model-agnostic*, they do not imply any assumptions on model structure,
- *interpretable representation*, model input is transformed into a feature space that is easier to understand. One of applications comes from image data, single pixels are not easy to interpret, thus the LIME method decompose image into a series of super pixels, that are easier to interpret to humans,
- *local fidelity* means that the explanations shall be locally well fitted to the black-box model.

Therefore the objective is to find a local model M^L that approximates the black box model f in the point x^*. As a solution the penalized loss function is used. The white-box model that is used for explanations satisfies following condition.

$$M^L(x^*) = \arg\min_{g \in G} L(f, g, \Pi_{x^*}) + \Omega(g)$$

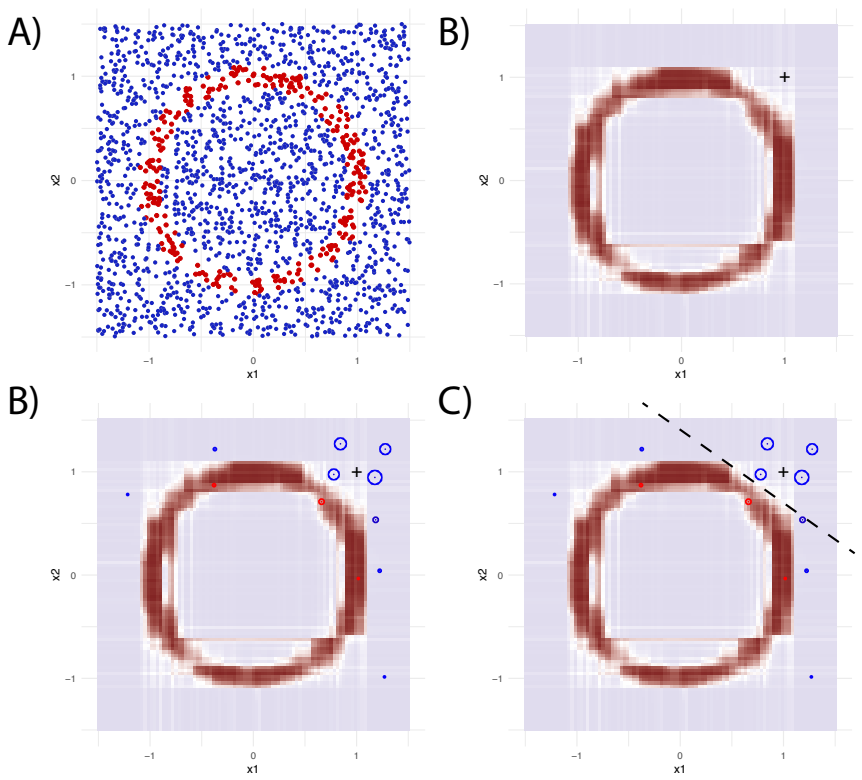


Figure 7.1: (fig:LIME1) A schematic idea behind local model approximations. Panel A shows training data, colors correspond to classess. Panel B showhs results fom the Random Forest model, whis is where the algorithm starts. Panel C shows new data sampled around the point of interest. Their color correspond to model response. Panel D shows fitted linear model that approximated the random forest model around point of interest

where G is a family of white box models (e.g. linear models), Π_{x^*} is neighbourhood of x^* and Ω stands for model complexity.

The algorithm is composed from three steps:

- Identification of interpretable data representations,
- Local sampling around the point of interest,
- Fitting a white box model in this neighbouhood

Identification of interpretable data representations

For image data, single pixel is not an interpretable feature. In this step the input space of the model is transformed to input space that is easier to understand for human. The image may be decomposed into parts and represented as presence/absence of some part of an image.

Local sampling around the point of interest

Once the interpretable data representation is identified, then the neighbourhood around point of interest needs to be explored.

Fitting a white box model in this neighbouhood

Any model that is easy to interpret may be fitted to this data, like decision tree or rule based system. However in practice the most common family of models are linear models.

7.2 HR dataset: Hire or Fire?

```
library("DALEX")
library("randomForest")
model <- randomForest(status ~ gender + age + hours + evaluation + salary, data = HR)
model

explainer_rf_fired <- explain(model,
                  data = HR,
                  y = HR$status == "fired",
                  predict_function = function(m,x) predict(m,x, type = "prob")[,1],
                  label = "fired")


new_observation <- data.frame(gender = factor("male", levels = c("male", "female")),
                        age = 57.7,
                        hours = 42.3,
                        evaluation = 2,
                        salary = 2)

predict(model, new_observation, type = "prob")


library(lime)
model_type.randomForest <- function(x, ...) "classification"
lime_rf <- lime(HR[,1:5], model)
lime::explain(new_observation[,1:5], lime_rf, n_labels = 1, n_features = 10)


library(iml)
mod = Predictor$new(model, data = HR[,1:5])
x.interest = new_observation
lemon = LocalModel$new(mod, x.interest = x.interest, k = 5)
lemon

lemon$results
plot(lemon)
```

7.3 Pros and cons

Local approximations are model agnostic, can be applied to any predictive model. Below we summarize key strengths and weaknesses of this approach.

Pros

- This method is highly adopted in text analysis and image analysis, in part thanks to the interpretable data representations.
- The intuition behind the model is straightforward
- Model explanations are sparse, thus only small number of features is used

Cons

- For continuous variables and tabular data it is not that easy to find interpretable representations

- The black-box model approximated the data and the white box model approximates the black box model. We do not have control over the quality of local fit of the white box model, thus the surrogate model may be misleading.
- Due to the *curse of dimensionality*, for high dimensional space points are sparse.

7.4 Code snippets for R

In this section we present example application of `lime` (**?**) and `live` (**?**) packages. Note that this method is also implemented in `iml` (**?**) and other packages. These pacakages differ in some details and also results in different explanations.

Model preparation

In this section we will present examples based on the `HR` dataset. See the Section **??** for more details.

```
library("DALEX")
head(HR)
```

```
##   gender      age    hours evaluation salary   status
## 1   male 32.58267 41.88626          3      1    fired
## 2 female 41.21104 36.34339          2      5    fired
## 3   male 37.70516 36.81718          3      0    fired
## 4 female 30.06051 38.96032          3      2    fired
## 5   male 21.10283 62.15464          5      3 promoted
## 6   male 40.11812 69.53973          2      0    fired
```

The problem here is to predict average price for square meter for an apartment. Let's build a random forest model with `randomForest` package (**?**).

```
library("randomForest")
rf_model <- randomForest(status ~ gender + age + hours + evaluation + salary, data = HR)
rf_model
```

```
##
## Call:
##  randomForest(formula = status ~ gender + age + hours + evaluation +     salary, data = HR)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 27.39%
## Confusion matrix:
##          fired   ok promoted class.error
## fired     2268  384      203   0.2056042
## ok         526 1257      438   0.4340387
## promoted   199  399     2173   0.2158066
```

```
new_observation <- data.frame(gender = factor("male", levels = c("male", "female")),
                      age = 57.7,
                      hours = 42.3,
                      evaluation = 2,
                      salary = 2)

predict(rf_model, new_observation, type = "prob")
```

```
##   fired    ok promoted
```

```
## 1 0.792 0.208        0
## attr(,"class")
## [1] "matrix" "votes"
```

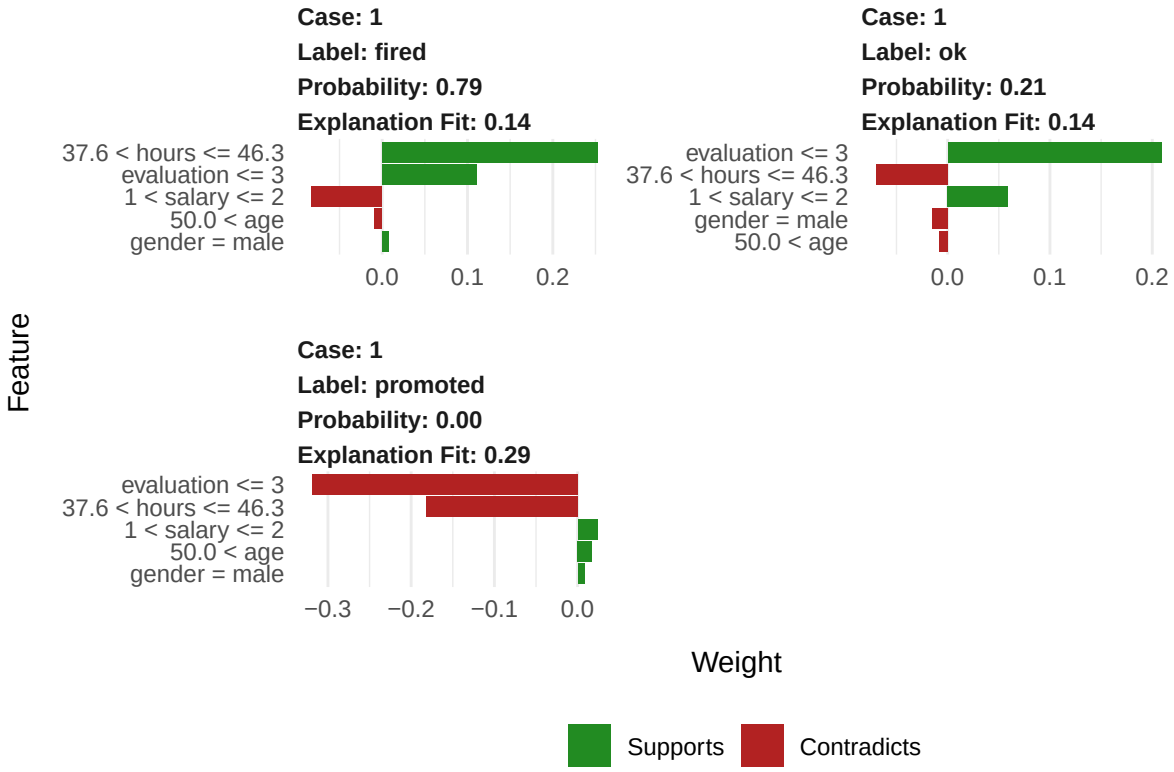
The lime pacakge

```
library("lime")
model_type.randomForest <- function(x, ...) "classification"
lime_rf <- lime(HR[,1:5], rf_model)
explanations <- lime::explain(new_observation[,1:5], lime_rf, n_labels = 3, n_features = 5)
explanations
```

```
##        model_type case    label label_prob  model_r2 model_intercept
## 1  classification    1    fired      0.792 0.1352405       0.2819122
## 2  classification    1    fired      0.792 0.1352405       0.2819122
## 3  classification    1    fired      0.792 0.1352405       0.2819122
## 4  classification    1    fired      0.792 0.1352405       0.2819122
## 5  classification    1    fired      0.792 0.1352405       0.2819122
## 6  classification    1       ok      0.208 0.1383425       0.2178898
## 7  classification    1       ok      0.208 0.1383425       0.2178898
## 8  classification    1       ok      0.208 0.1383425       0.2178898
## 9  classification    1       ok      0.208 0.1383425       0.2178898
## 10 classification    1       ok      0.208 0.1383425       0.2178898
## 11 classification    1 promoted      0.000 0.2909925       0.5002152
## 12 classification    1 promoted      0.000 0.2909925       0.5002152
## 13 classification    1 promoted      0.000 0.2909925       0.5002152
## 14 classification    1 promoted      0.000 0.2909925       0.5002152
## 15 classification    1 promoted      0.000 0.2909925       0.5002152
##    model_prediction    feature feature_value feature_weight
## 1        0.56022390     gender           2.0    0.007168961
## 2        0.56022390        age          57.7   -0.008946023
## 3        0.56022390      hours          42.3    0.252291949
## 4        0.56022390 evaluation           2.0    0.110295207
## 5        0.56022390     salary           2.0   -0.082498351
## 6        0.39184406     gender           2.0   -0.015519618
## 7        0.39184406        age          57.7   -0.008177973
## 8        0.39184406      hours          42.3   -0.070086417
## 9        0.39184406 evaluation           2.0    0.209072196
## 10       0.39184406     salary           2.0    0.058666032
## 11       0.04794627     gender           2.0    0.008356990
## 12       0.04794627        age          57.7    0.017126166
## 13       0.04794627      hours          42.3   -0.182121763
## 14       0.04794627 evaluation           2.0   -0.319425228
## 15       0.04794627     salary           2.0    0.023794891
##            feature_desc                      data          prediction
## 1         gender = male 2.0, 57.7, 42.3, 2.0, 2.0 0.792, 0.208, 0.000
## 2            50.0 < age 2.0, 57.7, 42.3, 2.0, 2.0 0.792, 0.208, 0.000
## 3  37.6 < hours <= 46.3 2.0, 57.7, 42.3, 2.0, 2.0 0.792, 0.208, 0.000
## 4       evaluation <= 3 2.0, 57.7, 42.3, 2.0, 2.0 0.792, 0.208, 0.000
## 5       1 < salary <= 2 2.0, 57.7, 42.3, 2.0, 2.0 0.792, 0.208, 0.000
## 6         gender = male 2.0, 57.7, 42.3, 2.0, 2.0 0.792, 0.208, 0.000
## 7            50.0 < age 2.0, 57.7, 42.3, 2.0, 2.0 0.792, 0.208, 0.000
## 8  37.6 < hours <= 46.3 2.0, 57.7, 42.3, 2.0, 2.0 0.792, 0.208, 0.000
## 9       evaluation <= 3 2.0, 57.7, 42.3, 2.0, 2.0 0.792, 0.208, 0.000
## 10      1 < salary <= 2 2.0, 57.7, 42.3, 2.0, 2.0 0.792, 0.208, 0.000
```

```
## 11        gender = male 2.0, 57.7, 42.3, 2.0, 2.0 0.792, 0.208, 0.000
## 12           50.0 < age 2.0, 57.7, 42.3, 2.0, 2.0 0.792, 0.208, 0.000
## 13 37.6 < hours <= 46.3 2.0, 57.7, 42.3, 2.0, 2.0 0.792, 0.208, 0.000
## 14      evaluation <= 3 2.0, 57.7, 42.3, 2.0, 2.0 0.792, 0.208, 0.000
## 15      1 < salary <= 2 2.0, 57.7, 42.3, 2.0, 2.0 0.792, 0.208, 0.000
```

```
plot_features(explanations)
```



The live package

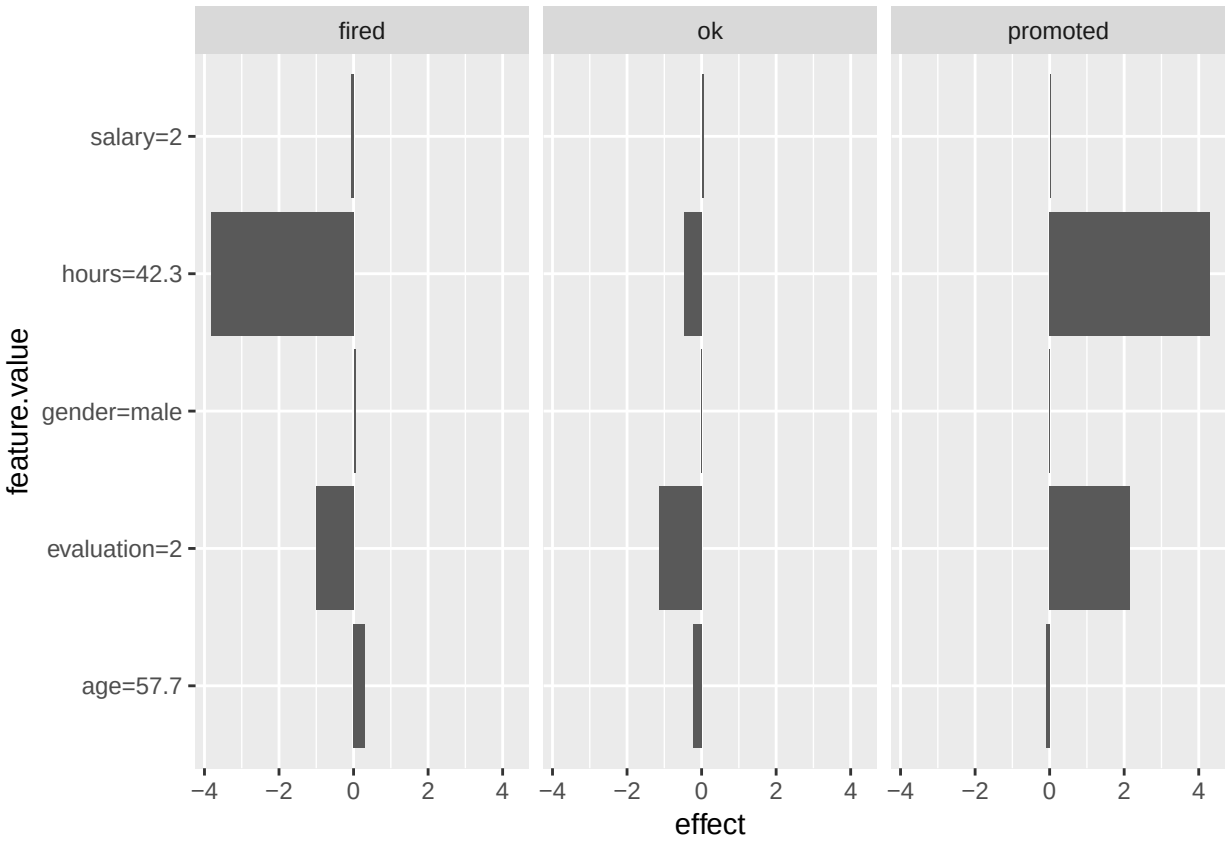
```
library("live")
new_observation$status <- "fired"

similar <- sample_locally2(data = HR,
                           explained_instance = new_observation,
                           explained_var = "status",
                           method = "lime",
                           size = 500)
similar <- sample_locally2(data = HR,
                           explained_instance = new_observation,
                           explained_var = "status",
                           size = 500)
similar1 <- add_predictions2(to_explain = similar,
                             black_box_model = rf_model,
                              predict_fun = function(m,x) predict(m,x,type="prob")[,1])
HR_explanation <- fit_explanation2(live_object = similar1,
                              white_box = "regr.glm")
HR_explanation

plot(HR_explanation, type = "forest")
```

The iml package

```
library("iml")

explainer_rf = Predictor$new(rf_model, data = HR[,1:5])
white_box = LocalModel$new(explainer_rf, x.interest = new_observation[,1:5], k = 5)
white_box
```

```
## Interpretation method:  LocalModel
##
##
## Analysed predictor:
## Prediction task: unknown
##
##
## Analysed data:
## Sampling from data.frame with 7847 rows and 5 columns.
##
## Head of results:
##           beta x.recoded      effect x.original     feature feature.value
## 1  0.053187250       1.0  0.05318725       male gender=male   gender=male
## 2  0.005287695      57.7  0.30510000       57.7         age      age=57.7
## 3 -0.090588272      42.3 -3.83188389       42.3       hours    hours=42.3
## 4 -0.508445662       2.0 -1.01689132          2  evaluation  evaluation=2
## 5 -0.037130808       2.0 -0.07426162          2      salary      salary=2
## 6 -0.026739834       1.0 -0.02673983       male gender=male   gender=male
##   .class
## 1  fired
## 2  fired
## 3  fired
## 4  fired
## 5  fired
## 6     ok
```

```
plot(white_box)
```


fired
ok
promoted
feature.value
salary=2
hours=42.3
gender=male
evaluation=2
age=57.7
−4
−2
0
2
4
effect

Chapter 8

Ceteris Paribus Principle

In this section we introduce tools based on Ceteris Paribus principle. The main goal for these tools is to help understand how changes in model input affect changes in model output.

Presented explainers are linked with the second law introduced in Section **??**, i.e. law for prediction's speculations. This is why these explainers are also known as *What-If model analysis* or *Individual Conditional EXpectations* (**?**). It turns out that it is easier to understand how blacx-box model is working if we can play with it by changing variable by variable.

Think of following usecases:

- Think about a model for hart attack. How the model response would change if a patient cuts the number of smoked cigarettes by half or increase physical activity.
- Think about a model for credit scoring. A customer gets a low score and is asking what he needs to change to increase this score to a certain level, to pass the bank criteria.
- Think about a model for apartment prices. An investor wants to know how much the price may increase if apartment standard is upgraded.

8.1 Introduction

Ceteris paribus is a Latin phrase meaning "other things held constant" or "all else unchanged". Using this principle we examine input variable per variable separatly, asumming that effects of all other variables are unchanged. See Figure **??**

Similar to the LIME method introduced in the section **??**, Ceteris Paribus profiles examine curvature of a model response function. The difference between these two methods that LIME approximates the model curvature with a simpler white-box model that is easier to present. Usually the LIME model is sparse, thus our attention may be limited to smaller number of dimensions. In contrary, the CP plots show conditional model response for every variable. In the last subsection we discuss pros and cons of this approach.

8.2 1D profiles

Let $f_M(x) : \mathcal{R}^d \to \mathcal{R}$ denote a predictive model, i.e. function that takes d dimensional vector and calculate numerical score. Symbol $x \in \mathcal{R}^d$ refers to a point in the feature space. We use subscript x_i to refer to a different data points and superscript x^j to refer to specific dimensions. Additionally, let x^{-j} denote all coordinates except j-th and let $x|^j = z$ denote a data point x^* with all coordinates equal to x except coordinate j equal to value z. I.e. $\forall_{i \neq j} x^i = x^{*,i}$ and $x^j = z$. In other words $x|^j = z$ denote a x with jth coordinate changed to z.

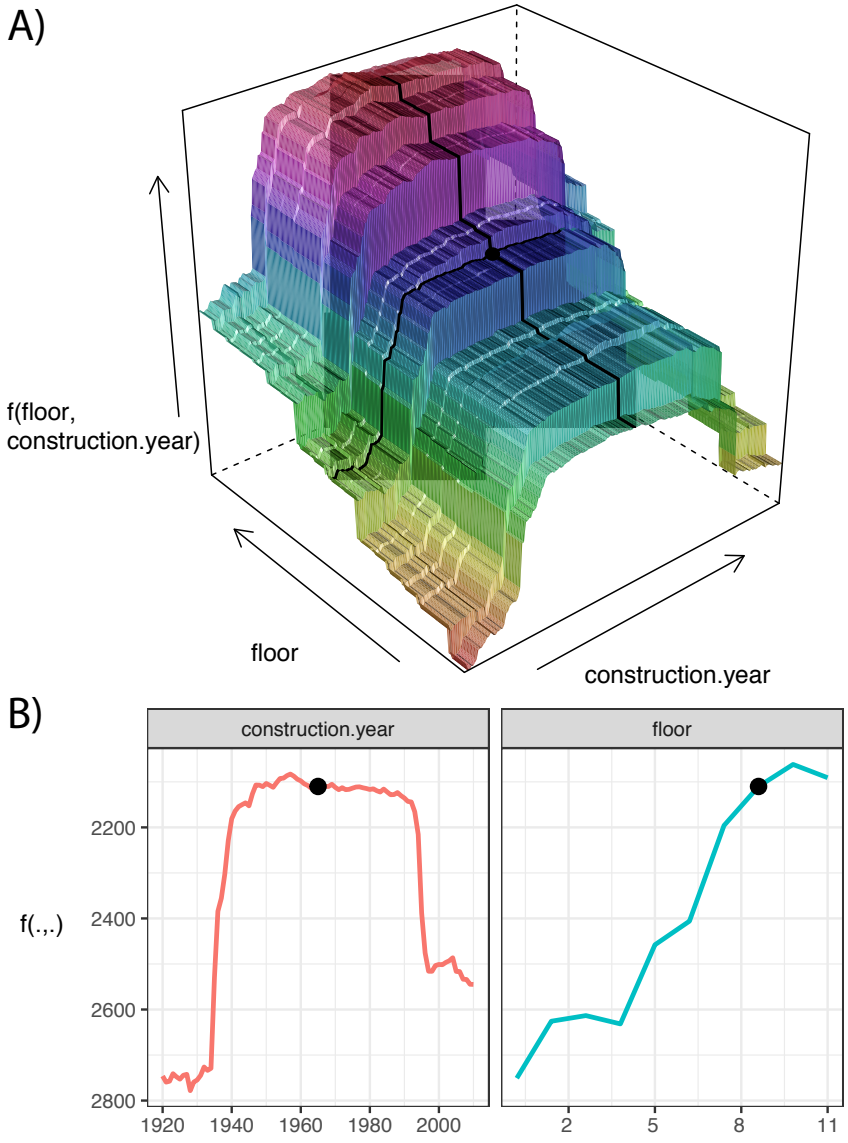


Figure 8.1: (fig:modelResponseCurveLine) A) Model response surface. Ceteris Paribus profiles marked with black curves helps to understand the curvature of the model response by updating only a single variable. B) CP profiles are individual conditional model responses

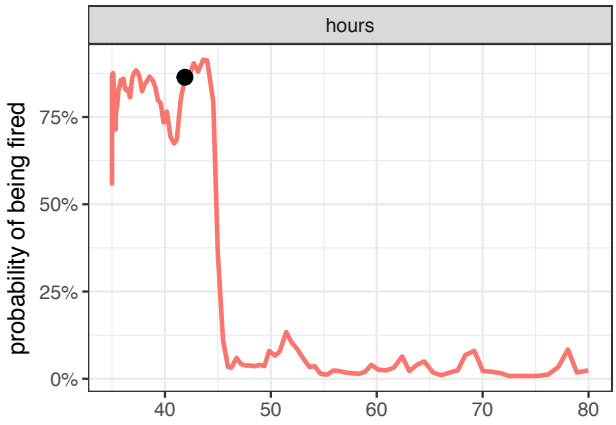


Figure 8.2: (fig:HRCPHiredHours) Ceteris Paribus profile for Random Forest model that assess the probability of being fired in call center as a function of average number of working hours

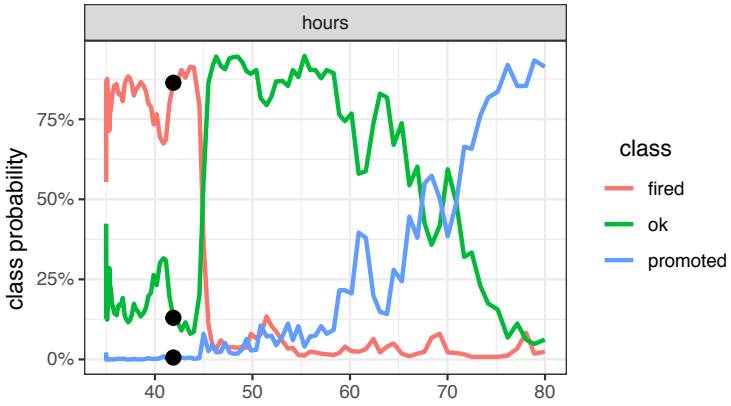


Figure 8.3: (fig:HRCPAllHours) Ceteris Paribus profiles for three classess predicted by the Random Forest model as a function of average number of working hours

Now we can define uni-dimensional Ceteris Paribus Profile for model f, variable j and point x as

$$CP^{f,j,x}(z) := f(x|^j = z).$$

I.e. CP profile is a model response obtained for observations created based on x with coordinate j changed and all other coordinates kept unchanged.

A natural way to visualise CP profiles is to use a profile plot as in Figure **??**.

Figure **??** shows an example of Ceteris Paribus profile. The black dot stands for prediction for a single observation. Grey line show how the model response would change if in this single observation coordinate `hours` will be changed to selected value. One thing that we can read is that the model response is not smooth and there is some variability along the profile. Second thing is that for this particular observation the model response would drop significantly if the variable `hours` will be higher than 45.

Since in the example dataset we are struggling with model for three classes, one can plot CP profiles for each class in the same panel. See an example in the Figure **??**.

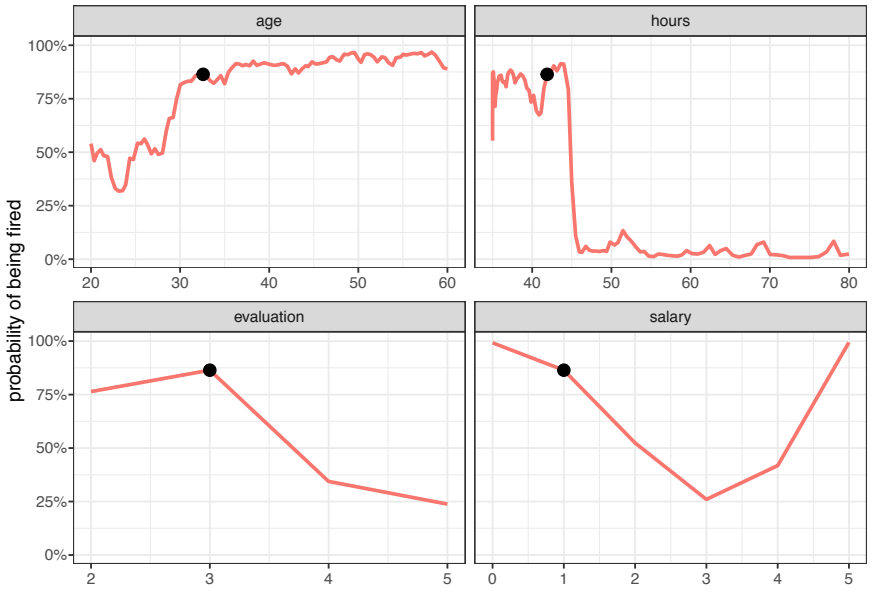


Figure 8.4: (fig:HRCPFiredAll) Ceteris Paribus profiles for all continuous variables

Usually model input consist many variables, then it is beneficial to show more variables at the same time. The easiest way to do so is to plot consecutive variables on separate panels. See an example in Figure **??**.

8.3 Profile oscillations

Visual examination of variables is insightful, but for large number of variables we end up with large number of panels, most of which are flat. This is why we want to asses variable importance and show only profiles for important variables. The advantage of CP profiles is that they lead to a very natural and intuitive way of assessing the variable importance for a single prediction. The intuition is: the more important variable the larger are changes along the CP profile. If variable is not important then model response will barely change. If variable is important the CP profile change a lot for different values of a variable.

Let's write it down in a more formal way.

Let $vip_j^{CP}(x)$ denotes variable importance calculated based on CP profiles in point x for variable j.

$$vip_j^{CP}(x) = \int_{-\inf}^{inf} |CP^{f,j,x}(z) - f(x)|dz$$

So it's an absolute deviation from $f(x)$. Note that one can consider different modification of this coefficient:

1. Deviations can be calculated not as a distance from $f(x)$ but from average $\bar{CP}^{f,j,x}(z)$.
2. The integral may be weighted based on the density of variable x^j.
3. Instead of absolute deviations one may use root from average squares.

TODO: we need to verify which approach is better. Anna Kozak is working on this

The straightforward estimator for $vip_j^{CP}(x)$ is

$$\widehat{vip_j^{CP}}(x) = \frac{1}{n}\sum_{i=1}^{n} |CP^{f,j,x}(x_i) - f(x)|.$$

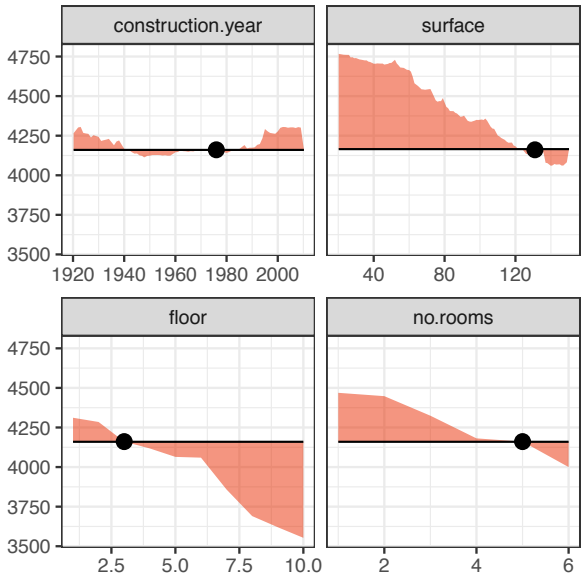


Figure 8.5: (fig:CPVIPprofiles) CP oscillations are average deviations between CP profiles and the model response

Figure **??** shows the idea behind measuring oscillations. The larger the highlighted area the more important is the variable.

Figure **??** summarizes variable oscillations. Such visuals help to quickly grasp how large are model oscillations around a specific point.

NOTE

Variable importance for single prediction may be very different than variable importance for the full model.

For example, consider a model

$$f(x_1, x_2) = x_1 * x_2$$

where variables x_1 and x_2 takes values in $[0, 1]$.

From the global perspective both variables are equally important.

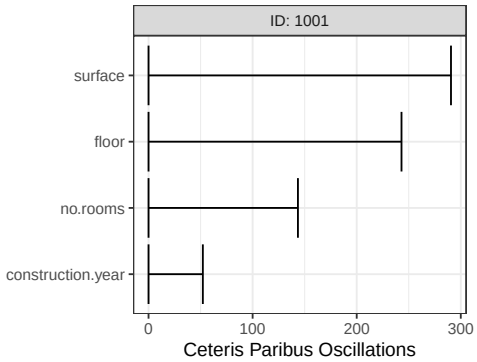


Figure 8.6: (fig:CPVIP1) Variable importance plots calculated for Ceteris Paribus profiles for observation ID: 1001

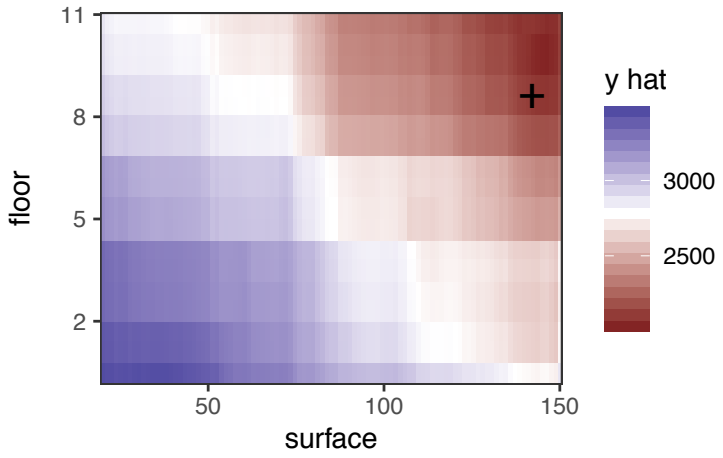


Figure 8.7: (fig:CP2Dsurflor) Ceteris Paribus plot for a pair of variales. Black cross marks coordinated for the observation of interest. Presented model estimates price of an appartment

But local variable importance is very different. Around point $x = (0, 1)$ the importance of x_1 is much larger than x_2. This is because profile for $f(z, 1)$ have larger oscillations than $f(0, z)$.

8.4 2D profiles

The definition of ceteris paribus profiles given in section **??** may be easily extended to two and more variables. Also definition of CP oscillations **??** have straight forward generalization for larger number of dimensions. Such generalisations are usefull when model is non additive. Presence of pairwise interactions may be detected with 2D Ceteris Paribus plots.

Let's define two-dimensional Ceteris Paribus Profile for model f, variables j and k and point x as

$$CP^{f,(j,k),x}(z_1, z_2) := f(x|^{(j,k)} = (z_1, z_2)).$$

I.e. CP profile is a model response obtained for observations created based on x with j and k coordinates changed to (z_1, z_2) and all other coordinates kept unchanged.

A natural way to visualise 2D CP profiles is to use a level plot as in Figure **??**.

If number of variables is small or moderate thein it is possible to present all pairs of variables. See an example in Figure **??**.

8.5 Local model fidelity

Ceteris Paribus profiles are also a useful tool to validate local model fidelity. It may happen that global performance of the model is good, while for some points the local fit is very bad. Local fidelity helps to understand how good is the model fit around point of interest.

How does it work?

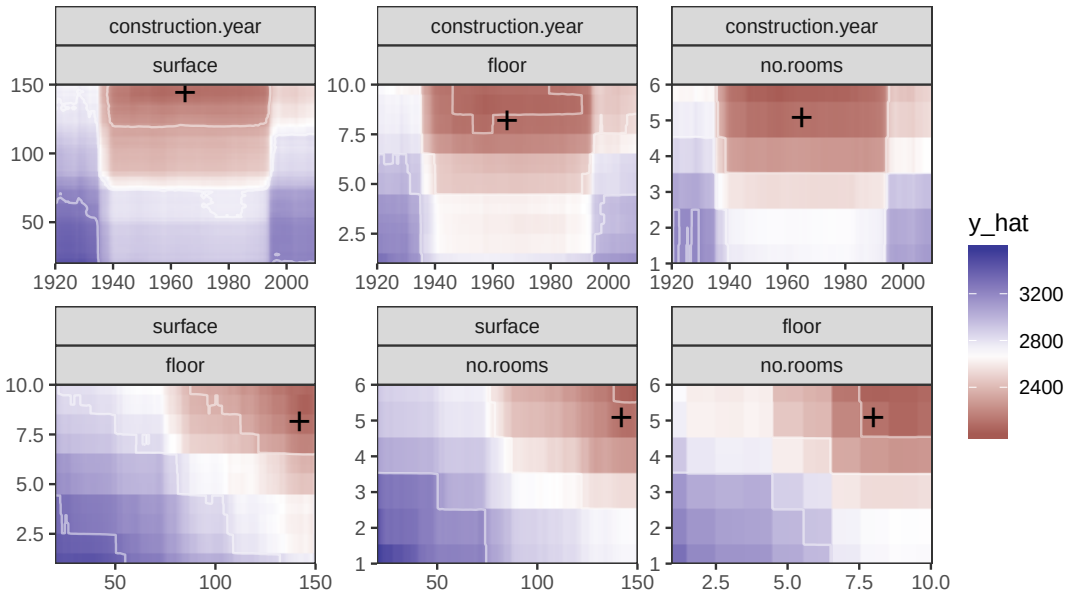


Figure 8.8: (fig:CP2Dall) Ceteris Paribus plot for all pairs of variales.

The idea behind fidelity plots is to select some number of points from the validation dataset that are closes to the point of interest. It's a similar approach as in k nearest neighbours. Then for these neighbours we may plot Ceteris Paribus Profiles and check how stable they are.

Also, if we know true taget values for points from the validation dataset we may plot residuals to show how large are residuals.

An example fidelity plot is presented in Figure **??**. Black line shows the CP profiles for the point of interest, while grey lines show CP profiles for neihgbors. Red intervals stand for residuals and in this example it looks like residuals for neighbours are all negative. Thus maybe model is biased around the point of interest.

This observation may be confirmed by plots that compare distribution of all residuals against distribution of residuals for neighbors.

See Figure **??** for an example. Here residuals for neighbors are shifted towards highest values. This suggests that the model response is biased around the observation of interest.

8.6 Pros and cons

Ceteris Paribus principle gives a uniform and extendable approach to model exploration. Below we summarize key strengths and weaknesses of this approach.

Pros

- Graphical representation of Ceteris Paribus profile is easy to understand.
- Ceteris Paribus profiles are compact and it is easy to fit many models or many variables in a small space.
- Ceteris Paribus profiles helps to understand how model response would change and how stable it is
- Oscillations calculated for CP profiles helps to select the most important variables.
- 2D Ceteris Paribus profiles help to identify pairwise interactions between variables.

Cons

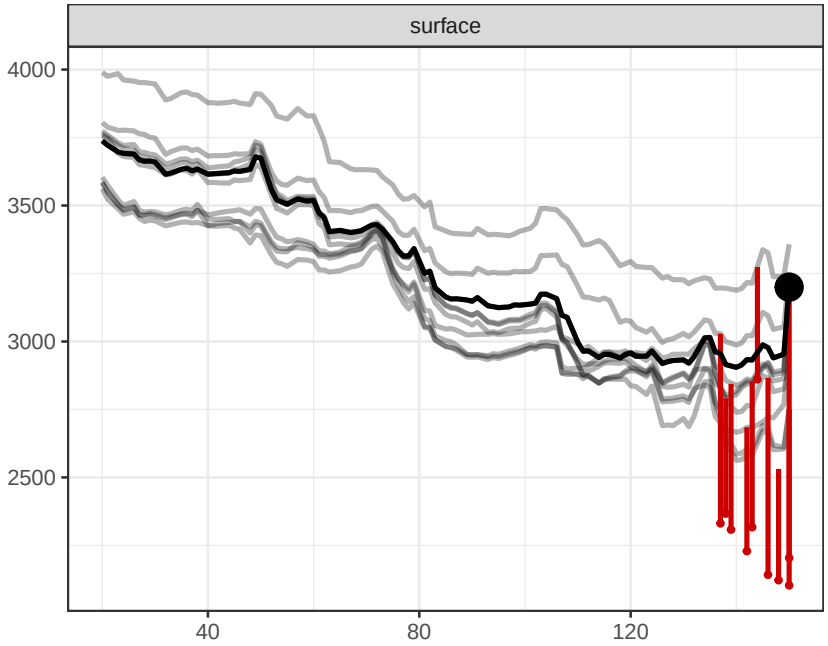


Figure 8.9: (fig:CPfidelity1) Local fidelity plots. Black line shows the CP profile for the point of interest. Grey lines show CP profiles for nearest neighbors. Red intervals correspond to residuals. Each red interval starts in a model prediction for a selected neighbor and ends in its true value of target variable.

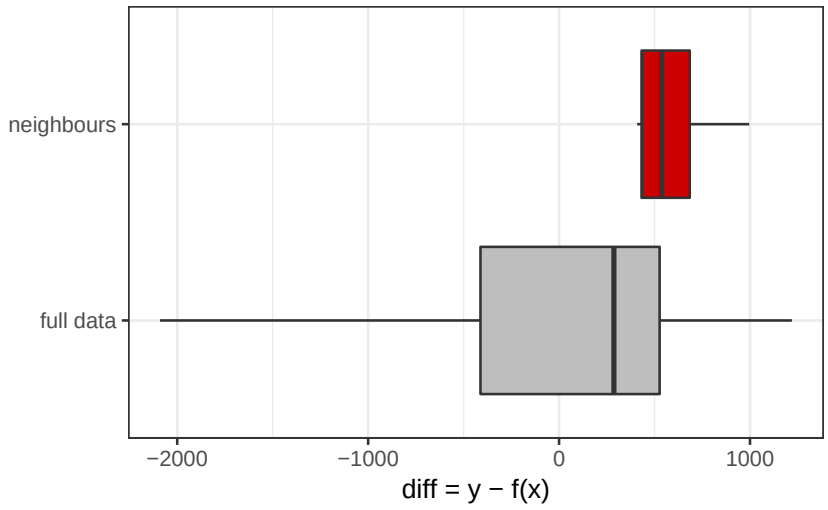


Figure 8.10: (fig:CPfidelityBoxplot) Distribution of residuals for whole validation data (grey boxplot) and for selected closes 15 neighbors (red boxplot).

- If variables are correlated (like surface and number of rooms) then the '*everything else kept unchanged*' approach leads to unrealistic settings.
- Interactions between variables are not visible in 1D plots.
- This tool is not suited for very wide data, like hundreds or thousands of variables.
- Visualization of categorical variables is non trivial.

8.7 Code snippets for R

In this section we present key features of the `ceterisParibus` package for R (**?**). This package covers all features presented in this chapter. It is available on CRAN and GitHub. Find more examples at the website of this package `https://pbiecek.github.io/ceterisParibus/`.

Model preparation

In this section we will present examples based on the `apartments` dataset. See section TODO for more details.

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

```
##   m2.price construction.year surface floor no.rooms    district
## 1     5897              1953      25     3        1 Srodmiescie
## 2     1818              1992     143     9        5     Bielany
## 3     3643              1937      56     1        2       Praga
## 4     3517              1995      93     7        3      Ochota
## 5     3013              1992     144     6        5     Mokotow
## 6     5795              1926      61     6        2 Srodmiescie
```

The problem here is to predict average price for square meter for an apartment. Let's build a random forest model with `randomForest` package (**?**).

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

```
##
## Call:
##  randomForest(formula = m2.price ~ construction.year + surface +     floor + no.rooms, data = apartments)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 1
##
##           Mean of squared residuals: 485190.1
##                     % Var explained: 40.92
```

Model exploration with `ceterisParibus` package is performed in four steps.

1. Create an explainer - wrapper around model and validation data.

Since all other functions work in a model agnostic fashion, first we need to define a wrapper around the model. Here we are using the `explain()` function from `DALEX` package (**?**).

```
library("DALEX")
explainer_rf <- explain(rf_model,
      data = apartmentsTest, y = apartmentsTest$m2.price)
explainer_rf
```

```
## Model label:  randomForest
## Model class:  randomForest.formula,randomForest
## Data head  :
##      m2.price construction.year surface floor no.rooms    district
## 1001     4644              1976     131     3        5 Srodmiescie
## 1002     3082              1978     112     9        4     Mokotow
```

2. Define point of interest.

Certeris Paribus profiles explore model around a single point.

```
new_apartment <- data.frame(construction.year = 1965, no.rooms = 5, surface = 142, floor = 8)
new_apartment
```

```
##   construction.year no.rooms surface floor
## 1              1965        5     142     8
```

```
predict(rf_model, new_apartment)
```

```
##        1
## 2315.707
```

3. Calculate CP profiles

The `ceteris_paribus()` function calculates CP profiles for selected model around selected observation.

By default CP profiles are calculated for all numerical variables. Use the `variables` argument to select subset of interesting variables. The result from `ceteris_paribus()`function is a data frame with model predictions for modified points around the point of interest.

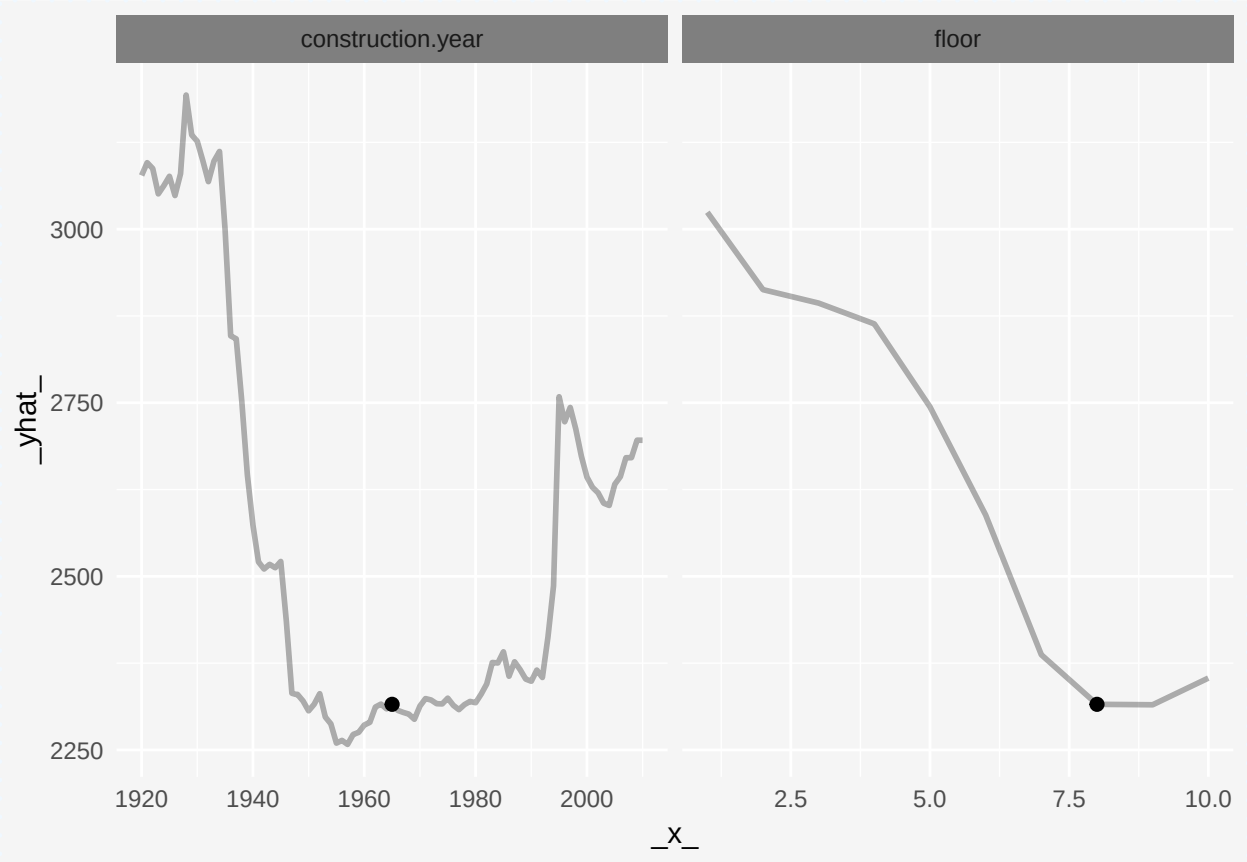
```
library("ceterisParibus")
cp_rf <- ceteris_paribus(explainer_rf, new_apartment,
                         variables = c("construction.year", "floor"))
cp_rf
```

```
## Top profiles    :
##     construction.year no.rooms surface floor   _yhat_           _vname_
## 1                1920        5     142     8 3077.322 construction.year
## 1.1              1921        5     142     8 3095.745 construction.year
## 1.2              1922        5     142     8 3087.349 construction.year
## 1.3              1923        5     142     8 3050.807 construction.year
## 1.4              1923        5     142     8 3050.807 construction.year
## 1.5              1924        5     142     8 3062.715 construction.year
##     _ids_      _label_
## 1       1 randomForest
## 1.1     1 randomForest
## 1.2     1 randomForest
## 1.3     1 randomForest
## 1.4     1 randomForest
## 1.5     1 randomForest
##
##
## Top observations:
##   construction.year no.rooms surface floor   _yhat_      _label_
## 1              1965        5     142     8 2315.707 randomForest
```

4. Plot CP profiles.

Generic `plot()` function plot CP profiles. It returns a `ggplot2` object that can be polished if needed. Use additional arguments of this function to select colors and sizes for elements visible in the plot.

```
plot(cp_rf)
```



One of very useful features of `ceterisParibus` explainers is that profiles for two or more models may be superimposed in a single plot. This helps in model comparisons.

Let's create a linear model for this dataset and repeat steps 1-3 for the lm model.

```
lm_model <- lm(m2.price ~ construction.year + surface + floor +
      no.rooms, data = apartments)
explainer_lm <- explain(lm_model,
      data = apartmentsTest, y = apartmentsTest$m2.price)
cp_lm <- ceteris_paribus(explainer_lm, new_apartment,
                         variables = c("construction.year", "floor"))
```

Now we can use function `plot()` to compare both models in a single chart. Additional argument `color = "_label_"` set color as a key for model.

```
plot(cp_rf, cp_lm, color = "_label_")
```

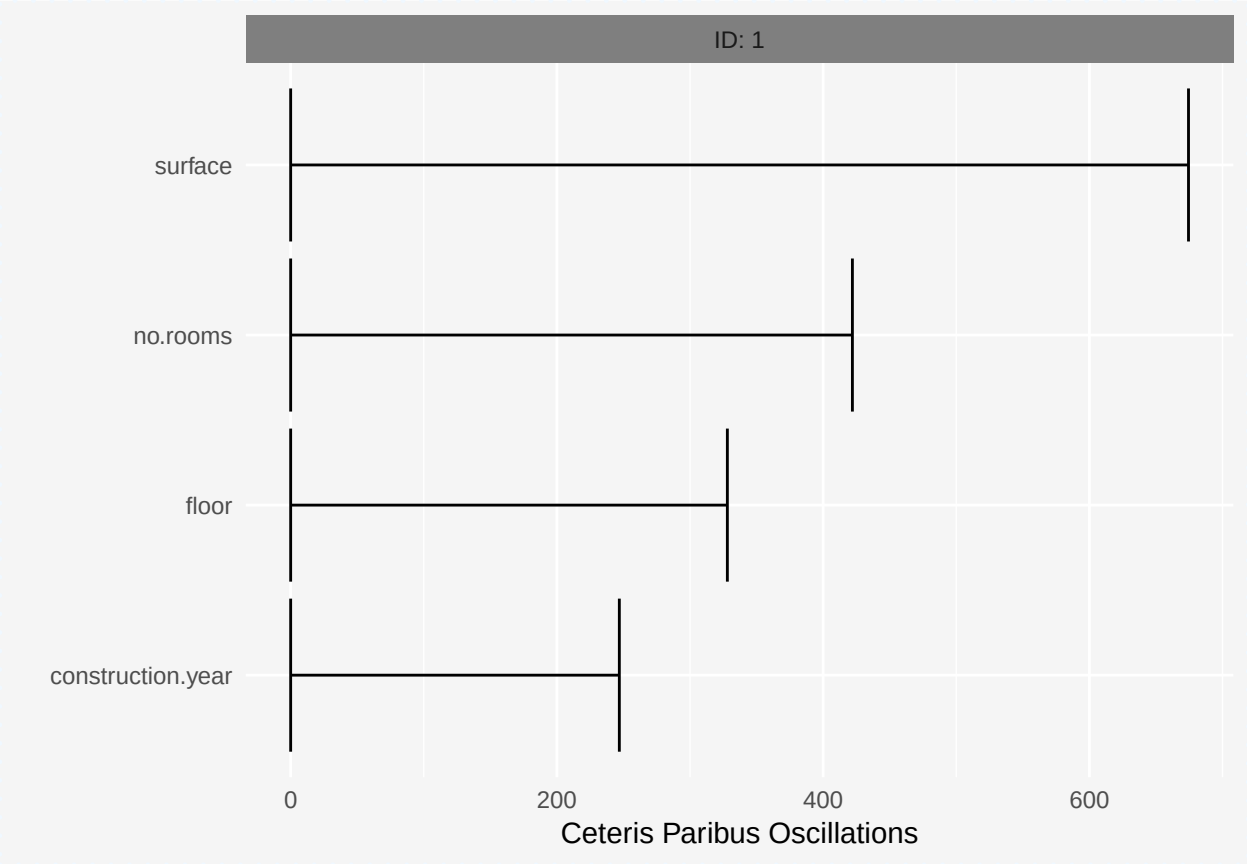
Oscillations

The `calculate_oscillations()` function calculates oscillations for CP profiles.

```
cp_rf_all <- ceteris_paribus(explainer_rf, new_apartment)
co_rf_all <- calculate_oscillations(cp_rf_all)
co_rf_all
```

```
##              _vname_ _ids_ oscillations
## 2            surface     1     674.4487
## 4          no.rooms     1     421.9685
## 3              floor     1     328.0328
## 1 construction.year     1     246.8978
```

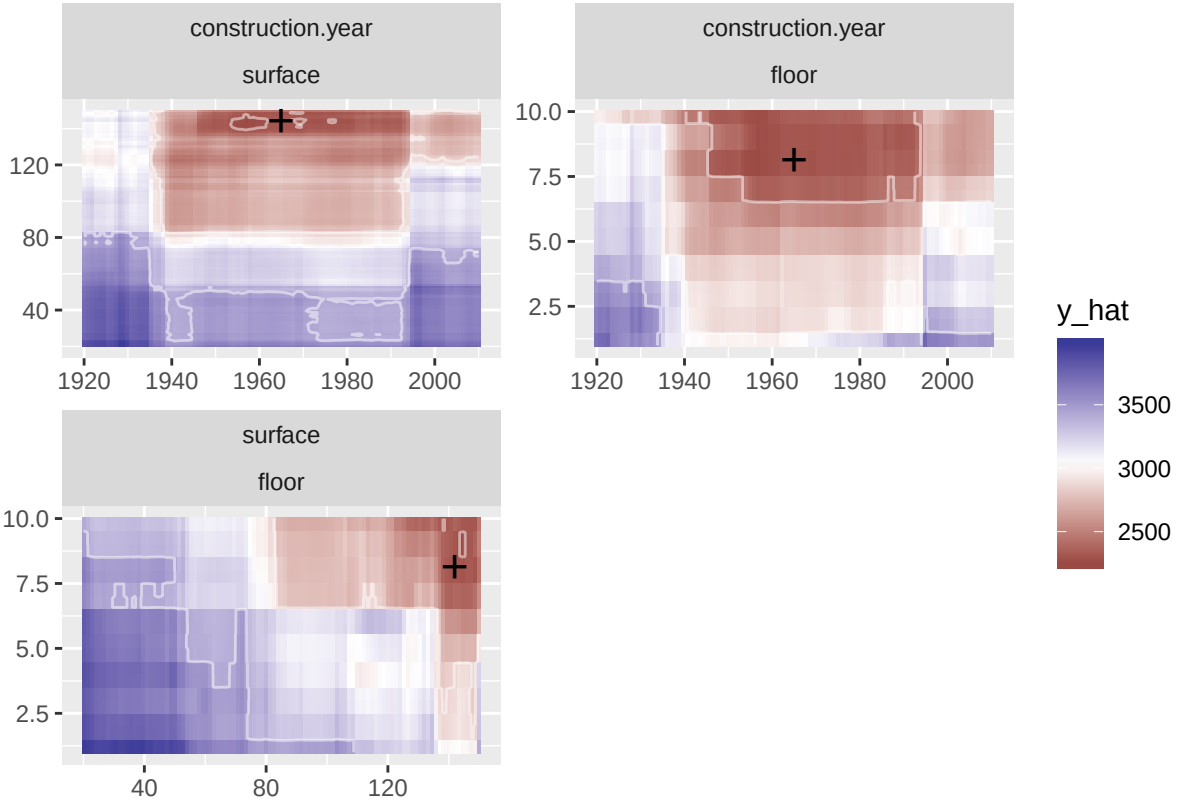
```
plot(co_rf_all)
```



2D Ceteris Paribus profiles

And the `what_if_2d()` function calculates 2D CP profiles.

```
wi_rf_2d <- what_if_2d(explainer_rf, observation = new_apartment,
                       selected_variables = c("surface","floor", "construction.year"))
plot(wi_rf_2d, split_ncol = 2)
```


construction.year
surface
120
80
40
1920
1940
1960
1980
2000
construction.year
floor
10.0
7.5
5.0
2.5
1920
1940
1960
1980
2000
surface
floor
10.0
7.5
5.0
2.5
40
80
120
y_hat
3500
3000
2500

Model level explanations

Chapter 9

Introduction

9.1 A bit of philosophy

Rights:

1. audit model residuals
2. variable importance
3. partial dependency plot

9.2 Example: Price prediction

(?)

(?)

(?)

```
library("factorMerger")
```

In this chapter we show examples for three predictive models trained on `apartments` dataset from the `DALEX` package. Random Forest model (elastic but biased), Support Vector Machines model (large variance on boundaries) and Linear Model (stable but not very elastic). Presented examples are for regression (prediction of square meter price), but the CP profiles may be used in the same way for classification.

```
library("DALEX")
# Linear model trained on apartments data
model_lm <- lm(m2.price ~ construction.year + surface + floor +
                      no.rooms + district, data = apartments)

library("randomForest")
set.seed(59)
# Random Forest model trained on apartments data
model_rf <- randomForest(m2.price ~ construction.year + surface + floor +
                      no.rooms + district, data = apartments)

library("e1071")
# Support Vector Machinesr model trained on apartments data
model_svm <- svm(m2.price ~ construction.year + surface + floor +
                         no.rooms + district, data = apartments)
```

For these models we use `DALEX` explainers created with `explain()` function. There exapliners wrap models, predict functions and validation data.

```
explainer_lm <- explain(model_lm,
                        data = apartmentsTest[,2:6], y = apartmentsTest$m2.price)
explainer_rf <- explain(model_rf,
                        data = apartmentsTest[,2:6], y = apartmentsTest$m2.price)
explainer_svm <- explain(model_svm,
                        data = apartmentsTest[,2:6], y = apartmentsTest$m2.price)
```

Examples presented in this chapter are generated with the `ceterisParibus` package in version 0.3.1.

```
library("ceterisParibus")
```

Chapter 10

Variable Importance

Feature selection

(**?**) (**?**) - variable importance

(**?**)

Beware Default Random Forest Importances

Terence Parr, Kerem Turgutlu, Christopher Csiszar, and Jeremy Howard March 26, 2018.

http://explained.ai/rf-importance/index.html

Chapter 11

Marginal Response

Feature extraction

11.1 Partial Dependency Plots

Accumulated Local Effects (ALE) Plots

(?) (?)

```
library(ALEPlot)
```

(?)

Interactions - extraction

11.2 Merging Path Plots

(?)

```
library(factorMerger)
```

Chapter 12

Performance Diagnostic

Model selection

Chapter 13

Residual Diagnostic

Model validation

(**?**)

```
library(auditor)
```

Chapter 14

Other topics

(?) (?) (?)

```
library(randomForestExplainer)
library(ICEbox)
library(ALEPlot)
```

(?)

```
library(modelDown)
```

Appendixes

Chapter 15

Data Sets

15.1 Hire or Fire? HR in Call Center

In this chapter we present an artificial dataset from Human Resources department in a Call Center to present pros and cons for different techniques of prediction level explainers.

The dataset is available in the `DALEX` package (**?**). Each row corresponds to a single employee of a call center. Features like gender, age, average number of working hours per week, grade from the last evaluation and level of salary are used as predictive features.

The problem here is to first build a model, that will determine when to fires and when to promote an employer, so it's a classification problem with three classes. But having a model we will use prediction level explainers to better understand how the model works for selected cases.

```
library("DALEX")
head(HR)
```

```
##   gender      age    hours evaluation salary   status
## 1   male 32.58267 41.88626          3      1    fired
## 2 female 41.21104 36.34339          2      5    fired
## 3   male 37.70516 36.81718          3      0    fired
## 4 female 30.06051 38.96032          3      2    fired
## 5   male 21.10283 62.15464          5      3 promoted
## 6   male 40.11812 69.53973          2      0    fired
```

In this book we are focused on model exploration rather than model building, thus for sake ok simplicity we will use two default models created with random forest (**?**) and generalized linear model (**?**).

```
set.seed(59)
library("randomForest")
model_rf <- randomForest(status ~ gender + age + hours + evaluation + salary, data = HR)

library("nnet")
model_glm <- multinom(status ~ gender + age + hours + evaluation + salary, data = HR)
```

```
## # weights:  21 (12 variable)
## initial  value 8620.810629
## iter  10 value 7002.127738
## iter  20 value 6239.478146
## iter  20 value 6239.478126
## iter  20 value 6239.478124
```

```
## final  value 6239.478124
## converged
```

15.2 How much does it cost? Price prediction for a square meter

In this chapter we present an artificial dataset related to prediction of prices for appartments in Warsaw. This dataset wil be used to discuss pros and cons for different techniques of model level explainers.

The dataset is available in the `DALEX` package (**?**). Each row corresponds to a single apartment. Features like surface, number of rooms, district or floor are used as predictive features.

The problem here is to predict price of a square meter for an appartment, so it's a regression problem with continouse outcome.

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

```
##   m2.price construction.year surface floor no.rooms    district
## 1     5897              1953      25     3        1 Srodmiescie
## 2     1818              1992     143     9        5     Bielany
## 3     3643              1937      56     1        2       Praga
## 4     3517              1995      93     7        3      Ochota
## 5     3013              1992     144     6        5     Mokotow
## 6     5795              1926      61     6        2 Srodmiescie
```

The problem here is to predict average price for square meter for an apartment. Let's build a random forest model with `randomForest` package (**?**).

```
library("randomForest")
model_rf <- randomForest(m2.price ~ construction.year + surface + floor + no.rooms + district, data = ap
model_rf
```

```
##
## 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: 82079.37
##                     % Var explained: 90.01
```

And a linear model.

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

```
##
## Call:
## lm(formula = m2.price ~ construction.year + surface + floor +
##     no.rooms + district, data = apartments)
##
## Coefficients:
##         (Intercept)    construction.year              surface
##            5020.139              -0.229             -10.238
##               floor             no.rooms      districtBielany
##             -99.482             -37.730              17.214
```

```
##     districtMokotow       districtOchota        districtPraga
##             918.380             926.254             -37.105
## districtSrodmiescie        districtUrsus      districtUrsynow
##            2080.611              29.942             -18.865
##        districtWola     districtZoliborz
##             -16.891             889.973
```