Predictive Models: Visual Exploration, Explanation and Debugging

With examples in R and Python

Przemyslaw Biecek and Tomasz Burzykowski

2019-05-15

Contents

List of Tables

List of Figures

0.1 Introduction

0.1.1 The aim of the book

Predictive models are used to guess (statisticians would say: predict) values of a variable of interest based on other variables. As an example, consider prediction of sales based on historical data, prediction of risk of heart disease based on patient's characteristics, or prediction of political attitudes based on Facebook comments.

Predictive models have been constructed through the whole human history. Ancient Egyptians, for instance, used observations of rising of Sirius to predict flooding of the Nile. A more rigorous approach to model construction may be attributed to the method of least squares, published more than two centuries ago by Legendre in 1805 and by Gauss in 1809. With time, the number of applications in economy, medicine, biology,and agriculture was growing. The term *regression* was coined by Francis Galton in 1886. Initially, it was referring to biological applications, while today it is used for various models that allow prediction of continuous variables. Prediction of nominal variables is called *classification*, and its beginning may be attributed to works of Ronald Fisher in 1936.

During the last century, many statistical models that can be used for predictive purposes have been developed. These include linear models, generalized linear models, regression and classification trees, rule-based models, and many others. Developments in mathematical foundations of predictive models were boosted by increasing computational power of personal computers and availability of large datasets in the era of „big data" that we have entered.

With the increasing demand for predictive models, model features such as flexibility, ability to perform internally some feature engineering, and high precision of predictions are of interest. To obtain robust models, ensembles of models are used. Techniques like bagging, boosting, or model stacking combine hundreds or thousands of small models into a one super-model. Large deep neural models have over a bilion of parameters.

There is a cost of this progress. Complex models may seem to operate like „black boxes'". It may be difficult, or even impopssible, to understand how thousands of coefficients affect the model prediction. At the same time, complex models may not work as good as we would like them to do. An overview of real problems with large black-box models may be found in an excellent book of Cathy O'Neil (**?**) or in her TED Talk „*The era of blind faith*

in big data must end". There is a growing number of examples of predictive models with performance that deteriorated over time or became biased in some sense. See, for instance, the issues related to the flu epidemic predictions by the Google Flu Trends model [Lazer et al Science 2014] or the problems with cancer recommndations based on the IBM Watson model [https://www.statnews.com/2017/09/05/watson-ibm-cancer/].

Today the true bottleneck in predictive modelling is not the lack of data, nor the lack of computational power, nor the lack of flexible models. It is the lack of tools for model validation, model exploration, and explanation of model decisions. Thus, in this book, we present a collection of methods that may be used for this purpose. As development of such methods is a very active area of research and new methods become available almost on a continuous basis, we do not aim at being exhaustive. Rather, we present the mind-set, key problems, and several examples of methods that can be used in model exploration.

Lack of interpretability often leads to harmful situations. *Models are not working properly and are hard to debug.* For example, very famous Watson for Oncology was criticized by oncologists for delivering unsafe and inaccurate recommendations (**?**). *Results are biased in a systematic ways.* For example, Amazon (giant in AI) failed with system for CV screening, as it was biased against woman (**?**). Or the COMPAS algorithm for predicting recidivism discriminates against race (**?**). These are serious violations of fairness and ethical principles. *Data drift leads to the deterioration in models performance.* For example, very popular model Google Flu after two years gave worse predictions than a baseline (**?**). The number of such examples grows rapidly. A reaction for some of these these problems are new legal regulations, like the General Data Protection Regulation (**?**) and the *„Right to Explanation"*, a civic right to be given an explanation for an output of the automated algorithm (**?**). Some are already in use while new are being suggested (**?**), (**?**). It is an important topic and surprisingly, we still do not have good enough tools for verification, exploration and explanation of machine learning models.

0.1.2 A bit of philosophy: three laws of model explanation

Seventy-six years ago Isaac Asimov forumlated 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.

Today's robots, like cleaning robots, robotic pets, or autonomous cars are far from being conscious enough to be under Asimov's ethics. However, we are more and more surrounded by complex predictive models and algoritmhs used for decision making. Machine learning models are used in health care, politics, education, justice, and many other areas. The models and algorithms have far larger influence on our lives than physical robots. Yet, applications of such models are left unregulated despite examples of their potential harmfulness. See *Weapons of Math Destruction* by Cathy O'Neil (**?**) for an excellent overview of selected problems.

It's clear that some we need to control the models and algorithms that may affect us. Thus, Asimov's laws are referred to in the context of the discussion around Ethics of Artifical

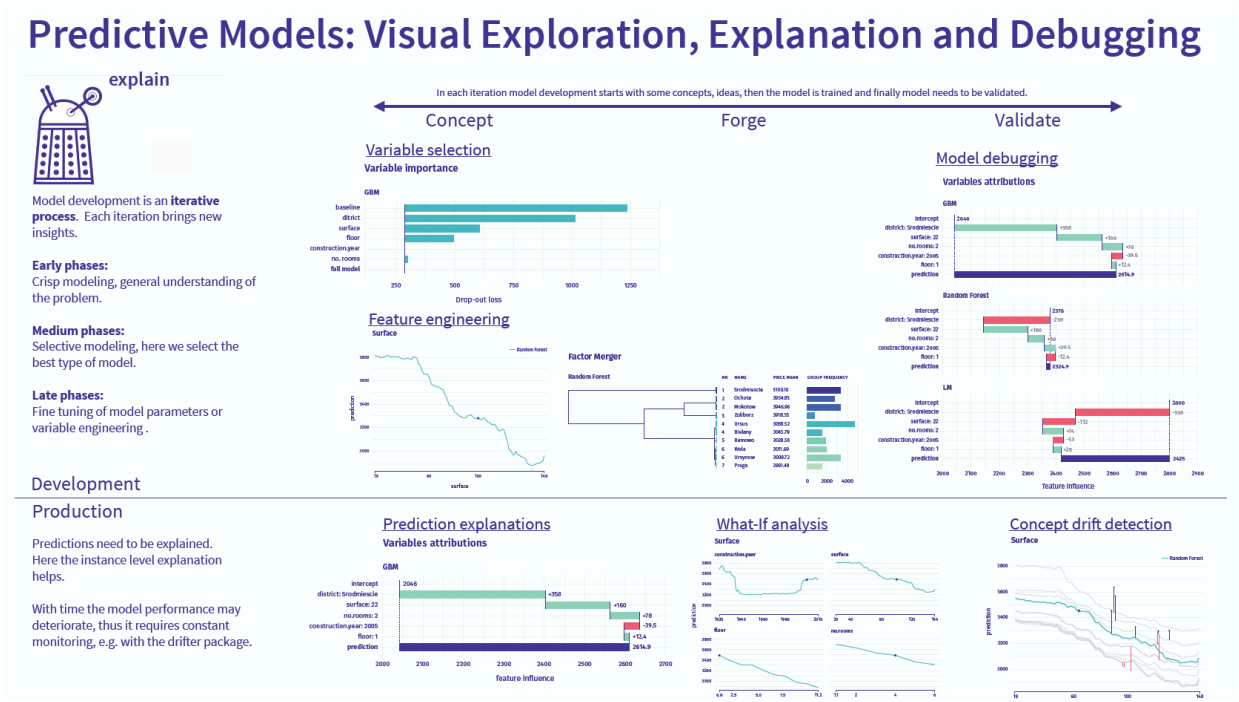


FIGURE 1 (fig:DrWhyAIPMVEE) Visual exploration of predictive models help in every phase of model life-cycle. Model level methods help in early crisp modeling. Instance level methods help in debugging. Feature effects help to cross-compare candidate models. Auditors help to identify weak sides of considered models.

Intelligence. Initiatives to formulate principles for the AI development have been undertaken, for instance, in the UK [Olhede & Wolfe, Significance 2018, 15: 6-7]. Following Asimov's approach, we could propose three requirements that any predictive model should fulfill:

- **Prediction's justification**. For every prediction of a model, one should be able to understand which variables affect the prediction and to which extent.
- **Prediction's speculation**. For every prediction of a model, one should be able to understand how the model prediction would change if input variables changed.
- **Prediction's validation** For every prediction of a model, one should be able to verify how strong is the evidence that confirms this particular prediction.

We see two ways to comply with these requirements. One is to use only models that fulfill these conditions by design. However, a reduction in performance may be the price for transparency. Another is to use tools that allow, perhaps by using approximations, to „explain" predictions for any model. In our book, we will focus on the latter.

0.1.3 Terminology

It is worth noting that, when it comes to predictive models, the same concepts have often been given different names in statistics and in machine learning. For instance, in the statistical-modelling literature, one refers to „explanatory variables," with „independent variables," „predictors," or „covariates" as often-used equivalents. Explanatory variables are used in the model as means to explain (predict) the „dependent variable," also called „predicted" variable or „response." In the machine-learning language, „input variables" or „features" are used to predict the „output" variable. In statistical modelling, models are fit to the data that contain „observations," whereas in the machine-learning world a dataset may contain „instances."

To the extent possible, in our book we try to consistently use the statistical-modelling terminology. However, the reader may expect references to a „feature" here and there. Somewhat inconsistently, we also introduce the term „instance-level" explanation. Instance-level explanation methods are designed to extract information about the behavior of the model related to a specific observation or instance. On the other hand, „global" explanation techniques allow obtaining information about the behavior of the model for an entire dataset.

We consider models for dependent variables that can be continuous or nominal. The values of a continuous variable can be represented by numbers with an ordering that makes some sense (zip codes or phone numbers are not considered as continuous variables). A continuous variable does not have to be continuous in the mathematical sense; counts (number of floors, steps, etc.) will be treated as continuous variables as well. A nominal variable can assume only a finite set of values that cannot be given numeric values.

In this book we focus on „black-box" models. We discuss them in a bit more detail in the next section.

0.1.4 White-box models vs. black-box models

Black-box models are models with a complex structure that is hard to understand by humans. Usually this refers to a large number of model coefficients. As humans may vary in their capacity of understanding complex models, there is no strict threshold for the number of coefficients that makes a model a black-box. In practice, for most humans this threshold is probably closer to 10 than to 100.

A „white-box" model, which is opposite to a „black-box" one, is a model that is easy to understand by a human (though maybe not by every human). It has got a simple structure and a limited number of coefficients. The two most common classess of white-box models are decision or regression trees (see an example in Figure **??**) or models with an additive structure, like the following model for mortality risk in melanoma patients:

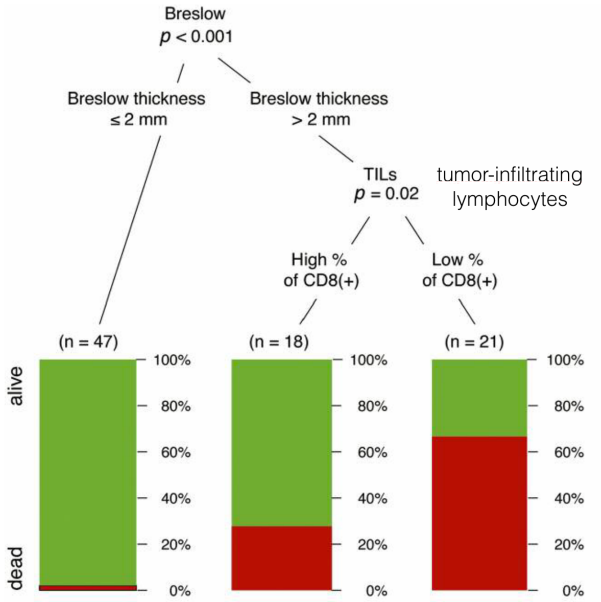
$$RelativeRisk = 1 + 3.6 * [Breslow > 2] - 2 * [TILs > 0]$$

In the model, two explanatory variables are used: an indicator whether the thickness of the lesion according to the Breslow scale is larger than 2 mm and an indicator whether the percentage of tumor-infiltrating lymphocytes (TILs) was larger than 0.

The structure of a white box-model is, in general, easy to understand. It may be difficult to collect the necessary data, build the model, fit it to the data, and/or perform model validation, but once the model has been developed its interpretation and mode of working is straightforward.

Why is it important to understand the model structure? There are several important advantages. If the model structure is clear, we can easily see which variables are included in the model and which are not. Hence, we may be able to, for instance, question the model when a particular explanatory variable was excluded from it. Also, in case of a model with a clear structure and a limited number of coefficients, we can easily link changes in model predictions with changes in particular explanatory variables. This, in turn, may allow us to challenge the model against the domain knowledge if, for instance, the effect of a particular variable on predictions is inconsistent with the previously established results. Note that linking changes in model predictions with changes in particular explanatory variables may be difficult when there are may variables and/or coefficients in the model. For instance, a classification tree with hundreds of nodes is difficult to understand, as is a linear regression model with hundreds of cofficients.

Getting the idea about the performance of a black-box model may be more challenging. The structure of a complex model like, e.g., a neural-network model, mmay be far from transparent. Consequently, we may not understand which features and how influence the model decisions. Consequently, it may be difficult to decide whether the model is consistent with the domain knowledge. In our book we present tools that can help in extracting the information necessary for the model evaluation for complex models.



0.1.5 Model visualization, exploration, and explanation

The lifecycle of a model can be divided, in general, in three different phases: development (or building), deployment, and maintenance.

Model development is the phase in which one is looking for the best available model. During this process, model exploration tools are useful. Exploration involves evaluation of the fit of the model, verification of the assumptions underlying the model (diagnostics), and assessment of the predictive performance of the model (validation). In our book we will focus on the visualization tools that can be useful in model exploration. We will not, however, discuss visualization methods for diagnostic purposes, as they are extensively discussed in many books devoted to statistical modelling.

Model deployment is the phase in which a predictive model is adopted for use. In this phase it is crucial that the users gain confidence in using the model. It is worth noting that the users might not have been involved in the model development. Moreover, they may only have got access to the software implementing the model that may not provide any insight in the details of the model structure. In this situation, model explanation tools can help to understand the factors that influence model predictions and to gain confidence in the model. The tools are one of the main focu point of our book.

Finally, a deployed model requires maintenance. In this phase, one monitors model's performance by, for instance, checking the validity of predictions for different datasets. If issues are detected, model explanation tools may be used to find the source of the problem and to suggest a modification of the structure of the model.

0.1.6 Model-agnostic vs. model-specific approach

Some classes of models have been developed for a long period of time or have attracted a lot of interest with an intensive research as a result. Consequently, those classes of models are equipped with very good tools for model exploration or visualisation. For example:

- There are many tools for diagnostics and evaluation of linear models. Model assumptions are formally defined (normality, linear structure, homogenous variance) and can be checked by using normality tests or plots (normal qq-plot), diagnostic plots, tests for model structure, tools for identification of outliers, etc.
- For many more advanced models with an additive structure, like the proportional hazards model, there also many tools that can be used for checking model assumptions.
- Random-forest model is equipped with the out-of-bag method of evaluation of performance and several tools for measuring variable importance (**?**). Methods have been developed to extract information from the model structure about possible interactions (**?**). Similar tools have been developed for other ensembles of trees, like xgboost models (**?**).
- Neural networks enjoy a large collection of dedicated model-explanation tools that use, for instance, the layer-wise relevance propagation technique (**?**), or saliency maps technique (**?**), or a mixed approach.

Of course, the list of model classes with dedicated collections of model-explanation and/or diagnostics methods is much longer. This variety of model-specific approaches does lead to issues, though. For instance, one cannot easily compare explanations for two models with different structures. Also, every time when a new architecture or a new ensemble of models is proposed, one needs to look for new methods of model exploration. Finally, for brand-new models no tools for model explanation or diagnostics may be immedaitely available.

For these reasons, in our book we focus on model-agnostic techniques. In particular, we prefer not to assume anything about the model structure, as we may be dealing with a black-box model with an unclear structure. In that case, the only operation that we may be able to perform is evaluation of a model for a selected observation.

However, while we do not assume anything about the structure of the model, we will assume that the model operates on p-dimensional vectors and, for a single vector, it returns a single value which is a real number. This assumption holds for a broad range of models for data such as tabular data, images, text data, videos, etc. It may not be suitable for, e.g., models with memory in which the model output does not depend only on the model input [TOMASZ: NOT SURE WHICH MODELS ARE MEANT HERE].

Note that the techniques considered in the book may not be sufficient to fully understand models in case p is large.

0.1.7 Code snippets

TODO: Here we should tell why we present examples for DALEX. And mention that there are also other functions that can be used.

0.1.8 The structure of the book

Our book is split in two parts. In the part *Instance-level explainers*, we present techniques for exploration and explanation of model predictions for a single observation. On the other hand, in the part *Global explainers*, we present techniques for exploration and explanation of model's performance for an entire dataset. In each part, every method is described in a separate section that has got the same structure: * Subsection *Introduction* explains the goal of and the general idea behind the method. * Subsection *The Algorithm* shows mathematical or computational details related to the method. This subsection can be skipped if you are not interested in the details. * Subsection *Example* shows an exemplary application of the method with discussion of results. * Subsection *Pros and Cons* summarizes the advantages and disadvantages of the method. It also provides some guideance regarding when to use the method. * Subsection *Code snippets* shows the implementation of the method in R and Python. This subsection can be skipped if you are not interested in the implementation.

TO DO: A SHORT REVIEW OF THE CONTENTS OF VARIOUS CHAPTERS

Finally, we would like to signal that, **in this book, we do show**

- how to determine features that affect model prediction for a single observation. In particular, we present the theory and examples of methods that can be used to explain prediction like break down plots, ceteris paribus profiles, local-model approximations, or Shapley values.
- techniques to examine fully-trained machine-learning models as a whole. In particular, we review the theory and examples of methods that can be used to explain model performance globally, like partial-dependency plots, variable-importance plots, and others.
- charts that can be used to present key information in a quick way.
- tools and methods for model comparison.
- code snippets for R and Python that explain how to use the described methods.

On the other hand, **in this book, we do not focus on**

- any specific model. The presented techniques are model agnostic and do not make any assumptions related to model structure.
- data exploration. There are very good books on this topic, like R for Data Science http://r4ds.had.co.nz/ or TODO
- the process of model building. There are also very good books on this topic, 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
- any particular tools for model building. These are discussed, for instance, in Applied Predictive Modeling By Max Kuhn and Kjell Johnson http://appliedpredictivemodeling.com/

0.1.9 Acknowledgements

Przemek's work on interpretability has started during research trips within the RENOIR project (691152 - H2020/2016-2019). So he would like to thank prof. Janusz Holyst for the chance to take part in this project.

Przemek would also like thank prof. Chris Drake for her hospitality. This book would have never been created without perfect conditions that Przemek found at Chris' house in Woodland.

This book has been prepared by using the **bookdown** package (**?**), created thanks to the amazing work of Yihui Xie.

0.2 DIY with R

In next chapters we will introduce different methods for global and instance level explanation and exploration of predictive models.

After each method there is a section with code snippets for R. Below you will find a short instruduction what do you need to replicate these results.

0.2.1 How to install dependencies

Obviously you need R (**?**). It is always better to use the newest version, but make sure that you have at least R in version 3.5. You can download R from https://cran.r-project.org/.

It is much easier to work with R if you have a good editor. There is plenty of choices, but if you just begun the journey then think about the RStudio editor - open source and enterprise ready tool for R. Download the latest version from https://www.rstudio.com/.

Once you get the R and the editor, now you need to install required packages. First which you need is the `DALEX` package, it's an entry point to solutions introduced in this book. Get this package with following line executed in the R command line.

```
install.packages("DALEX")
```

Along with `DALEX` all hard requirements will be installed, like `ggplot2` package for data visualization. To use all examples in this book you will need two additional packages `ingredients` and `iBreakDown`. The easiest way to get them and other useful weak dependencies is to call following line.

```
DALEX::install_dependencies()
```

Now you have installed everything that is needed to follow examples presented in this book.

0.2.2 How to work with DALEX

To do model exploration with DALEX you need to first create a model and then prepare the model for exploration.

There is lots of different packages in R that can be used to train a model. There are some structure specific packages, like randomForest for Random Forest Classification and Regression models (?), gbm for Generalized Boosted Regression Models (?), extensions for Generalized Linear Models (?) or many others. There is also a number of packages that works like model factories, can be used for training models of different structures. Most known are h2o which is a connector to H2O (?), caret (?) and its successor parsnip (?), very powerful and extensible mlr (?) or keras which is a wrapper to python library with the same name (?).

It's greater to have such large choice when it comes to modeling; unfortunately these packages have different interfaces and different parameters. Model-objects created with different libraries may have different internal structures. The main goal of the DALEX package (?) is to create a level of abstraction around a model that makes it easier to explore and explain the model.

Function DALEX::explain is THE function for model wrapping. The function requires five arguments:

- model, a model-object,
- data, a data frame with validation data,
- y, observed values of the dependent variable for the validation data, it an optional argument, required for explainers focused on model validation and benchmarking,
- predict_function, a function that returns prediction scores; if not specified, then a default predict() function is used. Note that for some models the default predict() function returns classes, in such cases you should provide a function that will return numerical scores.
- label, a name of a model; if not specified, then it is extracted from the class(model). This name will be presented in Figures so it is better to make it informative.

For example, let's train a random forest model for binary classification on titanic dataset.

```
library("DALEX")
titanic <- na.omit(titanic)

library("randomForest")
titanic_rf <- randomForest(survived ~ class + gender + age, data = titanic)
```

Here we create a wrapper for this model. We specified directly validation dataset (in the example below we use the training data as the validation dataset), the predict function and the label.

```
explain_titanic_rf <- explain(model = titanic_rf,
            data = titanic[, c("class", "age", "gender")],
            y = titanic$survived == "yes",
            predict_function = function(m, x) predict(m, x, type = "prob"),
            label = "Random Forest 3 vars")
```

For most popular models there are predefined `predict_function`'s so we can make the call shorter.

```
explain_titanic_rf <- explain(model = titanic_rf,
            data = titanic[, c("class", "age", "gender")])
```

You will find more examples in the **??** section.

0.2.3 How to work with `archivist`

In next chapters we will be focused on exploration of predictive models. We do not want to waste space nor time to replicate code for model development. This is where `archivist` will help.

The `archivist` package (**?**) is design to store, share and manage of R objects. We will use it to easily access R models and explainers.

To install the package just copy following line to the R command line.

```
install.pacakges("archivist")
```

Then you can use `aread()` function to get R objects from any remote repository.

In this book we use a GitHub repository `models` hosted at https://github.com/pbiecek/models. To download a model with md5 hash `ceb40` one can call following line.

```
archivist::aread("pbiecek/models/ceb40")
```

It returns a random forest model. Since md5 hash `ceb40` defines model uniquely, you will get exactly the same model and same explanations.

In next chapters precalculated model explainers will be accessed with `archivist` hooks.

0.3 DIY with Python

0.4 Data Sets

We illustrate the techniques presented in this book by using three datasets:

- *Sinking of the RMS Titanic*
- *Apartment prices*
- *Hire or Fire*

The first dataset will be used to illustrate the application of the techniques in the case of a predictive model for a binary dependent variable. The second one will provide an example

FIGURE 2 Titanic sinking by Willy Stöwer

for models for a continuous variable. Finally, the third dataset will be used for illustration of models for a categorical dependent variable.

In this chapter, we provide a short description of each of the datasets, together with results of exploratory analyses. We also introduce models that will be used for illustration purposes in subsequent chapters.

0.4.1 Sinking of the RMS Titanic

Sinking of the RMS Titanic is one of the deadliest maritime disasters in history (during peacetime). Over 1500 people died as a consequence of collision with an iceberg. Projects like *Encyclopedia titanica* `https://www.encyclopedia-titanica.org/` are a source of rich and precise data about Titanic's passengers. The data are available in a dataset included in the `stablelearner` package. The dataset, after some data cleaning and variable transformations, is also avaliable in the `DALEX` package. In particular, the 'titanic' data frame contains 2207 observations (for 1317 passengers and 890 crew members) and nine variables:

- *gender*, person's (passenger's or crew member's) gender, a factor (categorical variable) with two levels (categories)
- *age*, person's age in years, a numerical variable; for adults, the age is given in (integer)

years; for children younger than one year, the age is given as $x/12$, where x is the number of months of child's age
- *class*, the class in which the passenger travelled, or the duty class of a crew member; a factor with seven levels
- *embarked*, the harbor in which the person embarked on the ship, a factor with four levels
- *country*, person's home country, a factor with 48 levels
- *fare*, the price of the ticket (only available for passengers; 0 for crew members), a numerical variable
- *sibsp*, the number of siblings/spouses aboard the ship, a numerical variable
- *parch*, the number of parents/children aboard the ship, a numerical variable
- *survived*, a factor with two levels indicating whether the person survived or not

Models considered for this dataset will use *survived* as the (binary) dependent variable.

```
library("DALEX")
head(titanic, 2)
```

```
##   gender age class    embarked       country  fare sibsp parch survived
## 1   male  42   3rd Southampton United States  7.11     0     0       no
## 2   male  13   3rd Southampton United States 20.05     0     2       no
```

```
str(titanic)
```

```
## 'data.frame':    2207 obs. of  9 variables:
##  $ gender  : Factor w/ 2 levels "female","male": 2 2 2 1 1 2 2 1 2 2 ...
##  $ age     : num  42 13 16 39 16 25 30 28 27 20 ...
##  $ class   : Factor w/ 7 levels "1st","2nd","3rd",..: 3 3 3 3 3 3 2 2 3 3 ...
##  $ embarked: Factor w/ 4 levels "Belfast","Cherbourg",..: 4 4 4 4 4 4 2 2 2 4 ...
##  $ country : Factor w/ 48 levels "Argentina","Australia",..: 44 44 44 15 30 44 17 17 26 16 ...
##  $ fare    : num  7.11 20.05 20.05 20.05 7.13 ...
##  $ sibsp   : num  0 0 1 1 0 0 1 1 0 0 ...
##  $ parch   : num  0 2 1 1 0 0 0 0 0 0 ...
##  $ survived: Factor w/ 2 levels "no","yes": 1 1 1 2 2 2 1 2 2 2 ...
```

```
levels(titanic$class)
```

```
## [1] "1st"              "2nd"              "3rd"
## [4] "deck crew"        "engineering crew" "restaurant staff"
## [7] "victualling crew"
```

```
levels(titanic$embarked)
```

```
## [1] "Belfast"     "Cherbourg"   "Queenstown"  "Southampton"
```

0.4.1.1 Data exploration

It is always advisable to explore data before modelling. However, as this book is focused on model exploration, we will limit the data exploration part.

Before exploring the data, we first do some pre-processing. In particular, the value of variables *age*, *country*, *sibsp*, *parch*, and *fare* is missing for a limited number of observations (2, 81, 10, 10, and 26, respectively). Analyzing data with missing values is a topic on its own (Little and Rubin 1987; Schafer 1997; Molenberghs and Kenward 2007). An often-used approach is to impute the missing values. Toward this end, multiple imputation should be considered (Schafer 1997; Molenberghs and Kenward 2007; van Buuren 2012). However, given the limited number of missing values and the intended illustrative use of the dataset, we will limit ourselves to, admittedly inferior, single imputation. In particular, we replace the missing *age* values by the mean of the observed ones, i.e., 30. Missing *country* will be coded by "X". For *sibsp* and *parch*, we replace the missing values by the most frequently observed value, i.e., 0. Finally, for *fare*, we use the mean fare for a given *class*, i.e., 0 pounds for crew, 89 pounds for the 1st, 22 pounds for the 2nd, and 13 pounds for the 3rd class.

```
# missing age is replaced by average (30)
titanic$age[is.na(titanic$age)] = 30
# missing country is replaced by "X"
titanic$country <- as.character(titanic$country)
titanic$country[is.na(titanic$country)] = "X"
titanic$country <- factor(titanic$country)
# missing fare is replaced by class average
titanic$fare[is.na(titanic$fare) & titanic$class == "1st"] = 89
titanic$fare[is.na(titanic$fare) & titanic$class == "2nd"] = 22
titanic$fare[is.na(titanic$fare) & titanic$class == "3rd"] = 13
# missing sibsp, parch are replaced by 0
titanic$sibsp[is.na(titanic$sibsp)] = 0
titanic$parch[is.na(titanic$parch)] = 0
```

After imputing the missing values, we investigate the association between survival status and the other variables. Figures @ref(titanic_exploration_gender)-@ref(fig:titanic_exploration_fare) present graphically the proportion non- and survivors for different levels of the other variables. The height of the bars (on the y-axis) reflects the marginal distribution (proportions) of the observed levels of the variable. On the other hand, the width of the bars (on the x-axis) provides the information about the proportion of non- and survivors. Note that, to construct the graphs for *age* and *fare*, we categorized the range of the observed values.

Figures **??** and **??** indicate that the proportion of survivors was larger for females and children below 5 years of age. This is most likely the result of the "women and children first" principle that is often evoked in situations that require evacuation of persons whose life is in danger. The principle can, perhaps, partially explain the trend seen in Figures **??** and **??**, i.e., a higher proportion of survivors among those with 1-3 parents/children and 1-2 siblings/spouses aboard. Figure **??** indicates that passengers travelling in the first and second class had a higher chance of survival, perhaps due to the proximity of the location of their cabins to the deck. Interestingly, the proportion of survivors among crew deck was similar to the proportion of the first-class passengers. Figure **??** shows that the proportion of survivors increased with the fare, which is consistent with the fact that the proportion

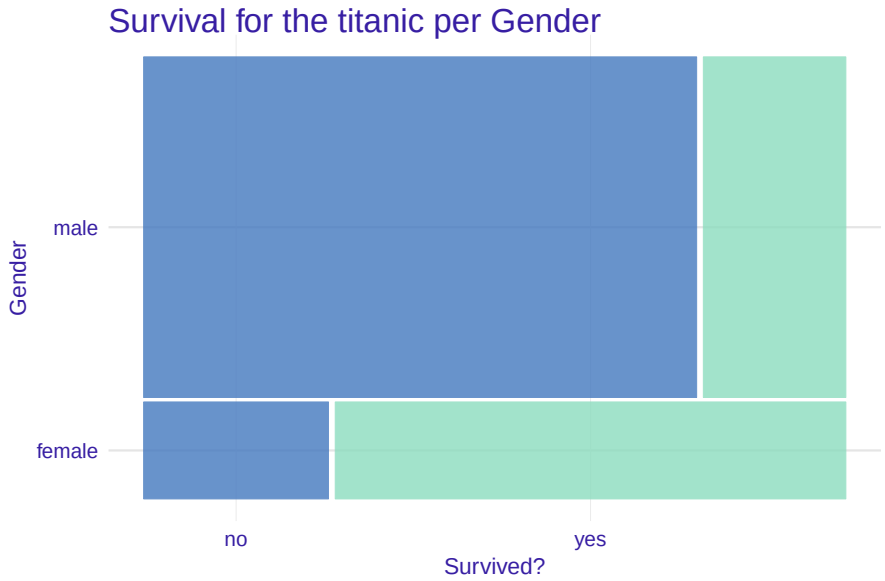


FIGURE 3 Survival in different genders for the titanic data.

was higher for passengers travelling in the first and second class. Finally, Figures **??** and **??** do not suggest any noteworthy trends.

0.4.1.2 Logistic regression

The dependent variable of interest, *survival*, is binary. Thus, a natural choice to build a predictive model is logistic regression. We do not consider country as an explanatory variable. As there is no reason to expect a linear relationship between age and odds of survival, we use linear tail-restricted cubic splines, available in the `rcs()` function of the `rms` package (**?**), to model the effect of age. We also do not expect linear relation for the `fare` variable, but because of it's skewness we have not used splines for this variable. The results of the model are stored in model-object `titanic_lmr_v6`, which will be used in subsequent chapters.

```
library("rms")
set.seed(1313)

titanic_lmr_v6 <- lrm(survived == "yes" ~ gender + rcs(age) + class + sibsp +
                   parch + fare + embarked, titanic)
titanic_lmr_v6
```

```
## Logistic Regression Model
##
##  lrm(formula = survived == "yes" ~ gender + rcs(age) + class +
##      sibsp + parch + fare + embarked, data = titanic)
##
##                         Model Likelihood     Discrimination    Rank Discrim.
##                            Ratio Test           Indexes           Indexes
```

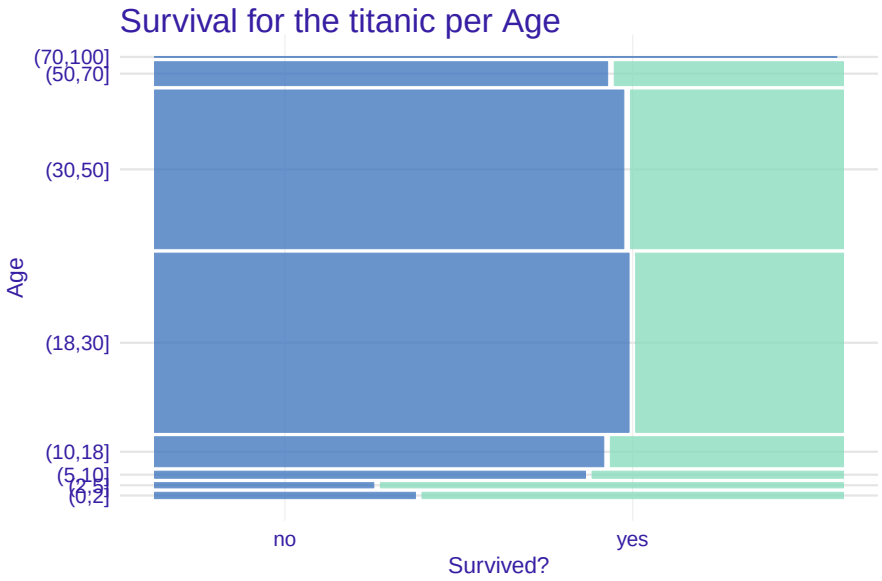


FIGURE 4 Survival in different age groups for the titanic data.

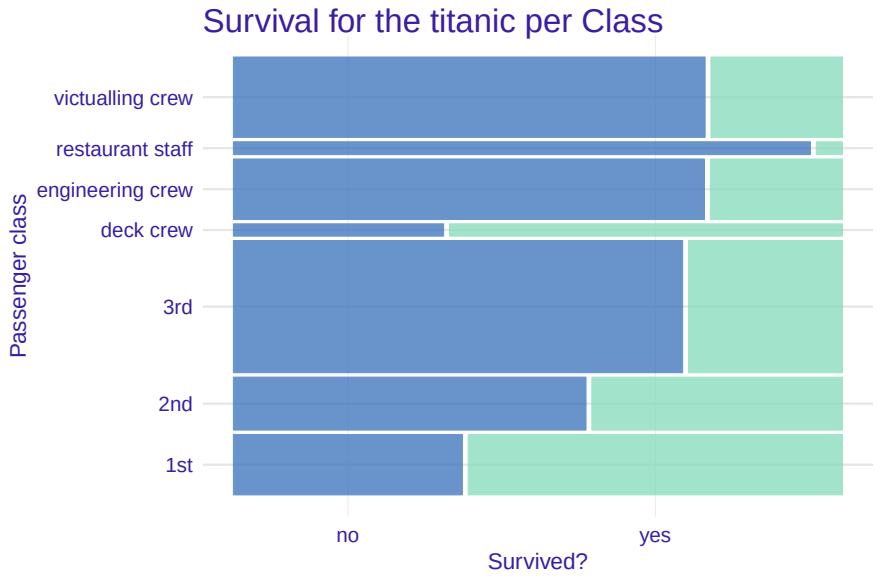


FIGURE 5 Survival for different classes in the titanic data.

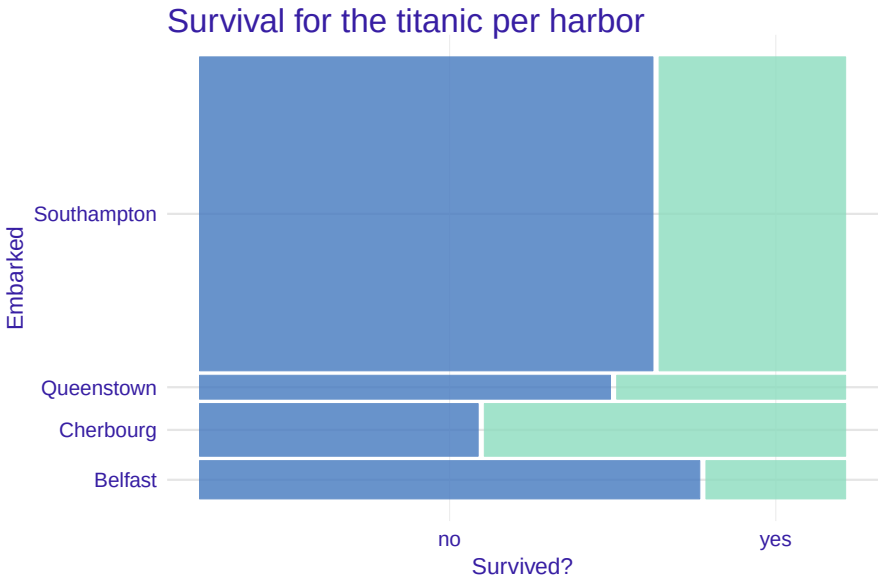


FIGURE 6 Survival for different port of embarking in the titanic data.

```
##  Obs            2207    LR chi2     752.06    R2       0.404    C       0.817
##   FALSE         1496    d.f.            17    g        1.647    Dxy     0.635
##   TRUE           711    Pr(> chi2) <0.0001    gr       5.191    gamma   0.636
##  max |deriv| 0.0001                           gp       0.282    tau-a   0.277
##                                               Brier    0.146
##
##                          Coef    S.E.   Wald Z Pr(>|Z|)
##  Intercept                4.5746 0.5480   8.35 <0.0001
##  gender=male             -2.7687 0.1586 -17.45 <0.0001
##  age                     -0.1180 0.0221  -5.35 <0.0001
##  age'                     0.6313 0.1628   3.88 0.0001
##  age''                   -2.6583 0.7840  -3.39 0.0007
##  age'''                   2.8977 1.0130   2.86 0.0042
##  class=2nd               -1.1390 0.2501  -4.56 <0.0001
##  class=3rd               -2.0627 0.2490  -8.28 <0.0001
##  class=deck crew          1.0672 0.3498   3.05 0.0023
##  class=engineering crew -0.9702 0.2648  -3.66 0.0002
##  class=restaurant staff -3.1712 0.6583  -4.82 <0.0001
##  class=victualling crew -1.0877 0.2596  -4.19 <0.0001
##  sibsp                   -0.4504 0.1006  -4.48 <0.0001
##  parch                   -0.0871 0.0987  -0.88 0.3776
##  fare                     0.0014 0.0020   0.70 0.4842
##  embarked=Cherbourg       0.7881 0.2836   2.78 0.0055
##  embarked=Queenstown      0.2745 0.3409   0.80 0.4208
##  embarked=Southampton     0.2343 0.2119   1.11 0.2689
##
```

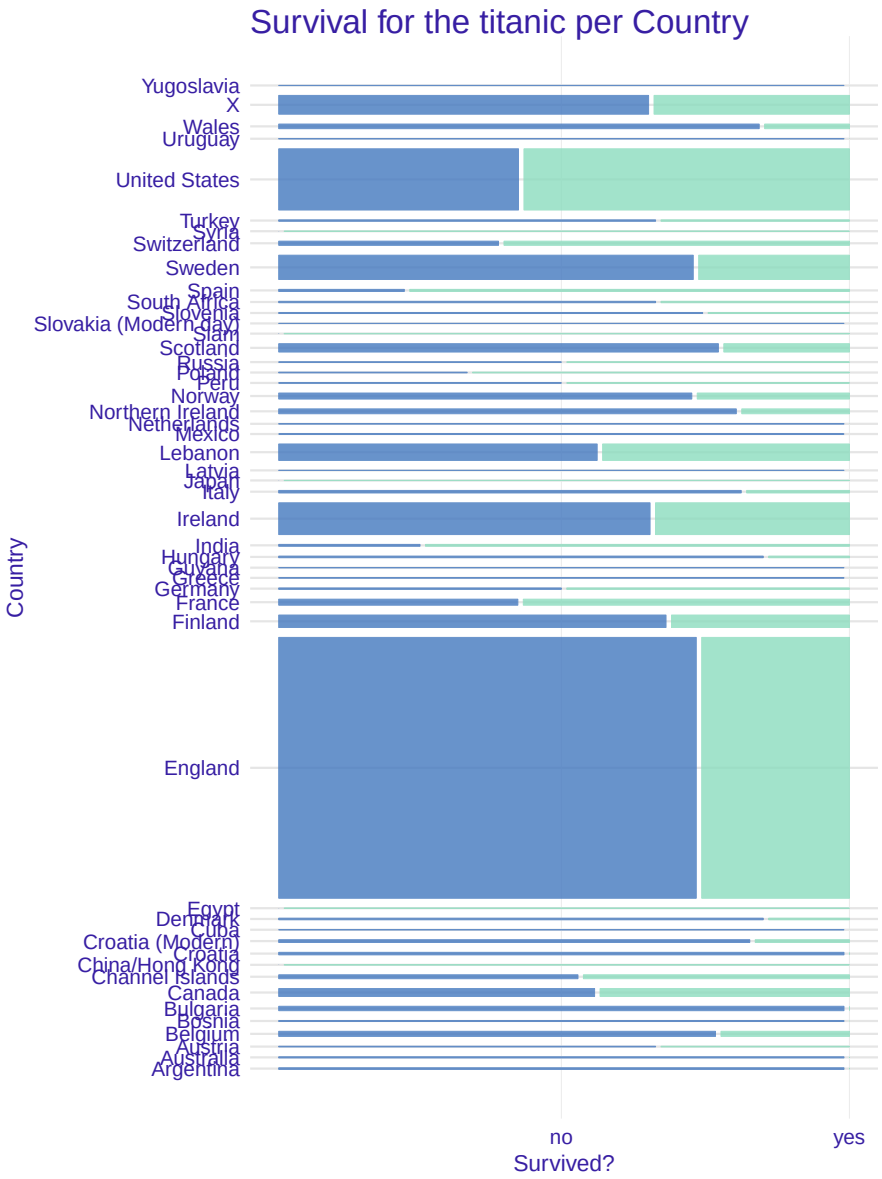


FIGURE 7 Survival for different countries in the titanic data.

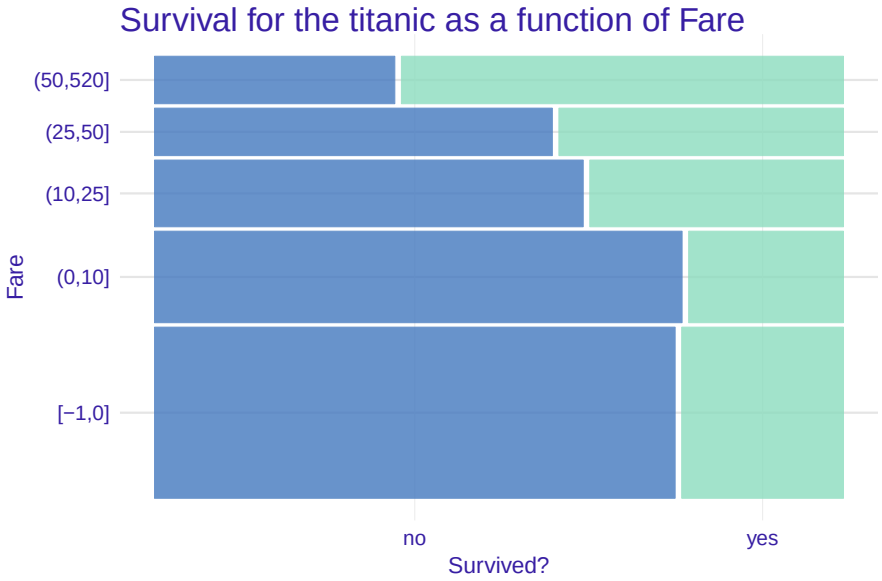


FIGURE 8 Survival as a function of fare in the titanic data.

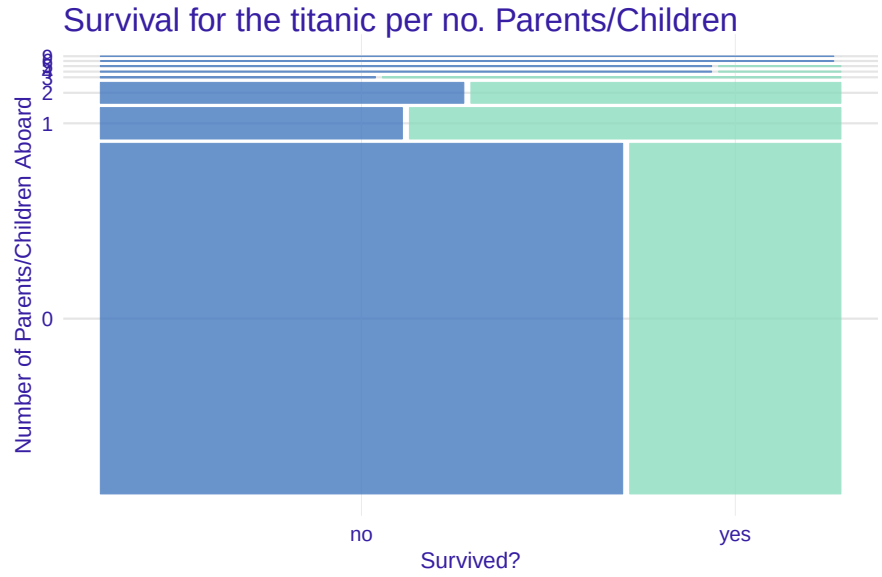


FIGURE 9 Survival in different numbers of parents/children for the titanic data.

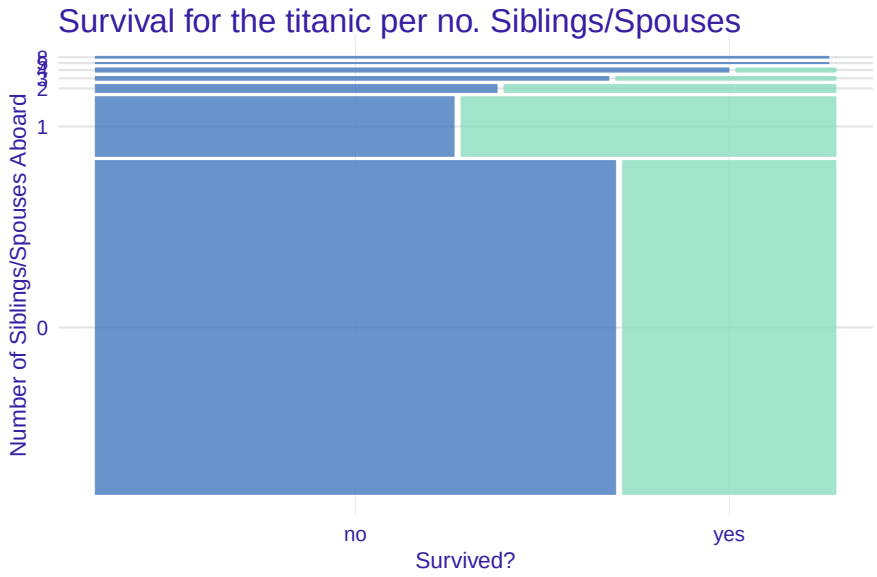


FIGURE 10 Survival in different numbers of siblings/spouses for the titanic data.

0.4.1.3 Random forest

As an alternative to a logistic regression model, we consider a random forest model. Random forest is known for good predictive performance, is able to grasp low-level variable interactions, and is quite stable (**?**). To fit the model, we apply the `randomForest()` function, with default settings, from the package with the same name (**?**).

In the first instance, we fit a model with the same set of explanatory variables as the logistic regression model. The results of the model are stored in model-object `titanic_rf_v6`.

```
library("randomForest")
set.seed(1313)

titanic_rf_v6 <- randomForest(survived ~ class + gender + age + sibsp + parch + fare + embarked,
                              data = titanic)
titanic_rf_v6
```

```
##
## Call:
##  randomForest(formula = survived ~ class + gender + age + sibsp +      parch + fare + embarked, data = titar
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 18.62%
## Confusion matrix:
##       no yes class.error
## no  1393 103  0.06885027
```

```
## yes  308 403  0.43319269
```

For comparison purposes, we also consider a model with only three explanatory variables: *class*, *gender*, and *age*. The results of the model are stored in model-object `titanic_rf_v3`.

```
titanic_rf_v3 <- randomForest(survived ~ class + gender + age, data = titanic)
titanic_rf_v3
```

```
##
## Call:
##  randomForest(formula = survived ~ class + gender + age, data = titanic)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 1
##
##         OOB estimate of  error rate: 21.02%
## Confusion matrix:
##       no yes class.error
## no  1367 129  0.08622995
## yes  335 376  0.47116737
```

0.4.1.4 Gradient boosting

Finally, we consider the gradient-boosting model. (**?**) The model is known for being able to accomodate higher-order interactions between variables. We use the same set of explanatory variables as for the logistic regression model. To fit the gradient-boosting model, we use the function `gbm()` from the `gbm` package (**?**). The results of the model are stored in model-object `titanic_gbm_v6`.

```
library("gbm")
set.seed(1313)

titanic_gbm_v6 <- gbm(survived == "yes" ~ class + gender + age + sibsp + parch + fare + embarked,
                      data = titanic, n.trees = 15000)
```

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

```
titanic_gbm_v6
```

```
## gbm(formula = survived == "yes" ~ class + gender + age + sibsp +
##     parch + fare + embarked, data = titanic, n.trees = 15000)
## A gradient boosted model with bernoulli loss function.
## 15000 iterations were performed.
## There were 7 predictors of which 7 had non-zero influence.
```

0.4.1.5 Model predictions

Let us now compare predictions that are obtained from the three different models. In particular, we will compute the predicted probability of survival for an 8-year-old boy who embarked in Belfast and travelled in the 1st class with no parents nor siblings with a ticket costing 72 pounds.

First, we create a data frame `johny_d` that contains the data describing the passenger.

```
johny_d <- data.frame(
            class = factor("1st", levels = c("1st", "2nd", "3rd", "deck crew", "engineering crew", "restaurant
            gender = factor("male", levels = c("female", "male")),
            age = 8,
            sibsp = 0,
            parch = 0,
            fare = 72,
            embarked = factor("Belfast", levels = c("Belfast","Cherbourg","Queenstown","Southampton"))
)
```

Subsequently, we use the generic function `predict()` to get the predicted probability of survival for the logistic regression model.

```
(pred_lmr <- predict(titanic_lmr_v6, johny_d, type = "fitted"))
```

```
##         1
## 0.7233413
```

The predicted probability is equal to 0.72.

We do the same for the random forest and gradient boosting models.

```
(pred_rf <- predict(titanic_rf_v6, johny_d, type = "prob"))
```

```
##      no   yes
## 1 0.626 0.374
## attr(,"class")
## [1] "matrix" "votes"
```

```
(pred_gbm <- predict(titanic_gbm_v6, johny_d, type = "response", n.trees = 15000))
```

```
## [1] 0.6700909
```

As a result, we obtain the predicted probabilities of 0.37 and 0.67, respectively.

The models lead to different probabilities. Thus, it might be of interest to understand the reason for the differences, as it could help us to decide which of the predictions we might want to trust.

Note that for some examples we will use another instance with lower chances of survival. Let's call him, Henry.

```
henry <- data.frame(
            class = factor("1st", levels = c("1st", "2nd", "3rd", "deck crew", "engineering crew", "restaurant
            gender = factor("male", levels = c("female", "male")),
            age = 47,
            sibsp = 0,
            parch = 0,
            fare = 25,
            embarked = factor("Cherbourg", levels = c("Belfast","Cherbourg","Queenstown","Southampton"))
)

(pred_gbm <- predict(titanic_gbm_v6, henry, type = "response", n.trees = 15000))
```

```
## [1] 0.4184165
```

0.4.1.6 Explainers

Model-objects created with different libraries may have different internal structures. Thus, first, we have got to create a wrapper around the model. Toward this end, we use the `explain()` function from the `DALEX` package (**?**). The function requires five arguments:

- `model`, a model-object
- `data`, a validation data frame
- `y`, observed values of the dependent variable for the validation data
- `predict_function`, a function that returns prediction scores; if not specified, then a default `predict()` function is used
- `label`, a function that returns prediction scores; if not specified, then it is extracted from the `class(model)`.

In the example below we create explainers for every model created above.

Each explainer wraps suitable `predict()` function, validation data set and the model i.e. all elements needed to create a model explanation. Thus in the next chapter we will use these explainers instead of models to keep code spinets more concise.

```
explain_titanic_lmr_v6 <- explain(model = titanic_lmr_v6,
                                  data = titanic[, -9],
                                  y = titanic$survived == "yes",
                                  predict_function = function(m, x) predict(m, x, type = "fitted"),
                                  label = "Logistic Regression v6")
explain_titanic_rf_v6 <- explain(model = titanic_rf_v6,
                                  data = titanic[, -9],
                                  y = titanic$survived == "yes",
                                  label = "Random Forest v6")
explain_titanic_rf_v3 <- explain(model = titanic_rf_v3,
                                  data = titanic[, -9],
                                  y = titanic$survived == "yes",
```

```
                                  label = "Random Forest v3")
explain_titanic_gbm_v6 <- explain(model = titanic_gbm_v6,
                                  data = titanic[, -9],
                                  y = titanic$survived == "yes",
                                  label = "Generalized Boosted Regression v6")
```

0.4.2 List of objects for the `titanic` example

In the previous sections we have built several predictive models for the `titanic` data set. The models will be used in the rest of the book to illustrate the model explanation methods and tools.

For the ease of reference, we summarize the models in the table below. The binary model-objects can be downloaded by using the attached `archivist` hooks (**?**). By calling a function specified in the last column you will recreate a selected model in your local R environment.

Table **??** summaries created models and corresponding explainers. Table **??** summaries data sets that are used in examples.

TABLE 0.1: Predictive models created for the `titanic` dataset.

Model name	Model generator	Variables	Archivist hooks
`titanic_lmr_v6`	`rms:: lmr` v.5.1.3	gender, age, class, sibsp, parch, fare, embarked	Get the model: `archivist:: aread("pbiecek/models/ceb40")`. Get the explainer: `archivist:: aread("pbiecek/models/2b9b6")`
`titanic_rf_v6`	`randomForest:: randomForest` v.4.6.14	gender, age, class, sibsp, parch, fare, embarked	Get the model: `archivist:: aread("pbiecek/models/1f938")`. Get the explainer: `archivist:: aread("pbiecek/models/9b971")`
`titanic_rf_v3`	`randomForest:: randomForest` v.4.6.14	gender, age, class	Get the model: `archivist:: aread("pbiecek/models/855c1")`. Get the explainer: `archivist:: aread("pbiecek/models/92754")`

Model name	Model generator	Variables	Archivist hooks
titanic_gbm_v6	gbm:: gbm v.2.1.5	gender, age, class, sibsp, parch, fare, embarked	Get the model: archivist:: aread("pbiecek/models/24e72"). Get the explainer: archivist:: aread("pbiecek/models/84d5f")

TABLE 0.2: Data frames created for the titanic example.

Description	No. rows	Variables	Link to this object
titanic dataset with imputed missing values	2207	gender, age, class, embarked, country, fare, sibsp, parch, survived	archivist:: aread("pbiecek/models/27e5c")
johny_d is 8 years old boy that travels in the 1st class without parents	1	class, gender, age, sibsp, parch, fare, embarked	archivist:: aread("pbiecek/models/e3596")
henry is 47 years old male passenger from the 1st class, paid 25 and embarked at Cherbourg	1	class, gender, age, sibsp, parch, fare, embarked	archivist:: aread("pbiecek/models/a6538")

0.4.3 Apartment prices

Predicting house prices is a common exercise used in machine-learning courses. Various datasets for house prices are available at websites like Kaggle (https://www.kaggle.com) or UCI Machine Learning Repository (https://archive.ics.uci.edu).

In this book we will work with an interesting version of this problem. The apartments dataset is an artificial dataset created to match key characteristics of real apartments in Warszawa, the capital of Poland. However, the dataset is created in a way that two very different models, namely linear regression and random forest, have almost exactly the same accuracy. The natural question is which model should we choose? We will show that the model-explanation tools provide important insight into the key model characteristics and are helpful in model selection.

FIGURE 11 Warsaw skyscrapers by Artur Malinowski Flicker

The dataset is available in the DALEX package (?). It contains 1000 observations (apartments) and six variables:

- *m2.price*, apatments price per meter-squared (in EUR), a numerical variable
- *construction.year*, the year of construction of the block of flats in which the apartment is located, a numerical variable
- *surface*, apartment's total surface in squared meters, a numerical variable
- *floor*, the floor at which the apartment is located (ground floor taken to be the first floor), a numerical integer variable with values from 1 to 10
- *no.rooms*, the total number of rooms, a numerical variable with values from 1 to 6
- *distric*, a factor with 10 levels indicating tha distric of Warszawa where the apartment is located

Models considered for this dataset will use *m2.price* as the (continuous) dependent variable.

```
library("DALEX")
head(apartments, 2)
```

```
##   m2.price construction.year surface floor no.rooms    district
## 1     5897              1953      25     3        1 Srodmiescie
## 2     1818              1992     143     9        5     Bielany
```

```
str(apartments)
```

```
## 'data.frame':    1000 obs. of  6 variables:
##  $ m2.price         : num  5897 1818 3643 3517 3013 ...
##  $ construction.year: num  1953 1992 1937 1995 1992 ...
##  $ surface          : num  25 143 56 93 144 61 127 105 145 112 ...
##  $ floor            : int  3 9 1 7 6 6 8 8 6 9 ...
##  $ no.rooms         : num  1 5 2 3 5 2 5 4 6 4 ...
##  $ district         : Factor w/ 10 levels "Bemowo","Bielany",..: 6 2 5 4 3 6 3 7 6 6 ...
```

```
table(apartments$floor)
```

```
##
##   1   2   3   4   5   6   7   8   9  10
##  90 116  87  86  95 104 103 103 108 108
```

```
table(apartments$no.rooms)
```

```
##
##   1   2   3   4   5   6
##  99 202 231 223 198  47
```

```
levels(apartments$district)
```

```
##  [1] "Bemowo"      "Bielany"     "Mokotow"     "Ochota"      "Praga"
##  [6] "Srodmiescie" "Ursus"       "Ursynow"     "Wola"        "Zoliborz"
```

Model predictions will be obtained for a set of six apartments included in data frame `apartments_test`, also included in the DALEX package.

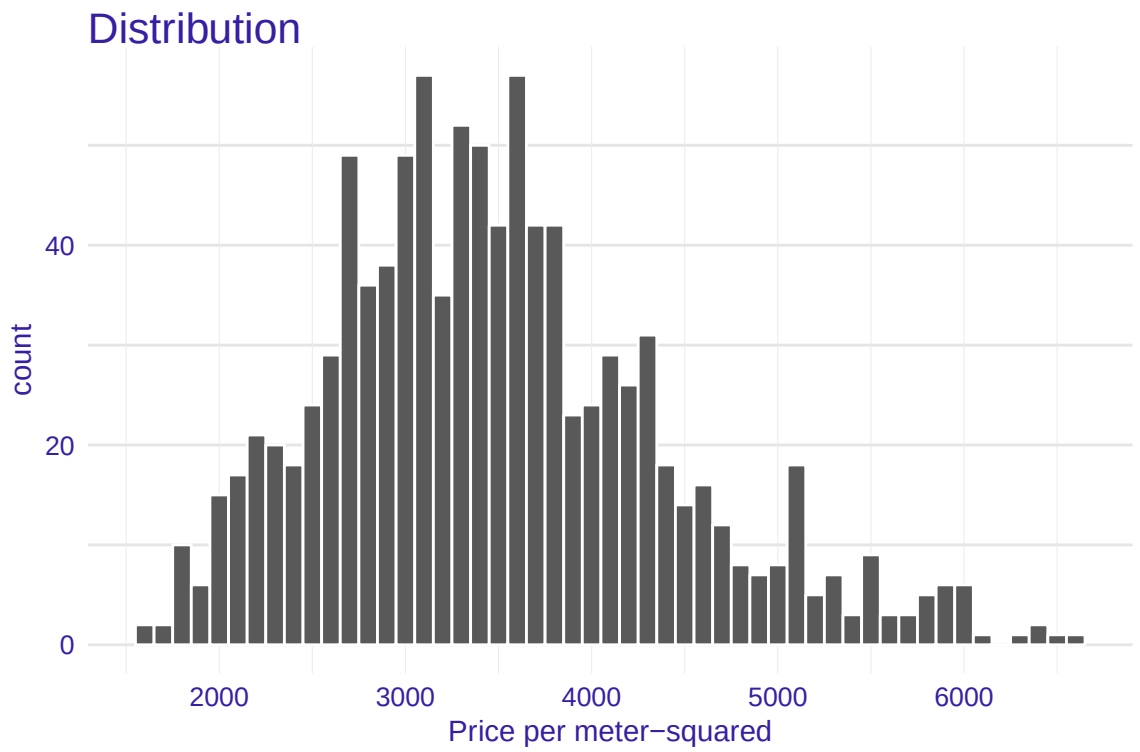


FIGURE 12 (#fig:appartments_exploration_mi2)(fig:appartments_exploration_mi2) Distribution of the price per meter-squared in the apartments data.

```
head(apartments_test)
```

```
##      m2.price construction.year surface floor no.rooms    district
## 1001     4644              1976     131     3        5 Srodmiescie
## 1002     3082              1978     112     9        4     Mokotow
## 1003     2498              1958     100     7        4     Bielany
## 1004     2735              1951     112     3        5        Wola
## 1005     2781              1978     102     4        4      Bemowo
## 1006     2936              2001     116     7        4      Bemowo
```

0.4.3.1 Data exploration

Note that this is an artificial dataset created to illustrate and explain differences between random forest and linear regression. Hence, the structure of the data, the form and strength of association between variables, plausibility of distributional assumptions, etc., is better than in a real-life dataset. In fact, all these characteristics of the data are known. Nevertheless, we conduct some data exploration to illustrate the important aspects of the data.

The variable of interest is *m2.price*, the price per meter-squared. The histogram presented in Figure @ref(fig:appartments_exploration_mi2) indicates that the distribution of the variable is slightly skewed to the right.

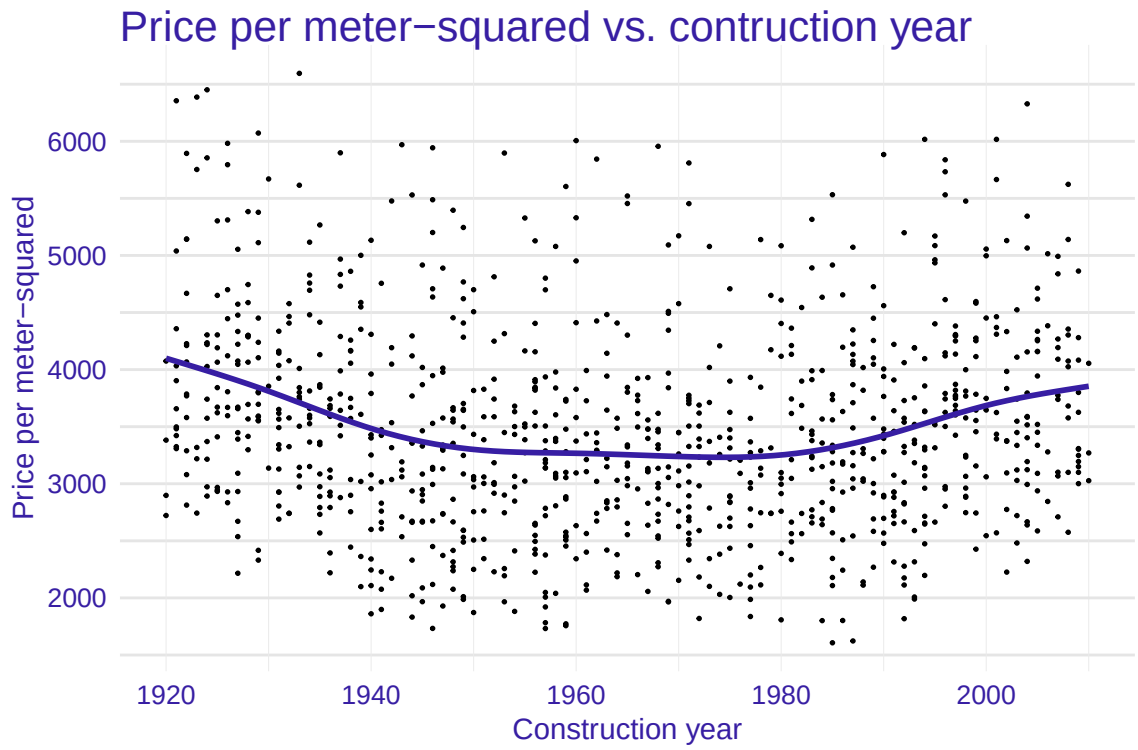


FIGURE 13 (fig:appartmentsMi2Construction) Price per meter-squared vs. construction year

Figure **??** suggests (possibly) a nonlinear relation between *construction.year* and *m2.price*.

Figure **??** indicates a linear relation between *surface* and *m2.price*.

Relation between *floor* and *m2.price* is also close to linear, as seen in Figure **??**.

There is a close to linear relation between *no.rooms* and *m2.price*, as suggested by Figure **??**. It is worth noting that, quite naturally, surface and number of rooms are correlated (see Figure appartments_surface_norooms).

Prices depend on district. Violin plots in Figure **??** indicate that the highest prices per meter-squeared are observed in Srodmiescie (Downtown).

0.4.3.2 Linear regression

The dependent variable of interest, *m2.price*, is continuous. Thus, a natural choice to build a predictive model is linear regression. We treat all the other variables in the `apartments` data frame as explanatory and include them in the model. The results of the model are stored in model-object `apartments_lm_v5`.

```
apartments_lm_v5 <- lm(m2.price ~ ., data = apartments)
apartments_lm_v5
```

```
##
## Call:
```

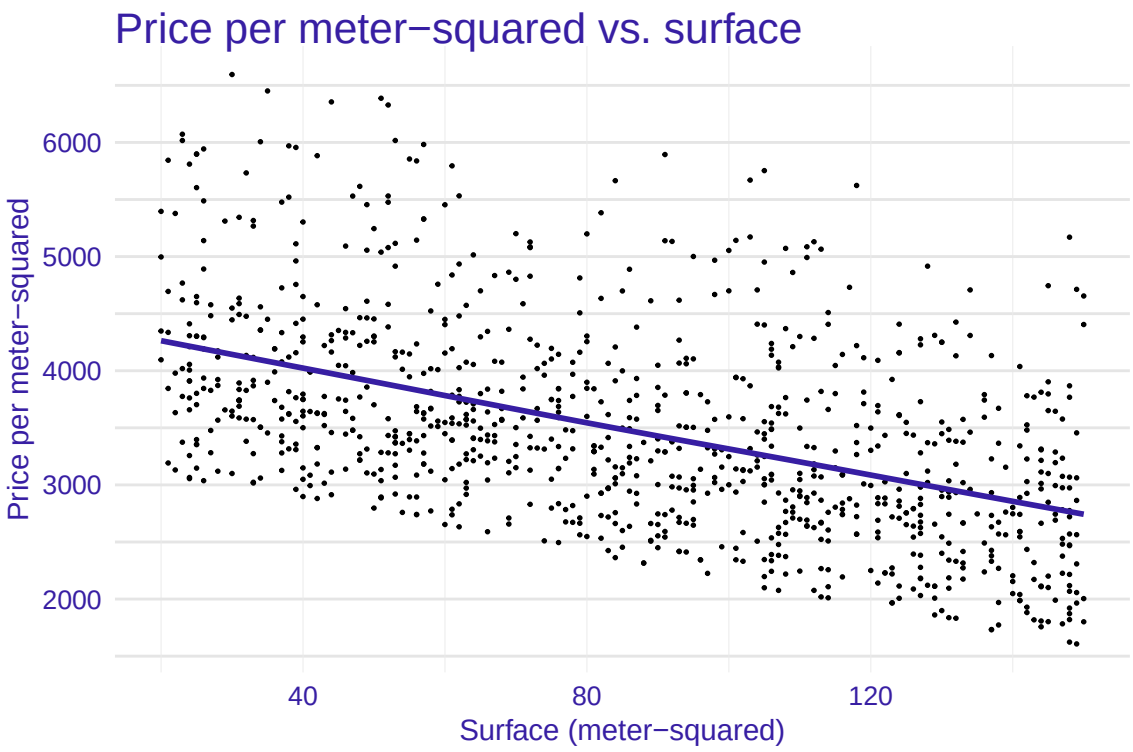


FIGURE 14 (fig:appartmentsMi2Surface) Price per meter-squared vs. surface

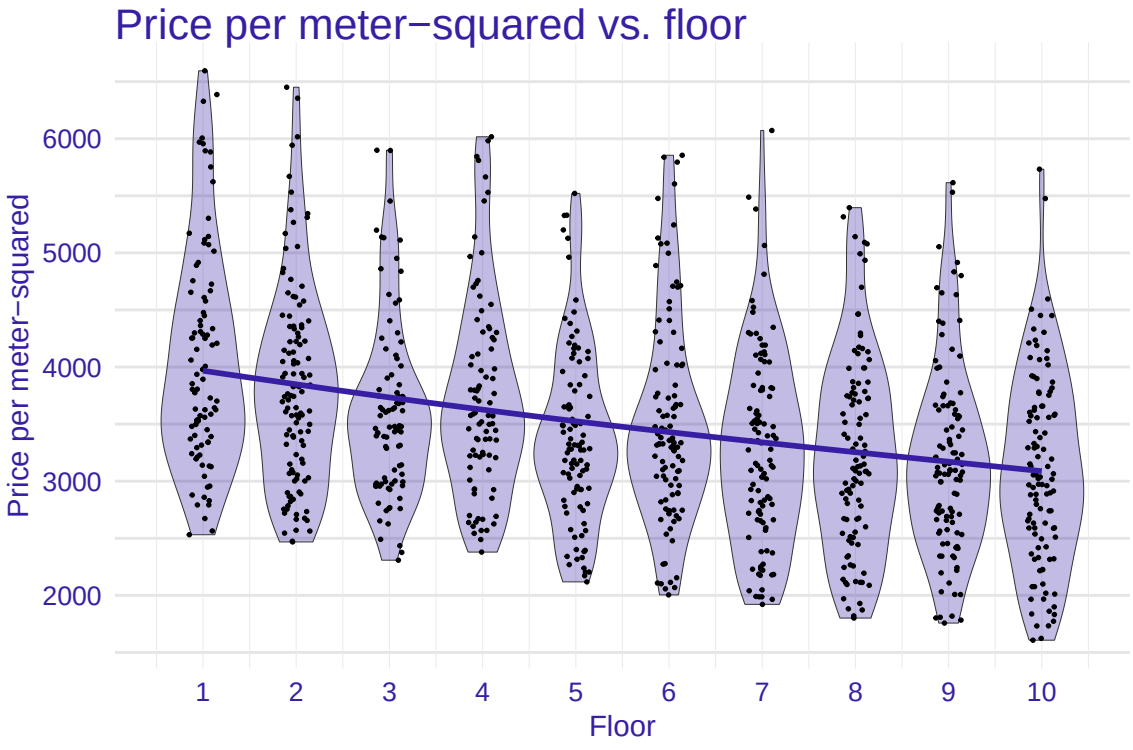


FIGURE 15 (fig:appartmentsMi2Floor) Price per meter-squared vs. floor

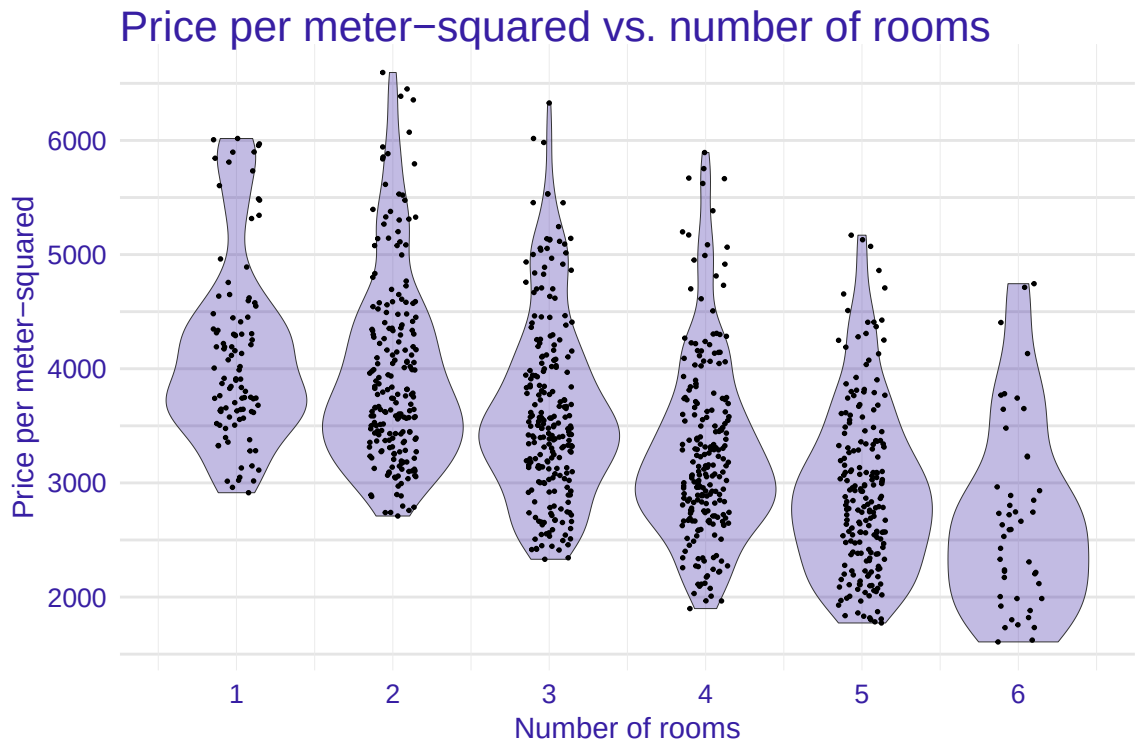


FIGURE 16 (fig:appartmentsMi2Norooms) Price per meter-squared vs. number of rooms

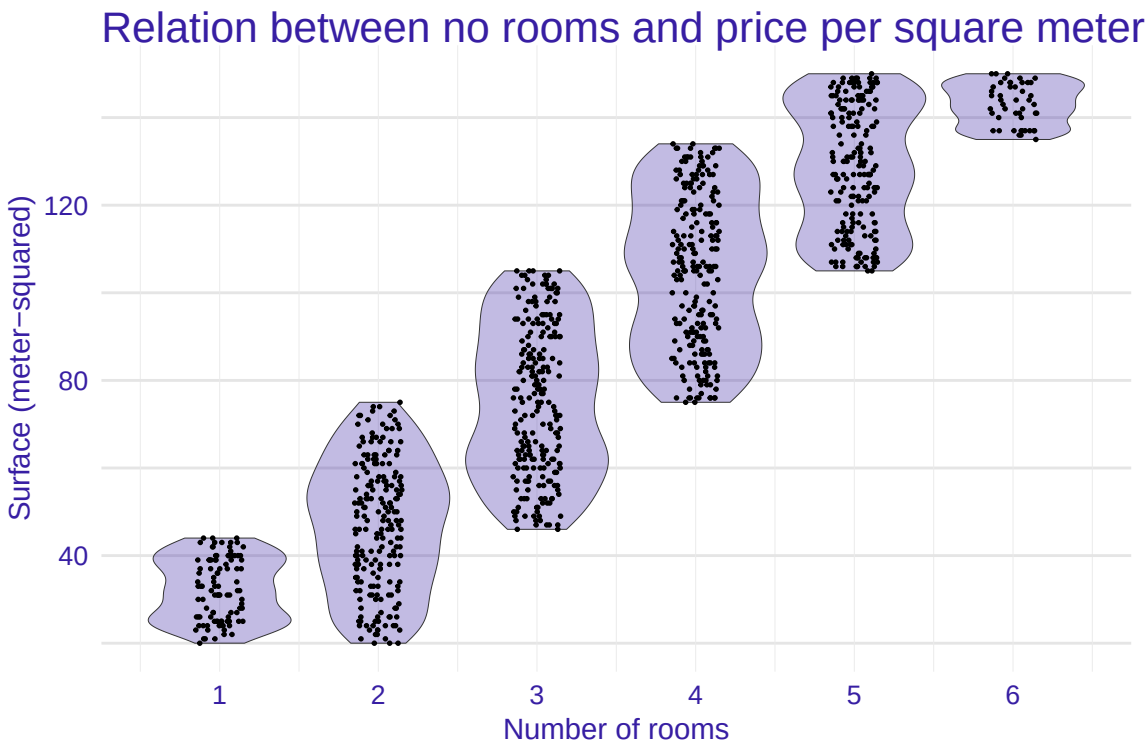


FIGURE 17 (fig:appartmentsSurfaceNorooms) Surface vs. number of rooms

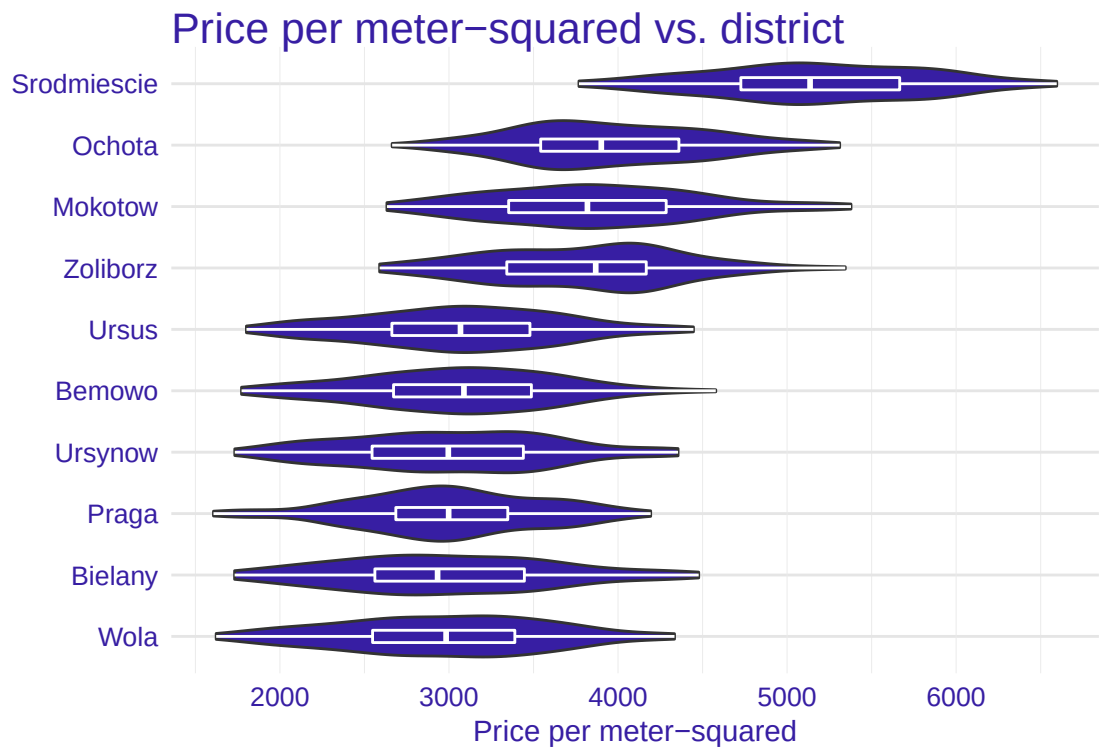


FIGURE 18 (fig:appartmentsMi2District) Price per meter-squared vs. district

```
## lm(formula = m2.price ~ ., data = apartments)
##
## Coefficients:
##         (Intercept)    construction.year              surface
##            5003.248               -0.229              -10.238
##               floor             no.rooms      districtBielany
##             -99.482              -37.730              34.106
##       districtPraga      districtUrsynow       districtBemowo
##             -20.214               -1.974              16.891
##       districtUrsus     districtZoliborz      districtMokotow
##              46.833              906.865              935.271
##      districtOchota  districtSrodmiescie
##             943.145             2097.502
```

0.4.3.3 Random forest

As an alternative to linear regression, we consider a random forest model. To fit the model, we apply the `randomForest()` function, with default settings, from the package with the same name (**?**).
The results of the model are stored in model-object `apartments_rf_v5`.

```
library("randomForest")
set.seed(72)
apartments_rf_v5 <- randomForest(m2.price ~ ., data = apartments)
apartments_rf_v5
```

```
##
## Call:
##  randomForest(formula = m2.price ~ ., data = apartments)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 1
##
##           Mean of squared residuals: 79789.39
##                     % Var explained: 90.28
```

0.4.3.4 Model predictions

By aplying the `predict()` function to model-object `apartments_lm_v5` with `apartments_test` as the data frame for which predictions are to be computed, we obtain the predicted prices for the testing set of six apartments for the linear regression model. Subsequently, we compute the mean squared difference between the predicted and actual prices for the test apartments. We repeat the same steps for the random forest model.

```
predicted_apartments_lm <- predict(apartments_lm_v5, apartments_test)
rmsd_lm <- sqrt(mean((predicted_apartments_lm - apartments_test$m2.price)^2))
rmsd_lm
```

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

```
predicted_apartments_rf <- predict(apartments_rf_v5, apartments_test)
rmsd_rf <- sqrt(mean((predicted_apartments_rf - apartments_test$m2.price)^2))
rmsd_rf
```

```
## [1] 282.9519
```

For the random forest model, the square-root of the mean squared difference is equal to 283. It is only minimally smaller than the value of 283.1, obtained for the linear regression model. Thus, the question we may face is: should we choose the model complex, but flexible random-forest model, or the simpler and easier to interpret linear model? In the subsequent chapters we will try to provide an answer to this question.

0.4.4 List of objects for the `apartments` example

In the previous sections we have built several predictive models for the `apartments` data set. The models will be used in the rest of the book to illustrate the model explanation methods and tools.

For the ease of reference, we summarize the models in the table below. The binary model-objects can be downloaded by using the attached `archivist` hooks (**?**). By calling a function specified in the last column you will recreate a selected model in your local R environment.

TABLE 0.3: Predictive models created for the `apartments` dataset.

Model name	Model generator	Variables	Archivist hooks
`apartments_lm_v5`	`stats:: lm` v.3.5.3	construction .year, surface, floor, no.rooms, district	Get the model: `archivist:: aread("pbiecek/models/55f19")`. Get the explainer: TODO: add if needed
`apartments_rf_v5`	`randomForest:: randomForest` v.4.6.14	construction .year, surface, floor, no.rooms, district	Get the model: `archivist:: aread("pbiecek/models/fe7a5")`. Get the explainer: TODO: add if needed

0.4.5 Hire or fire

Predictive models can be used to support decisions. For instance, they could be used in a human-resources department to decide whether, for instance, promote an employee. An advantage of using a model for this purpose would be the objectivity of the decision, which would not be subject to personal preferences of a manager. However, in such a situation, one would most likely want to understand what influences the model's prediction.

To illustrate such a situation, we will use the `HR` dataset that is available in the `DALEX` package (**?**). It is an artificial set of data from a human-resources department of a call center. It contains 7847 observations (employees of the call center) and six variables:

- *gender*, person's gender, a factor with two levels
- *age*, person's age in years, a numerical variable
- *hours*, average number of working hours per week, a numerical variable
- *evaluation*, the last evaluation score, a numerical variable with values 2 (fail), 3 (satisfactory), 4 (good), and 5 (very good)
- *salary*, the salary level, a numerical variable with values from 0 (lowest) to 5 (highest)
- *status*, a factor with three indicating whether the employee was fired, retained, or promoted

Models considered for this dataset will use *status* as the (categorical) dependent variable.

```
library("DALEX")
head(HR, 4)
```

```
##   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
```

```
str(HR)
```

```
## 'data.frame':    7847 obs. of  6 variables:
##  $ gender    : Factor w/ 2 levels "female","male": 2 1 2 1 2 2 1 2 1 1 ...
##  $ age       : num  32.6 41.2 37.7 30.1 21.1 ...
##  $ hours     : num  41.9 36.3 36.8 39 62.2 ...
##  $ evaluation: num  3 2 3 3 5 2 4 2 2 4 ...
##  $ salary    : num  1 5 0 2 3 0 0 4 4 4 ...
##  $ status    : Factor w/ 3 levels "fired","ok","promoted": 1 1 1 1 3 1 3 2 1 3 ...
```

```
table(HR$evaluation)
```

```
##
##    2    3    4    5
## 2371 2272 1661 1543
```

```
table(HR$salary)
```

```
##
##    0    1    2    3    4    5
## 1105 1417 1461 1508 1316 1040
```

0.4.5.1 Data exploration

As it was the case for the `Apartments` dataset (see Section **??**), the `HR` data were simulated. Despite the fact that characteristics of the data are known, we conduct some data exploration to illustrate the important aspects of the data.

Figure **??** indicates that young females and older males were fired more frequently than older females and younger males.

Figure **??** indicates that the proportion of promoted employees was the lowest for the lowest and highest salary level. At the same time, the proportion of fired employees was the highest for the two salary levels.

Figure **??** indicates that the chance of being fired was larger for evaluation scores equal to 2 or 3. On the other hand, the chance of being promoted substantially increased for scores equal to 4 or 5.

0.4.5.2 Multinomial logistic regression

The dependent variable of interest, *status*, is categorical with three categories. Thus, a simple choice is to consider a multinomial logistic regression model (**?**). We fit the model with the

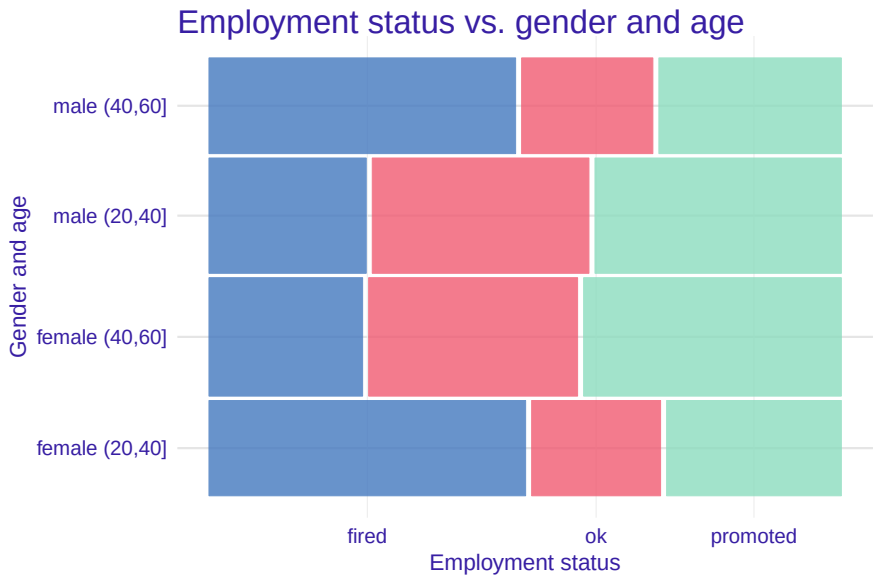


FIGURE 19 Employment status for age-groups and gender.

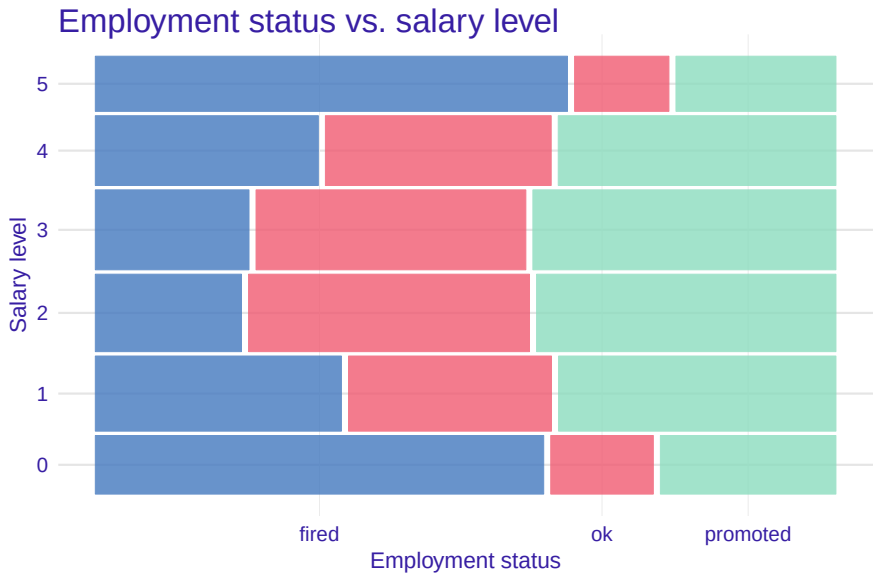


FIGURE 20 Employment status for different salary levels.

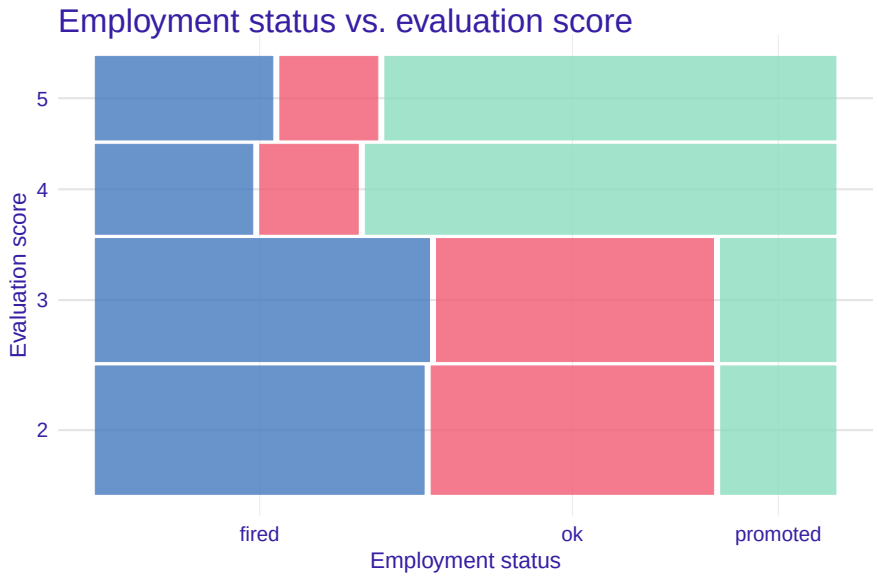


FIGURE 21 Employment status for different evaluation scores.

help of function `multinom` from package `nnet`. The function fits multinomial log-linear models by using the neural-networks approach.We chose this model to have a complex model that is smooth in contrary to random forest that relies on binary splits for continuous variables. We treat all variables other than `status` in the `HR` data frame as explanatory and include them in the model. The results of the model are stored in model-object `HR_glm_v5`.

```
library("nnet")
set.seed(1313)

HR_glm_v5 <- 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
```

```
HR_glm_v5
```

```
## Call:
## multinom(formula = status ~ gender + age + hours + evaluation +
##     salary, data = HR)
##
## Coefficients:
##          (Intercept) gendermale         age      hours  evaluation
```

```
## ok         -3.199741 0.05185293 0.001003521 0.06628055 -0.03734345
## promoted  -12.677639 0.11838037 0.003436872 0.16253343  1.26109093
##              salary
## ok       0.01680039
## promoted 0.01507927
##
## Residual Deviance: 12478.96
## AIC: 12502.96
```

0.4.5.3 Random forest

As an alternative to multinomial logisitc regression, we consider a random forest model. To fit the model, we apply the `randomForest()` function, with default settings, from the package with the same name (**?**). The results of the model are stored in model-object `HR_rf_v5`.

```
library("randomForest")
set.seed(1313)

HR_rf_v5 <- randomForest(status ~ gender + age + hours + evaluation + salary, data = HR)
HR_rf_v5
```

```
##
## 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: 27.46%
## Confusion matrix:
##          fired   ok promoted class.error
## fired     2270  388      197   0.2049037
## ok         530 1243      448   0.4403422
## promoted   202  390     2179   0.2136413
```

0.4.5.4 Model predictions

Let us now compare predictions that are obtained from the multinomial regression and random forest models. In particular, we will compute the predicted probabilities of being fired, retained in service, or promoted for Dilbert, a 58-year old male working around 42 hours per week for a salary at level 2, who got evaluation score equal to 2. Data frame `dilbert` contains the data describing the employee.

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

```
                evaluation = 2,
                salary = 2)
```

By aplying the `predict()` function to model-objects `HR_rf_v5` and `HR_glm_v5`, with `dilbert` as the data frame for which predictions are to be computed and argument `type="prob"`, we obtain the predicted probabilities of being fired, retained in service, or promoted for Dilbert.

```
pred_HR_rf <- predict(HR_rf_v5, dilbert, type = "prob")
pred_HR_rf
```

```
##   fired    ok promoted
## 1 0.778 0.218    0.004
## attr(,"class")
## [1] "matrix" "votes"
```

```
pred_HR_glm <- predict(HR_glm_v5, dilbert, type = "prob")
pred_HR_glm
```

```
##      fired         ok   promoted
## 0.56369601 0.40630786 0.02999612
```

For both models, the predicted probability of promotion is low; it is more likely that Dilbert will be fired. It is of interest to understand why such prediction is made? Moreover, random forest yields a higher probability of firing (0.78) than the multinomial regression model (0.56). We may want to learn where does this difference come from? We will try to answer these questions in subsequent chapters.

0.4.6 List of objects for the HR example

In the previous sections we have built several predictive models for the `HR` data set. The models will be used in the rest of the book to illustrate the model explanation methods and tools.

For the ease of reference, we summarize the models in the table below. The binary model-objects can be downloaded by using the attached `archivist` hooks (**?**). By calling a function specified in the last column you will recreate a selected model in your local R environment.

TABLE 0.4: Predictive models created for the `HR` dataset.

Model name	Model generator	Variables	Archivist hooks
`HR_rf_v5`	`randomForest::randomForest` v.4.6.14	gender, age, hours, evaluation, salary	Get the model: `archivist::aread("pbiecek/models/1ecfd")`. Get the explainer: TODO: add if needed

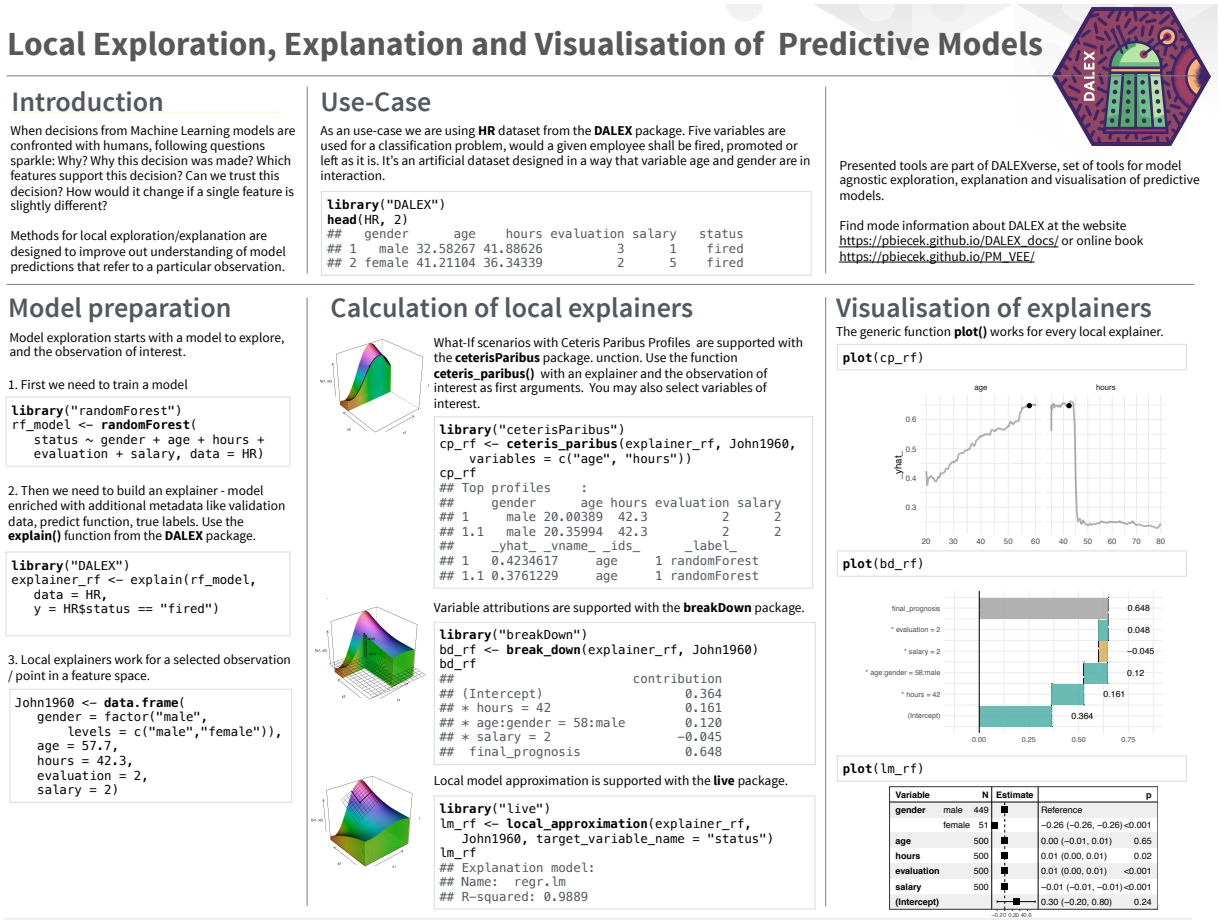


FIGURE 22 (fig:localDALEXsummary) Summary of three approaches to local model exploration and explanation.

Model name	Model generator	Variables	Archivist hooks
`HR_glm_v5`	`stats:: glm` v.3.5.3	gender, age, hours, evaluation, salary	Get the model: `archivist:: aread("pbiecek/models/f0244")`. Get the explainer: TODO: add if needed

Instance-level explanation

0.5 Introduction

Instance-level explainers help to understand how a model yields a prediction for a single observation. We can think about the following situations as examples:

- We may want to evaluate the effects of explanatory variables on model predictions. For instance, we may be interested in predicting the risk of heart attack based on person's age, sex, and smoking habits. A model may be used to construct a score (for instance, a linear combination of the explanatory variables representing age, sex, and smoking habits) that could be used for the purposes of prediction. For a particular patient We may want to learn how much the different variables contribute to the patient's score?
- We may want to understand how models predictions would change if values of some of the explanatory variables changed. For instance, what would be the predicted risk of heart attack if the patient cut the number of cigarettes smoked per day by half?
- We may discover that the model is providing incorrect predictions and we may want to find the reason. For instance, a patient with a very low risk-score experiences heart attack. What has driven that prediction?

A model is a function with a p-dimensional vector x as an argument. The plot of the value(s) of the function can be constructed in a $p+1$-dimensional space. An example with $p = 2$ is presented in Figure **??**. We will use it as an illustration of key ideas. The plot provides an information about the values of the function in the vicinity of point x^*.

There are many different tools that may be used to explore the predictions of the model around a single point x^*. In the following sections we will describe the most popular approaches. They can be divided into three classes.

- One approach is to investigate how the model prediction changes if the value of a single explanatory variable changes. It is useful in the so-called „What-If" scenarios. In particular, we can construct plots presenting the change of model-based predictions in function of a single variable. Such plots are usually called Ceteris Paribus profiles. They are presented in Chapter **??**. An example is provided in panel A of Figure **??**.
- Another approach is to analyze the curvature of the response surface (see Figure **??**) around the point of interest x^*. Treating the model as a function, we are interested in the local behavior of this function around x^*. In case of a black-box model, we approximate it with a simpler white-box model around x^*. An example is provided in panel B of Figure **??**. In Chapter **??** we present the Local Interpretable Model-agnostic Explanations (LIME) method that exploits the concept of a „local model."
- Yet another approach is to analyze how the model prediction for point x^* is different from the average model prediction and how the difference can be distributed among the different dimensions (explanatory variables). It is often called the „variable attributions"

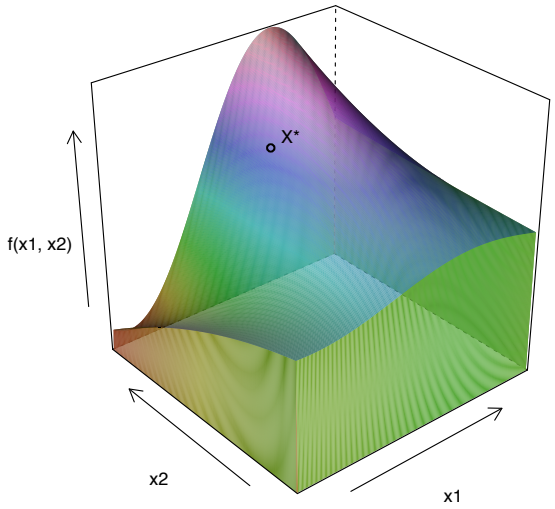


FIGURE 23 (fig:cutsSurfaceReady) Response surface for a model that is a function of two variables. We are interested in understanding the response of a model in a single point x*

approach. An example is provided in panel C of Figure **??**. In Chapter **??** we present two methods implementing this approach, sequential conditioning and average conditioning (also called Shapley values).

0.6 Ceteris-paribus Profiles and What-If Analysis

0.6.1 Introduction

Ceteris paribus is a Latin phrase meaning "other things held constant" or "all else unchanged." In this chapter, we introduce a technique for model exploration based on the Ceteris paribus principle. In particular, we examine the influence of each explanatory variable, assuming that effects of all other variables are unchanged. The main goal is to understand how changes in a single explanatory variable affects model predictions.

Explanation tools (explainers) presented in this chapter are linked to the second law introduced in Section **??**, i.e. the law of "Prediction's speculation." This is why the tools are also known as *What-If model analysis* or *Individual Conditional Expectations* (**?**). It turns out

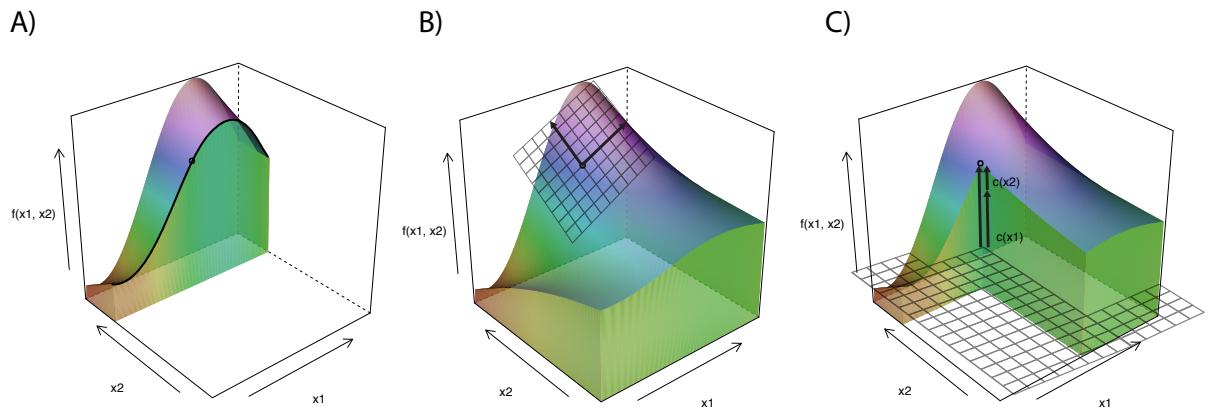


FIGURE 24 (fig:cutsTechnikiReady) Illustration of different approaches to instance-level explanation. Panel A presents a What-If analysis with Ceteris Paribus profiles. The profiles plot the model response as a function of a value of a single variable, while keeping the values of all other explanatory variables fixed. Panel B illustrates the concept of local models. A simpler white-box model is fitted around the point of interest. It describes the local behaviour of the black-box model. Panel C illustrates the concept of variable attributions. Additive effects of each variable show how the model response differs from the average.

that it is easier to understand how a black-box model is working if we can explore the model by investigating the influence of explanatory variables separately, changing one at a time.

0.6.2 Intuition

Panel A of Figure **??** presents response (prediction) surface for the `titanic_lmr_v6` model for two explanatory variables, *age* and *class*, from the *titanic* dataset (see Section **??**). We are interested in the change of the model prediction induced by each of the variables. Toward this end, we may want to explore the curvature of the response surface around a single point with *age* equal to 47 and *class* equal to "1st," indicated in the plot. Ceteris-paribus (CP) profiles are one-dimensional profiles that examine the curvature across each dimension, i.e., for each variable. Panel B of Figure **??** presents the profiles corresponding to *age* and *class*. Note that, in the CP profile for *age*, the point of interest is indicated by the black dot. In essence, a CP profile shows a conditional expectation of the dependent variable (response) for the particular explanatory variable.

CP technique is similar to the LIME method (see Chapter **??**). LIME and CP profiles examine the curvature of a model response-surface. The difference between these two methods lies in the fact that LIME approximates the black-box model of interest locally with a simpler white-box model. Usually, the LIME model is sparse, i.e., contains fewer variables, and thus we have got to graphically investigate a smaller number of dimensions. On the other hand, the CP profiles present conditional predictions for every variable and, in most cases, are easier to interpret.

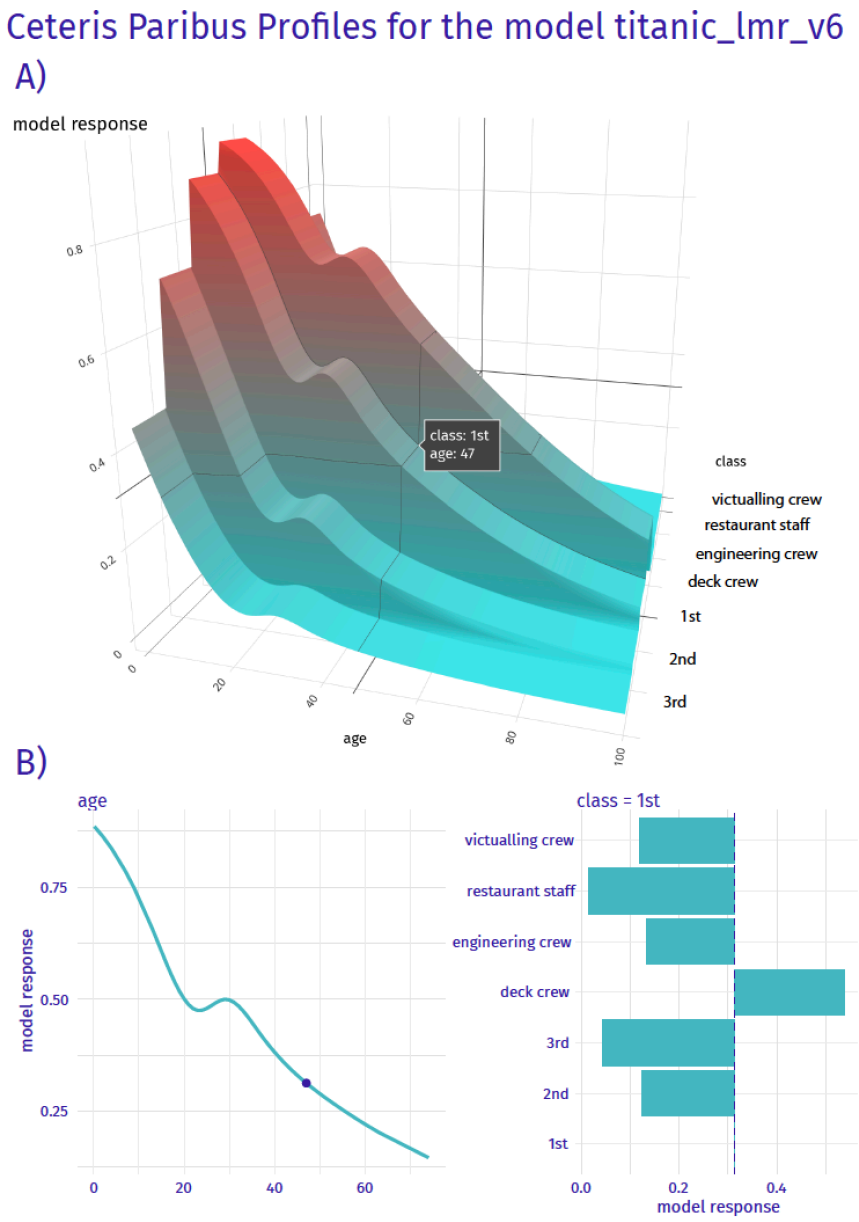


FIGURE 25 (fig:modelResponseCurveLine) A) Model response (prediction) surface. Ceteris-paribus (CP) profiles marked with black curves help to understand the curvature of the surface while changing only a single explanatory variable. B) CP profiles for individual variables, age (continuous) and class (categorical).

0.6.3 Method

In this section we introduce more formally one-dimensional CP profiles.

In predictive modeling, we are interested a distribution of a dependent variable Y given vector x^* that contains the values of explanatory variables. In the ideal world we would like to know the conditional distribution of Y given x^*, $Y|x^*$. In practical applications we usually do not predict the entire distribution, but just some of its characteristics like the expected (mean) value, a quantile, or variance. Without loss of generality we will assume that we model the expected value $E_Y(Y|x^*)$.

Assume that we have got model $f()$, for which $f(x^*)$ is an approximation of $E_Y(Y|x^*)$, i.e., $E_Y(Y|x^*) \approx f(x^*)$. Note that we do not assume that it is a "good" model, nor that the approximation is precise. We simply assume that we have got a model that is used to estimate the expected value and that to form predictions of the dependent variable. Our interest lies in the evaluation of the quality of the predictions. If the model offers a "good" approximation of the expected value, it should be reflected in its satisfactory predictive performance.

We will use subscript x_i^* to refer to the vector corresponding to the i-th observation in a dataset. We will use superscript x^{*j} to refer to the j-th element of x^*, i.e., the j-th variable. Additionally, let x^{*-j} denote a vector resulting from removing the j-th element from vector x^*. Moreover, let $x^{*|j} = z$ denote a vector in which the j-th element is equal to z (a scalar).

We define a one-dimensional CP profile for the model $f()$, j-th explanatory variable, and point x^* as follows:

$$CP^{f,j,x^*}(z) \equiv f(x^{*|j} = z).$$

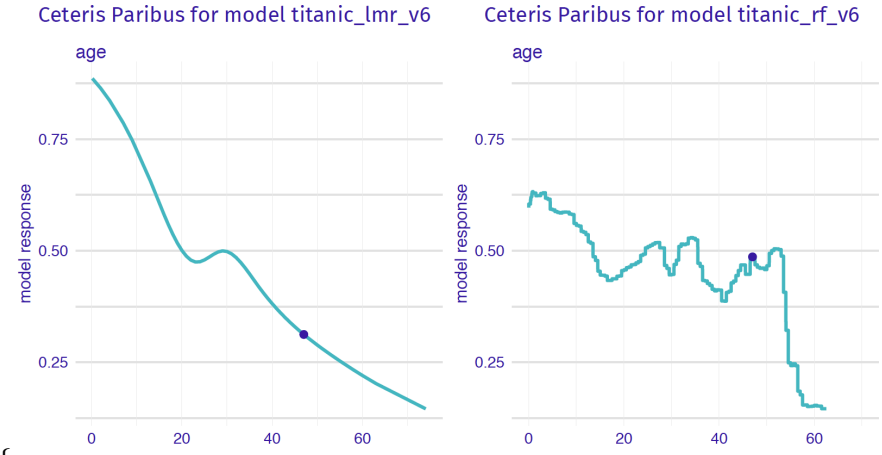
That is, CP profile is a function that provides the dependence of the approximated expected value (prediction) of the model for Y on the value of j-th explanatory variable z. Note that z is taken to go through the range of values typical for the variable, while values of all other explanatory variables are kept fixed at the values given by x^*.

0.6.4 Example: Titanic data

For continuous explanatory variables, a natural way to represent the CP function is to use a profile plot similar to the ones presented in Figure **??**. In the figure, the dot on the curves marks an instance prediction, i.e., prediction $f(x^*)$ for a single observation x^*. The curve itself shows how the prediction would change if the value of a particular explanatory variable changed.

Figure **??** presents CP profiles for the *age* variable in the logistic regression and random forest models for the Titanic dataset (see Section **??**). It is worth observing that the profile for the logistic regression model is smooth, while for the random forest model it shows more variability. For this instance (observation), the prediction for both models would increase substantially if the value of the explanatory variable became lower than 20.

\begin{figure}



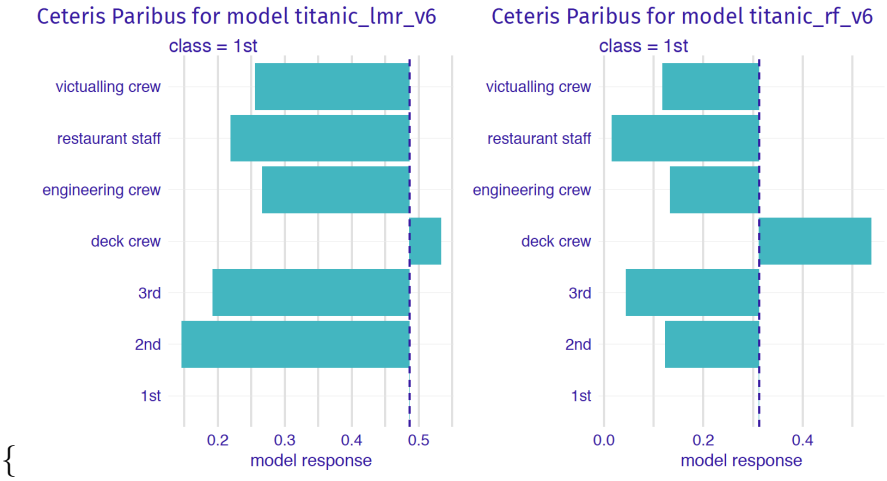
{

}

\caption{(fig:profileAgeRf) Ceteris-paribus profiles for variable `age` for the logistic regression (`titanic_lmr_v6`) and random forest (`titanic_rf_v6`) models that predict the probability of surviving based on the Titanic data} \end{figure}

For a categorical explanatory variable, a natural way to represent the CP function is to use a barplot similar to the ones presented in Figure **??**. The barplots in Figure **??** present CP profiles for the *class* variable in the logistic regression and random forest models for the Titanic dataset (see Section **??**). For this instance (observation), the predicted probability for the logistic regression model would decrease substantially if the value of *class* changed to "2nd". On the other hand, for the random forest model, the largest change would be marked if *class* changed to "restaurant staff".

\begin{figure}



{

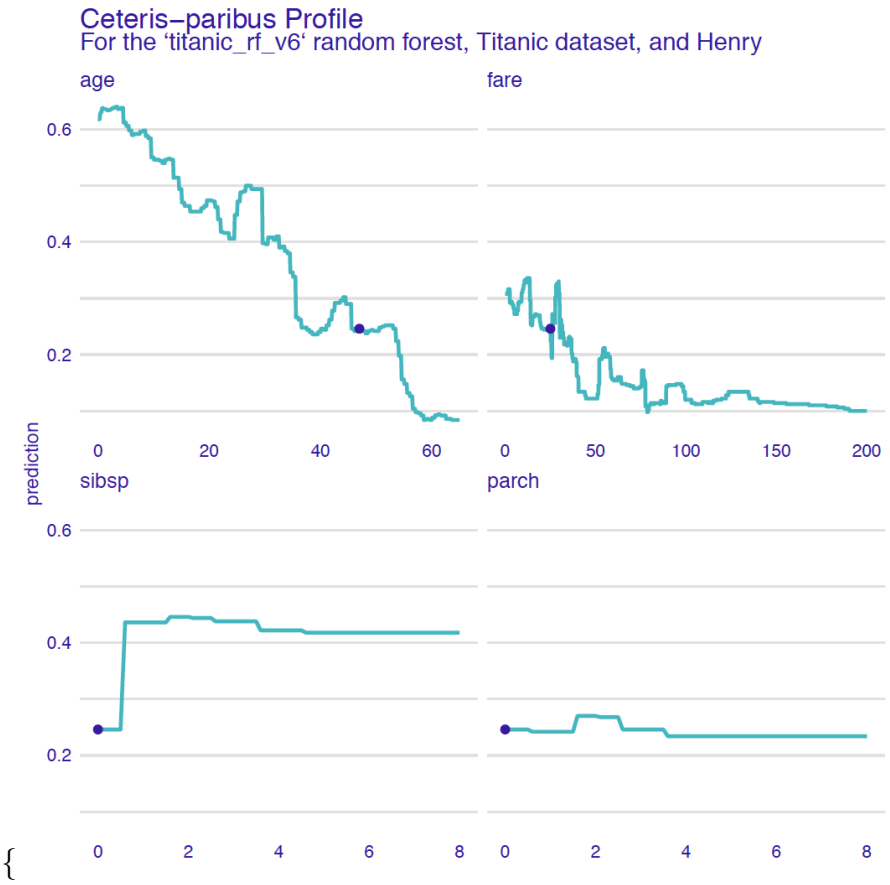
}

\caption{(fig:profileAgeRf2) Ceteris-paribus profiles for variable `class` for the logistic

regression (`titanic_lmr_v6`) and random forest (`titanic_rf_v6`) models that predict the probability of surviving based on the Titanic data} \end{figure}

Usually, black-box models contain a large number of explanatory variables. However, CP profiles are legible even for tiny subplots, created with techniques like sparklines or small multiples (**?**). In this way we can display a large number of profiles at the same time keeping profiles for consecutive variables in separate panels, as shown in Figure **??** for the random forest model for the Titanic dataset. It helps if these panels are ordered so that the most important profiles are listed first. We discuss a method to assess the importance of CP profiles in the next chapter.

\begin{figure}



{

}

\caption{(fig:profileV4Rf) Ceteris-paribus profiles for all continuous explanatory variables for the random forest (`titanic_rf_v6`) model for the `titanic` dataset} \end{figure}

0.6.5 Pros and cons

One-dimensional CP profiles, as presented in this chapter, offer a uniform, easy to communicate and extendable approach to model exploration. Their graphical

representation is easy to understand and explain. It is possible to show profiles for many variables or models in a single plot.

There are several issues related to the use of the CP profiles. If explanatory variables are correlated, then changing one variable implies a change in the other. In such case, the application of the *Ceteris paribus* principle may lead to unrealistic settings, as it is not possible to keep one variable fixed while varying the other one. Special cases are interactions, which require the use of two-dimensional CP profiles that are more complex than one-dimensional ones. Also, in case of a model with hundreds or thousands of variables, the number of plots to inspect may be daunting. Finally, while barplots allow visualization of CP profiles for factors (categorical explanatory variables), their use becomes less trivial in case of factors with many nominal (unordered) categories (like, for example, a ZIP-code).

0.6.6 Code snippets for R

In this section we present key features of the R package `ingredients` (**?**) which is a part of `DrWhy.AI` universe and covers all methods presented in this chapter. More details and examples can be found at https://modeloriented.github.io/ingredients/.

Note that there are also other R packages that offer similar functionality, like `condvis` (**?**), `pdp` (**?**), `ICEbox` (**?**), `ALEPlot` (**?**), `iml` (**?**).

In this section, we use two classification models developed in the chapter **??**, namely logistic regression (**?**) model `titanic_lmr_v6` and the random forest (**?**) model `titanic_rf_v6`. They are trained to predict probability of survival from sinking of Titanic. Instance level explanations are calculated for a single observation `henry` - 47 years old passenger that travels 1st class.

`DALEX` explainers for both models and the Henry data are retrieved via `archivist` hooks as listed in Chapter **??**.

```
library("rms")
explain_lmr_v6 <- archivist::aread("pbiecek/models/2b9b6")

library("randomForest")
explain_rf_v6 <- archivist::aread("pbiecek/models/9b971")

library("DALEX")
henry <- archivist::aread("pbiecek/models/a6538")
henry
```

```
##   class gender age sibsp parch fare  embarked
## 1   1st   male  47     0     0   25 Cherbourg
```

0.6.6.1 Basic usage for the `ceteris_paribus` function

The easiest way to create and plot Ceteris Paribus profiles is to call `ceteris_paribus()` function and then the generic `plot()` function. By default profiles for all variables are being calculated and all numeric features are being plotted. One can limit the number of variables that should be considered with the `variables` argument.

To obtain CP profiles, we use the `ceteris_paribus()` function. It requires the explainer-object and the instance data frame as arguments.

As a result, the function yields an object od the class `ceteris_paribus_explainer`. It is a data frame with model predictions.

```
library("ingredients")
cp_titanic_lmr <- ceteris_paribus(explain_lmr_v6, henry)
cp_titanic_lmr
```

```
## Top profiles    :
##                class gender age sibsp parch fare  embarked     _yhat_
## 1                3rd   male  47     0     0   25 Cherbourg 0.08809759
## 1.1              2nd   male  47     0     0   25 Cherbourg 0.19569070
## 1.2              1st   male  47     0     0   25 Cherbourg 0.43182446
## 1.3 engineering crew   male  47     0     0   25 Cherbourg 0.22363294
## 1.4 victualling crew   male  47     0     0   25 Cherbourg 0.20389924
## 1.5 restaurant staff   male  47     0     0   25 Cherbourg 0.03090131
##     _vname_ _ids_               _label_
## 1     class     1 Logistic Regression v6
## 1.1   class     1 Logistic Regression v6
## 1.2   class     1 Logistic Regression v6
## 1.3   class     1 Logistic Regression v6
## 1.4   class     1 Logistic Regression v6
## 1.5   class     1 Logistic Regression v6
##
##
## Top observations:
##   class gender age sibsp parch fare  embarked    _yhat_
## 1   1st   male  47     0     0   25 Cherbourg 0.4318245
##                  _label_ _ids_
## 1 Logistic Regression v6     1
```

To obtain a graphical representation of CP profiles, the generic `plot()` function can be applied to the data frame returned by the `ceteris_paribus()` function. It returns a `ggplot2` object that can be processed further if needed. In examples below we use `ggplot2` functions like `ggtitle()` or `ylim()` to modify plot's title or range of OY axis.

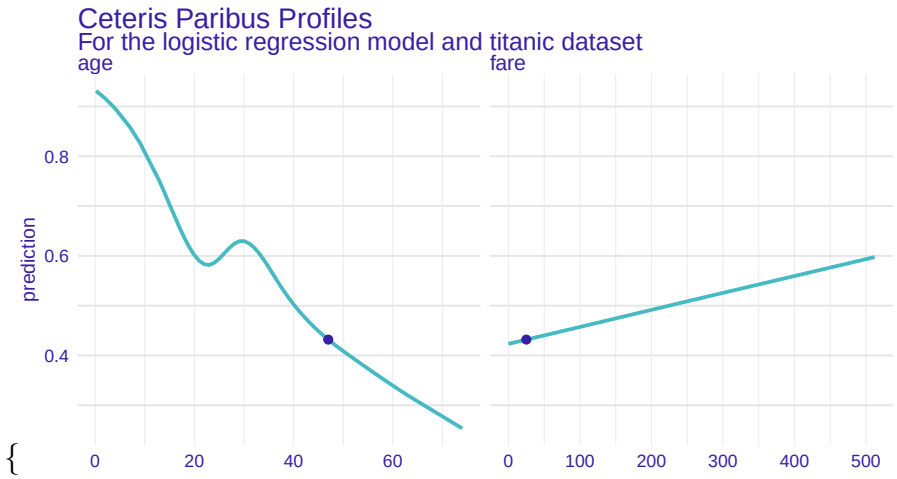
The resulting plot can be enriched with additional data by applying functions `ingredients::show_rugs()` (adds rugs for the selected points), `ingredients::show_observations`

(adds dots that shows observations), or `ingredients::show_aggreagated_profiles` (see Chapter **??**). All these functions can take additional arguments to modify size, color or linetype.

Below we show an R snippet that can be used to replicate plots presented in the left panel of Figure **??**.

```
library("ggplot2")
plot(cp_titanic_lmr, variables = c("age", "fare")) +
  show_observations(cp_titanic_lmr, variables = c("age", "fare")) +
  ggtitle("Ceteris Paribus Profiles", "For the logistic regression model and titanic dataset")
```

\begin{figure}



{

}

\caption{Ceteris-paribus profiles for `age` and `fare` variables and the `titanic_lmr_v6` model.}
\end{figure}

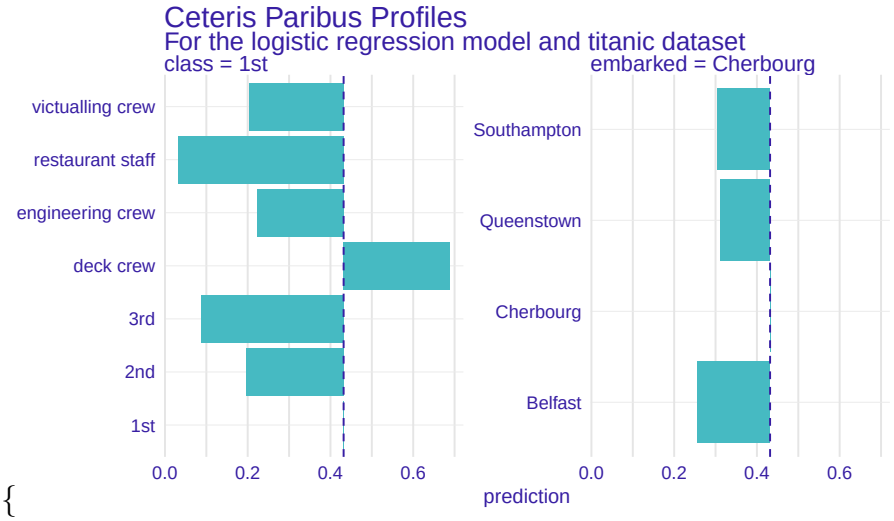
By default all numerical variables are plotted. For categorical variables we use bar plots instead of line plots. Default `ggplot2` plots cannot handle facets with different geoms, and for this reason we need to plot categorical variables separately.

To plot CP profiles for categorical variables we need to add the `only_numerical = FALSE` argument to the `plot()` function.

Here we recreate plot from Figure **??**.

```
plot(cp_titanic_lmr, variables = c("class", "embarked"), only_numerical = FALSE) +
  ggtitle("Ceteris Paribus Profiles", "For the logistic regression model and titanic dataset")
```

\begin{figure}



{

}

\caption{Ceteris-paribus profiles for `class` and `embarked` variables and the `titanic_lmr_v6` model.} \end{figure}

0.6.6.2 Advanced usage for the `ceteris_paribus` function

The `ceteris_paribus()` is a very flexible function. To better understand how it can be used let's scan it's arguments.

- `x`, `data`, `predict_function`, `label` - information about a model. If `x` is created with the `DALEX::explain` function then other arguments are extracted from `x`, this is how we use this function in this chapter. Otherwise we need to specify directly model, validation data, predict function and the model label.
- `new_observation` - instances, one or more, for which we want to calculate CP profiles. It should be a data frame with same variables as in the validation data.
- `y` - observed labels for `new_observation`. We will show how to make use of this argument in the chapter **??**,
- `variables` - names of variables for which Ceteris Paribus profiles shall be calculated. By default they will be calculated for all variables, which may be time consuming.
- `variable_splits` - it's a list which specifies for what values Ceteris Paribus shall be calculated. By default these are all values for categorical variables. For continuous variables they are uniformly placed values and one can specify how many with the `grid_points` argument (default is 101).

In the example below we replicate plot **??** for `henry` and the random forest model `titanic_rf_v6`.

The argument `variable_splits` specifies for which variables (here `age` and `fare`) CP profiles are to be calculated and in which points they shall be evaluated.

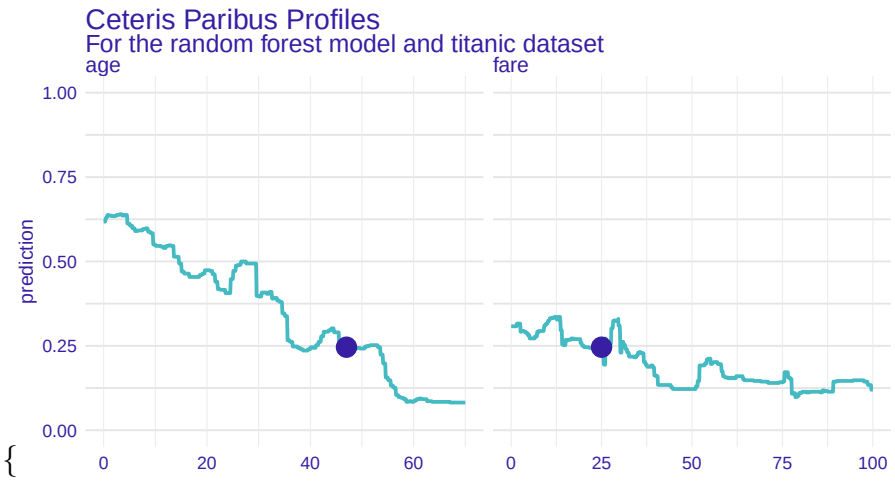
```
cp_titanic_rf <- ceteris_paribus(explain_rf_v6, henry,
```

```
                variable_splits = list(age = seq(0, 70, 0.1),
                                       fare = seq(0, 100, 0.1)))
```

Random Forest model is especially interesting case for such investigations, as the CP profiles are piece-wise constant. Moreover, as we see below, some profiles are quite unstable.

```
plot(cp_titanic_rf) +
  show_observations(cp_titanic_rf, variables = c("age", "fare"), size = 5) +
  ylim(0, 1) +
  ggtitle("Ceteris Paribus Profiles", "For the random forest model and titanic dataset")
```

\begin{figure}



{

}

\caption{Ceteris-paribus profiles for `class` and `embarked` variables and the `titanic_rf_v6` model. Blue dot stands for `henry`.} \end{figure}

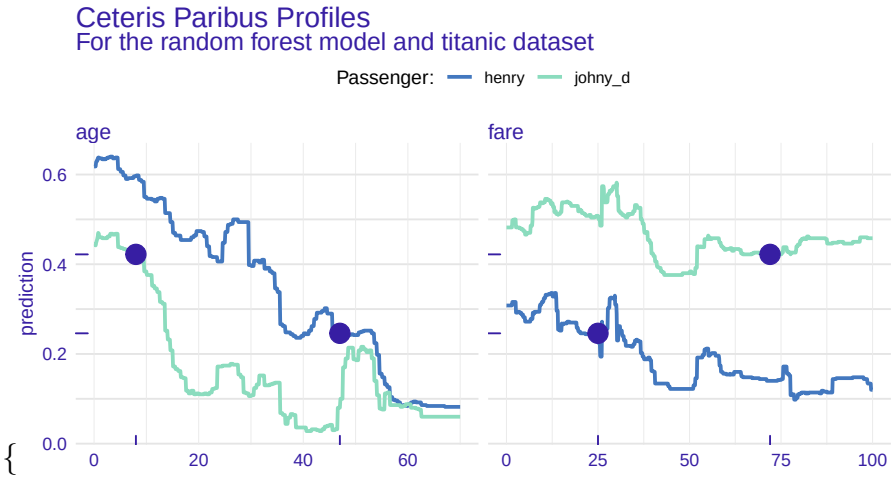
There are few functions that one can use. The generic `plot` creates a `ggplot2` object with a single `geom_line` layer. Function `show_observations` adds `geom_point` layer, `show_rugs` adds `geom_rugs` while `show_profiles` adds another `geom_line`. All these functions take as the first argument an object created with `ceteris_paribus` function. They can be combined freely combined to superpose profiles for different models or observations.

In the example below we create profiles for two passengers, `henry` and `johny_d`. Their profiles are plot in a single chart **??**. We add `scale_color_manual` function to add names of passengers to the plot, control colors and positions.

```
johny_d <- archivist::aread("pbiecek/models/e3596")
cp_titanic_rf2 <- ceteris_paribus(explain_rf_v6, rbind(henry, johny_d))
```

```
plot(cp_titanic_rf2, color = "_ids_") +
  show_observations(cp_titanic_rf2, size = 5, variables = c("age", "fare")) +
  show_rugs(cp_titanic_rf2, sides = "bl", variables = c("age", "fare")) +
  scale_color_manual(name = "Passenger:", breaks = 1:2, values = c("#4378bf", "#8bdcbe"), labels = c("henry" ,
  ggtitle("Ceteris Paribus Profiles", "For the random forest model and titanic dataset")
```

\begin{figure}



}

\caption{Ceteris-paribus profiles for the `titanic_rf_v6` model. Profiles for different passangers are color-coded.} \end{figure}

0.6.6.3 Champion-challenger analysis

One of the most interesting use cases for explainers is contrastive comparison of two or larger number of models.

Let's see how to do this. First we calculate profiles for two models. We can do this for larger number of models, here we limit to two models for larger clarity.

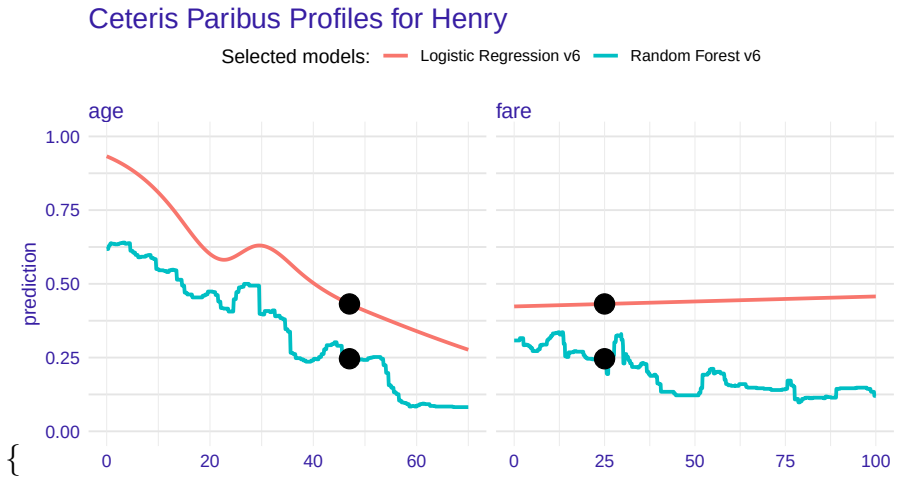
```
cp_titanic_rf <- ceteris_paribus(explain_rf_v6, henry)
cp_titanic_lmr <- ceteris_paribus(explain_lmr_v6, henry)
```

Every `plot` and `show_*` function can take a collection of explainers as arguments. Profiles for different models will be plot in a single chart. Here models are color-coded as we set argument `color = "_label_"`. Here `_label_` is the name of a column in CP explainer that contains information about model label.

Figure **??** shows differences between CP profiles for both considered models.

```
plot(cp_titanic_rf, cp_titanic_lmr, color = "_label_") +
  show_observations(cp_titanic_rf, cp_titanic_lmr, color = "black", variables = c("age", "fare"), size = 5) +
  scale_color_discrete(name = "Selected models:") + ylim(0,1) +
  ggtitle("Ceteris Paribus Profiles for Henry")
```

\begin{figure}



{

}

\caption{Champion-challenger comparison of `titanic_lmr_v6` and `titanic_rf_v6` models. Profiles for different models are color-coded.} \end{figure}

0.7 Ceteris-paribus Oscillations and Local Variable-importance

0.7.1 Introduction

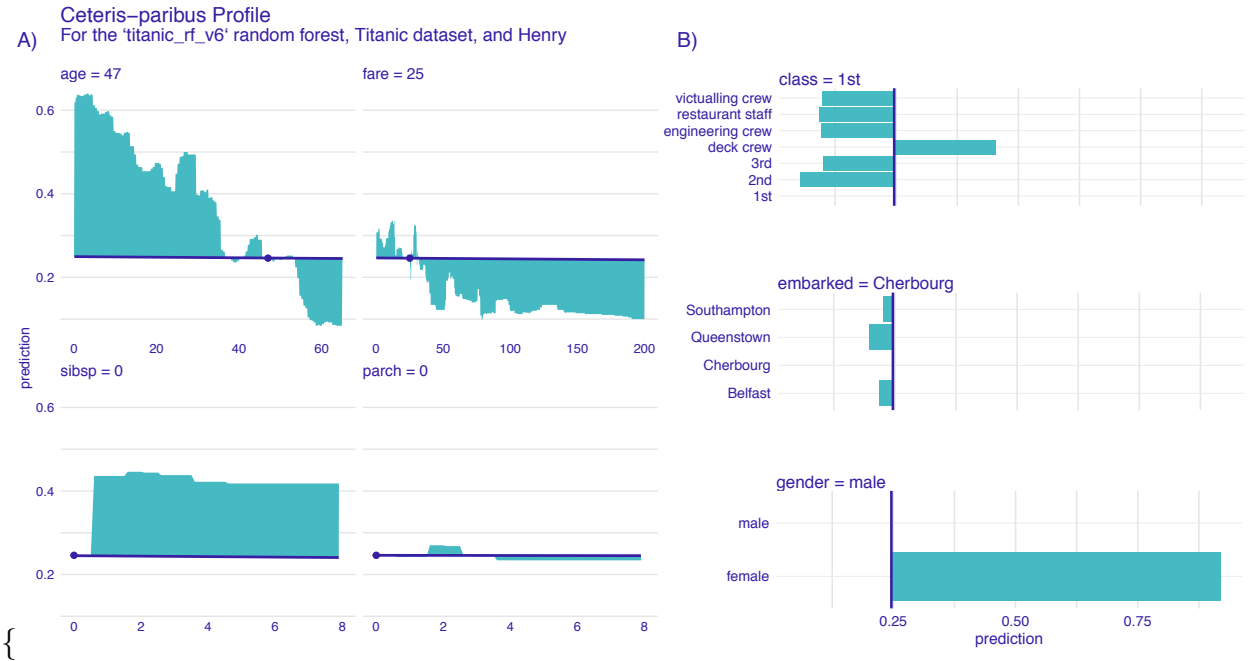
Visual examination of Ceteris-paribus (CP) profiles is insightful, but for a model with a large number of explanatory variables we may end up with a large number of plots which may be overwhelming. To prioritize between the profiles we need a measure that would summarize the impact of a selected variable on model's predictions. In this chapter we describe a solution closely linked with CP profiles. An alternative is also discussed in the Chapters **??** and **??**.

0.7.2 Intuition

To assign importance to CP profiles, we can use the concept of profile oscillations. In particular, the larger influence of an explanatory variable on prediction at a particular instance, the larger the fluctuations along the corresponding CP profile. For a variable that exercises little or no influence on model prediction, the profile will be flat or will barely change. Figure **??** illustrates the idea behind measuring oscillations. The figure corresponds to the CP profiles presented in Figure **??**.

The larger the highlighted area, the more important is the variable.

\begin{figure}



{

}

\caption{(fig:CPVIPprofiles) The value of the colored area summarizes the CP oscillations and provides the mean of the absolute deviations between the CP profile and the instance prediction. Panel A shows continuous variables while panel B shows categorical variables. Example for the `titanic_rf_v6` model.} \end{figure}

0.7.3 Method

Let us formalize this concept now. Denote by $g^j(z)$ the probability density function of the distribution of the j-th explanatory variable. The summary measure of the variable's importance for model prediction at point x, $vip_j^{CP}(x)$, computed based on the variable's CP profile, is defined as follows:

$$vip_j^{CP}(x^*) = \int_{\mathcal{R}} |CP^{f,j,x^*}(z) - f(x^*)| g^j(z) dz = E_{X_j}[|CP^{f,j,x^*}(X_j) - f(x^*)|].$$

Thus, $vip_j^{CP}(x^*)$ is the expected absolute deviation of the CP profile from the model prediction for x^* over the distribution of the j-th explanatory variable. A straightforward estimator of $vip_j^{CP}(x^*)$ is

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

where index i goes through all observations in a dataset.

Note that the importance of an explanatory variable for instance prediction may be very different for different points x^*. For example, consider model

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

where x_1 and x_2 take values in $[0, 1]$. Consider prediction for an observation described by vector $x^* = (0, 1)$. In that case, the importance of X_1 is larger than X_2. This is because the CP profile for the first variable, given by the values of function $f(z, 1) = z$, will have oscillations, while the profile for the second variable will show no oscillation, because it is given by function $f(0, z) = 0$. Obviously, the situation is reversed for $x^* = (1, 0)$.

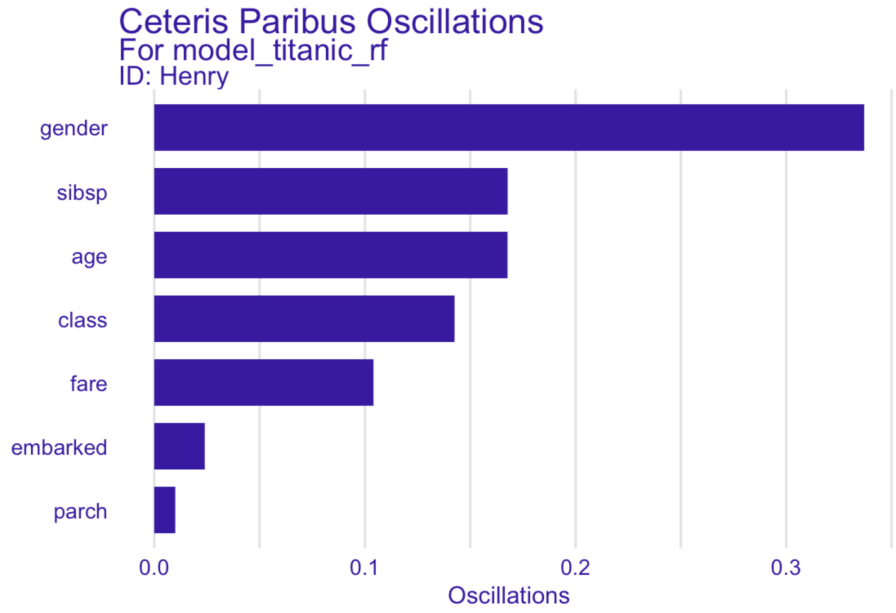
0.7.4 Example: Titanic data

Figure **??** provides a bbarplot of variable importance measures for different continuous explanatory variables for the logistic regression model (`titanic_lmr_v6`) for `henry`.

The longer the bar, the larger the CP-profile oscillations for a particular explanatory variable. Thus, Figure **??** indicates that the most important variable for prediction for the selected observation are `gender` and `sibsp`, followed by `age`.

If Henry were older, this would significantly lower the chances of survival. One the other hand, Henry were not travelling alone, this would increase these chances.

\begin{figure}



{

}

\caption{(fig:CPVIP1) Variable-importance measures calculated for Ceteris-paribus oscillations for `henry` based on the `titanic_rf_v6` model} \end{figure}

0.7.5 Pros and cons

Oscillations of CP profiels are easy to interpret and understand. By using the average of oscillations it is possible to select the most important variables for instance prediction. The methodology can easily be extended to two or more variables.

There are several issues related to the use of the CP oscillations. For instance, the oscillations may not be of help in situations when the use of CP profiles may itself be problematic (e.g., in the case of correlated explanatory variables or interactions - see Section **??**). An important issue is that the local variable importances do not sum up to the instsance prediction for which they are calculated. In Chapters **??** and **??** we will introduce measures that address this problem.

0.7.6 Code snippets for R

In this section we present key features of the R package `ingredients` which is a part of `DrWhy.AI` universe and covers all methods presented in this chapter. More details and examples can be found at https://modeloriented.github.io/ingredients/.

In this section, we use a random forest classification model developed in the chapter **??**, namely the `titanic_rf_v6` model. It is trained to predict probability of survival from sinking of Titanic. Instance level explanations are calculated for a single observation `henry` - 47 years old passenger that travels 1st class.

`DALEX` explainers for both models and the Henry data are retrieved via `archivist` hooks as listed in Chapter **??**.

```
library("randomForest")
explain_rf_v6 <- archivist::aread("pbiecek/models/9b971")

library("DALEX")
henry <- archivist::aread("pbiecek/models/a6538")
henry
```

0.7.6.1 Basic usage for the `calculate_oscillations` function

To calculate CP oscillations, we have got to calculate CP profiles for the selected observation. Let us use `henry` as the instance prediction of interest.

CP profiles are calculated by applying the `ceteris_paribus()` function to the wrapper object. Resulting object can be then processed with `calculate_oscillations()`.

```
library("ingredients")
library("ggplot2")

cp_titanic_rf <- ceteris_paribus(explain_rf_v6, henry)
```

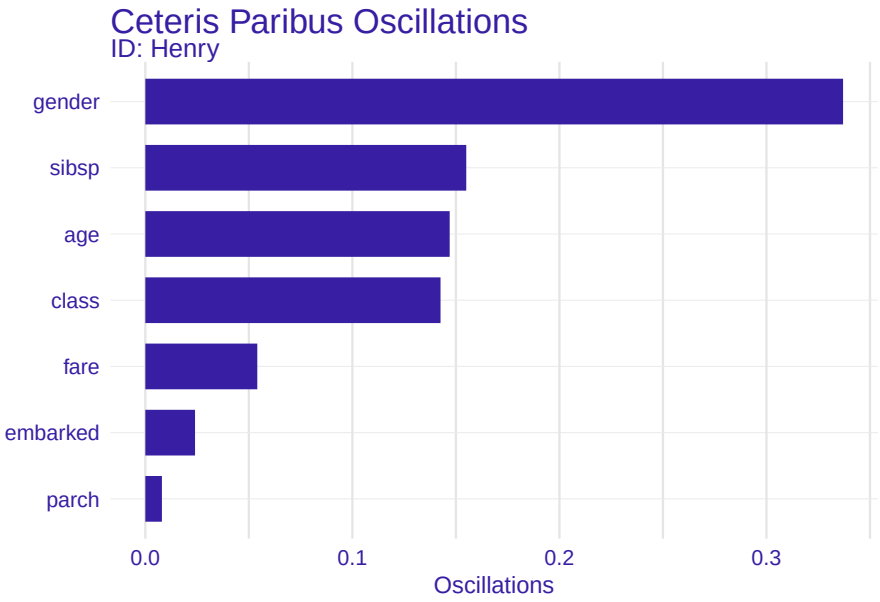
The `calculate_oscillations()` function returns an object of class `ceteris_paribus_oscillations`, which has a form of a data frame, but has also an overloaded `plot()` function. We can use the latter function to plot the local variable-importance measures for the instance of interest.

```
oscillations_titanic_rf <- calculate_oscillations(cp_titanic_rf)
oscillations_titanic_rf
```

```
##    _vname_ _ids_ oscillations
## 2   gender     1   0.33700000
## 4    sibsp     1   0.15500000
## 3      age     1   0.14700000
## 1    class     1   0.14257143
## 6     fare     1   0.05407273
## 7 embarked     1   0.02400000
## 5    parch     1   0.00800000
```

It can be then plotted with the generic `plot()`.

```
oscillations_titanic_rf$`_ids_` <- "Henry"
plot(oscillations_titanic_rf) + ggtitle("Ceteris Paribus Oscillations")
```



0.7.6.2 Advanced usage for the `calculate_oscillations` function

Note, that the `calculate_oscillations` function calculates $\widehat{vip_j^{CP}(x^*)}$ as an average absolute difference between Ceteriws Paribus profile and the model prediction for a selected instance over selected grid of values.

The choice of grid of values may influence results as the expectation will be calculated over different distributions.

The default behavior is to use all unique values as grid points. But let us consider two alternative approaches.

One would be to calculate the expectation over uniform distribution for continuous variables.

To do this we manually specify a dense uniform grid of values for each variable. Here we are doing it by setting the `variable_splits` argument.

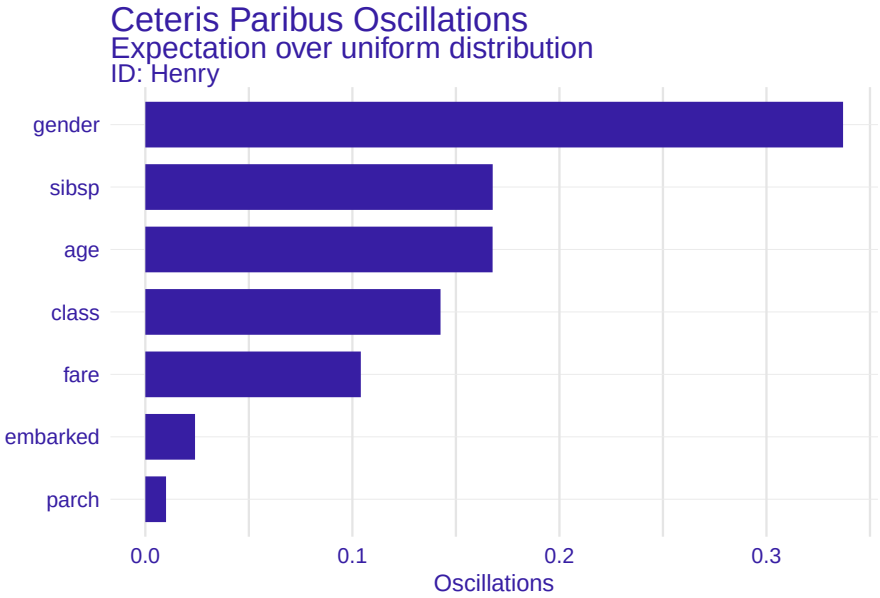
```
cp_titanic_rf_uniform <- ceteris_paribus(explain_rf_v6, henry,
              variable_splits = list(age = seq(0, 65, 0.1),
                                     fare = seq(0, 200, 0.1),
                                     sibsp = seq(0, 8, 0.1),
                                     parch = seq(0, 8, 0.1),
                                     gender = unique(titanic$gender),
                                     embarked = unique(titanic$embarked),
                                     class = unique(titanic$class)))
```

Calculation of oscillations is easy.

```
oscillations_uniform <- calculate_oscillations(cp_titanic_rf_uniform)
oscillations_uniform$`_ids_` <- "Henry"
oscillations_uniform
```

```
##    _vname_ _ids_ oscillations
## 5   gender Henry    0.3370000
## 3    sibsp Henry    0.1677778
## 1      age Henry    0.1677235
## 7    class Henry    0.1425714
## 2     fare Henry    0.1040790
## 6 embarked Henry    0.0240000
## 4    parch Henry    0.0100000
```

```
plot(oscillations_uniform) + ggtitle("Ceteris Paribus Oscillations", "Expectation over uniform distribution")
```



Another approach would be to average over empirical distribution of each variable.

To do this we manually specify grid of values as a sample from validation data.

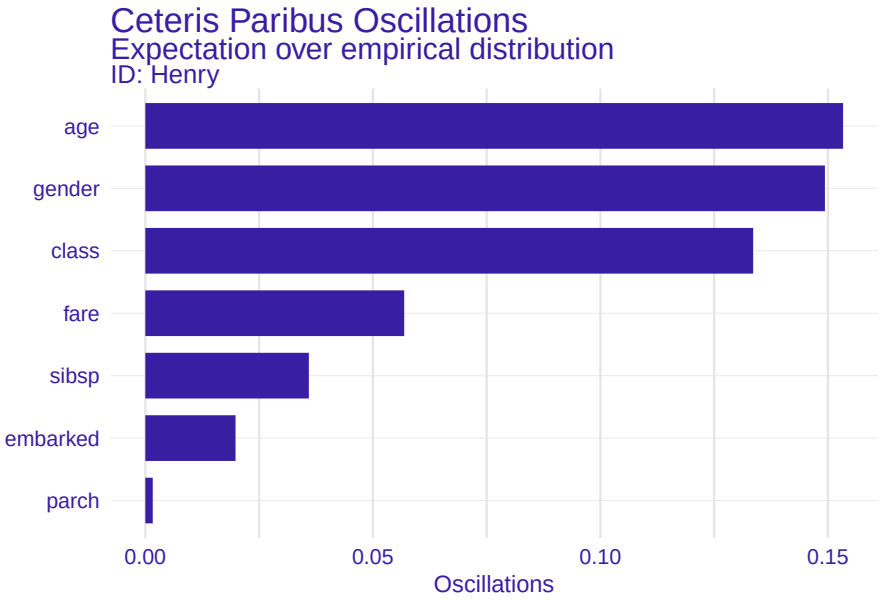
```
titanic <- na.omit(titanic)

cp_titanic_rf_empirical <- ceteris_paribus(explain_rf_v6, henry,
              variable_splits = list(age = titanic$age,
                                     fare = titanic$fare,
                                     sibsp = titanic$sibsp,
                                     parch = titanic$parch,
                                     gender = titanic$gender,
                                     embarked = titanic$embarked,
                                     class = titanic$class))
```

Having profiles we can then calculate oscillations with `calculate_oscillations`.

```
oscillations_empirical <- calculate_oscillations(cp_titanic_rf_empirical)
oscillations_empirical$`_ids_` <- "Henry"
oscillations_empirical
```

```
##    _vname_ _ids_ oscillations
## 1      age Henry  0.153323969
## 5   gender Henry  0.149336656
## 7    class Henry  0.133567739
## 2     fare Henry  0.056883552
## 3    sibsp Henry  0.035932034
## 6 embarked Henry  0.019818758
## 4    parch Henry  0.001623924
```

```
plot(oscillations_empirical) + ggtitle("Ceteris Paribus Oscillations", "Expectation over empirical distributior
```



0.8 Local Diagnostics With Ceteris-paribus Profiles

0.8.1 Introduction

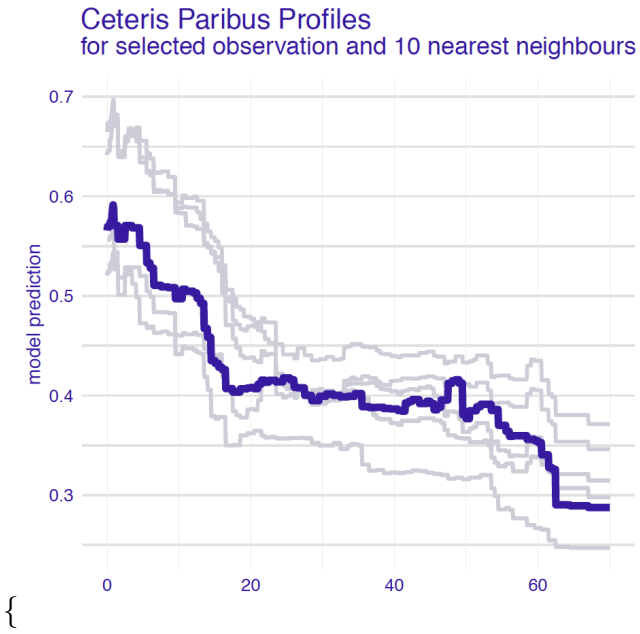
It may happen that, while the global predictive performance of the model is good, model predictions for some observations are very bad. In this chapter, we present two local-diagnostics techniques that can address this issue. IN particulat, we focus on fidelity plots: the plot of CP profiles for nearest neighbors and the local-fidelity plot.

The idea behind fidelity plots is to select a number of observatons ("neighbors") from the validation dataset that are closest to the instance (observation) of interest. Then, for the selected observations, we plot CP profiles and check how stable they are. Additionally, if we know true values of the dependent variable for the selected neighbors, we may add residuals to the plot to evaluate the local fit of the model.

0.8.2 Intuition

One approach to local model diagnostics is to examine how the model behaves for observations from the training dataset. Figure **??** presents CP profiles for the instance of interest and its 10 nearest neighbors. The profiles are almost parallel and very close to each other. This suggests that model predictions are stable around the instance of interest, because small changes in the explanatory variables (represented by the nearest neighbors) have not got much influence on the predictions.

\begin{figure}



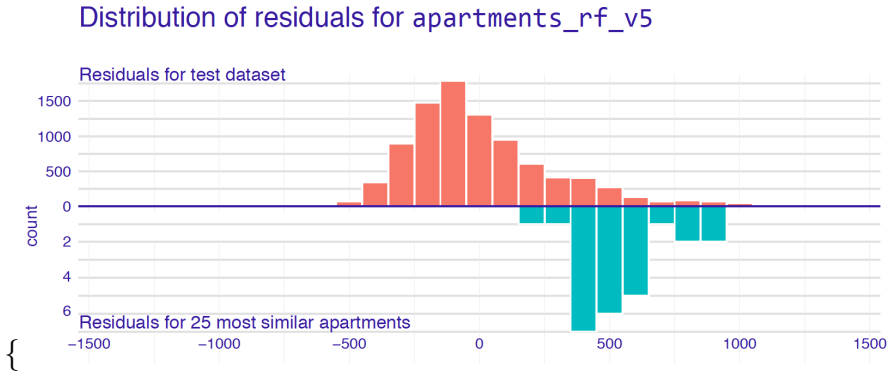
{

}

\caption{(fig:profileWith10NN) Ceteris-paribus profiles for a selected instance (dark violet line) and 10 nearest neighbors (light grey lines) for the `titanic_rf_b6` model. The profiles are almost parallel and close to each other what suggests the stability of the model.}
\end{figure}

Once we have selected the nearest neighbors, we can also look closer at the model fit around the point of interest. Figure **??** presents histograms of residuals for the entire dataset and the selected neighbors. The distribution of residuals for the entire dataset is rather symmetric and centered around 0, suggesting a reasonable average performance of the model. On the other hand, the residuals for the selected neighbors are centered around the value of 500. This sugests that, on average, the model predictions are biased for the instance of interest.

\begin{figure}



{

}

\caption{(fig:profileBack2BackHist) Histograms of residuals for the `apartments_rf_v5`. Upper panel: residuals calculated for all observations from the dataset. Bottom panel: residuals calculated for 25 nearest neighbors of the instance of interest.} \end{figure}

0.8.3 Method

The proposed method is based on three elements:

- identification of nearest neighbors,
- calculation and visualization of CP profiles for the selected neighbors, and
- analysis of residuals for the neighbors.

In what follows we discuss each of the elements in more detail.

0.8.3.1 Nearest neighbors

There are two important questions related to the selection of the neighbors "nearest" to the instance (observation) of interest:

- How many neighbors should we choose?
- What metric should be used to measure the "proximity" of observations?

The answer to both questions is, of course, *it depends*.

- The smaller the number of neighbors, the more local is the analysis. However, a very small number will lead to a larger variability of the results. In many cases we found that 20 works fine, but one shall always take into account computational time, size od data and other possibly important factors.
- The metric if very important. The more explanatory variables, the more important is the choice. In particular, the metric should be capable of accomodating variables of different natuer (categorical, continuous). Our default choice is the Gower similarity measure:

$$d_{gower}(x_i, x_j) = \frac{1}{p}\sum_{k=1}^{p} d_k(x_i^k, x_j^k),$$

where x_i is a p-dimensional vector of explanatory covariates for the i-th observation and $d_k(x_i^k, x_j^k)$ is the distance between values of the k-th variable for the i-th and j-th observations. Note that $d_k()$ depends on the nature of the variable. For instance, for a continuous variable it is equal to $|x_i^k - x_j^k|/\{max(x_1^k, \ldots, x_n^k) - min(x_1^k, \ldots, x_n^k)\}$, i.e., the absolute difference scaled by the observed range of the variable. On the other hand, for a categorical variable, it is simply $I(x_i^k = x_j^k)$, where $I()$ is the indicator function. Note that p may be equal to the number of all explanatory variables included in the model, or only a subset of them.

Once we have decided on the numer of neighbors, we can use the chosen metric to select the required number observations "closest" to the one of interest.

0.8.3.2 Profiles for neighbors

Once nearest neighbors are identified, we can graphically compare CP profiles for selected (or all) variables.

For a model with a large number of variables we may end up with a large number of plots. If the number of features is large, then we may easily end up with a large number of plots. In such a case a better strategy is to focus only on K most important variables, selected by using the variable-importance measure (see Chapter **??**).

0.8.3.3 Local fidelity plot

CP profiles are helpful to assess the model stability. In addition, we can enhance the plot by adding residuals to it to allow evaluation of the local model fit. For model $f()$ and observation i described by the vector of explanatory variables x_i, the residual is the difference between the observed and predicted value of the dependent variable Y_i, i.e.,

$$r_i = y_i - f(x_i).$$

Note that, for a binary variable `y`, the resiudal is the difference between 0 or 1, depending on how we code "success," and the predicted probability of "success."

The plot that includes CP profiles for the nearest neighbors and the corresponding residuals is called a local fidelity plot.

0.8.4 Example: Titanic data

As an example, we will use the predictions for the random forest model for the Titanic data (see Section @ref(model_titanic_rf)). We show profiles only for a single continuous explanatory variable, *age*. This makes the example easier to read without reducing the generality. [TOMASZ: NOT SURE. WOULD BE USEFUL TO SEE AN EXAMPLE FOR A FACTOR.]

Figure **??** presents a detailed explanation of the elements of a local fidelity plot. The plot includes eight nearest neighbors of Henry (see Section @ref(predictions_titanic)). Profiles are quite apart from each other, which indicates potential instability of model predictions. However, the residuals included in the plots are positive and negative, indicating that, on average, the instance prediction should not be biased.

0.8.5 Pros and cons

Local fidelity plots may be very helpful to validate if

- the model is locally additive, as for such models the CP profiles should be parallel
- the model is locally stable, as in that case the CP profiles should be close to each other

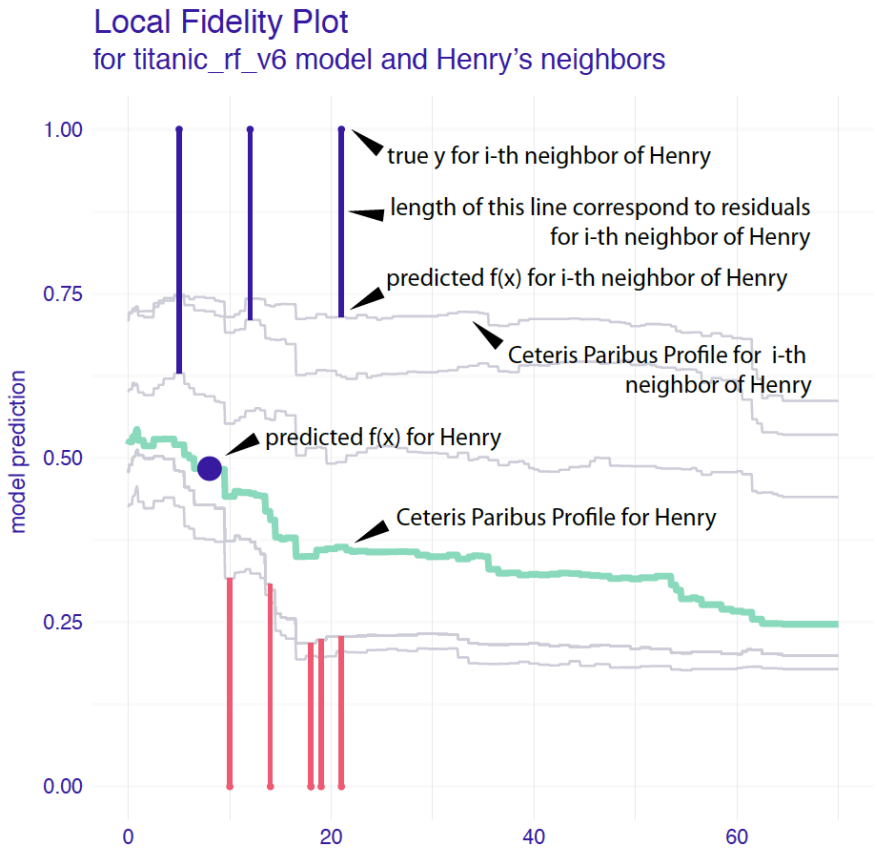


FIGURE 26 (fig:localFidelityPlots) Elements of a local fidelity plot. The green line shows the Ceteris-paribus profile for the instance of intererst. Profiles forthe nearest neighbors are marked with grey lines. The vertical intervals correspond to residuals; the shorter the interval, the smaller the residual and the more accurate prediction of the model. Blue intervals correspond to positive residuals, red intervals to negative intervals. Stable model will have profiles close to each other; additive model will have parallel lines.

- the model fit for the instance of interest is good, as in that case the residuals should be small and balanced around 0.

The drawback is that such plots are quite complex and lack objective measures of the quality of the model fit. Thus, they are mainly suitable for exploratory analysis.

0.8.6 Code snippets for R

In this section, we show how to use the R package `ingredients` to construct local fidelity plots. More details and examples can be found at `https://modeloriented.github.io/ingredients/`.

We use the random forest (**?**) model `titanic_rf_v6` developed for the Titanic dataset (see Section @ref(model_titanic_rf)) as the example. Recall that we deal with a binary classification problem - we want to predict the probability of survival for a selected passenger.

`DALEX` explainers for the model and the `Henry` data are retrieved via `archivist` hooks as listed in Chapter **??**.

```
library("randomForest")
explain_rf_v6 <- archivist::aread("pbiecek/models/9b971")

library("DALEX")
henry <- archivist::aread("pbiecek/models/a6538")
henry
```

```
##   class gender age sibsp parch fare  embarked
## 1   1st   male  47     0     0   25 Cherbourg
```

We are going to recreate Figure **??**. To do this we need some number of passangers most similar to `henry`. The easiest way to do this is to use `ingredients::select_neighbours()` function. It returns `n` observations (by default 20) most similar to a observation of interest taking `distance` measure into account (by default Gower distance is used).

In the example below we find from `titanic` dataset 10 nearest neighbors to `henry`, and the similarity is based only on two explanatory covariates, *gender*, and *class*.

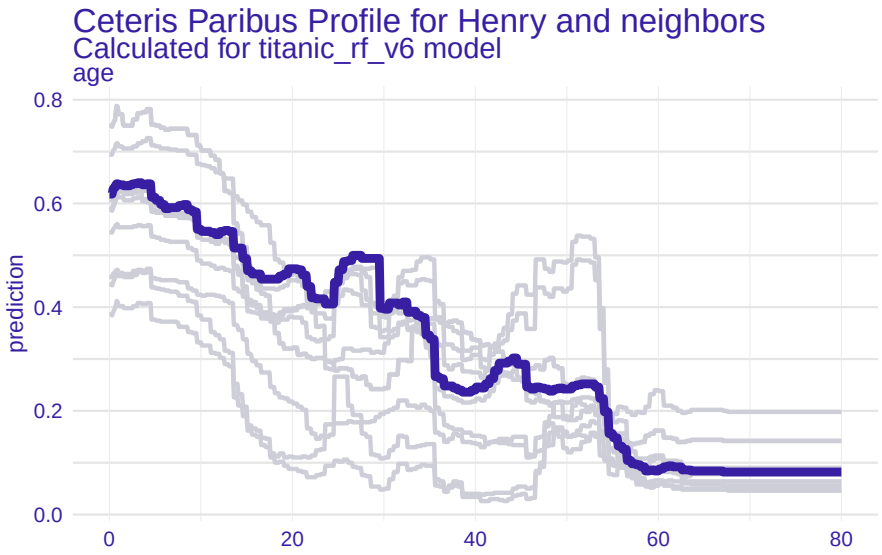
```
library("ingredients")

henry_neighbors <- select_neighbours(titanic,
                          henry,
                          n = 10,
                          variables = c("class", "gender"))
```

Now we are ready to plot profiles for henry and his neighbors. First we need to calculate corresponding profiles with `ceteris_paribus()` function introduced in Section **??**.

```
cp_henry <- ceteris_paribus(explain_rf_v6,
                            henry,
                            variable_splits = list(age = seq(0,80,0.1)))
cp_henry_neighbors <- ceteris_paribus(explain_rf_v6,
                            henry_neighbors,
                            variable_splits = list(age = seq(0,80,0.1)))
```

And we can plot both profiles. In Figure ??fig:titanicCeterisParibus05) we do this only for a single variable `age`.

```
library("ggplot2")

plot(cp_henry_neighbors, color = '#ceced9') +
  show_profiles(cp_henry, size = 2)  +
  ggtitle("Ceteris Paribus Profile for Henry and neighbors", "Calculated for titanic_rf_v6 model")
```



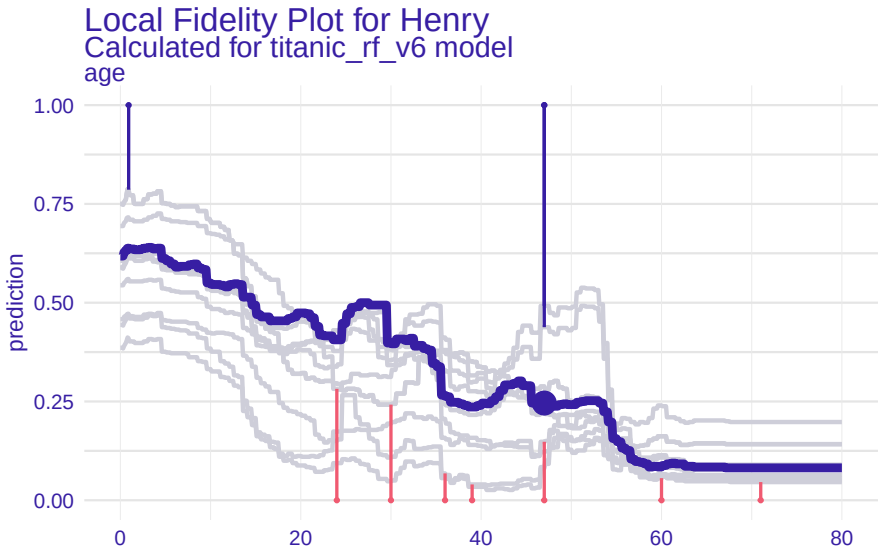
Now we will recreate the fidelity plot presented in Figure ??. To do this we need to add information about true values for selected neighbors. It can be done through the `y` argument in the `ceteris_paribus()` function.

The `y` argument takes numerical values. For our binary classification we have values `yes/no` in the `survived` variable, so to convert them to numerical values withe the `survived === "yes"` expression.

```
cp_henry_neighbors <- ceteris_paribus(explain_rf_v6,
                            henry_neighbors,
                            y = henry_neighbors$survived == "yes",
                            variable_splits = list(age = seq(0,80,0.1)))
```

Now we can add residuals the the plot with the `show_residuals()` function. As a result, we obtain the local fidelity plot for `henry`.

```
plot(cp_henry_neighbors, color = '#ceced9') +
  show_profiles(cp_henry, size = 2) +
  show_observations(cp_henry, variables = "age", size = 5) +
  show_residuals(cp_henry_neighbors, variables = "age") +
  ggtitle("Local Fidelity Plot for Henry","Calculated for titanic_rf_v6 model")
```



0.9 Break Down for Additive Variable Attributions

In the Section **??** we introduced a method for assessment of local variable importance based on Ceteris Paribus Profiles. But the main disadvantage of this method is that importance scores do not sum up to final model predictions.

In this chapter we introduce Break Down Plots which solve this problem. Note that the described method is also similar to the EXPLAIN algorithm introduced in (**?**) and implemented in (**?**) package.

0.9.1 Intuition

For any model we may repeat the intuition presented in the Section **??** to calculate variable contribution as shifts in expected model response after conditioning over consecutive variables. This intuition is presented in Figure **??**.

Panel A shows distribution of model responses. The row `all data` shows the model response of the validation dataset. The red dot stands for average model response and it is an estimate of expected model response $E[f(x)]$.

Since we want to calculate effects of particular values of selected variables we then

condition over these variables in a sequential manner. The next row in panel A corresponds to average model prediction for observations with variable `class` fixed to value `1st`. The next for corresponds to average model prediction with variables `class` set to `1st` and `age` set to `0`, and so on. The last row corresponds to model response for x^*.

Black lines in the panel A show how prediction for a single point changes after coordinate i is replaced by the x_i^*. But finally we are not interested in particular changes, not even in distributions but only in averages - expected model responses.

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 red bar. They sum up to final model prediction, which is denoted by a violet bar in this example.

0.9.2 Method

Again, as in previous chapter, let $v(f, x^*, i)$ stands for the contribution of variable x_i to 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.

Unfortunately, for non-additive models, variable contributions depend 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 $f()$.

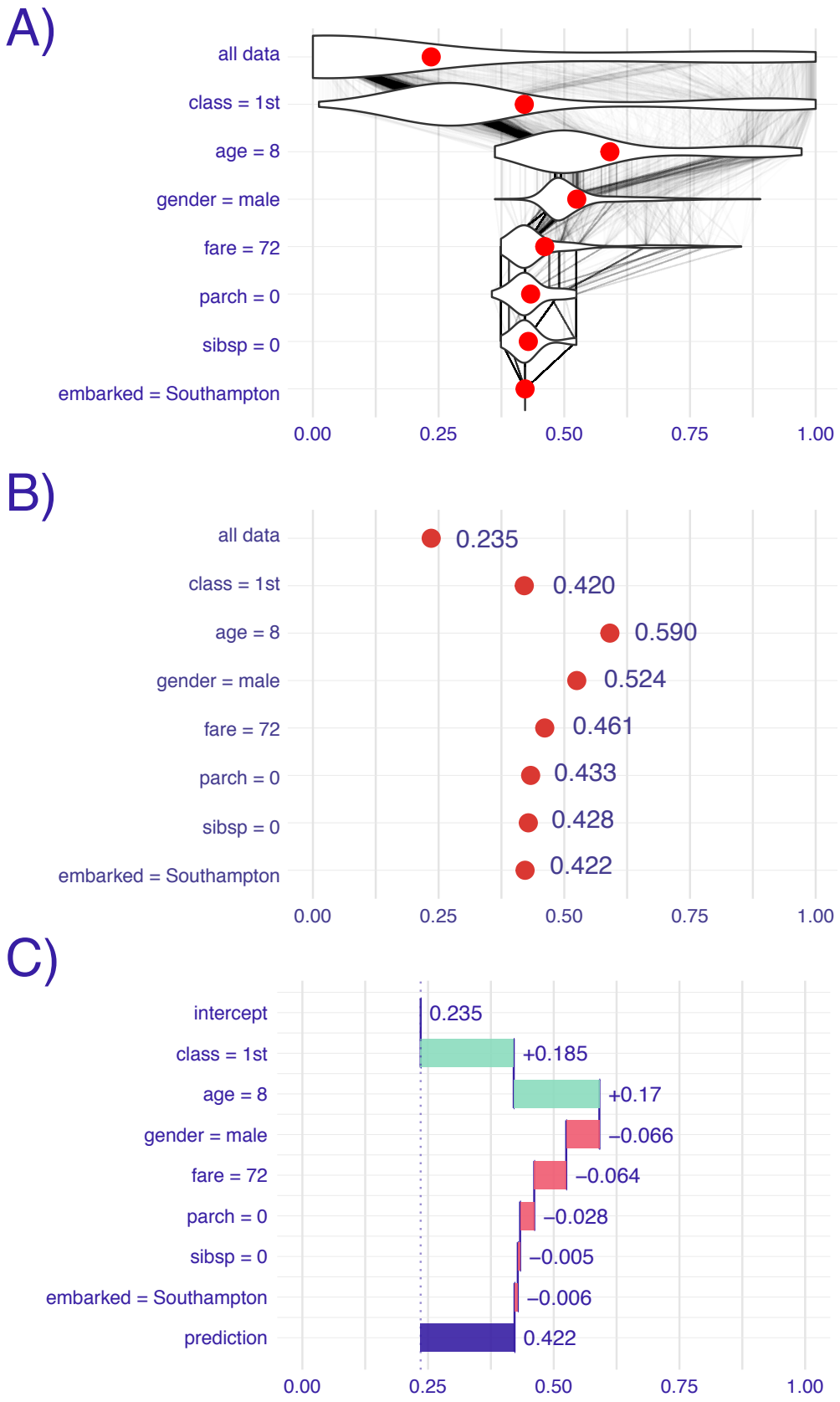


FIGURE 27 (fig:BDPrice4) Break Down Plots show how variables move the model prediction from population average to the model prognosis for a single observation. A) The first row

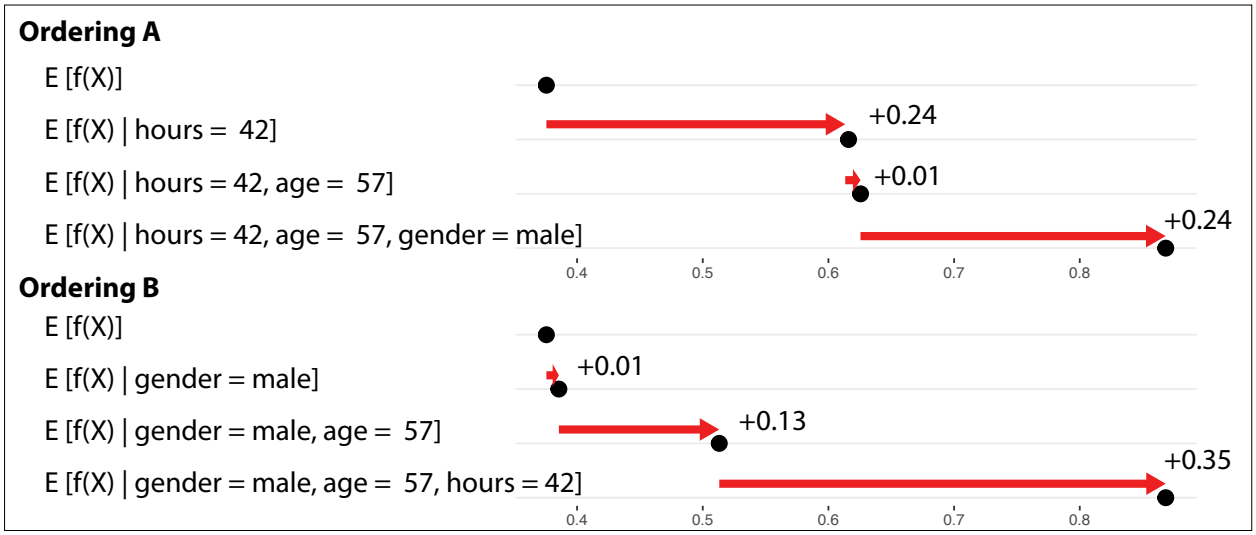


FIGURE 28 (fig:ordering) Two different paths between average model prediction and the model prediction for a selected observation. Black dots stand for conditional average, red arrows stands for changes between conditional averages.

There are different attempts to solve the problem with the ordering.

A. choose an ordering in which variables with largest contributions are first. In this chapter we will describe a heuristic behind this approach. B. identify interactions that causes difference in attributions for different orderings and show these interactions. In the chapter **??** we will describe a heuristic behind this idea. C. calculate average across all possible orderings. There is $p!$ possible orderings, be the may quite accurately approximate the average. This approach will be presented in the chapter **??**.

So, let's start with approach A. The easiest way to solve this problem is to use two-step procedure. In the first step variables are ordered and in the second step the consecutive conditioning is applied to ordered variables.

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 $O(p)$.

0.9.3 Example: Titanic data

PBI: in this section, should we replicate figures and data already presented in Figure fig:BDPrice4 ?

Old:

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
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

0.9.4 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.
- 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.

0.9.5 Code snippets for R

In this section we present key features of the `iBreakDown` package for R (**?**) which is a part of `DrWhy.AI` universe. 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://modeloriented.github.io/iBreakDown/`.

In this section, we use a random forest classification model developed in the chapter **??**, namely the `titanic_rf_v6` model. It is trained to predict probability of survival from sinking of Titanic. Instance level explanations are calculated for a single observation `henry` - 47 years old passenger that travels 1st class.

`DALEX` explainers for both models and the Henry data are retrieved via `archivist` hooks as listed in Chapter **??**.

```
library("randomForest")
explain_rf_v6 <- archivist::aread("pbiecek/models/9b971")

library("DALEX")
johny_d <- archivist::aread("pbiecek/models/e3596")
johny_d
```

0.9.5.1 Basic usage for the `break_down` function

The `iBreakDown::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 additive attributions for selected observation.

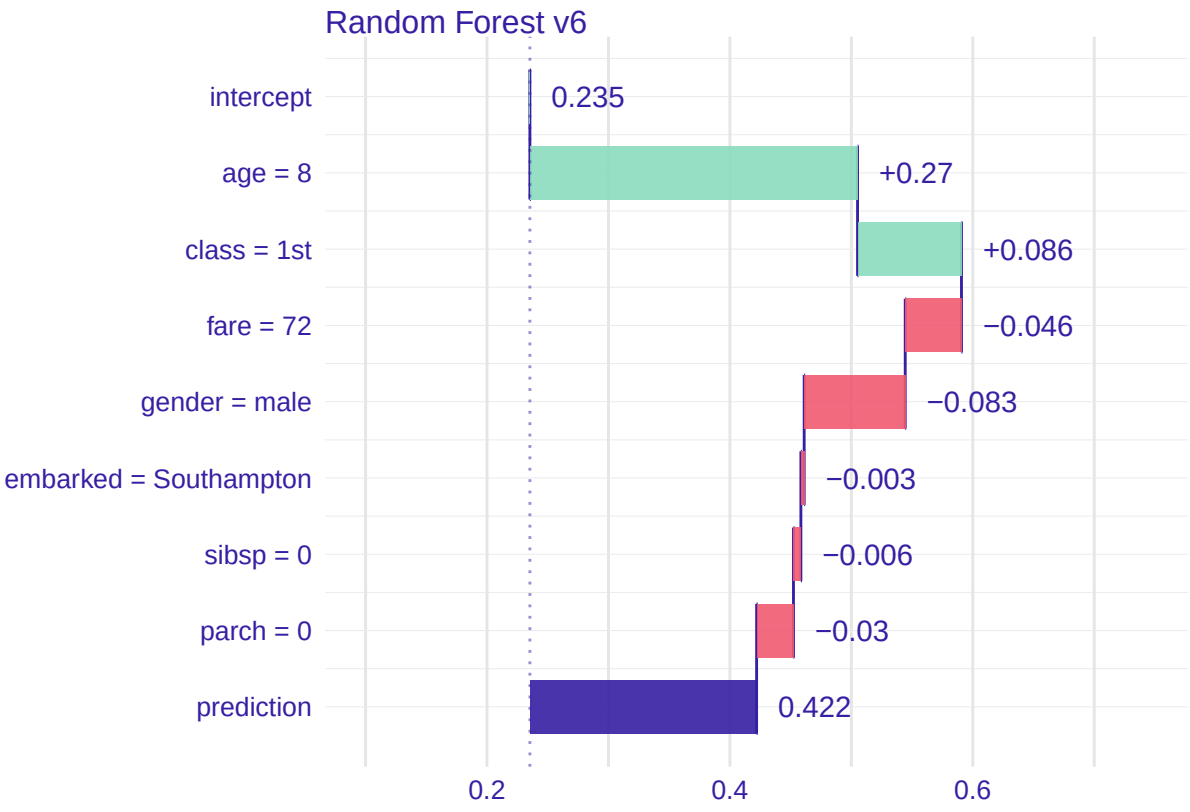
The simplest use case is to set only the arguments - model explainers and observation of interest.

```
library("iBreakDown")
bd_rf <- break_down(explain_rf_v6,
                    johny_d)
bd_rf
```

```
##                                          contribution
## Random Forest v6: intercept                     0.235
## Random Forest v6: age = 8                       0.270
## Random Forest v6: class = 1st                   0.086
## Random Forest v6: fare = 72                    -0.046
## Random Forest v6: gender = male                -0.083
## Random Forest v6: embarked = Southampton       -0.003
## Random Forest v6: sibsp = 0                    -0.006
## Random Forest v6: parch = 0                    -0.030
## Random Forest v6: prediction                    0.422
```

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

```
plot(bd_rf)
```



0.9.5.2 Advanced usage for the `break_down` function

The function `break_down()` can take more arguments. The most commonly used are:

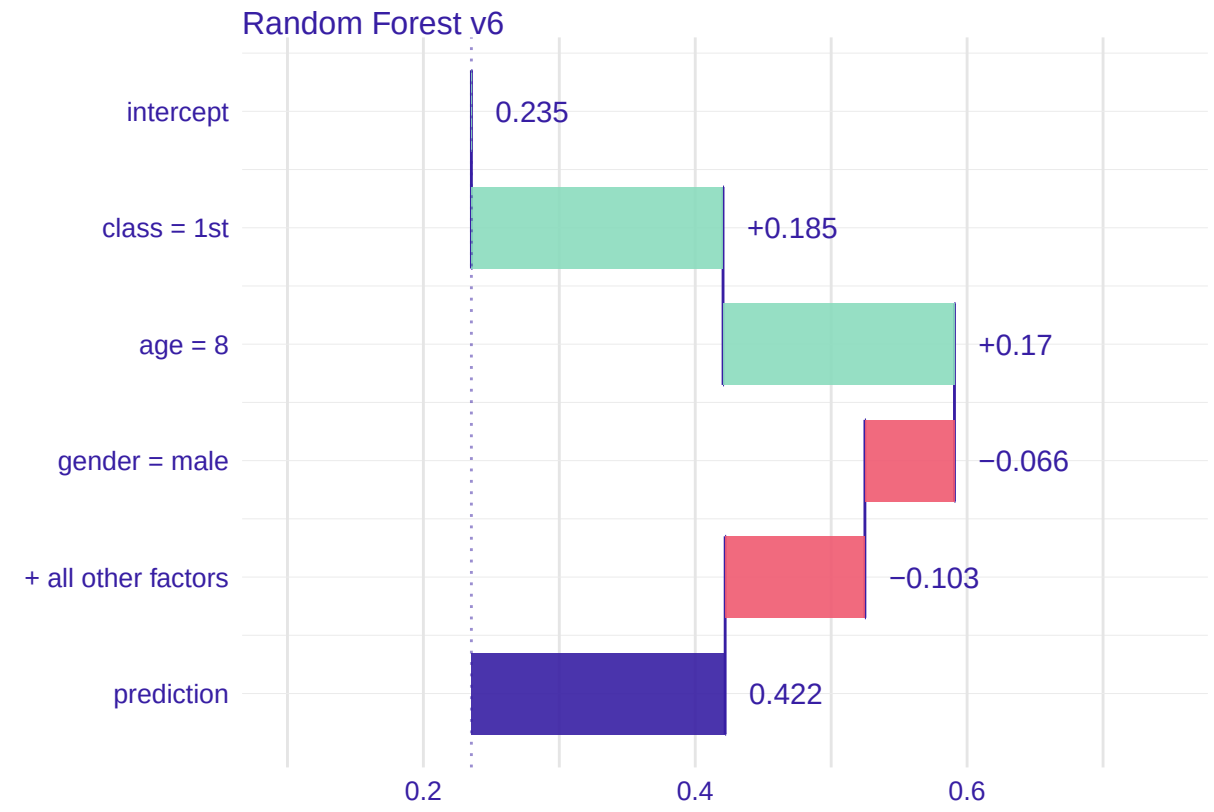
- `x` a wrapper over a model created with function `DALEX::explain()`,

- `new_observation` an observation to be explained is should be a data frame with structure that matches the training data,
- `order` if specified then it can be a vector of characters (column names) or integers (column indexes) that specify order of variable conditioning. If not specified (default) then a one-step heuristic is used to determine the order,
- `keep_distributions` logical value.
  if `TRUE`, then additional diagnostic information is about conditional distributions is stored and can be plotted with the generic `plot()` function.

Let's see these additional arguments in action.

First we will specify order. You can use integer indexes or variable names. Note that the second option is in most cases better because of higher readability. Additionally, to reduce clutter in the plot we set `max_features = 3` argument in the `plot()` function.

```
library("iBreakDown")
bd_rf_order <- break_down(explain_rf_v6,
                 johny_d,
                 order = c("class", "age", "gender", "fare", "parch", "sibsp", "embarked"))
plot(bd_rf_order, max_features = 3)
```

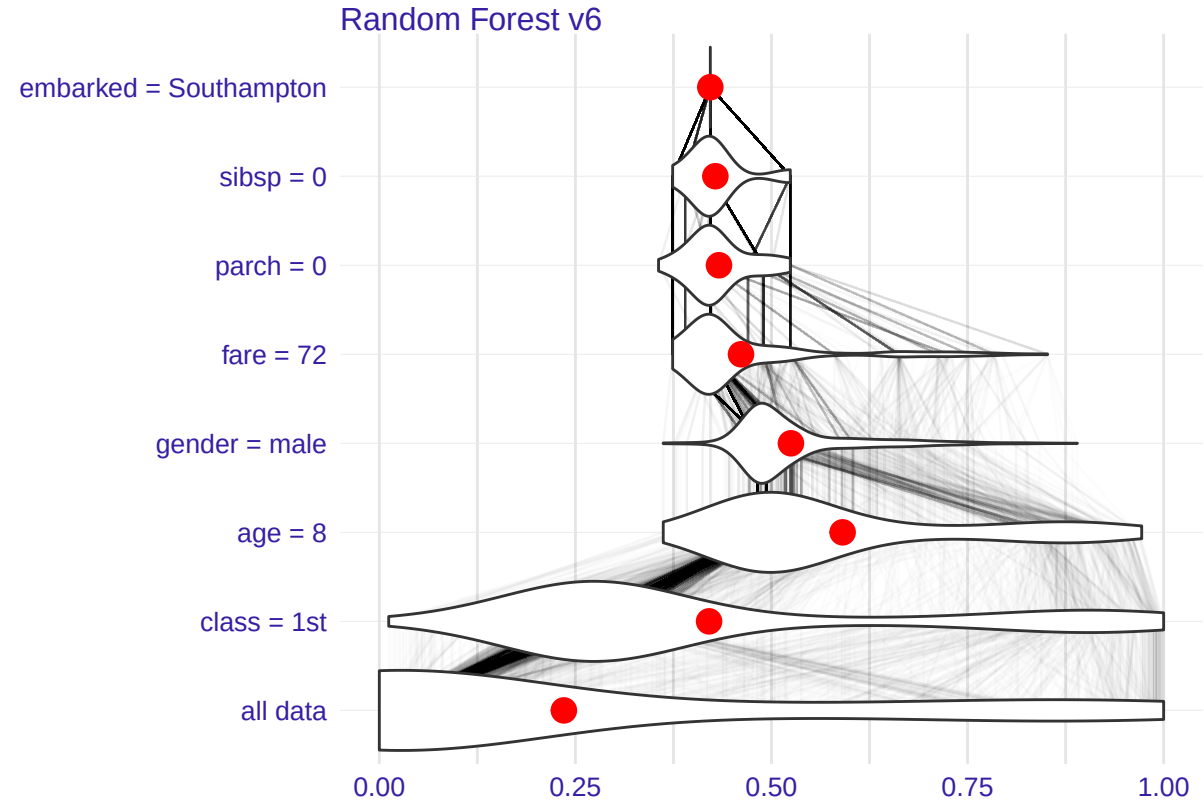


The `plot_distributions = TRUE` argument of `break_down()` function enriches model response with additional information about conditional distribution.

It can be presented after setting `plot_distributions = TRUE` in the `plot()` function. Conditional distributions are presented as vioplots. Red dots stand for conditional average

model response. Thin black lines between vioplots correspond to predictions for individual observations. With them we can trace how model predictions change after consecutive conditionings.

```
bd_rf_distr <- break_down(explain_rf_v6,
                  johny_d,
                  order = c("class", "age", "gender", "fare", "parch", "sibsp", "embarked"),
                  keep_distributions = TRUE)
plot(bd_rf_distr, plot_distributions = TRUE)
```



0.10 iBreakDown for Variable Attributions with Interactions

PBI: note that we are using completly different notations here and in the prev chapter. Let's keep only one, but which one?

In the Section **??** we presented a model agnostic approach to additive decomposition of model predictions. We also showed that for non-additive moedls the proposed attribution depends on the ordering of variables.

Lack of additivness means, that effect of one variable is modulated by another variable(s). In such pair (or larger tuple) a single variable does not contribute independently, therefore in model explanations they should be presented together.

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.

0.10.1 Intuition

First, let's see an example of an interaction between two variables. We will use real data from the Titanic dataset. Table **??** shows survival statistics for men on Titanic. For the sake of simplicity, in this example we consider only two variables - `age` and `class`. In the data `age` is a continuous variable, but again, for simplicity we have dychotomized it into two levels: boys (0-16 years old) and adults (17+ years old).

Suppose that we would like to explain factors that contribute to the survival of kids from the second class. As we can read, the survival for young passengers from 2nd class is 91.7% (survived 11 out of 12 male passengers in this group). It is higher than survival for men on titanic which is 20.5% (survived 352 out of 1716 men). So the question is, how age and class contribute to this higher survival?

Let us consider two explanations, that correspond to two different orderings:

- Overall men survival is 20.5%, but when we condition on male passengers from 2nd class the survival is even lower, i.e. 13.5%. Thus effect of the 2nd class is negative, it decreases the probability of survival by 7 percent points. Being a kid in the 2nd class is very lucky though. It changes of survival increase from 13.5% (male 2nd class) to 91.7% (boys from 2nd class). It's an increase by 78.2 percent points. So the contributions are -7% for class and +78.2% for age.
- Overall men survival is 20.5%, but when we condition on young man then the survival is higher, i.e. 40.7%. Thus the effect of age is positive, being a boy increases the probability of survival by 20.2 percent points. Being a kid in 2nd class is even better, it changes the survival increase from 40.7% (boys) to 91.7% (boys from 2nd class). It's an increase by 51 percent points. So the contributions are +51% for class and +20.2% for age.

As we see these two paths leads to two very different explanations. They differ not only in the size but also in the sign of attributed importance. It has happend because one variable modulate effect on the second variable as the moedl is not additive. Below we will show how to deal with such cases.

TABLE 0.7: Survival rates for men on Titanic. Ratios show how many survived in each Class/Age category.

Class / Age	Kids (0-16)	Adults (>16)	Total
1st	5/5 = 100%	57/175 = 32.6%	62/180 = 34.4%
2nd	11/12 = 91.7%	13/166 = 7.8%	24/178 = 13.5%
3rd	17/61 = 27.9%	58/430 = 13.5%	75/491 = 15.3%
deck crew		43/66 = 65.2%	43/66 = 65.2%

Class / Age	Kids (0-16)	Adults (>16)	Total
engineering crew		71/324 = 21.9%	71/324 = 21.9%
restaurant staff		1/67 = 1.5%	1/67 = 1.5%
victualling crew	0/3 = 0%	76/407 = 18.7%	76/410 = 18.5%
Total	33/81 = 40.7%	319/1635 = 19.5%	352/1716 = 20.5%

The key intuition behind an iBreakDown algorithm is to include variable interactions to the visual explanations. To do this we need to identify some candidated for interactions. Here we propose a very simple algorithm that will do this in two steps.

1. First it will calculate variable contributions for each variable independently.
2. Second it will calculate joint effect for each pair of variables. If this effect is different than the sum of separate variables then such pair is identified as candidate for an interaction.

0.10.2 Method

Identification of interactions in the model is performed in three steps (**?**)

- Calculate a *single-step* contribution for each variable.
- Calculate a *single-step* contribution for every pair of variables. Subtract individual contributions to assess the size of non additivness.
- Order interaction effects and additive effects in a list to determin the final order for conditioning/explanations.

This simple intuition may be generalized into higher order interactions.

0.10.2.1 Single step contributions

For a feature x_i we may define a single-step contribution as

$$\Delta_i = \mathbb{E}[f(x)|x_i = x_i^*] - \mathbb{E}[f(x)].$$

The expected model prediction $\mathbb{E}[f(x)]$ is sometimes called baseline or intercept and may be denoted as $\Delta_\varnothing$.

Expected value $\mathbb{E}[f(x)|x_i = x_i^*]$ corresponds to an average prediction of a model f if feature x_i is fixed on x_i^* coordinate from the observation to explain x^*.

I.e. the Δ_i is the difference between expected model response after conditioning on i variable minus the expected model response. Δ_i measures a naive single-step local variable importance, it indicates how much the average prediction of model f changes if feature x_i is set on x_i^*.

0.10.2.2 Two steps contributions

For a pair of variables x_i, x_j we introduce a single-step contribution as

$$\Delta_{ij} = \mathbb{E}[f(x)|x_i = x_i^*, x_j = x_j^*] - \mathbb{E}[f(x)].$$

And non additive component of this contribution as

$$\Delta_{ij}^I = \mathbb{E}[f(x)|x_i = x_i^*, x_j = x_j^*] - \mathbb{E}[f(x)|x_i = x_i^*] - \mathbb{E}[f(x)|x_j = x_j^*] + \mathbb{E}[f(x)].$$

Or equivalently

$$\Delta_{ij}^I = \Delta_{ij} - \Delta_i - \Delta_j.$$

0.10.2.3 Sequential contributions

The Δ_{ij}^I is the difference between collective effect of variables x_i and x_j denoted as Δ_{ij} and their additive effects Δ_i and Δ_j. Therefore, Δ_{ij}^I measures the importance of local lack-of-additivnes (aka. interaction) between features i and j. For additive models Δ_{ij}^I should small~for any i, j.

Note that contributions Δ_i do not sum to final model prediction. We only use them to determine the order of features in which the instance shall be explained. To calculate contributions that have the property of *local accuracy* we need to introduce one more symbol, that corresponds to the added contribution of feature i to the set of features~J.

$$\Delta_{i|J} = \mathbf{E}[f(X)|x_{J\cup\{i\}} = x^*_{J\cup\{i\}}] - \mathbf{E}[f(X)|x_J = x^*_J] = \Delta_{J\cup\{i\}} - \Delta_J.$$

And for pairs of features

$$\Delta_{ij|J} = \mathbf{E}[f(X)|x_{J\cup\{i,j\}} = x^*_{J\cup\{i,j\}}] - \mathbf{E}[f(X)|x_J = x^*_J] = \Delta_{J\cup\{i,j\}} - \Delta_J.$$

Once the order of single-step importance is determined based on Δ_i and Δ_{ij}^I scores, the final explanation is the attribution to the sequence of $\Delta_{i|J}$ scores. These contributions sum up to the model predictions, because

$$\Delta_{1,2...p} = f(x^*) - E[f(X)].$$

This approach can be generalized to interactions between any number of variables.

The complexity of the calculation of single step attributions is $O(p)$ where p stands for the number of variables, wile complexity for all pairs is $O(p^2)$. The complexity of the consecutive conditioning is $O(p)$, thus the complexity of whole algorithm is $O(p^2)$.

0.10.3 Example: Titanic

In this example we will use a random forest model for Titanic data and Johny D example - an 8 years old boy from 1st class.

In Table **??** we showed expected model responses $\mathbb{E}[f(x)|x_i = x_i^*, x_j = x_j^*]$, single-step effects Δ_{ij} and non-additive effects Δ_{ij}^I for each variable and each pair of variables. All these values are calculated locally for Johny D. These values are sorted along local importance, most important to the top.

Based on this ordering a following sequence of variables are indentified as informative: `age`, `fare:class`, `gender`. `embarked`, `sibsp` and `parch`.

Once the ordering is specified, in the table **??** we showed how the sequential attribution is calculated. These values are then presented in the iBreakDown plot **??**.

TABLE 0.8: For each variable and each pair of variables we calculated the expected conditional model response, the difference between conditional model response and the baseline and for pairs of variables the non-additive contribution. Rows are sorted according to the absolute value of the last column (if provided).

Variable	$E[f(x)\|x_{ij} = x_{ij}^*]$	Δ_{ij}	Δ_{ij}^I
age	0.505	0.270	
fare:class	0.333	0.098	-0.231
class	0.420	0.185	
fare:age	0.484	0.249	-0.164
fare	0.379	0.143	
gender	0.110	-0.125	
age:class	0.591	0.355	-0.100
age:gender	0.451	0.215	0.070
fare:gender	0.280	0.045	0.027
embarked	0.225	-0.011	
embarked:age	0.504	0.269	0.010
parch:gender	0.100	-0.136	-0.008
sibsp	0.243	0.008	
sibsp:age	0.520	0.284	0.007
sibsp:class	0.422	0.187	-0.006
embarked:fare	0.374	0.138	0.006
sibsp:gender	0.113	-0.123	-0.005
fare:parch	0.380	0.145	0.005
parch:sibsp	0.236	0.001	-0.004
parch	0.232	-0.003	
parch:age	0.500	0.264	-0.002
embarked:gender	0.101	-0.134	0.002

Variable	$E[f(x)\|x_{ij} = x^*_{ij}]$	Δ_{ij}	Δ^I_{ij}
embarked:parch	0.223	-0.012	0.001
fare:sibsp	0.387	0.152	0.001
embarked:class	0.409	0.173	-0.001
gender:class	0.296	0.061	0.001
embarked:sibsp	0.233	-0.002	0.001
parch:class	0.418	0.183	0.000

TABLE 0.9: Based on identified order we calculated the expected conditional model response and the difference between conditional model response with and without a specified variable. These values are plotted in the iBreakDown plot.

Variable	$\Delta_{i\|J}$	$\Delta_{J\cup\{i\}}$
intercept		0.235
age = 8	0.269	0.505
fare:class = 72:1st	0.039	0.544
gender = male	-0.083	0.461
embarked = Southampton	-0.002	0.458
sibsp = 0	-0.006	0.452
parch = 0	-0.030	0.422

0.10.4 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.

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 pairs grow faster than number of variables.
- Identification of interaction is not based on significance testing, it's purely based on absolute empirical effects, thus for small samples this procedure is prone to errors.

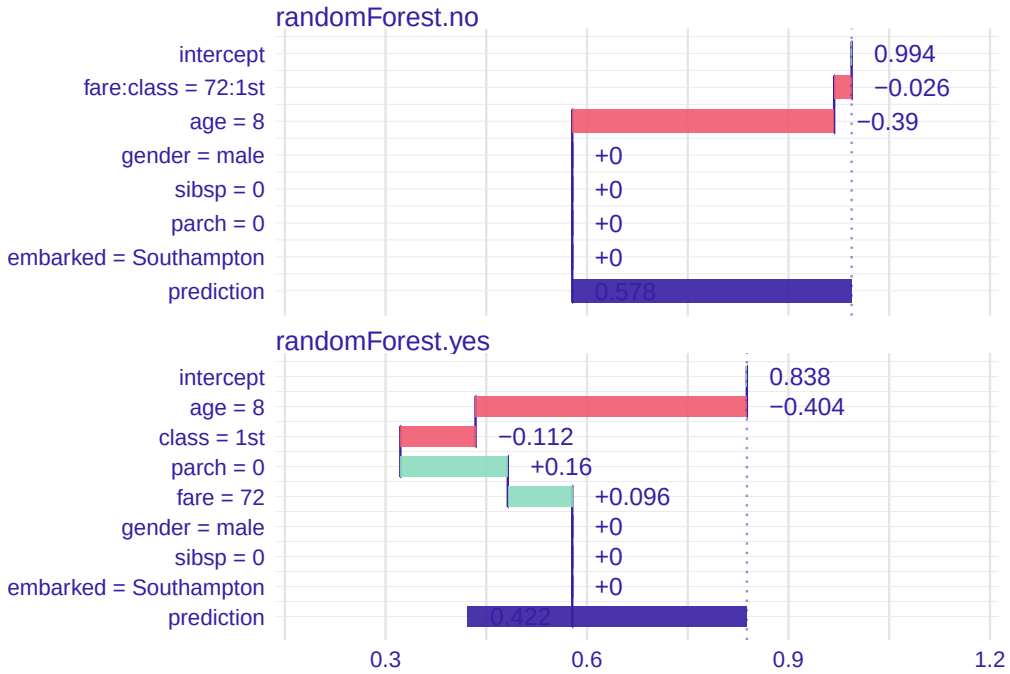


FIGURE 29 (fig:iBreakDownTitanicExamplePlot) Break Down Plots with interactions for Johny D.

0.10.5 Code snippets for R

In this section we present key features of the `iBreakDown` 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://modeloriented.github.io/iBreakDown/`.

All steps are very similar to these presented in the previouse chapter for variable attributions. The only difference is that the `break_down()` function will now take `interactions = FALSE` argument.

Model preparation

As in previous chapters we will use the random forest (**?**) model `titanic_rf_v6` developed for the Titanic dataset (see Section **??**). Using the same model will help us (1) to understand how the Break Down method works, (2) to compare these explanations against methods presented in previous chapters.

So let restore the `titanic_rf_v6` model.

```
library("DALEX")
library("randomForest")

titanic <- archivist::aread("pbiecek/models/27e5c")
titanic_rf_v6 <- archivist::aread("pbiecek/models/31570")
```

Model exploration with the `iBreakDown` 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 (**?**).

This function was introduced in detailes in the **??** section.

```
explain_titanic_rf <- explain(model = titanic_rf_v6,
                              data = titanic[,-9],
                              y = titanic$survived == "yes",
                              label = "Random Forest v6")
```

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.

Here again we will use a data frame `johny_d` with a single row, that describes an 8-years old boy that travels in the first class without parents and siblings. Then, we obtain the model prediction for this instance with the help of the 'predict()' function.

```
johny_d <- data.frame(
  class = factor("1st", levels = c("1st", "2nd", "3rd", "deck crew", "engineering crew",
                                   "restaurant staff", "victualling crew")),
  gender = factor("male", levels = c("female", "male")),
  age = 8,
  sibsp = 0,
  parch = 0,
  fare = 72,
  embarked = factor("Southampton", levels = c("Belfast", "Cherbourg", "Queenstown", "Southampton"))
)

predict(explain_titanic_rf, johny_d)
```

```
## [1] 0.422
```

3. Calculate Break Down decomposition

The `iBreakDown::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 additive attributions for selected observation.

The simplest use case is to set only the arguments - model explainers and observation of interest.

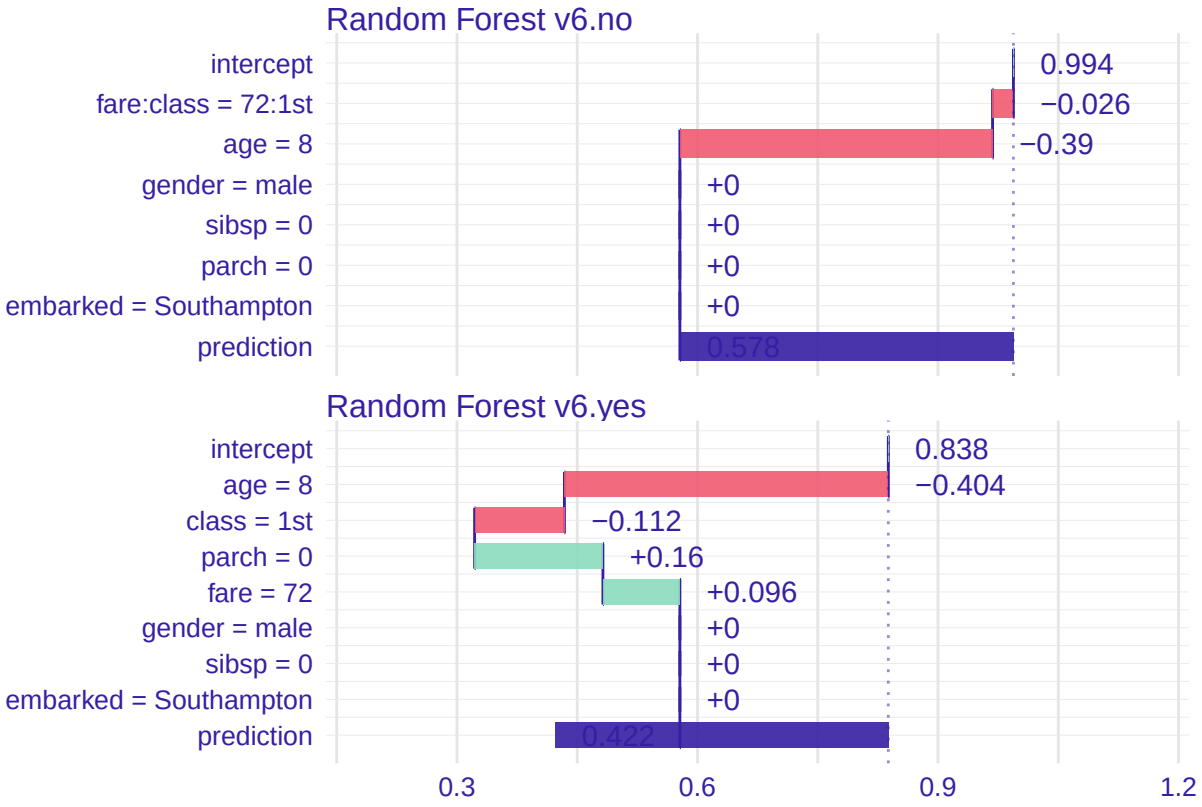
By default only additive attributions are calculated. Use `interactions = TRUE` argument to look for interactions.

```
library("iBreakDown")
bd_rf <- break_down(explain_titanic_rf,
                    johny_d,
                    interactions = TRUE)

bd_rf
```

```
##                                             contribution
## Random Forest v6.no: intercept                     0.994
## Random Forest v6.no: fare:class = 72:1st          -0.026
## Random Forest v6.no: age = 8                      -0.390
## Random Forest v6.no: gender = male                 0.000
## Random Forest v6.no: sibsp = 0                     0.000
## Random Forest v6.no: parch = 0                     0.000
## Random Forest v6.no: embarked = Southampton        0.000
## Random Forest v6.no: prediction                    0.578
## Random Forest v6.yes: intercept                    0.838
## Random Forest v6.yes: age = 8                     -0.404
## Random Forest v6.yes: class = 1st                 -0.112
## Random Forest v6.yes: parch = 0                    0.160
## Random Forest v6.yes: fare = 72                    0.096
## Random Forest v6.yes: gender = male                0.000
## Random Forest v6.yes: sibsp = 0                    0.000
## Random Forest v6.yes: embarked = Southampton       0.000
## Random Forest v6.yes: prediction                   0.422
```

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

```
plot(bd_rf)
```



0.11 SHapley Additive exPlanations (SHAP) and Average Variable Attributions

In the Section **??** we show a procedure that attributes parts of model prediction to input features. We also show that in the presence of interactions attributions depend on the feature ordering. One solution to this problem is introduced in section **??** - just find an ordering that put most important features to the front. Other solution is introduced in the Section **??** - identify interactions and show interactions in model explanations.

In this section we introduce another, very popular approach that deal with feature ordering. Basically, the problem of ordering is solved by averaging over all possible orderings. Or at least some large number of sampled orderings. Additionally, such average is closely linked with Shapley values developed originally for cooperative games.

This approach was first introduced in (**?**) and (**?**). Wide adoption of this method comes with a NIPS 2017 paper (**?**) and python library SHAP (**?**). Authors of the SHAP (SHapley Additive exPlanations) method introduced an efficient algorithm for tree-based models (**?**) and show that Shapley values is an unification of a collection of different commonly used techniques for model explanations.

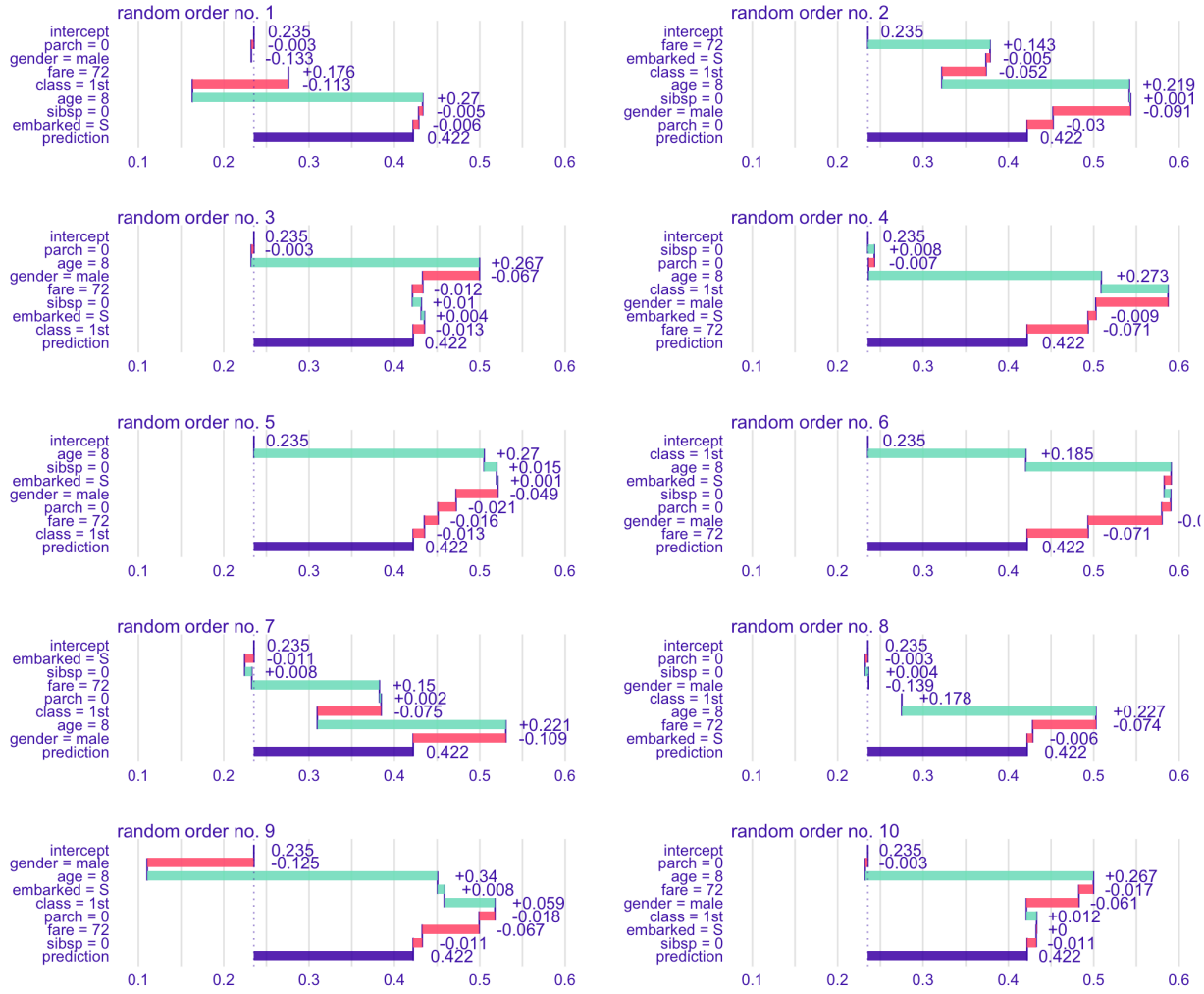


FIGURE 30 (fig:shap10orderings) Break Down plots for 10 random orderings. Each panel shows a single ordering

0.11.1 Intuition

Figure **??** shows Break Down attributions for 10 random orderings for Titanic dataset. As we see there are differences, most striking are that depending on the order features like `fare` or `class` may have positive or negative effect.

SHAP attributions are averages across all (or at least large number) of different orderings. See for example Figure **??**. In a single plot we summarize all orderings from Figure **??**. Violet boxplots show distributions for attributions for a selected variable.

0.11.2 Method

SHapley Additive exPlanations are based on *Shapley Values*, a solution concept in cooperative game theory developed by Lloyd Shapley.

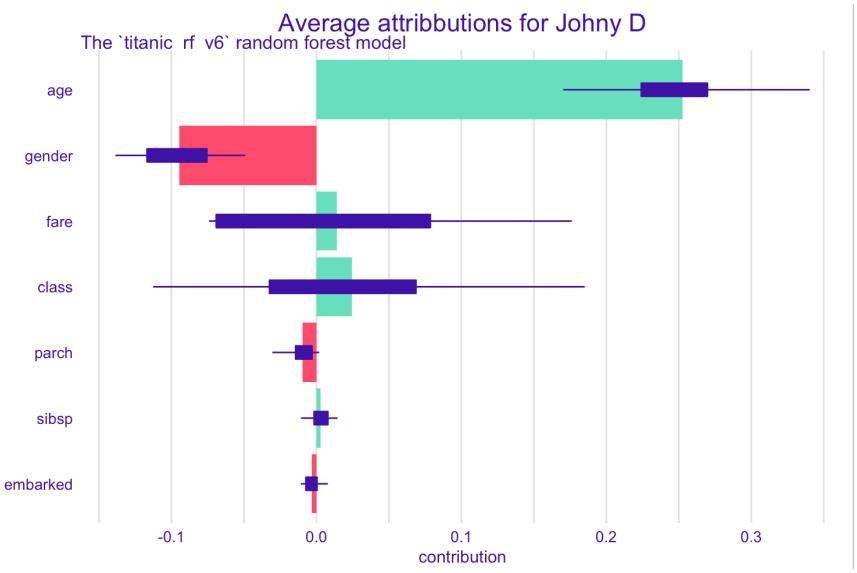


FIGURE 31 (fig:shapOrdering) Summary for 10 random orderings. Boxplots show distribution of feature attributions. Bars stand for average attributions.

Consider a following problem. A coalition of players cooperates, and obtains a certain overall gain from that cooperation. Players are not identical, different players may have different importance. Cooperation is beneficial, from cooperation they got more than from individual actions. The problem to solve is how to distribute the generated surplus among the players? The Shapley value provides one possible fair answer to this question (**?**).

Now let's translate this problem to machine learning settings. Instead of players we have features and instead of coalitions we have specific settings of values for features in the coalition. The payoff from a coalition is the model response for a selected setting. Problem to solve: how to distribute model response to particular features. Shapley values are defined for a single instance x^*. The idea of using Shapley values for feature attribution was introduced in (**?**). Here we present a different notation more suited with approach presented in previous sections.

Let $v(S)$ stand for value of coalition of S features, defined as

$$v(x^*, S) = E[f(X)|X_S = x^*_S].$$

The value is defined as expected model response given features in the set S are set to values in the selected instance x^*. Expected value averages across all features that are not in the set S.

A special case is for empty coalition. Its value is an expected model response

$$v(x^*, \emptyset) = E[f(X)].$$

Shapley values may be defined as

$$\varphi(i) = \frac{1}{p!}\sum_{\pi}[v(x^*, \pi(i) \cup \{i\}) - v(x^*, \pi(i))]$$

where p is a number of all features, $p!$ is number of all possible orderings, π is an ordering and $\pi(i)$ are all features in the ordering π that appear before feature i. Thus the $v(\pi(i) \cup \{i\}) - v(\pi(i))$ corresponds to a difference in value of a coalition $v(\pi(i))$ when feature i is added to it.

Of course for large p it is not feasible to consider all $p!$ permutations. A Monte Carlo estimator of this value was introduced in (**?**) and efficient implementation of Shapley values was introduced in (**?**). Later in this chapter we will use a crude estimator on $\varphi(i)$ in which instead of all $p!$ permutations we average across B randomly selected permutations.

Alternative formulation of Shapley values averages across coalitions not orderings.

$$\varphi(i) = \frac{1}{p} \sum_{S \subseteq \{1:p\} \setminus \{i\}} \binom{p-1}{|S|}^{-1} [v(x^*, S \cup \{i\}) - v(x^*, S)]$$

Note that the number of all subsets is 2^{p-1} is much smaller than number of all orderings $p!$. Binomial coefficients weight according to number of ordering with selected prefix coalition.

Properties

Shapley values are proven to be fair. And here fairness means that they are a single unique solution with following properties. Proved for cooperative games and then translated to machine learning.

- Symmetry. If two features are interchangeable, i.e. contribute equally to all coalitions

$$\forall_S v(x^*, S \cup \{i\}) = v(x^*, S \cup \{j\})$$

then they should have equal Shapley values

$$\varphi(i) = \varphi(j).$$

- Dummy feature. If a features does not contribute to any coalitions

$$\forall_S v(x^*, S \cup \{i\}) = v(x^*, S)$$

then it should have Shapley value equal to 0

$$\varphi(i) = 0.$$

- Additivity. If a model f is sum of two other models g and h then Shapley value calculated for model f is a sum of Shapley values for g and h.

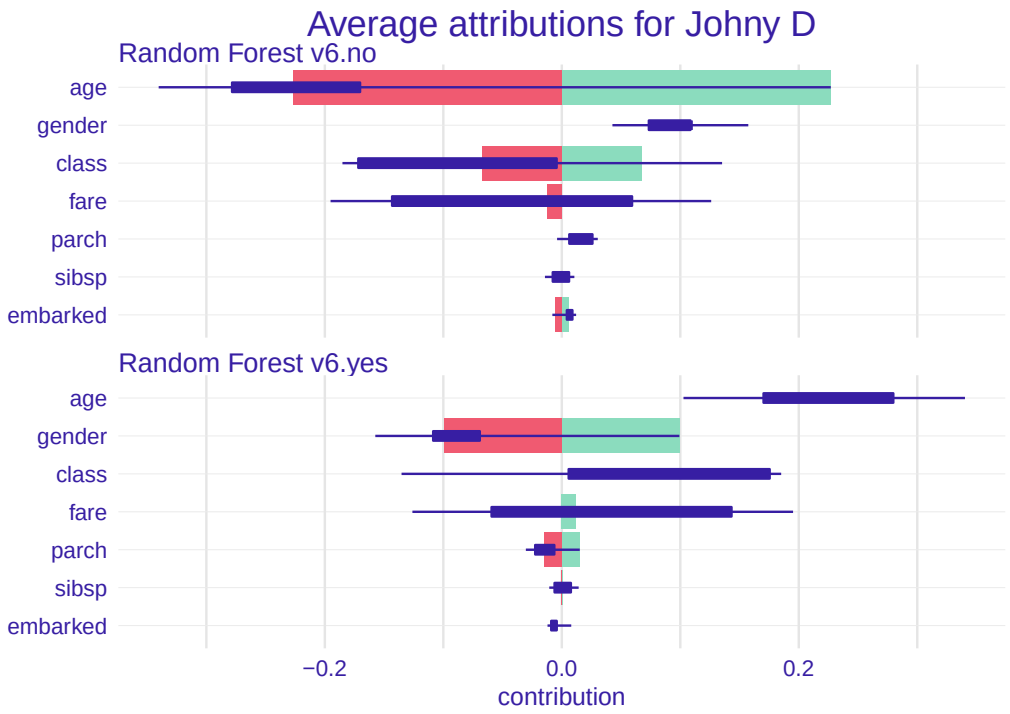


FIGURE 32 (fig:shappJohny02) Average attributions for Johny D. Violet boxplots show distributions of attributions.

- Local accuracy. Sum of Shapley values is equal to the model response

$$f(x^*) - v(x^*, \emptyset) = \sum_{i=1}^{p} \varphi(i).$$

0.11.3 Titanic

Let us again consider explanation for prediction of the `titanic_rf_v6` model for *Johny D*, an 8-years old boy from 1st class.

In Figure **??** we have presented distribution of attributions for random 25 orderings. As we see, young age of Johny D has positive effect in all orderings. An average age-effect is equal 0.2525. Similarly, effect of being male is in all cases negative for this model, on average the negative effect is −0.0908.

Things get complicated for `fare` and `class` features. Depending on the order one or another is largely positive or negative. In the section **??** we showed them as a pair, which should not be separated. Here we show average attributions for each feature.

Note, that in most applications the detailed information about distribution of orderings will be unnecessary complicated. So it is more common to keep only information about Shapley values as it is presented in Figure **??**.

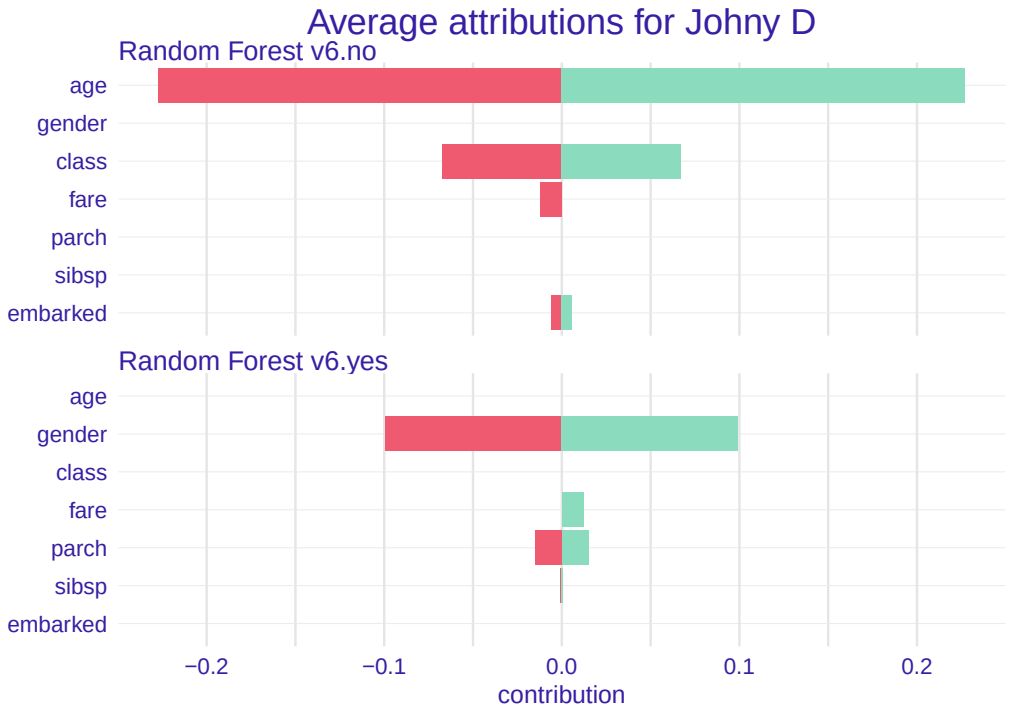


FIGURE 33 (fig:shappJohny01) Average attributions for Johny D.

Table **??** shows average attributions for Johny D.

TABLE 0.10: Average attributions for Johny D.

feature	avg. attribution
age = 8	0.2525
class = 1st	0.0246
embarked = Southampton	-0.0032
fare = 72	0.0140
gender = male	-0.0943
parch = 0	-0.0097
sibsp = 0	0.0027

0.11.4 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, like DeepLIFT, Layer-Wise Relevance Propagation, LIME.
- There is an efficient implementation available for Python and ports or reimplementations for R.

Note that there are also other R packages that offer similar functionality, like `shapper` (?) which is a wrapper over SHAP python library (?) and `iml` (?).

In this section, we use the random forest (?) model `titanic_rf_v6` developed for the Titanic dataset (see Section ??).

So let restore the `titanic_rf_v6` model and explainer created with the `explain()` function from `DALEX` package (?).

```
library("DALEX")
library("randomForest")

titanic <- archivist::aread("pbiecek/models/27e5c")
titanic_rf_v6 <- archivist::aread("pbiecek/models/31570")
explain_titanic_rf <- explain(model = titanic_rf_v6,
                              data = titanic[,c(1:4,6:8)],
                              y = titanic$survived == "yes",
                              label = "Random Forest v6")
```

Here again we will use a data frame `johny_d` with a single row, that describes an 8-years old boy that travels in the first class without parents and siblings. Then, we obtain the model prediction for this instance with the help of the 'predict()' function.

```
johny_d <- data.frame(
  class = factor("1st", levels = c("1st", "2nd", "3rd", "deck crew", "engineering crew",
                                   "restaurant staff", "victualling crew")),
  gender = factor("male", levels = c("female", "male")),
  age = 8,
  sibsp = 0,
  parch = 0,
  fare = 72,
  embarked = factor("Southampton", levels = c("Belfast", "Cherbourg", "Queenstown", "Southampton"))
)

predict(explain_titanic_rf, johny_d)
```

```
## [1] 0.422
```

First, we will recreate Figure ??. To do this we use function `iBreakDown::shap()` that calculates `B` random orderings and average Shapley contributions. This function takes an explainer created with `DALEX::explain()` function and an observation for which attributions shall be calculated. Additionally one can specify `B` number of orderings to sample.

The generic function `plot()` shows Shapley values with corresponding boxplots.

```
library("iBreakDown")

shap_johny <- shap(explain_titanic_rf, johny_d, B = 25)
plot(shap_johny)
```

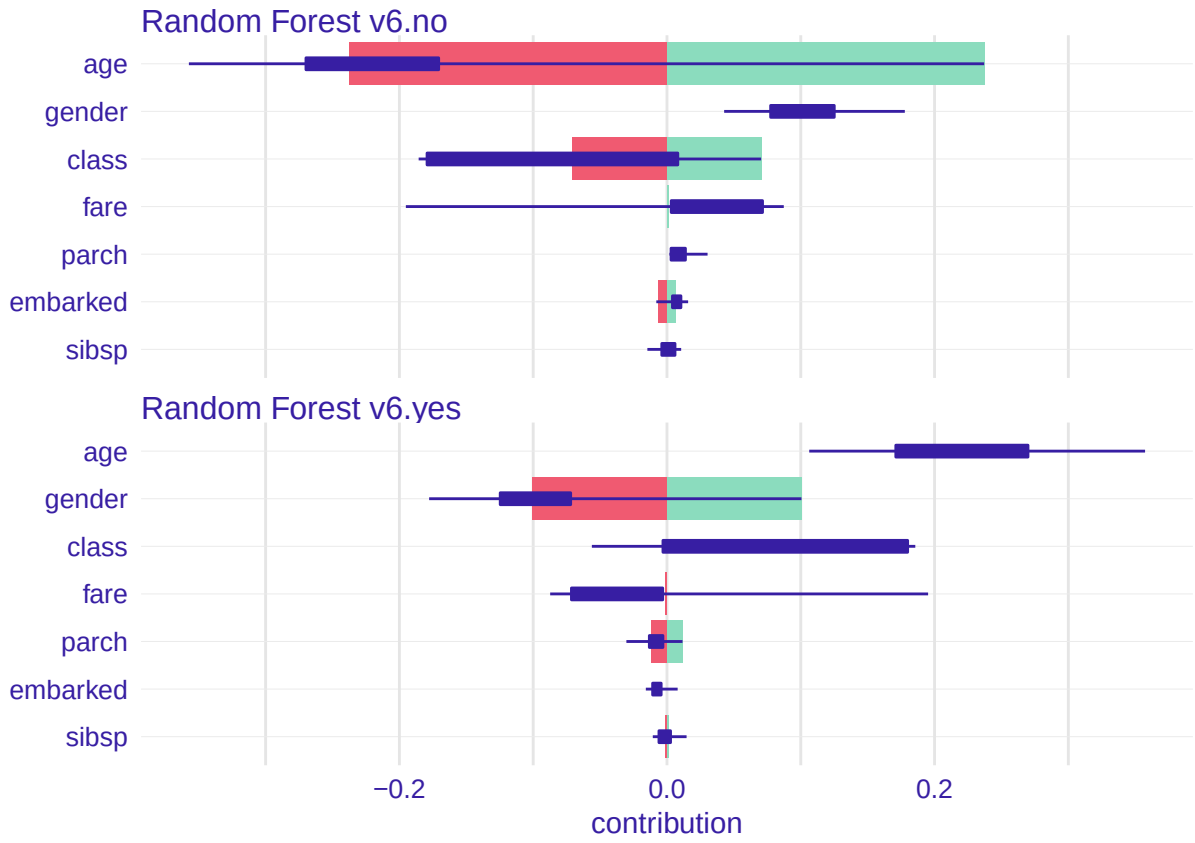
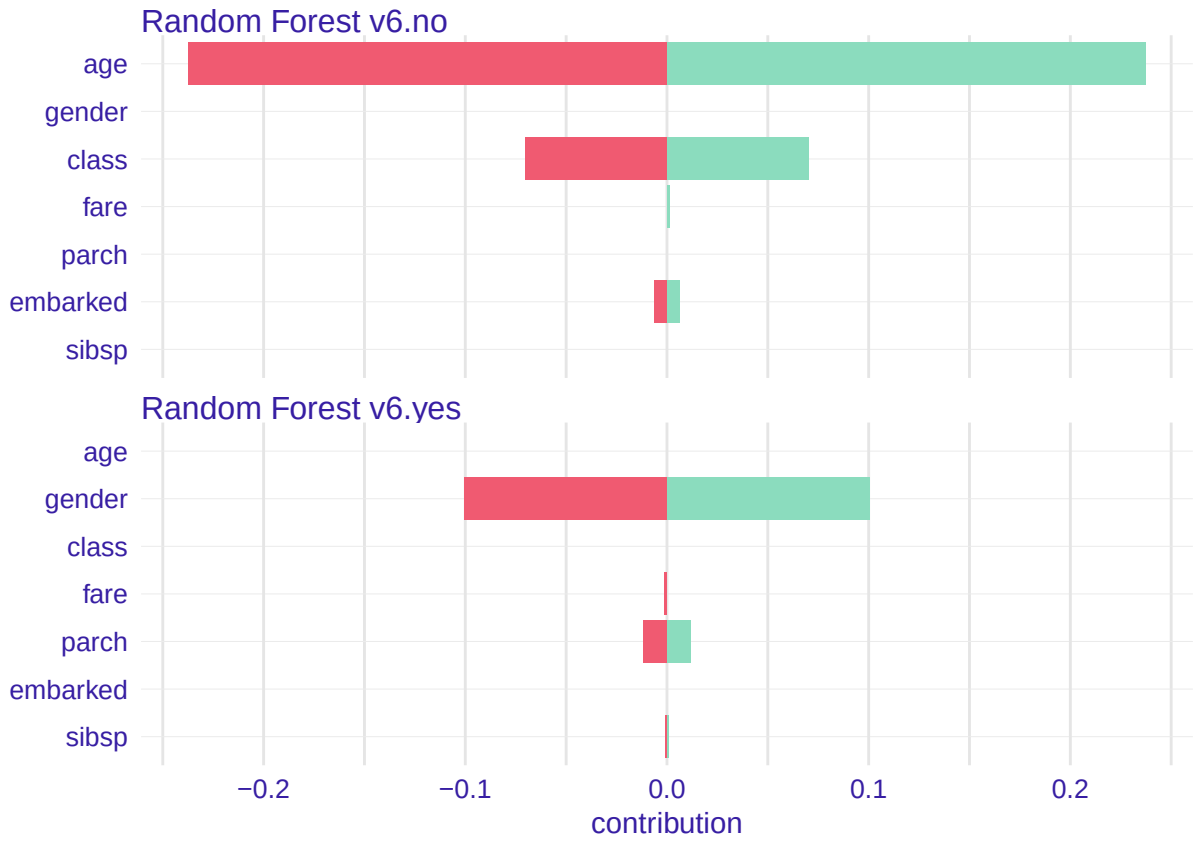


Figure ?? is generated in the same way. The only difference is that boxplots are not plotted. Use the `show_boxplots` argument to decide whatever they shall be added or not.

```
plot(shap_johny,  show_boxplots = FALSE)
```



Function `shap()` results a data frame with attributions for every ordering. Having all these values we can calculated not only Shapley values (averages) but also some other statistics, like quintiles or range for feature attributions.

```
shap_johny
```

```
##                                      min           q1       median
## Random Forest v6.no-age       -0.357347531 -0.269811509 -0.237055659
## Random Forest v6.no-class     -0.185606706 -0.179299502 -0.059222474
## Random Forest v6.no-embarked  -0.007968283  0.004115995  0.007224286
## Random Forest v6.no-fare      -0.195174445  0.003210340  0.045135931
## Random Forest v6.no-gender     0.042726778  0.077494336  0.105988219
## Random Forest v6.no-parch      0.001842320  0.003043951  0.009595831
## Random Forest v6.no-sibsp     -0.014658813 -0.003875850  0.001937472
## Random Forest v6.yes-age       0.106317173  0.171056638  0.267388310
## Random Forest v6.yes-class    -0.056152243 -0.002998641  0.059222474
## Random Forest v6.yes-embarked -0.015808790 -0.010705936 -0.007719982
## Random Forest v6.yes-fare     -0.087299502 -0.071392841 -0.045135931
## Random Forest v6.yes-gender   -0.177782510 -0.124633439 -0.105988219
## Random Forest v6.yes-parch    -0.030339828 -0.013175351 -0.009595831
## Random Forest v6.yes-sibsp    -0.010595378 -0.005975532 -0.001872225
##                                       mean           q3         max
## Random Forest v6.no-age       -0.2194959808 -0.170654282 0.237055659
```

```
## Random Forest v6.no-class     -0.0651330279  0.008076575 0.070343670
## Random Forest v6.no-embarked   0.0058560473  0.010402356 0.015808790
## Random Forest v6.no-fare       0.0014576167  0.071392841 0.087299502
## Random Forest v6.no-gender     0.1004931944  0.125022202 0.177782510
## Random Forest v6.no-parch      0.0116046398  0.013774354 0.030339828
## Random Forest v6.no-sibsp      0.0008288174  0.005975532 0.010595378
## Random Forest v6.yes-age       0.2370556593  0.269811509 0.357347531
## Random Forest v6.yes-class     0.0703436701  0.179858632 0.185606706
## Random Forest v6.yes-embarked -0.0063245310 -0.004355233 0.007968283
## Random Forest v6.yes-fare     -0.0014576167 -0.003210340 0.195174445
## Random Forest v6.yes-gender   -0.0930492541 -0.072002719 0.100493194
## Random Forest v6.yes-parch    -0.0107450368 -0.003043951 0.011604640
## Random Forest v6.yes-sibsp    -0.0007674235  0.002659719 0.014658813
```

```
library("dplyr")
shap_johny %>%
  group_by(variable) %>%
  summarise(avg = mean(contribution),
            q10 = quantile(contribution, 0.1),
            q90 = quantile(contribution, 0.9)) %>%
  arrange(-abs(avg))
```

```
## # A tibble: 7 x 4
##   variable       avg      q10     q90
##   <fct>        <dbl>    <dbl>   <dbl>
## 1 class    -1.15e-17 -0.184   0.184
## 2 sibsp    -8.05e-18 -0.00870 0.00870
## 3 embarked -4.17e-18 -0.0107  0.0107
## 4 parch    -3.60e-18 -0.0170  0.0170
## 5 gender    2.94e-18 -0.125   0.125
## 6 fare     -2.78e-18 -0.138   0.138
## 7 age       1.07e-18 -0.283   0.283
```

0.12 Local Interpretable Model-Agnostic Explanations (LIME)

A different approach to explanations of a single instance is through surrogate models. Models that easy to understand and are similar to black box model around the instance 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 plots is that they explore changes along single dimension or pairs of dimensions.

In this section we describe 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 (**?**).

0.12.1 Intuition

The intuition is presented in Figure **??**. We want to understand some complex model as the one presented in panel B in a point makes with a black cross. As the model may be complex and describe in high dimension, we need to perform two things: find a interpretable representation of these features, fit a simple - easier to interpret model that can be used to better understand the model.

Interpretable representation is case specific, for image data one can use some super-pixels or other large chunks of the image, for tabular data it's still not clear how to construct interpretable features.

Local model is usually a simple model like linear regression of decision tree that can be directly interpret.

0.12.2 Method

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.

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

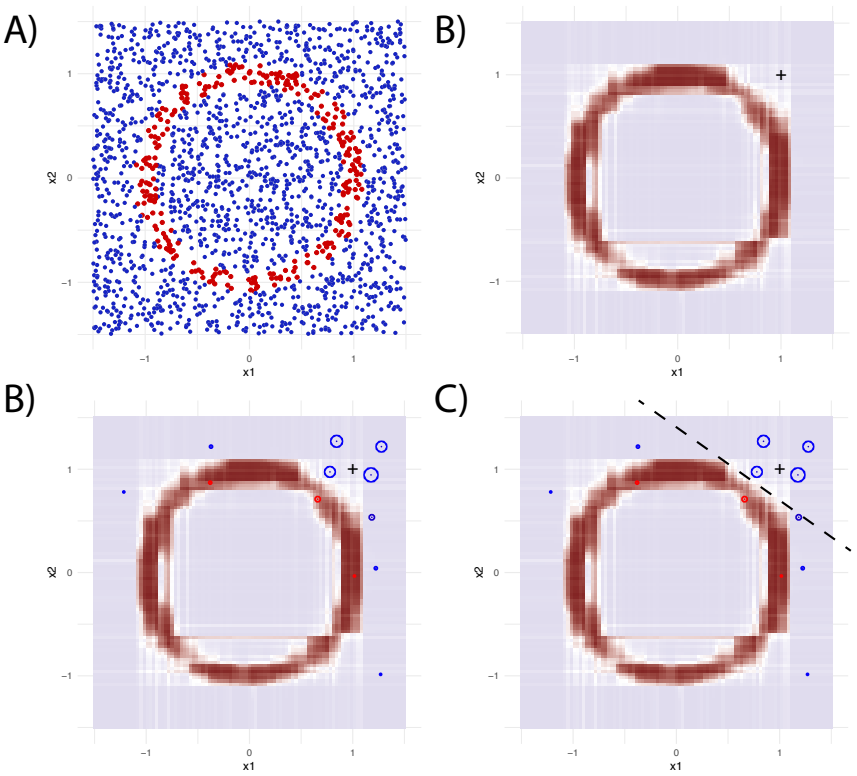


FIGURE 34 (fig:LIME1) A schematic idea behind local model approximations. Panel A shows training data, colors correspond to classes. Panel B shows results from the Random Forest model, here the LIME algorithm starts. Panel C shows new data sampled around the point of interest. The color corresponds to 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 neighborhood

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 neighborhood around point of interest needs to be explored.

Fitting a white box model in this neighborhood

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.

0.12.3 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 what makes them easier to read.

Cons

- For continuous variables and tabular data it is not that easy to find interpretable representations. IMHO this problem is not solved yet.
- 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. Measuring of being local is tricky.

0.12.4 Code snippets for R

In this section we present key features of the R package `iBreakDown` (?) which is a part of `DrWhy.AI` universe and covers methods presented in this chapter.

Note that there are also other R packages that offer similar functionality, like `lime` (?) which is a port of LIME python library (?), `live` (?) and `localModel` (?) and `iml` (?). These packages are different in a way how they handle continuous variables (`lime` performs global discretization, `localModel` local discretization while `live` and `iml` works directly on continuous variables), what kind of local model is fit to the black-box model and how new instances are being sampled. For these reasons these packages results different explanations.

Below we present explanations returned for these four methods for the Johny D and `titanic_rf_v6` model.

```
library("DALEX")
library("randomForest")

titanic <- archivist::aread("pbiecek/models/27e5c")
titanic_rf_v6 <- archivist::aread("pbiecek/models/31570")
johny_d <- archivist::aread("pbiecek/models/e3596")
```

0.12.4.1 The lime pacakge

An example code snippet for the `lime` package is presented below. Key parts are function `lime::lime` which creates an explainer and `lime::explain` which evaluates explanations.

Resulting explanations are presented in Figure ??.

For continuous variables `lime` package discretizes features by using quartiles. This is why in the explanation we have `age < 22`.

```
library("lime")
model_type.randomForest <- function(x, ...) "classification"
lime_rf <- lime(titanic[,colnames(johny_d)], titanic_rf_v6)
lime_expl <- lime::explain(johny_d, lime_rf, labels = "yes", n_features = 4, n_permutations = 1000)
lime_expl

#      model_type case label label_prob  model_r2 model_intercept model_prediction
#1 classification    1    no      0.602 0.5806297       0.5365448        0.5805939
#2 classification    1    no      0.602 0.5806297       0.5365448        0.5805939
#3 classification    1    no      0.602 0.5806297       0.5365448        0.5805939
#4 classification    1    no      0.602 0.5806297       0.5365448        0.5805939
#  feature feature_value feature_weight  feature_desc                 data   prediction
#1    fare            72     0.00640936  21.00 < fare 1, 2, 8, 0, 0, 72, 4 0.602, 0.398
#2  gender             2     0.30481181 gender = male 1, 2, 8, 0, 0, 72, 4 0.602, 0.398
#3   class             1    -0.16690730   class = 1st 1, 2, 8, 0, 0, 72, 4 0.602, 0.398
#4     age             8    -0.10026475     age <= 22 1, 2, 8, 0, 0, 72, 4 0.602, 0.398
```

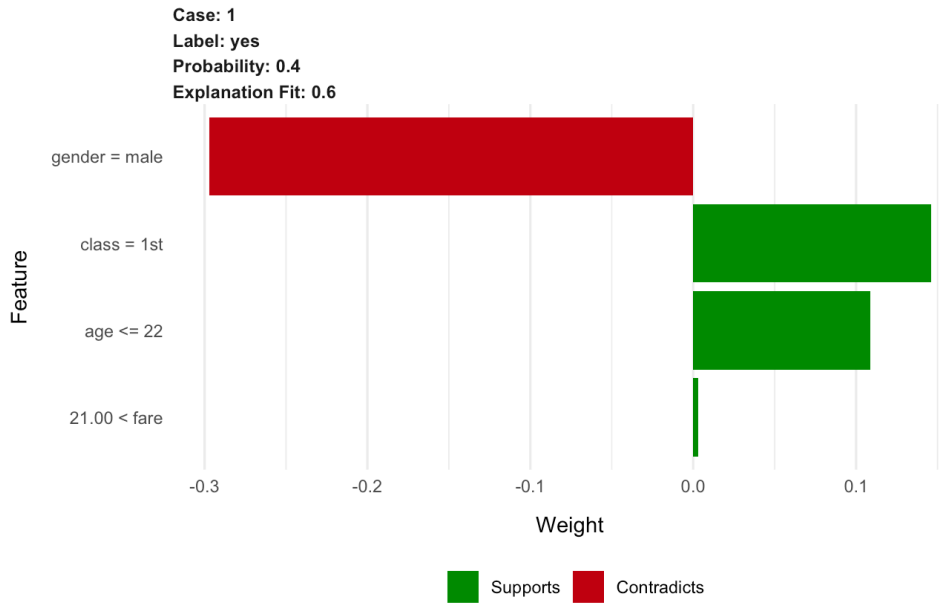


FIGURE 35 (fig:limeExplLIMETitanic) Explanations for Johny D generated by the lime package.

```
plot_features(lime_expl)
```

0.12.4.2 The localModel package

An example code snippet for the `localModel` package is presented below. Key parts are function `DALEX::explain` which creates an explainer and `individual_surrogate_model` which fits a local model.

Resulting explanations are presented in Figure **??**.

For continuous variables `localModel` package discretizes features by using local Ceteris Paribus Profiles. This is why in the explanation we have `age < 15`. As we show in the Ceteris Paribus Chapter **??** the largest drop in survival is observed around 15 years old boys.

```
library("localModel")

localModel_rf <- DALEX::explain(model = titanic_rf_v6,
                      data = titanic[,colnames(johny_d)])
localModel_lok <- individual_surrogate_model(localModel_rf, johny_d,
                                       size = 1000, seed = 1313)
localModel_lok
#   estimated                   variable dev_ratio response
#1 0.23479837                (Model mean) 0.6521442
#2 0.14483341                 (Intercept) 0.6521442
```

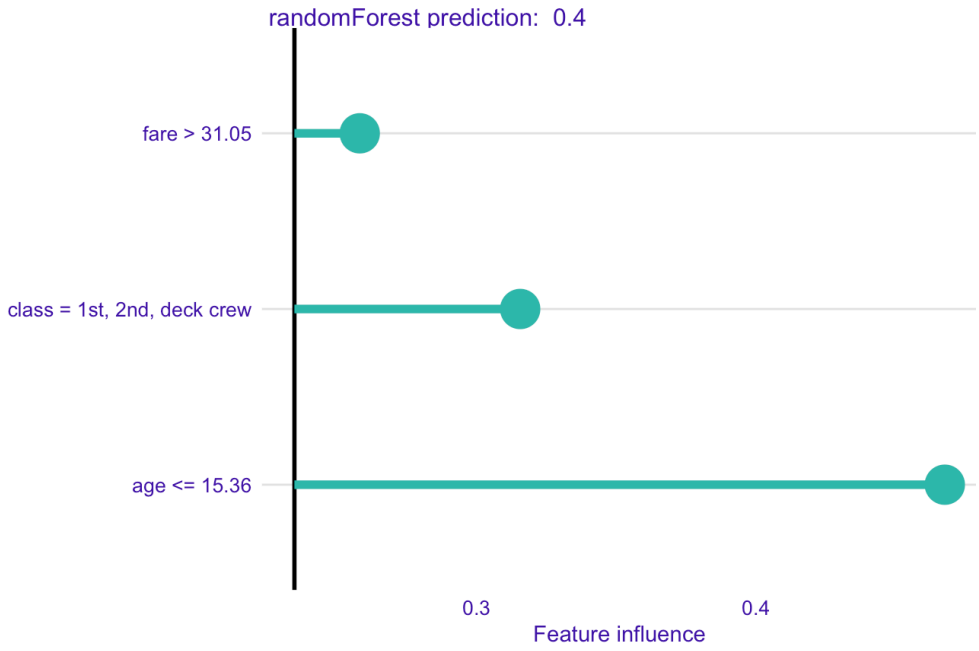


FIGURE 36 (fig:limeExplLocalModelTitanic) Explanations for Johny D generated by the localModel package.

```
#3 0.08081853 class = 1st, 2nd, deck crew 0.6521442
#4 0.00000000     gender = female, NA, NA 0.6521442
#5 0.23282293                 age <= 15.36 0.6521442
#6 0.02338929                 fare > 31.05 0.6521442
plot(localModel_lok)
```

0.12.4.3 The iml package

An example code snippet for the `iml` package is presented below. Key parts are function `Predictor$new` which creates an explainer and `LocalModel$new` which fits local model.

Resulting explanations are presented in Figure **??**.

For continuous variables like `age` the `iml` package approximates this feature without any discretization. As we showed in the Ceteris Paribus Chapter **??**, the profile for age is constant in the interval 0-15, this is why here age is not an important feature.

```
library("iml")
iml_rf = Predictor$new(titanic_rf_v6, data = titanic[,colnames(johny_d)])
iml_white_box = LocalModel$new(iml_rf, x.interest = johny_d, k = 6)
iml_white_box
#Interpretation method:  LocalModel
#
#Analysed predictor:
```

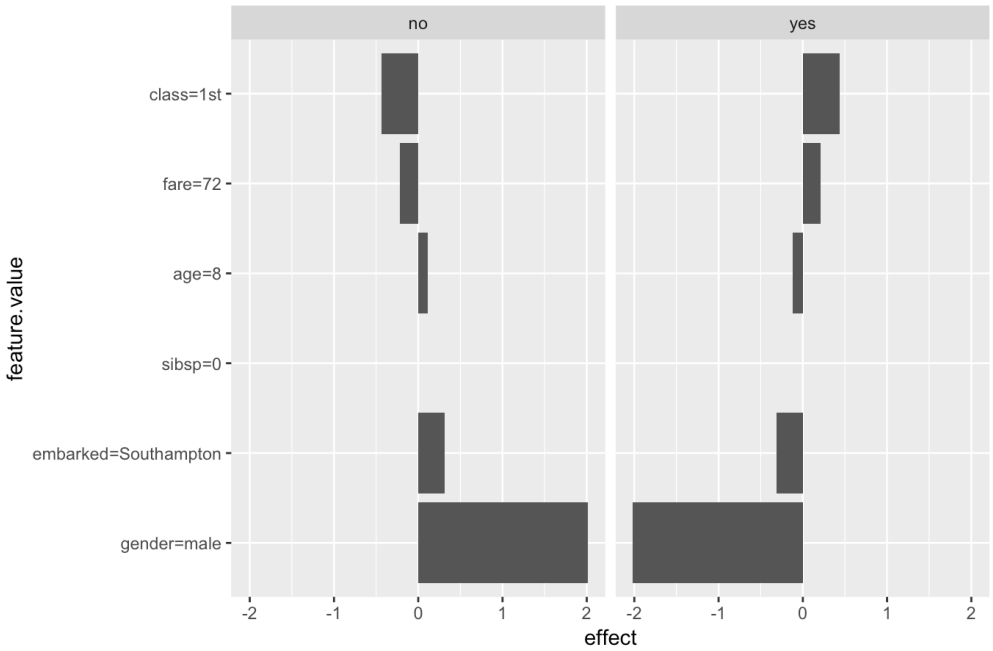


FIGURE 37 (fig:limeExplIMLTitanic) Explanations for Johny D generated by the iml package.

```
#Prediction task: unknown
#
#Analysed data:
#Sampling from data.frame with 2207 rows and 7 columns.
#
#Head of results:
#          beta x.recoded     effect  x.original              feature
#1 -0.158368701         1 -0.1583687         1st            class=1st
#2  1.739826204         1  1.7398262        male          gender=male
#3  0.018515945         0  0.0000000           0                sibsp
#4 -0.001484918        72 -0.1069141          72                 fare
#5  0.131819869         1  0.1318199 Southampton embarked=Southampton
#6  0.158368701         1  0.1583687         1st            class=1st

plot(iml_white_box)
```

0.13 Comparision of instance level explainers

TODO compare pros and cons of different techniques

TODO: Sparse model approximation / variable selection / feature ranking

TODO comparison of difrerent approach for Johny D

TODO Champion-Challenger explainers

0.13.1 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.

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

Model level explanations

0.14 Introduction

Model level explainers help to understand how the model works in general, for some population of interest. This is the main difference from the instance level explainers that were focused on a model behaviour around a single observation. Model level explainers work in the context of a population or subpopulation.

Think about following use-cases

- One wants to know which variables are important in the model. Think about model for heart accident in which features come from additional medical examinations. Knowing

which examinations are not important one can reduce a model by removing unnecessary variables.
- One wants to understand how a selected variable affects the model response. Think about a model for prediction of apartment prices. You know that apartment location is an important factor, but which locations are better and how much a given location is worth? Model explainers help to understand how values of a selected variable affect the model response.
- One wants to know if there are any unusual observations that do not fit to the model. Observations with unusually large residuals. Think about a model for survival after some very risky treatment. You would like to know if for some patients the model predictions are extremely incorrect.

All cases mentioned above are linked with either model diagnostic (checking if model behaves alog our expectations) or knowledge extraction (model was trained to extract some knowledge about the discipline).

0.14.1 Approaches to model explanations

Model level explanations are focused on four main aspects of a model.

- Model performance. Here the question is how good is the model, is it good enough (better than some predefined threshold), is a model A better than model B?
- Variable importance. How important are variables, which are the most important and which are not important at all?
- Variable effects. What is the relation between a variable and model response, can the variable be transformed to create a better model?
- Model residuals. Is there any unusual pattern related to residuals, are they biased, are they correlated with some additional variable?

0.14.2 A bit of philosophy: Three Laws for Model Level Explanations

In the spirit of three laws introduces in the chapter **??** here we propose three laws for model level explanations.

- **Variable importance.** For every model we shall be able to understand which variables are important and which are not.
- **Model audit.** For every model we shall be able to verify basic check like if residuals are correlated with variables and if there are unusual observations.
- **Second opinion.** For every model we shall be able to compare it against other models to verify if they capture different stories about the data.

0.15 Feature Importance

Methods presented in this chapter are useful for assessment of feature importance. There are many possible applications of such methods, for example:

- Feature importance scores may be used for feature filtering. Features that are not important may be removed from the model training procedure. Removal of the noise shall lead to better models.
- Identification of the most important features may be used as a validation of a model against domain knowledge. Just to make sure that it's not like a single random feature dominates model predictions.
- Identification of the most important features may leads to new domain knowledge. Well, we have identified important features.
- Comparison of feature importance between different models helps to understand how different models handle particular features.
- Ranking of feature importance helps to decide in what order we shall perform further model exploration, in what order we shall examine particular feature effects.

There are many methods for assessment of feature importance. In general we may divide them into two groups, methods that are model specific and methods that are model agnostic.

Some models like random forest, gradient boosting, linear models and many others have their own ways to assess feature importance. Such method are linked with the particular structure of the model. In terms of linear models such specific measures are linked with normalized regression coefficients of p-values. For tree based ensembles such measures may be based on utilization of particular features in particular trees, see (**?**) for gradient boosting or (**?**) for random forest.

But in this book we are focused on methods that are model agnostic. The may reason for that is

- First, be able to apply this method to any predictive model or ensemble of models.
- Second, (which is maybe even more important) to be able to compare feature importance between models despite differences in their structure.

Model agnostic methods cannot assume anything about the model structure and we do not want to refit a model. The method that is presented below is described in details in the (**?**). The main idea is to measure how much the model fit will decrease if a selected feature or group of features will be cancelled out. Here cancellation means perturbations like resampling from empirical distribution of just permutation.

The method can be used to measure importance of single features, pairs of features or larger tuples For the simplicity below we describe algorithm for single features, but it is straight forward to use it for larger subsets of features.

0.15.1 Permutation Based Feature Importance

The idea behind is easy and in some sense borrowed from Random Forest (**?**). If a feature is important then after permutation model performance shall drop. The larger drop the more important is the feature.

Let's describe this idea in a bit more formal way. Let $\mathcal{L}(f(x), y)$ be a loss function that assess goodness of fit for a model $f(x)$ while let $\mathcal{X}$ be a set of features.

1. For each feature $x_i \in \mathcal{X}$ do steps 2-5
2. Create a new data $x^{*,-i}$ with feature x_i resampled (or permutated).
3. Calculate model predictions for the new data $x^{*,-i}$, they will be denoted as $f(x^{*,-i})$.
4. Calculate loss function for models predictions on perturbed data

$$L^{*,-i} = \mathcal{L}(f(x^{*,-i}), y)$$

5. Feature importance may be calculated as difference or ratio of the original loss and loss on perturbed data, i.e. $vip(x_i) = L^{*,-i} - L$ or $vip(x_i) = L^{*,-i}/L$.

Note that ranking of feature importance will be the same for the difference and the ratio since the loss L is the same.

Note also, that the main advantage of the step 5 is that feature importance is kind of normalized. But in many cases such normalization is not needed and in fact it makes more sense to present raw $L^{*,-i}$ values.

0.15.2 Example: Titanic

Let's use this approach to a random forest model created for the Titanic dataset. The goal is to predict passenger survival probability based on their sex, age, class, fare and some other features available in the `titanic` dataset.

```
head(titanic)
```

```
##   gender age class    embarked       country  fare sibsp parch survived
## 1   male  42   3rd Southampton United States  7.11     0     0       no
## 2   male  13   3rd Southampton United States 20.05     0     2       no
## 3   male  16   3rd Southampton United States 20.05     1     1       no
## 4 female  39   3rd Southampton       England 20.05     1     1      yes
## 5 female  16   3rd Southampton        Norway  7.13     0     0      yes
## 6   male  25   3rd Southampton United States  7.13     0     0      yes
##   age_cat fare_cat
## 1 (30,50]   (0,10]
## 2 (10,18]  (10,25]
## 3 (10,18]  (10,25]
## 4 (30,50]  (10,25]
## 5 (10,18]   (0,10]
```

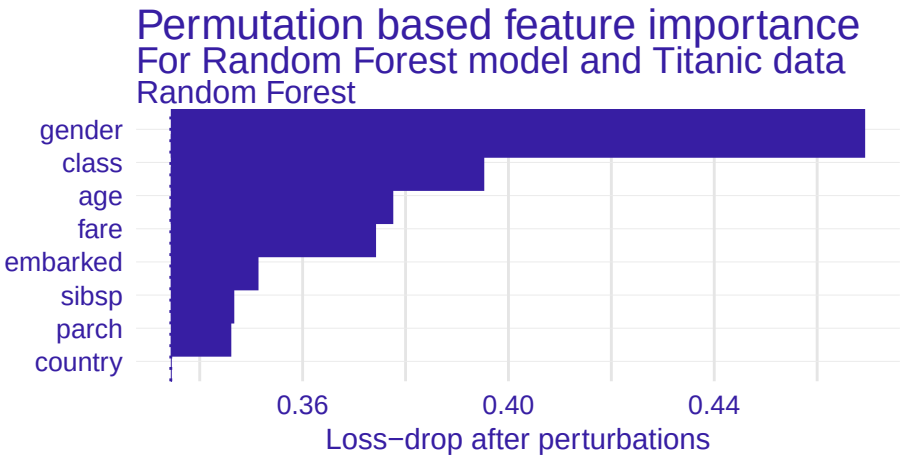


FIGURE 38 Feature importance. Each interval presents the difference between original model performance (left end) and the performance on a dataset with a single feature perturbed

```
## 6 (18,30]   (0,10]
```

Permutation based feature importance can be calculated with the `feature_importance{ingredients}`. By default it permutes values feature by feature.

Instead of showing normalized feature importance we plot both original L and loss after permutation $L^{*,-i}$. This way we can read also how good was the model, and as we will see in next subsection it will be useful for model comparison.

```
library("ingredients")
fi_rf <- feature_importance(explain_titanic_rf)
plot(fi_rf) + ggtitle("Permutation based feature importance", "For Random Forest model and Titanic data")
```

It's interesting that the most important variable for Titanic data is the Sex. So it have been „women first" after all. Then the three features of similar importance are passenger class (first class has higher survival), age (kids have higher survival) and fare (owners of more pricy tickets have higher survival).

Note that drawing permutations evolves some randomness. Thus to have higher repeatability of results you may either set a seed for random number generator or replicate the procedure few times. The second approach has additional advantage, that you will learn the uncertainty behind feature importance assessment.

Here we present scores for 10 repetition of the process.

```
fi_rf10 <- replicate(10, feature_importance(explain_titanic_rf), simplify = FALSE)
do.call(plot, fi_rf10) + ggtitle("Permutation based feature importance", "For Random Forest model and Titanic d
```

It is much easier to assess feature importance if they come with some assessment of the uncertainty. We can read from the plot that Age and passenger class are close to each other.

Note that intervals are useful for model comparisons. In the Figure @ref{titanic5} we can read feature importance for random forest, gradient boosting and logistic regression models.

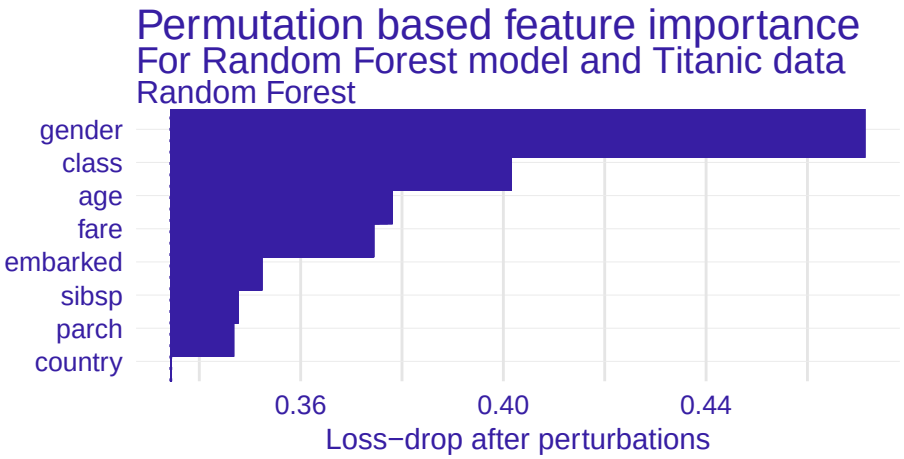


FIGURE 39 Feature importance for 10 replication of feature importance assessment

Best results are achieved by the random forest model and also this method consume more features than others. A good example is the *Fare* variable, not used in gradient boosting not logistic regression (as a feature highly correlated with passenger class) but consumed in the random forest model.

```
fi_rf <- feature_importance(explain_titanic_rf)
fi_gbm <- feature_importance(explain_titanic_gbm)
fi_glm <- feature_importance(explain_titanic_lmr)

plot(fi_rf, fi_gbm, fi_glm)
```

0.15.3 Example: Price prediction

Let's create a regression model for prediction of apartment prices.

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

A popular loss function for regression model is the root mean square loss

$$L(x, y) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - y_i)^2}$$

```
loss_root_mean_square(
  predict(model_rf, apartments),
  apartments$m2.price
)
```

FIGURE 40 Feature importance for random forest, gradient boosting and logistic regression models

```
## [1] 193.8477
```

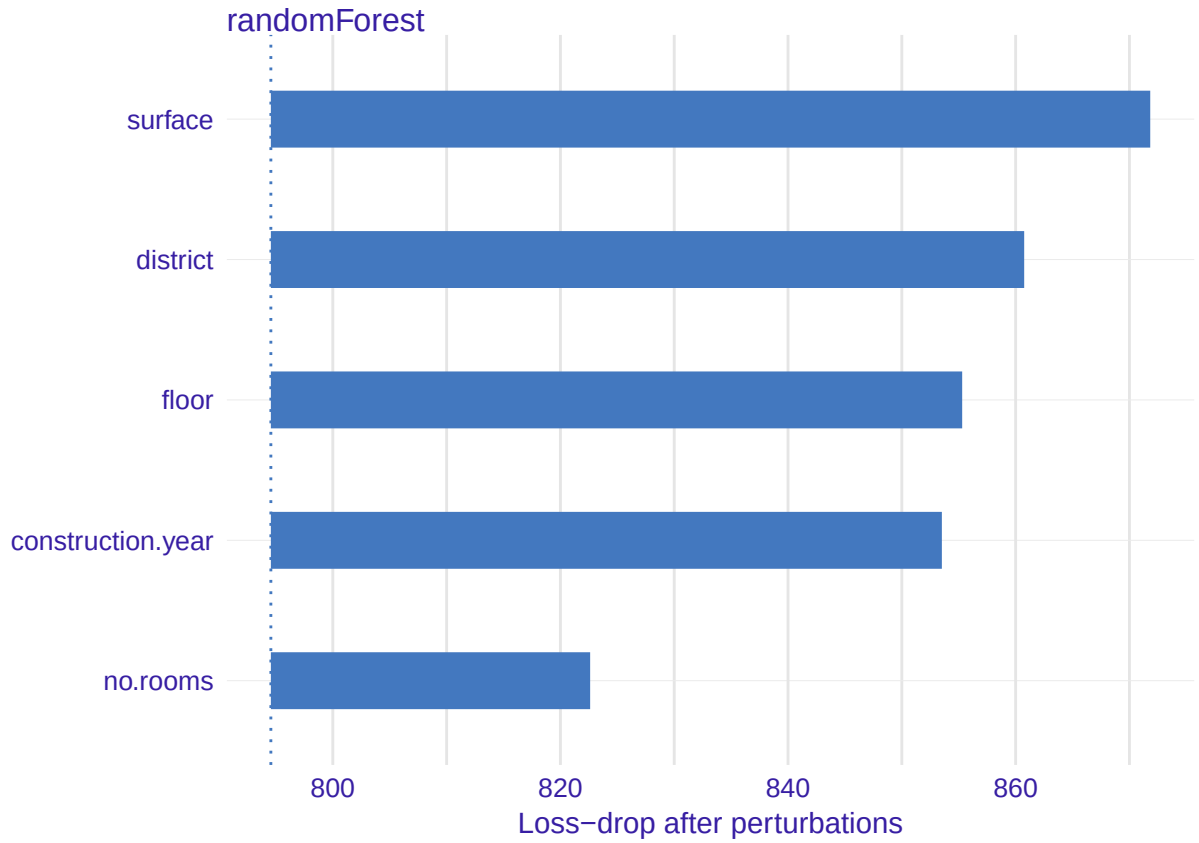
Let's calculate feature importance

```
explainer_rf <- explain(model_rf,
            data = apartmentsTest[,2:6], y = apartmentsTest$m2.price)
vip <- variable_importance(explainer_rf,
            loss_function = loss_root_mean_square)
vip
```

```
##             variable dropout_loss        label
## 1      _full_model_     794.5552 randomForest
## 2          no.rooms     822.6120 randomForest
## 3 construction.year     853.5165 randomForest
## 4             floor     855.3036 randomForest
## 5          district     860.7501 randomForest
## 6           surface     871.8285 randomForest
## 7        _baseline_    1130.2765 randomForest
```

On a diagnostic plot is useful to present feature importance as an interval that start in a loss and ends in a loss of perturbed data.

```
plot(vip)
```



0.15.4 More models

Much more can be read from feature importance plots if we compare models of a different structure. Let's train 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.

Let's fit these three models.

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

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

library("e1071")
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. These explainers wrap models, predict functions and validation data.

```
explainer_lm <- explain(model_lm,
                        data = apartmentsTest[,2:6], y = apartmentsTest$m2.price)
vip_lm <- variable_importance(explainer_lm,
            loss_function = loss_root_mean_square)
vip_lm
```

```
##            variable dropout_loss label
## 1      _full_model_     282.0062    lm
## 2 construction.year     281.9007    lm
## 3          no.rooms     292.8398    lm
## 4             floor     492.0857    lm
## 5           surface     614.9198    lm
## 6          district    1002.3487    lm
## 7        _baseline_    1193.6209    lm
```

```
explainer_rf <- explain(model_rf,
                        data = apartmentsTest[,2:6], y = apartmentsTest$m2.price)
vip_rf <- variable_importance(explainer_rf,
            loss_function = loss_root_mean_square)
vip_rf
```

```
##            variable dropout_loss        label
## 1      _full_model_     799.9382 randomForest
## 2          no.rooms     827.8470 randomForest
## 3 construction.year     852.1447 randomForest
## 4          district     857.3774 randomForest
## 5             floor     874.5364 randomForest
## 6           surface     898.5794 randomForest
## 7        _baseline_    1104.9754 randomForest
```

```
explainer_svm <- explain(model_svm,
                         data = apartmentsTest[,2:6], y = apartmentsTest$m2.price)
vip_svm <- variable_importance(explainer_svm,
            loss_function = loss_root_mean_square)
vip_svm
```

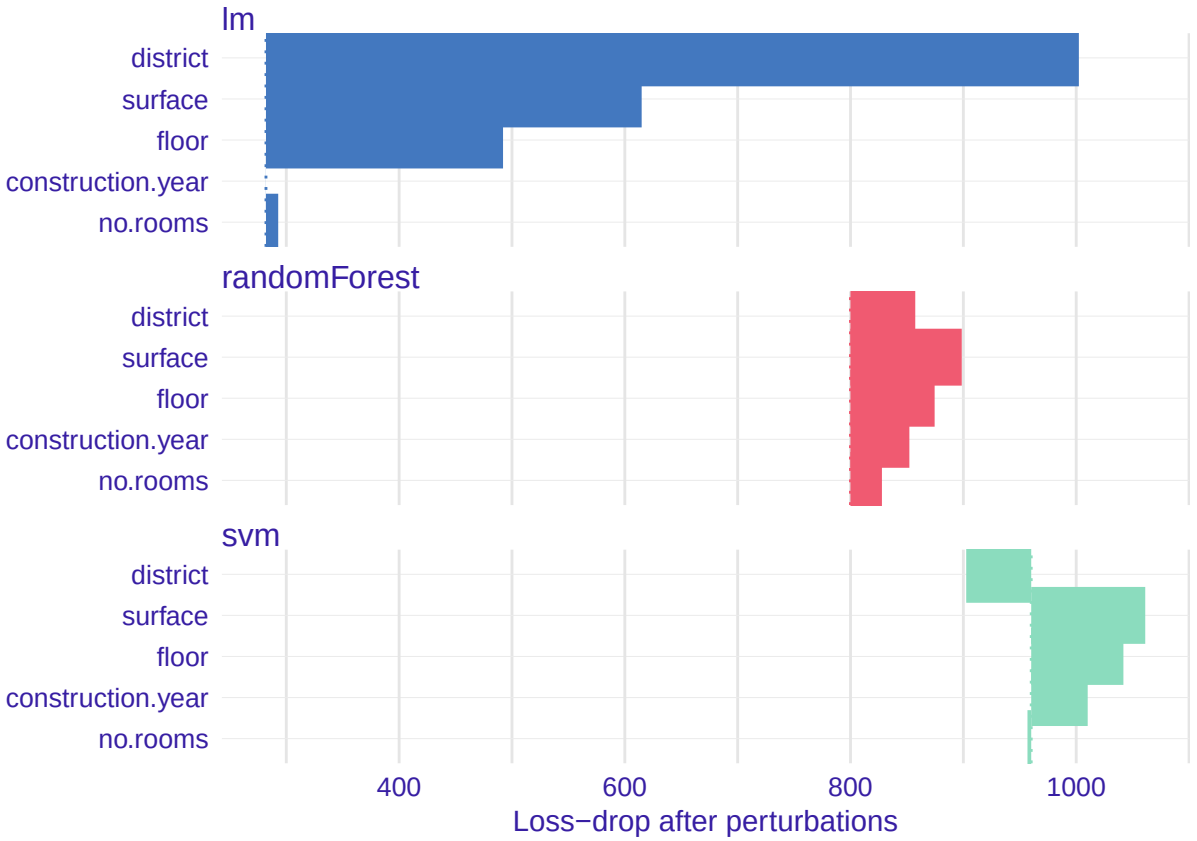
```
##            variable dropout_loss label
## 1      _full_model_     960.1219   svm
## 2          district     902.5403   svm
## 3          no.rooms     956.8193   svm
## 4 construction.year    1010.1792   svm
## 5             floor    1041.8232   svm
## 6           surface    1061.1809   svm
## 7        _baseline_    1248.4173   svm
```

Let's plot feature importance for all three models on a single plot.

Intervals start in a different values, thus we can read that loss for SVM model is the lowest.

When we compare other features it looks like in all models the `district` is the most important feature followed by `surface` and `floor`.

```
plot(vip_rf, vip_svm, vip_lm)
```



There is interesting difference between linear model and others in the way how important is the `construction.year`. For linear model this variable is not importance, while for remaining two models there is some importance.

In the next chapter we will see how this is possible.

0.15.5 Level frequency

What does the feature importance mean? How it is linked with a data distribution.

0.16 Feature effects

In following chapters we introduce tools for extraction of the information between model response and individual model inputs. These tools are useful to summarize how „in general" model responds to the input of interest. All presented approaches are based on Ceteris Ceteris Paribus Profiles introduced in Chapter @ref{ceterisParibus} but they differ in a way how individual profiles are merged into a global model response.

We use the term „feature effect" to refer to global model response as a function of single or

small number of model features. Methods presented in this chapter are useful for extraction information of feature effect, i.e. how a feature is linked with model response. There are many possible applications of such methods, for example:

- Feature effect may be used for feature engineering. The crude approach to modeling is to fit some elastic model on raw data and then use feature effects to understand the relation between a raw feature and model output and then to transform model input to better fit the model output. Such procedure is called surrogate training. In this procedure an elastic model is trained to learn about link between a feature and the target. Then a new feature is created in a way to better utilized the feature in a simpler model (**?**). In the next chapters we will show how feature effects can be used to transform a continuous variable in to a categorical one in order to improve the model behavior.
- Feature effect may be used for model validation. Understanding how a model utilizes a feature may be used as a validation of a model against domain knowledge. For example if we expect monotonic relation or linear relation then such expectations can be verified. Also if we expect smooth relation between model and its inputs then the smoothness can be visually examined. In the next chapters we will show how feature effects can be used to warn a model developer that model is unstable and should be regularized.
- In new domains an understanding of a link between model output and the feature of interest may increase our domain knowledge. It may give quick insights related to the strength or character of the relation between a feature of interest and the model output.
- The comparison of feature effects between different models may help to understand how different models handle particular features. In the next chapters we will show how feature effects can be used learn limitations of particular classes of models.

0.16.1 Global level vs instance level explanations

The plot below shows Ceteris Paribus Profiles for the random forest `rf_5` for 10 selected passengers. Different profiles behave differently. In following chapter we discuss different approaches to aggregation of such profiles into model level feature effects.

0.17 Partial Dependency Profiles

One of the first and the most popular tools for inspection of black-box models on the global level are Partial Dependence Plots (sometimes called Partial Dependence Profiles).

PDP were introduced by Friedman in 2000 in his paper devoted to Gradient Boosting Machines (GBM) - new type of complex yet effective models (**?**). For many years PDP as sleeping beauties stay in the shadow of the boosting method. But this has changed in recent years. PDP are very popular and available in most of data science languages. In this chapter we will introduce key intuitions, explain the math beyond PDP and discuss strengths and weaknesses.

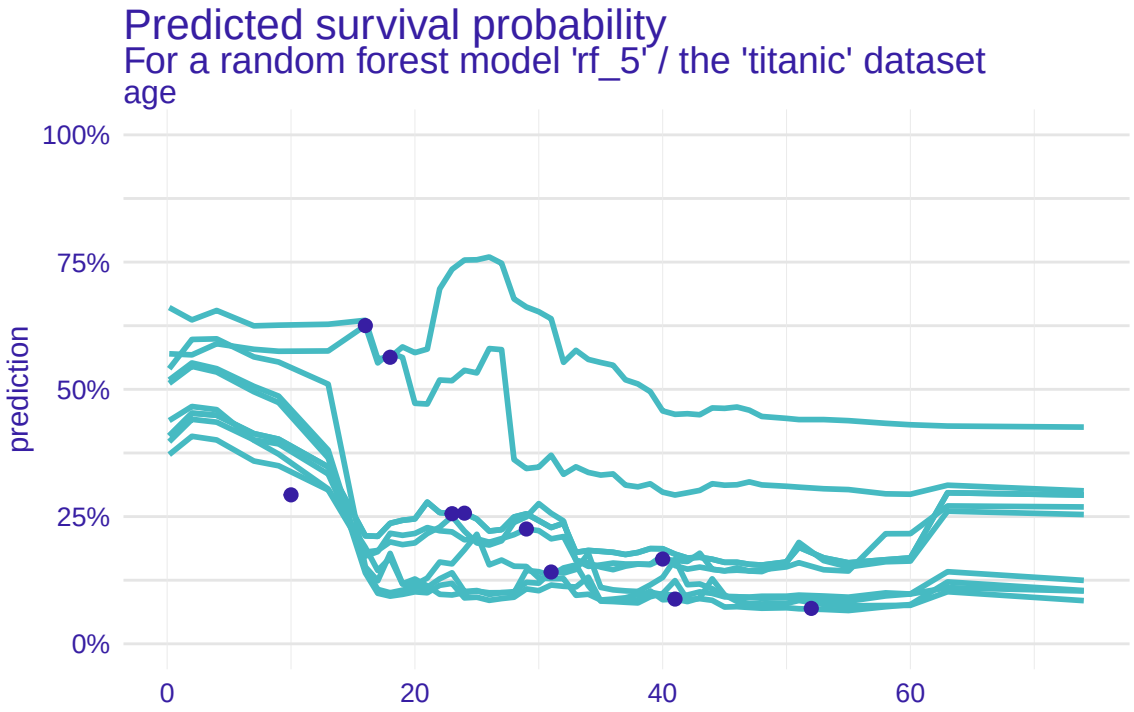


FIGURE 41 (#fig:pdp_part_1A)Ceteris Paribus profiles for 10 passangers and the random forest model

General idea is to show how the expected model response behaves as a function of a selected feature. Here the term „expected" will be estimated simply as the average over the population of individual Ceteris Paribus Profiles introduced in **??**.

0.17.1 Definition

Partial Dependency Profile for for a model f and a variable x^j is defined as

$$g_{PD}^{f,j}(z) = E[f(x^j = z, X^{-j})] = E[f(x|^j = z)].$$

So it's an expected value for $x^j = z$ over **marginal** distribution X^{-j} or equivalently expected value of f after variable x^j is set to z.

Exercise

Let $f = x_1 + x_2$ and distribuion of (x_1, x_2) is given by $x_1 \sim U[0, 1]$ and $x_2 = x_1$.

Calculate $g_{PD}^{f,1}(z)$.

Answer $g_{PD}^{f,1}(z) = z + 0.5$.

0.17.2 Estimation

Let's see how they are constructed step by step. Here we will use a random forest `rf_5` model for the *titanic* dataset. Examples are related to a single variable *age*.

The expectation cannot be calculated directly as we do not know fully neither the distribution of X^{-j} nor the $f()$. Yet this value may be estimated by as average from CP profiles.

$$\hat{g}_{PD}^{f,j}(z) = \frac{1}{n}\sum_{i=1}^{N} f(x_i^j = z, x_i^{-j})] = \frac{1}{n}\sum_{i=1}^{N} f(x_i|^j = z).$$

1. Calculate Ceteris Paribus Profiles for observations from the dataset

As it was introduced in @ref{ceterisParibus} Ceteris Paribus profiles are calculated for observations. They show how model response change is a selected variable in this observation is modified.

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

Such profiles can be calculated for example with the `ceteris_paribus{ingredients}` function.

So for a single model and a single variable we get a bunch of *what-if* profiles. In the figure @ref{pdp_part_1} we show an example for 100 observations. Despite some variation (random forest are not as stable as we would hope) we see that most profiles are decreasing. So the older the passengers is the lower is the survival probability.

2. Aggregate Ceteris Paribus into a single Partial Dependency Profile

Simple pointwise average across CP profiles. If number of CPprofiles is large, it is enoug to sample some number of them to get resonably accurate PD profiles.

Here we show profiles calculated with `ingredients` package, but find siilar implementation in the `pdp` package (**?**), `ALEPlots` package (**?**) or `iml` (**?**) package.

Such average can be calculated with the `aggregate_profiles{ingredients}` function.

```
pdp_rf <- aggregate_profiles(cp_rf)
plot(pdp_rf) +
  ggtitle("Partial Dependency profile", "For a random forest model / Titanic data")
```

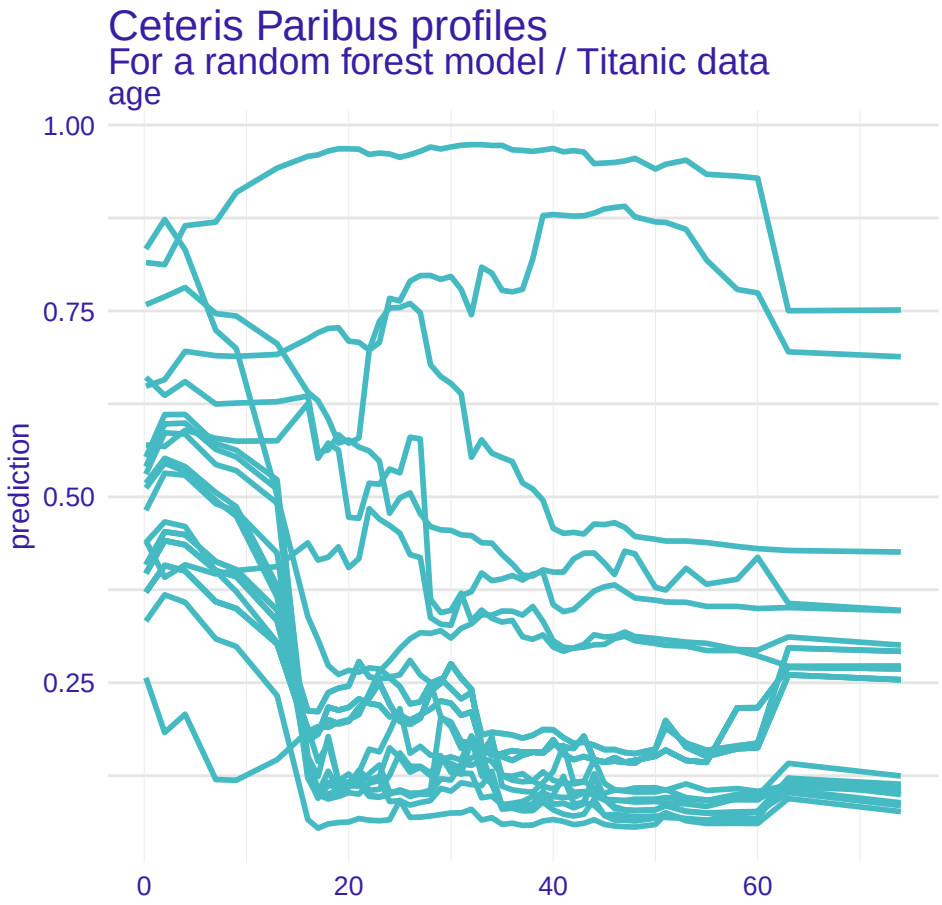
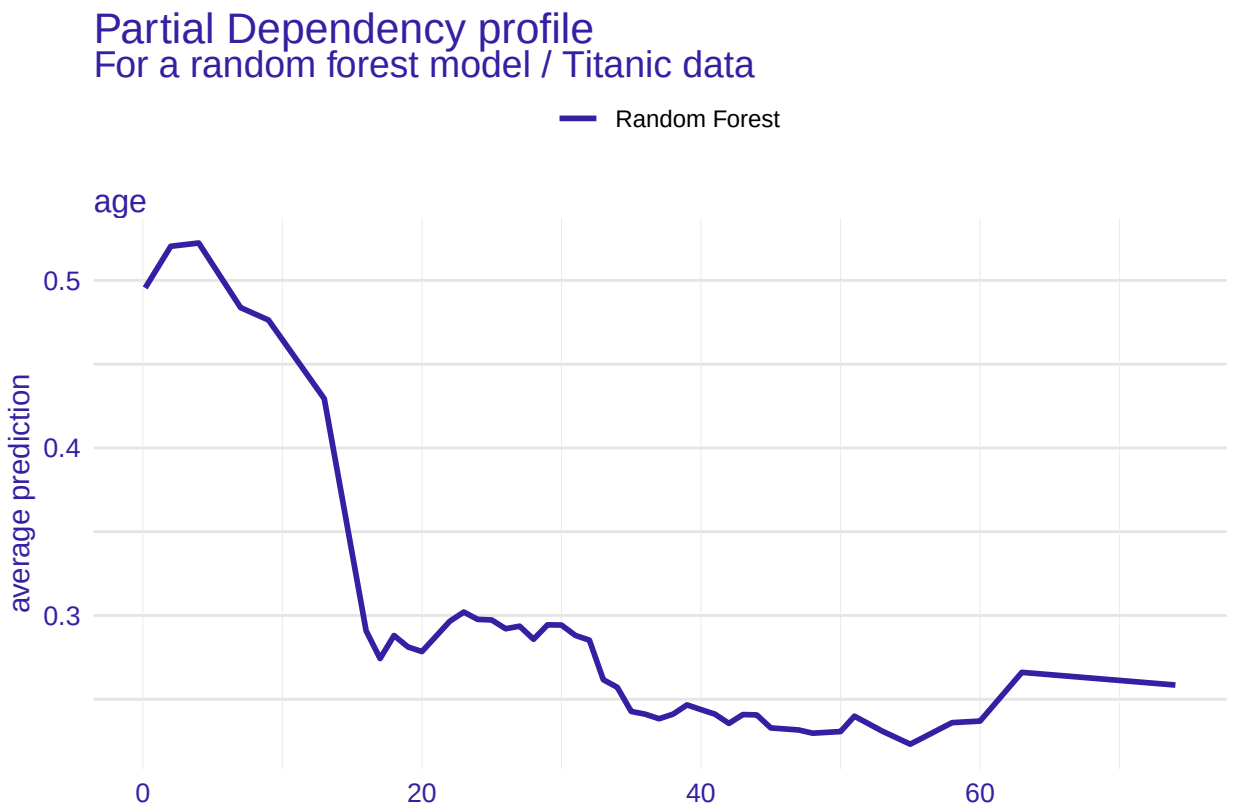


FIGURE 42 (#fig:pdp_part_1)Ceteris Paribus profiles for 100 observations, the age variable and the random forest model



So for a single model and a single variable we get a profile. See an example in figure @ref{pdp_part_2}. It is much easier than following 100 separate curves, and in cases in which Ceteris Paribus are more or less parallel, the Partial Dependency is a good summary of them.

The average response is of course more stable (as it's an average) and in this case is more or less a decreasing curve. It's much easier to notice that the older the passenger is the lower the survival probability. Moreover it is easier to notice that the largest drop in survival changes happen for teenagers. On average the survival for adults is 30 percent points smaller than for kids.

0.17.3 Clustered Partial Dependency Profiles

As we said in the previous section, Partial Dependency is a good summary if Ceteris Paribus profiles are similar, i.e. parallel. But it may happen that the variable of interest is in interaction with some other variable. Then profiles are not parallel because the effect of variable of interest depends on some other variables.

So on one hand it would be good to summaries all this Ceteris Paribus profiles with smaller number of profiles. But on another hand a single aggregate may not be enough. To deal with this problem we propose to cluster Ceteris Paribus profiles and check how homogenous are these profiles.

The most straightforward approach would be to use a method for clustering, like k-means

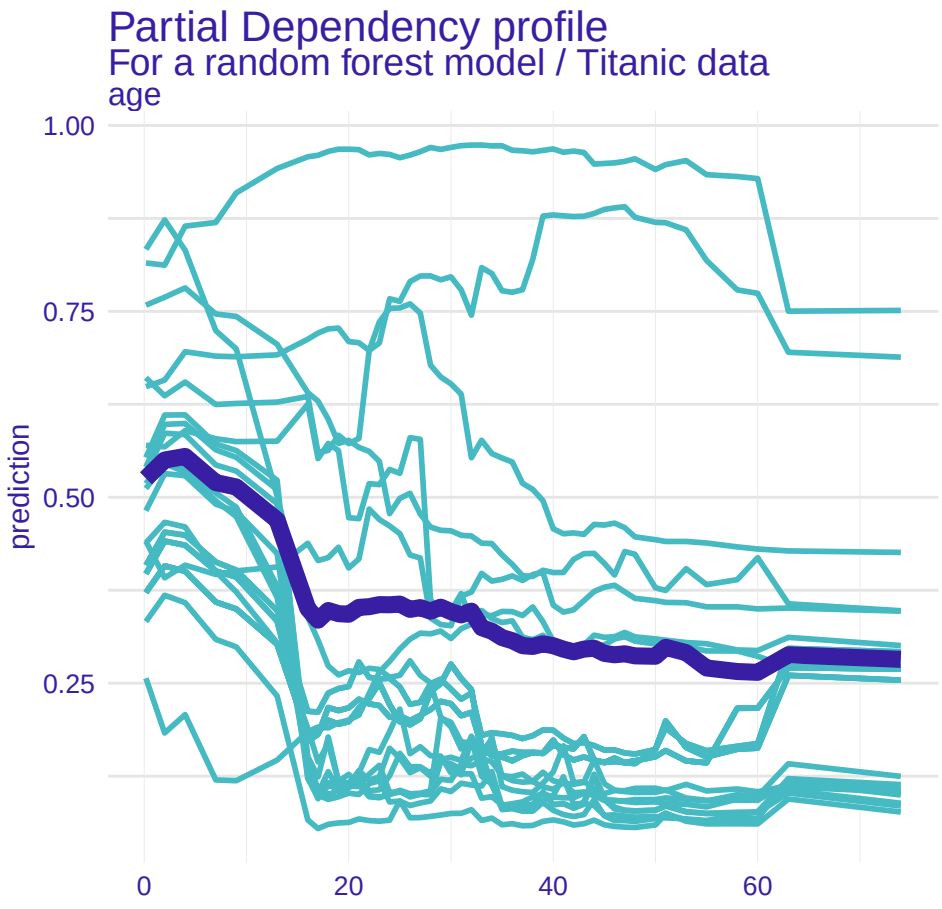


FIGURE 43 (#fig:pdp_part_2)Partial Dependency profile as an average for 100 observations

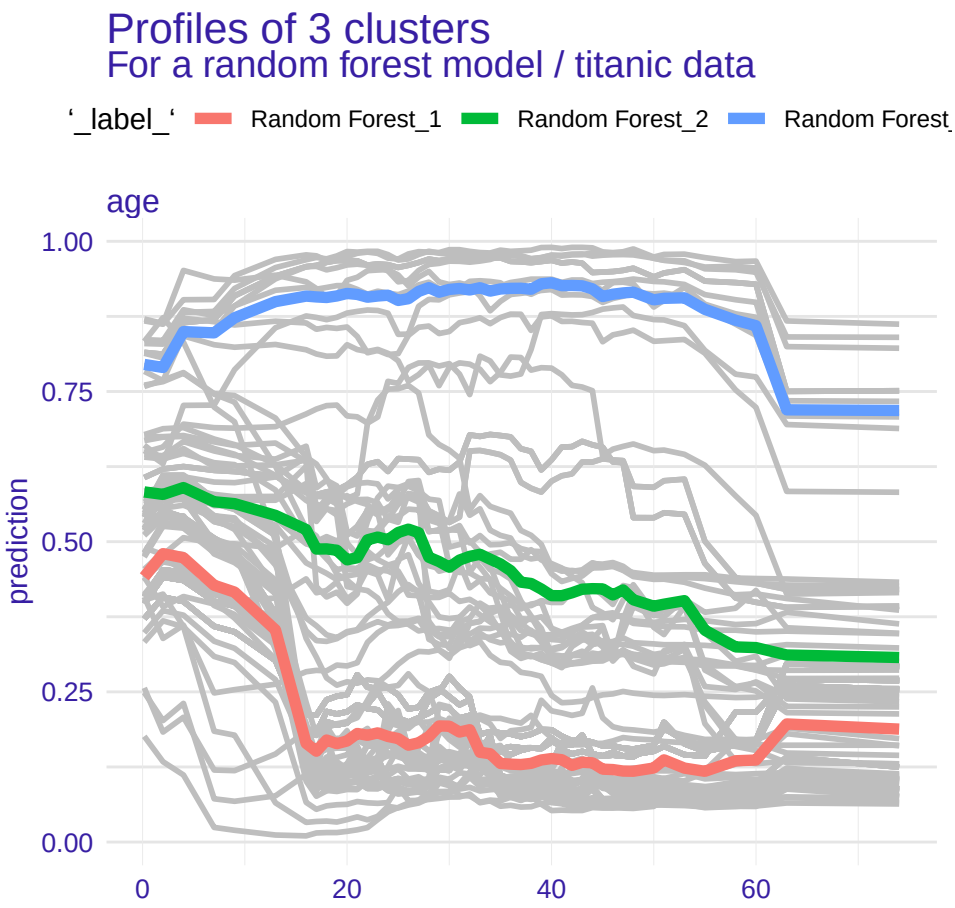


FIGURE 44 (#fig:pdp_part_4)Cluster profiles for 3 clusters over 100 Ceteris Paribus profiles

algorithm or hierarchical clustering, and see how these cluster of profiles behave. Once clusters are established we can aggregate within clusters in the same way as in case of Partial Dependency Plots.

Such clusters can be calculated with the `cluster_profiles{ingredients}` function. We choose the hierarchical clustering with Ward linkage as it gives most stable results.

So for a single model and a single variable we get k profiles. The common problem in clustering is the selection of k. However in our case, as it's an exploration, the problem is simpler, as we are interesting if $k = 1$ (Partial Dependency is a good summary) or not (there are some interactions).

See an example in Figure @ref{pdp_part_4}. It is easier to notice that Ceteris Paribus profiles can be groups in three clusters. Group of passengers with a very large drop in the survival (cluster 1), moderate drop (cluster 2) and almost no drop in survival (cluster 3). Here we do not know what other factors are linked with these clusters, but some additional exploratory analysis can be done to identify these factors.

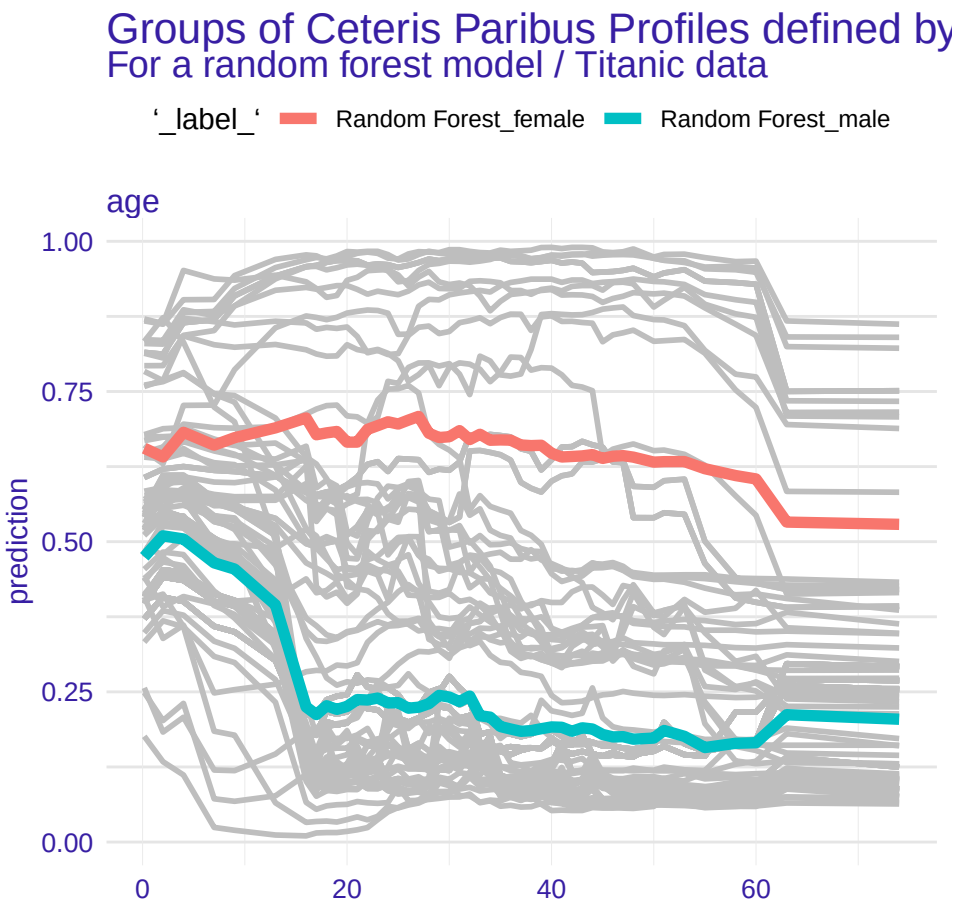


FIGURE 45 (#fig:pdp_part_5)Grouped profiles with respect to the gender variable

0.17.4 Grouped Partial Dependency Profiles

Once we see that variable of interest may be in interaction with some other variable, it is tempting to look for the factor that distinguish clusters.

The most straightforward approach is to use some other variable as a grouping variable. This can be done by setting the `groups` argument in the `aggregate_profiles{ingredients}` function.

```
library("ingredients")
selected_passangers <- select_sample(titanic, n = 100)
cp_rf <- ceteris_paribus(explain_titanic_rf, selected_passangers)
pdp_Sex_rf <- aggregate_profiles(cp_rf, variables = "age",
                groups = "gender")
```

See an example in Figure @ref{pdp_part_5}. Clearly there is an interaction between Age and Sex. The survival for woman is more stable, while for man there is more sudden drop in Survival for older passengers. Check how the interaction for `Pclass` (passenger class) looks like.

0.17.5 Contrastive Model Comparisons

Contrastive comparisons of Partial Dependency Plots are useful not only for subgroups of observations but also for model comparisons.

Why one would like to compare models? There are at least three reasons for it.

- *Agreement of models will calm us.* Some models are known to be more stable other to be more elastic. If profiles for models from these two classes are not far from each other we can be more convinced that elastic model is not over-fitted.
- *Disagreement of models helps to improve.* If simpler interpretable model disagree with an elastic model, this may suggest a feature transformation that can be used to improve the interpretable model. For example if random forest learned non linear relation then it can be captures by a linear model after suitable transformation.
- *Validation of boundary conditions.* Some models are know to have different behavior on the boundary, for largest or lowest values. Random forest is known to shrink predictions towards the average, while support vector machines are known to have larger variance at edges. Contrastive comparisons may help to understand differences in boundary behavior.

Generic `plot{ingredients}` function handles multiple models as consecutive arguments.

See an example in Figure @ref{pdp_part_7}. Random forest is compared with gradient boosting model and generalized linear model (logistic regression). All three models agree when it comes to a general relation between Age and Survival. Logistic regression is of course the most smooth. Gradient boosting has on average higher predictions than random forest.

0.18 Conditional Dependency Profiles

One of the largest advantages of the Partial Dependency Profiles is that they are easy to explain, as they are just an average across Ceteris Paribus profiles. But one of the largest disadvantages lies in expectation over marginal distribution which implies that x^j is independent from x^{-j}. In many applications this assumption is violated. For example, for the *apartments* dataset one can expect that features like $surface$ and $number.or.rooms$ are strongly correlated as apartments with larger number of rooms usually have larger surface. It may makes no sense to consider an apartment with 10 rooms and 20 square meters, so it may be misleading to change $x^{surface}$ independently from $x^{number.of.rooms}$. In the `titanic` dataset we shall expect correlation between `fare` and `passanger class` as tickets in the 1st class are the most expensive.

There are several attempts to fix this problem. Here we introduce Local Dependency Profiles presented in the (**?**) under the name M-profiles.

The general idea is to use conditional distribution instead of marginal distribution to accomodate for the dependency between x^j and x^{-j}.

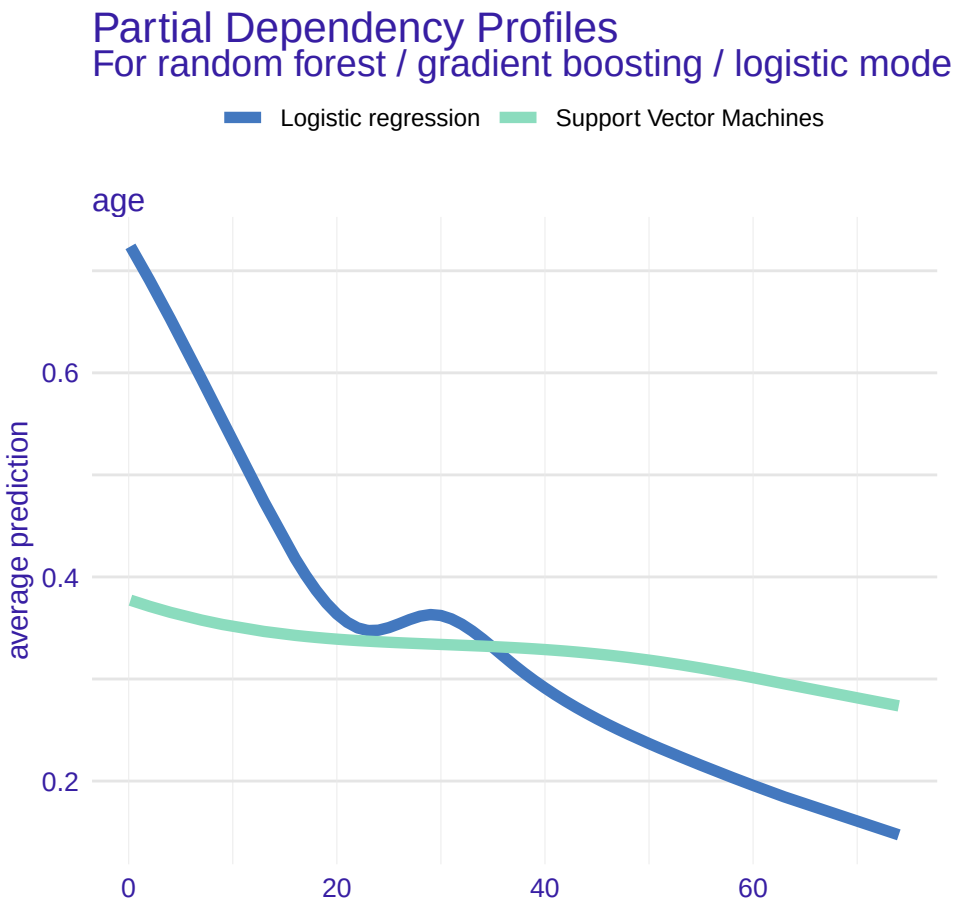


FIGURE 46 (#fig:pdp_part_7)Comparison on three predictive models with different structures.

0.18.1 Definition

Conditional Dependency Profile for a model f and a variable x^j is defined as

$$g_{CD}^{f,j}(z) = E[f(X^j, X^{-j})|X^j = z].$$

So it's an expected value over **conditional** distribution $(X^j, X^{-j})|X^j = z$.

Exercise

Let $f = x_1 + x_2$ and distribuion of (x_1, x_2) is given by $x_1 \sim U[0, 1]$ and $x_2 = x_1$.

Calculate $g_{CD}^{f,1}(z)$.

Answer $g_{CD}^{f,1}(z) = 2 * z$.

0.18.2 Estimation

Partial Dependency Profiles are defined as an expected value from Ceteris Paribus Profiles.

$$g_i^{PD}(z) = E_{X_{-i}}[f(x|^i = z, x^{-i})].$$

And can be estimated as average from CP profiles.

$$\hat{g}_i^{PD}(z) = \frac{1}{n}\sum_{j=1}^{n} f(x|^i = z, x_j^{-i}).$$

As it was said, if X_i and X_{-i} are related it may have no sense to average CP profiles over marginal X_{-i}. Instead, an intuitive approach would to use a conditional distribution $X_{-i}|X_i = x_i$.

$$g_i^{M}(z) = E_{X_{-i}|X_i=x_i}[f(x|^i = z, x^{-i})].$$

0.18.3 Example

See Figure **??** for illustration of difference between marginal and conditional distribution. Such profiles are called Conditional Dependency Profiles and are estimated as

$$\hat{g}_i^{M}(z) = \frac{1}{|N_i|}\sum_{j \in N_i} f(x|^i = z, x_j^{-i}).$$

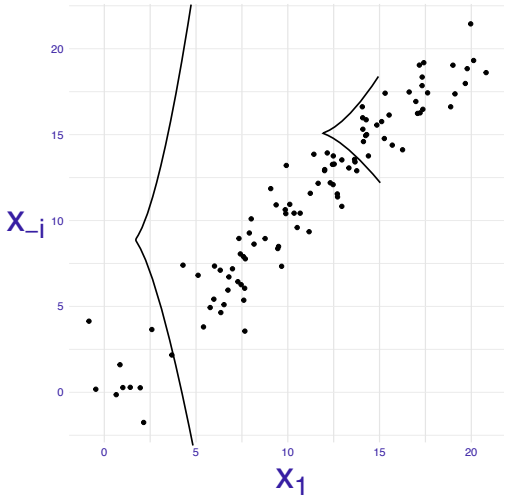


FIGURE 47 (fig:accumulatedCor)

where N_i is the set of observations with x_i close to z.

As it is justified in (**?**), there is a serious problem with this approach, illustrated by a following observation. If y depends on x_2 but not x_1 then the correlation between x_1 and x_2 will produce a *false* relation in the Marginal profiles for feature x_1. This problem is also illustrated in the Figure **??**.

0.19 Accumulated Local Profiles

As we showed in the previous chapter, Conditional Dependency Profiles takes into account dependency between features, but it is both advantage and disadvantage. The advantage is that in some cases the dependency is real and should be taken into account when computing expected value of f. The disadvantage is that in the Conditional Dependency Profiles we see both effects of the feature of interest x^j and other features that are dependent on it. Accumulated Local Profiles disentangle effects of a feature of interest and features correlated with it.

For example, for the *apartments* dataset one can expect that features like $surface$ and $number.or.rooms$ are correlated but we can also imagine that each of these variables affect the apartment price somehow. Partial Dependency Profiles show how the average price changes as a function of surface, keeping all other variables unchanged. Conditional Dependency Profiles show how the average price changes as a function of surface adjusting all other variables to the current value of the surface. Accumulated Local Profiles show how the average price changes as a function of surface adjusting all other variables to the current value of the surface but extracting changes caused by these other features.

Accumulated Local Dependency Profiles presented in the (**?**) paper.

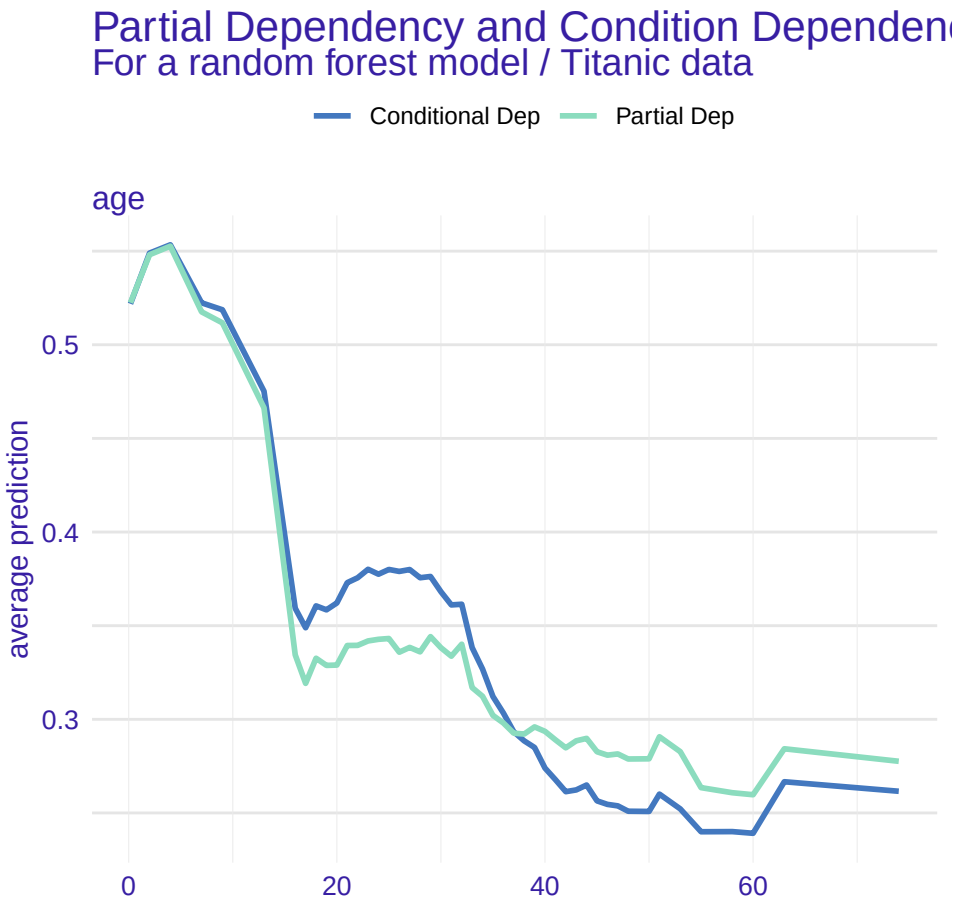


FIGURE 48 (#fig:mp_part_1)Conditional Dependency Profile for 100 observations

The general idea is to accumulate local changes in model response affected by single feature x^j.

0.19.1 Definition

Accumulated Local Profile for a model f and a variable x^j is defined as

$$g_{AL}^{f,j}(z) = \int_{z_0}^{z} E\left[\frac{\partial f(X^j, X^{-j})}{\partial x_j} | X^j = v\right] dv + c,$$

where z_0 if the lower boundry of x^j. The profile $g_{AL}^{f,j}(z)$ is calculated up to some constant c. Usually the constant c is selected to keep average $g_{AL}^{f,j}$ equal to 0 or average f.

The equation may be a bit complex, but the intuition is not that complicated. Instead of aggregation of Ceteris Paribus we just look locally how quickly CP profiles are changing. And AL profile is reconstructed from such local partial changes.

So it's an cummulated expected change of the model response along where the expected values are calculated over **conditional** distribution $(X^j, X^{-j})|X^j = v$.

Exercise

Let $f = x_1 + x_2$ and distribuion of (x_1, x_2) is given by $x_1 \sim U[0, 1]$ and $x_2 = x_1$.

Calculate $g_{AD}^{f,1}(z)$.

Answer $g_{AD}^{f,1}(z) = z$.

0.20 How PD, CD and AL Profiles are different and which to choose

In previous chapters we introduced different was to calculate model level explainers for feature effects. A natural question is how these approaches are different and which one should we choose.

An example that illustrate differences between these approaches is presented in Figure @ref{accumulatedLocalEffects}. Here we have a model $f(x_1, x_2) = x_1 * x_2 + x_2$ and what is important features are correlated $x_1 \sim U[-1, 1]$ and $x_2 = x_1$.

We have 8 points for which we calculated instance level profiles.

x_1	x_2
-1	-1
-0.71	-0.71
-0.43	-0.43

x_1	x_2
-0.14	-0.14
0.14	0.14
0.43	0.43
0.71	0.71
1	1

Panel A) shows Ceteris Paribus for 8 data points, the feature x_1 is on the OX axis while f is on the OY. Panel B) shows Partial Dependency Profiles calculated as an average from CP profiles.

$$g_{PD}^{f,1}(z) = E[z * x^2 + x^2] = 0$$

Panel C) shows Conditional Dependency Profiles calculated as an average from conditional CP profiles. In the figure the conditioning is calculated in four bins, but knowing the formula for f we can calculated it directly as.

$$g_{CD}^{f,1}(z) = E[X^1 * X^2 + X^2 | X^1 = z] = z^2 + z$$

Panel D) shows Accumulated Local Effects calculated as accumulated changes in conditional CP profiles. In the figure the conditioning is calculated in four bins, but knowing the formula for f we can calculated it directly as.

$$g_{AL}^{f,1}(z) = \int_{z_0}^{z} E\left[\frac{\partial(X^1 * X^2 + X^2)}{\partial x_1} | X^1 = v\right] dv = \int_{z_0}^{z} E\left[X^2 | X^1 = v\right] dv = \frac{z^2 - 1}{2},$$

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

0.21 Merging Path Plots and Others

(?)

(?) (?)

(?)

(?) (?) - variable importance

(?)

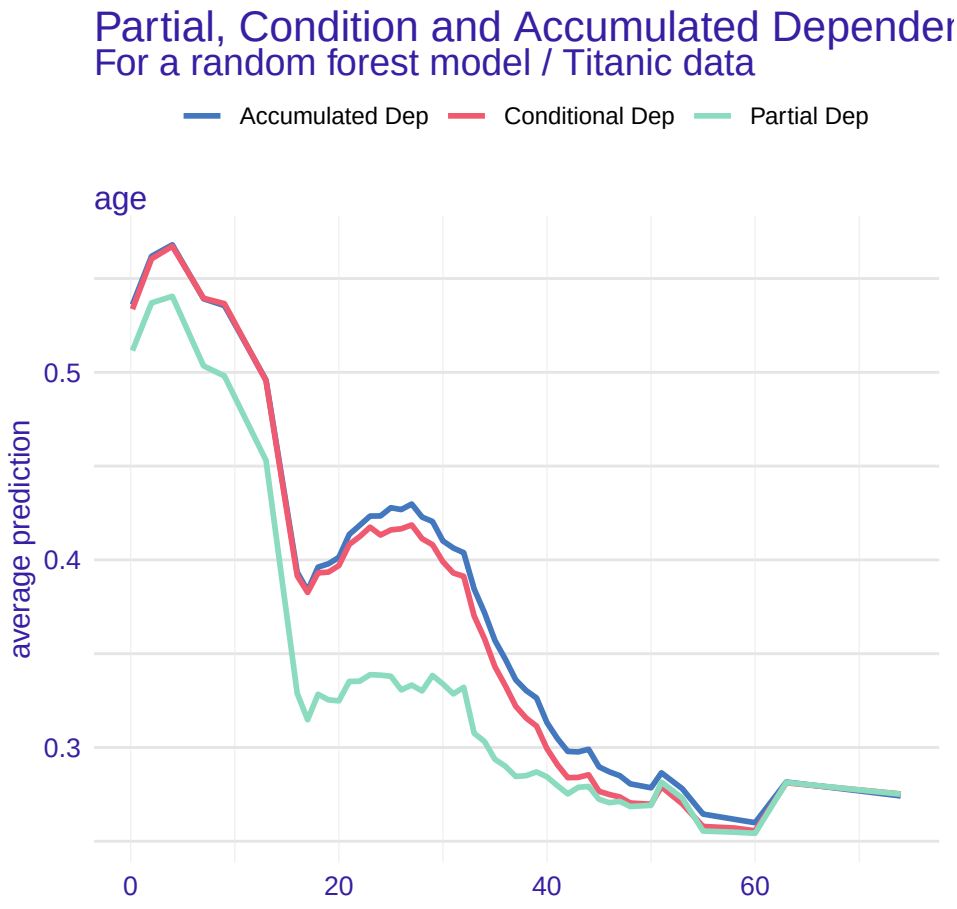


FIGURE 49 (#fig:ale_part_1)Accumulated Local Effects for 100 observations

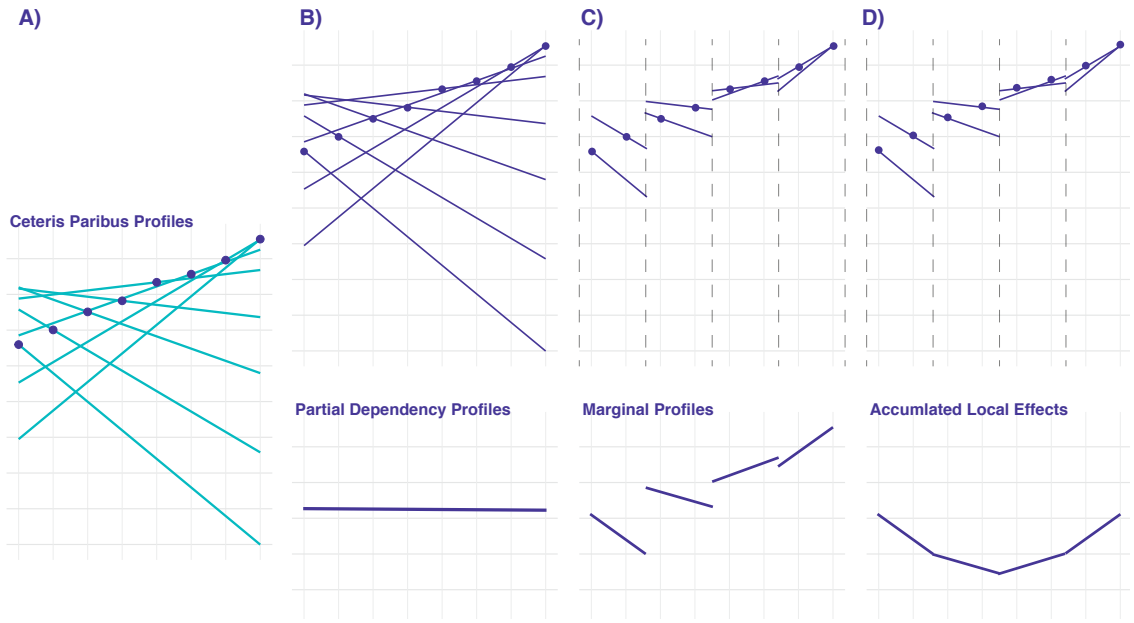


FIGURE 50 (fig:accumulatedLocalEffects) Differences between Partial Dependency, Marginal and Accumulated Local Effects profiles. Panel A) shows Ceteris Paribus Profiles for 8 points. Panel B) shows Partial Dependency profiles, i.e. an average out of these profiles. Panel C shows Marginal profiles, i.e. an average from profiles similar to the point that is being explained. Panel D shows Accumulated Local Effects, i.e. effect curve that takes into account only changes in the Ceteris Paribus Profiles.

Beware Default Random Forest Importances

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

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

```
library(factorMerger)
```

0.22 Other topics

(?) (?) (?)

(?)

0.23 Performance Diagnostic

Goal: how good is the model, which is better

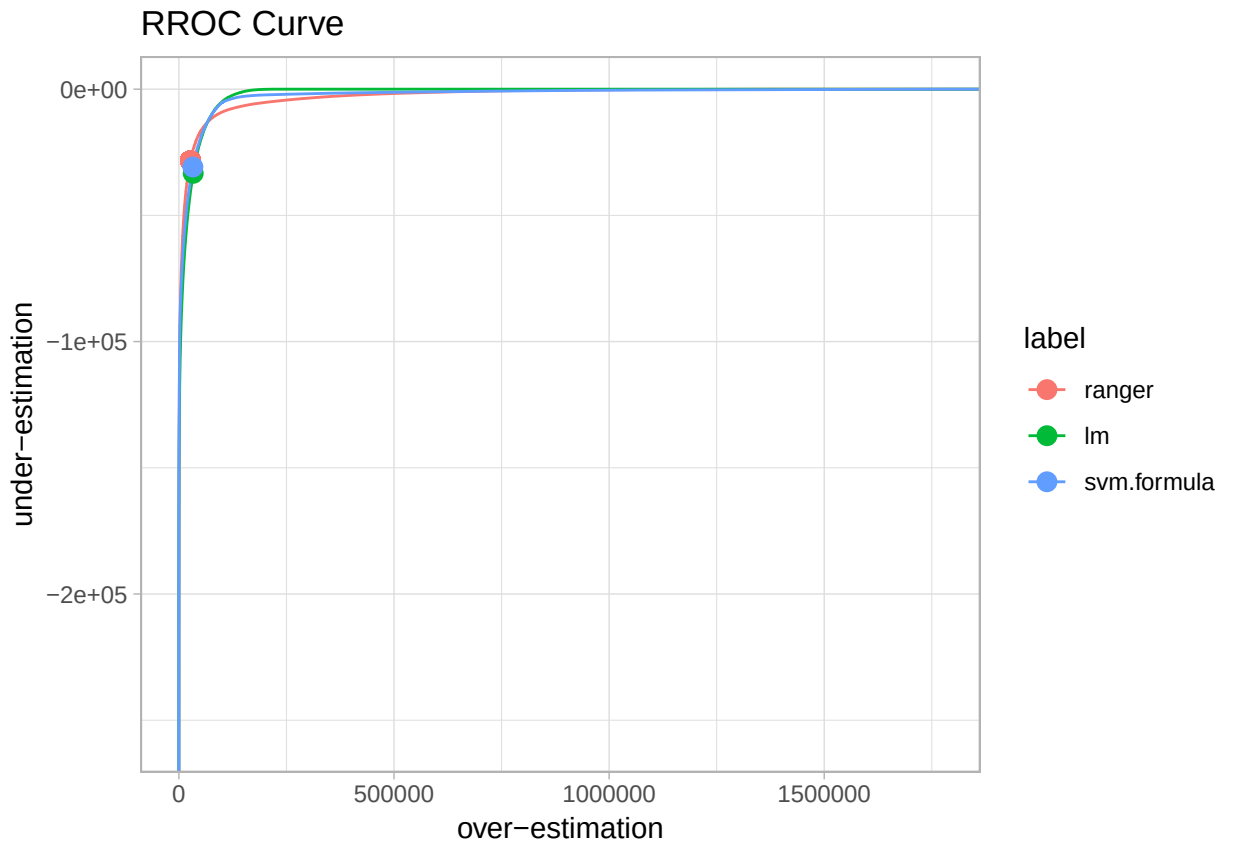
(?) how good is the tree explainer

Model selection

- ROC / RROC / LIFT

```
library("auditor")
library("DALEX2")
library("ranger")
library("e1071")

rf_model <- ranger(life_length ~ ., data = dragons)
lm_model <- lm(life_length ~ ., data = dragons)
svm_model <- svm(life_length ~ ., data = dragons)

predict_function <- function(m,x,...) predict(m, x, ...)$predictions
rf_au <- audit(rf_model, data = dragons, y = dragons$life_length,
           predict.function = predict_function)
lm_au <- audit(lm_model, data = dragons, y = dragons$life_length)
svm_au <- audit(svm_model, data = dragons, y = dragons$life_length)

plotResidualBoxplot(rf_au, lm_au, svm_au)
```


```
plotRROC(rf_au, lm_au, svm_au)
```



0.24 Residual Diagnostic

Goal: verify if model is ok

(?)

```
library("auditor")
library("DALEX2")
library("ranger")

rf_model <- ranger(life_length ~ ., data = dragons)
predict_function <- function(m,x,...) predict(m, x, ...)$predictions
rf_au <- audit(rf_model, data = dragons, y = dragons$life_length,
           predict.function = predict_function)
#check_residuals(rf_au)

#plotResidualBoxplot(rf_au)
#plotResidual(rf_au, variable = "Observed response")
plotScaleLocation(rf_au)
plotRROC(rf_au)
plotAutocorrelation(rf_au)
```

0.25 Concept Drift

Machine learning models are often fitted and validated on historical data under silent assumption that data are stationary. The most popular techniques for validation (k-fold cross-validation, repeated cross-validation, and so on) test models on data with the same distribution as training data.

Yet, in many practical applications, deployed models are working in a changing environment. After some time, due to changes in the environment, model performance may degenerate, as model may be less reliable.

Concept drift refers to the change in the data distribution or in the relationships between variables over time. Think about model for energy consumption for a school, over time the school may be equipped with larger number of devices of with more power-efficient devices that may affect the model performance.

In this chapter we define basic ideas behind concept drift and propose some solutions.

0.25.1 Introduction

In general, concept drift means that some statistical properties of variables used in the model change over time. This may result in degenerated performance. Thus the early detection of concept drift is very important, as it is needed to adapt quickly to these changes.

The term `concept` usually refers to target variable, but generally, it can also refer to model input of relations between variables.

The most general formulation of a concept drift refers to changes in joint distribution of $p(X, y)$. It is useful to define also following measures.

- Conditional Covariate Drift as change in $p(X|y)$
- Conditional Class Drift as change in $p(y|X)$
- Covariate Drift or Concept Shift as changes in $p(X)$

Once the drift is detected one may re-fit the model on newer data or update the model.

0.25.2 Covariate Drift

Covariate Drift is a change in distribution of input, change in the distribution of $p(X)$. The input is a p-dimensional vector with variables of possible mixed types and distributions.

Here we propose a simple one-dimensional method, that can be applied to each variable separately despite of its type. We do not rely on any formal statistical test, as the power of the test depends on sample size and for large samples the test will detect even small differences.

We also consider an use-case for two samples. One sample gathers historical „old" data, this may be data available during the model development (part of it may be used as training and part as test data). Second sample is the current „new" data, and we want to know is the distribution of X_{old} differs from the distribution of X_{new}.

There is a lot of distances between probability measures that can be used here (as for example Wasserstein, Total Variation and so on). We are using the Non-Intersection Distance due to its easy interpretation.

For categorical variables P and Q non-intersection distance is defined as

$$d(P, Q) = 1 - \sum_{i \in \mathcal{X}} \min(p_i, q_i)$$

where $\mathcal{X}$ is a set of all possible values while p_i and q_i are probabilities for these values in distribution P and Q respectively. An intuition behind this distance is that it's amount of the distribution P that is not shared with Q (it's symmetric). The smaller the value the closes are these distributions.

For continuous variables we discretize their distribution in the spirit of χ^2 test.

0.25.3 Code snippets

Here we are going to use the `drifter` package that implements some tools for concept drift detection.

As an illustration we use two datasets from the `DALEX2` package, namely `apartments` (here we do not have drift) and `dragons` (here we do have drift).

```
library("DALEX2")
library("drifter")

# here we do not have any drift
head(apartments, 2)
```

```
##   m2.price construction.year surface floor no.rooms    district
## 1     5897              1953      25     3        1 Srodmiescie
## 2     1818              1992     143     9        5     Bielany
```

```
d <- calculate_covariate_drift(apartments, apartments_test)
d
```

```
##                   Variable  Shift
##   -------------------------------------
##                   m2.price    4.9
##          construction.year    6.0
##                    surface    6.8
##                      floor    4.9
##                   no.rooms    2.8
##                   district    2.6
```

```
# here we do have drift
head(dragons, 2)
```

```
##   year_of_birth   height   weight scars colour year_of_discovery
## 1         -1291 59.40365 15.32391     7    red              1700
## 2          1589 46.21374 11.80819     5    red              1700
##   number_of_lost_teeth life_length
## 1                   25    1368.433
## 2                   28    1377.047
```

```
d <- calculate_covariate_drift(dragons, dragons_test)
d
```

```
##                   Variable  Shift
##   -------------------------------------
##              year_of_birth    8.9
##                     height   15.3  .
##                     weight   14.7  .
##                      scars    4.6
```

```
##                     colour   17.9  .
##          year_of_discovery   97.5  ***
##       number_of_lost_teeth    6.3
##                life_length    8.6
```

0.25.4 Residual Drift

Perhaps the most obvious negative effect of the concept drift is that the model performance degrades over time.

But this is also something that is straightforward to verify. One can calculate distribution of residuals on new data and compare this distribution with residuals obtained on old data.

Again, we have two samples, residuals calculated on the old dataset

$$r_{old} = y_{old} - \hat{y}_{old} = y_{old} - f_{old}(X_{old})$$

versus residuals calculated on the new dataset

$$r_{new} = y_{new} - \hat{y}_{new} = y_{new} - f_{old}(X_{new})$$

We can use any distance between distributions to compare r_{new} and r_{old}, for example the non-intersection distance.

0.25.5 Code snippets

Here we are going to use the `drifter` package.

```
library("DALEX2")
library("drifter")
library("ranger")

data_old <- apartments_test[1:4000,]
data_new <- apartments_test[4001:8000,]

predict_function <- function(m,x,...) predict(m, x, ...)$predictions
model_old <- ranger(m2.price ~ ., data = apartments)
calculate_residuals_drift(model_old,
                      data_old, data_new,
                      data_old$m2.price,
                      data_new$m2.price,
                      predict_function = predict_function)
```

```
##                   Variable  Shift
##    ------------------------------------
##                  Residuals    3.9
```

0.25.6 Model Drift

Model Drift is a change in the relation between target variable and input variables, change in $p(y|X)$. The input is a p-dimensional vector with variables of possible mixed types and distributions.

Here we propose a simple one-dimensional method based on Partial Dependency Plots introduced in the Chapter **??**. PDP profiles summaries marginal relation between $\hat{y}$ and variable x_i. The idea behind concept drift is to compare two models, the old model f_{old} and model refitted on the new data f_{new} and compare these models through PDP profiles.

For each variable we can obtain scores for drift calculated as L_2 distance between PDP profiles for both models.

$$drift_i = \frac{1}{|Z_i|} \int_{z \in Z_i} (PDP_i(f_{old}) - PDP_i(f_{new}))^2 dz$$

where Z_i is the set of values for variable x_i (for simplicity we assume that it's an interval) while $PDP_i(f_{new})$ is the PDP profile for variable i calculated for the model f_{new}.

0.25.7 Code snippets

Here we are going to use the `drifter` package. Instead of using `old` and `new` data here we compare model trained on data with males versus new dataset that contain data for females.

But, because of the interaction of gender and age, models created on these two datasets are different.

```
library("DALEX2")
library("drifter")
library("ranger")

predict_function <- function(m,x,...) predict(m, x, ..., probability=TRUE)$predictions[,1]
data_old = HR[HR$gender == "male", -1]
data_new = HR[HR$gender == "female", -1]
model_old <- ranger(status ~ ., data = data_old, probability = TRUE)
model_new <- ranger(status ~ ., data = data_new, probability = TRUE)
calculate_model_drift(model_old, model_new,
                  HR_test,
                  HR_test$status == "fired",
                  max_obs = 1000,
                  predict_function = predict_function)

library("ceterisParibus2")
prof_old <- individual_variable_profile(model_old,
                  data = data_new,
```

```
                  new_observation = data_new[1:1000,],
                  label = "model_old",
                  predict_function = predict_function)
prof_new <- individual_variable_profile(model_new,
                  data = data_new,
                  new_observation = data_new[1:1000,],
                  label = "model_new",
                  predict_function = predict_function)
plot(prof_old, prof_new,
     variables = "age", aggregate_profiles = mean,
     show_observations = FALSE, color = "_label_", alpha = 1)
```

Appendixes

0.26 Data Sets

0.26.1 Hire or Fire? HR in Call Center

0.27 Packages

0.27.1 Arguments

Here we present list of arguments in explainers from `DrWhy`. All explainers use unified set of arguments. All of them are generic with two specific implementations `*.explainer` and `*.default`. The first one is working for objects created with `DALEX2::explain()` function.

Common core of arguments

- `x` a model to be explained, or an explainer created with function `DALEX2::explain()`.
- `data` validation dataset. Used to determine univariate distributions, calculation of quantiles, correlations and so on. It will be extracted from `x` if it's an explainer.
- `predict_function` predict function that operates on the model `x`. Since the model is a black box, the `predict_function` is the only interface to access values from the model. It should be a function that takes at least a model `x` and `data` and returns vector of predictions. If model response has more than a single number (like multiclass models) then this function should return a marix/data.frame of the size `m` x `d`, where `m` is the number of observations while `d` is the dimensionality of model response. It will be extracted from `x` if it's an explainer.
- `new_observation` an observation/observations to be explained. Required for local/instance level explainers. Columns in should correspond to columns in the `data` argument.

- ... other parameters.
- `label` name of the model. By default it's extracted from the `class` attribute of the model

Function specific arguments

- `keep_distributions` if `TRUE`, then distributions of partial predictions is stored and can be plotted with the generic `plot()`.

0.28 Ceteris-paribus Two-dimensional Profiles - a Tool for Pairwise Interactions

0.28.1 Introduction

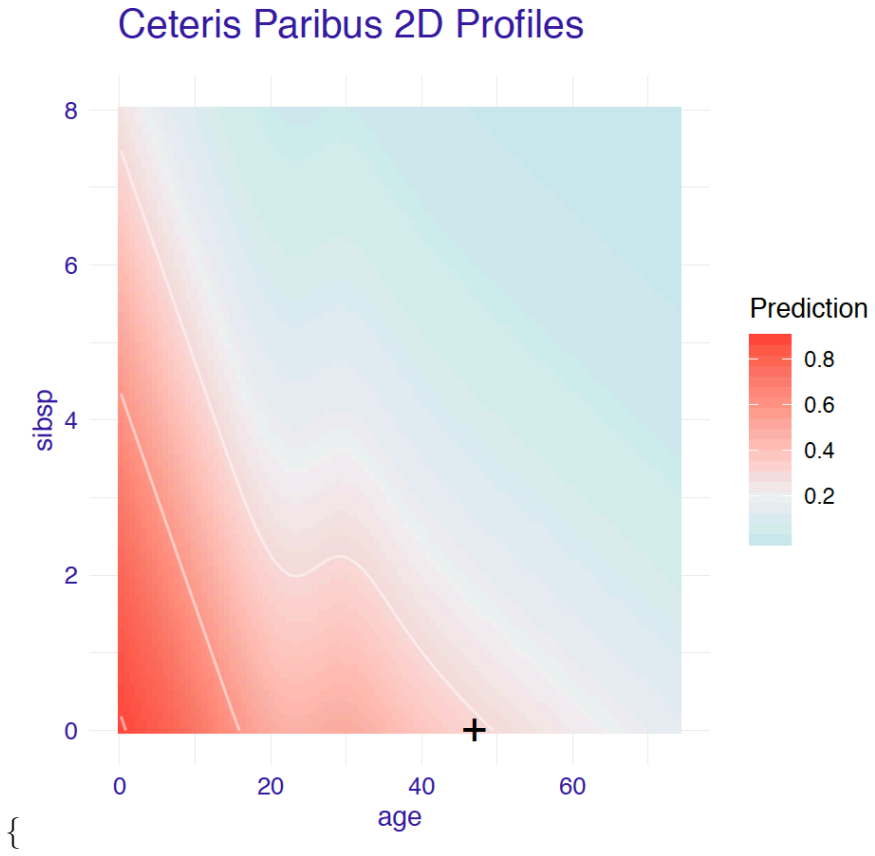
The definition of Ceteris-paribus (CP) profiles, given in Section **??**, may be easily extended to two or more explanatory variables. Also, the definition of the variable importance measure $vip_j^{CP}(x^*)$ have a straightforward extension for a larger number of variables. The extensions are useful to identify or visualize pairwise interactions between explanatory variables.

0.28.2 Intuition

Figure **??** presents response (prediction) surface for the `titanic_lmr_v6` model for two explanatory variables, *age* and *sibsp*, from the *titanic* dataset (see Section **??**). We are interested in the change of the model prediction induced jointly by the variables.

[TOMASZ: THIS IS A BIT WEAK. WHAT INTUITIVE IS ABOUT THE PLOT? WHAT CAN BE SEEN DIFFERENTLY THAN IN AN 1D CP PROFILE? WHICH STRUCTURE WOULD WE LOOK FOR?]

\begin{figure}



{

}

\caption{(fig:profile2d) Ceteris-paribus profile for `age` and `sibsp` explanatory variables for the `titanic_lmr_v6` model.} \end{figure}

0.28.3 Method

The definition of one-dimensional CP profiles (see Section **??**) may be easily extended to two or more explanatory variables. A two-dimensional CP profile for model $f()$, explanatory variables j and k, and point x^* is defined as follows:

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

Thus, a two-dimensional (2D) CP profile is a function that provides the dependence of the instance prediction of the model on the values of j-th and k-th explanatory variables Z_1 and Z_2, respectively. The values of Z_1 and Z_2 are taken to go through the range of values typical for the variables. All other explanatory variables are kept fixed at the values given by x^*.

The corresponding variable importance measure is defined as follows:

$$vip^{CP}_{j,k}(x^*) = \int_{\mathcal{R}}\int_{\mathcal{R}} |CP^{f,(j,k),x^*}(z_1,z_2) - f(x^*)| g^{j,k}(z_1,z_2) dz_1 dz_2 = E_{X_j,X_k}[|CP^{f,j,x^*}(X_j,X_k) - f(x^*)|],$$

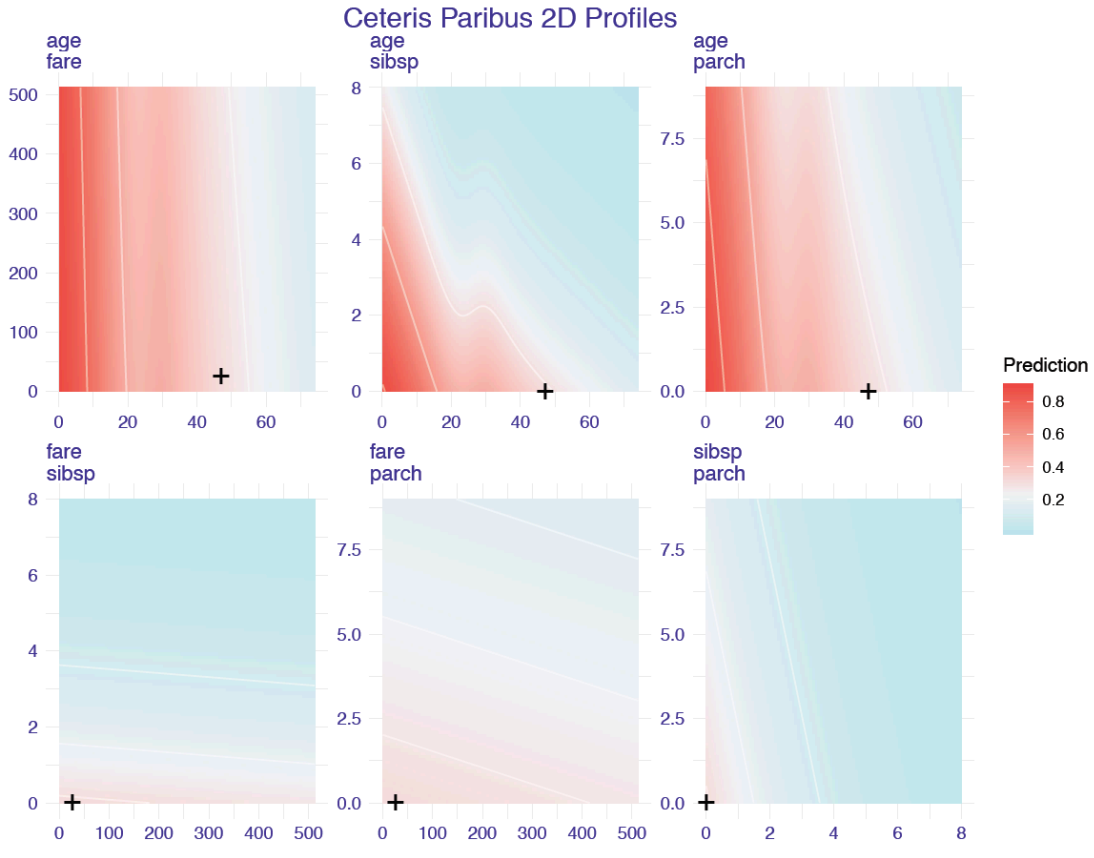
where the expected value is taken over the joint distribution of the j-th and k-th explanatory variable.

Such multi-dimensional extensions are useful to check if, for instance, the model involves interactions. In particular, presence of pairwise interactions may be detected with 2D CP profiles.

0.28.4 Example: Titanic data

A natural way to visualize 2D CP profiles is to use a heat map for all pairs of explanatory variables as, in Figure **??**.

\begin{figure}



{

}

\caption{(fig:profile2dAll) Two-dimensional ceteris-paribus profiles for all pairs of explanatory variables for the `titanic_lmer_v6` model. Black-cross marks the point of interest.} \end{figure}

If the number of pairs of explanatory variables is small or moderate, then it is possible to present 2D CP profiles for all pairs of variables.

If the number of pairs is large, we can use the variable importance measure to order the pairs based on their importance and select the most important pairs for purposes of illustration.

[TOMASZ: WE SHOULD INCLUDE HERE A MORE SUBSTANTIVE DISCUSSION REFERRING TO "HENRY".]

0.28.5 Pros and cons

Two-dimensional CP profiles can be used to identify the presence and the influence of pairwise interactions in a model. However, for models with a large number of explanatory variables, the number of pairs will be large. Consequently, inspection of all possible 2D CP profiles may be challenging. Moreover, the profiles are more difficult to read and interpret than the 1D CP profiles.

[TOMASZ: 2D CP PROFILES FOR FACTORS?]

0.28.6 Code snippets for R

In this section, we present key features of the R package `ingredients` (**?**) which is a part of `DALEXverse` and covers all methods presented in this chapter. More details and examples can be found at `https://modeloriented.github.io/ingredients/`.

There are also other R packages that offer similar functionality, like `condvis` (**?**) or `ICEbox` (**?**).

We use the random forest model `titanic_rf_v6` developed for the Titanic dataset (see Section @ref(model_titanic_rf)) as the example. Recall that we deal with a binary classification problem - we want to predict the probability of survival for a selected passenger.

```
titanic <- archivist::aread("pbiecek/models/27e5c")
titanic_rf_v6 <- archivist::aread("pbiecek/models/31570")
```

First, we have got to create a wrapper around the model (see Section **??**).

```
library("DALEX")
library("randomForest")
explain_titanic_rf <- explain(model = titanic_rf_v6,
                              data = titanic[,-9],
                              y = titanic$survived == "yes",
                              label = "Random Forest v6")
```

To calculate oscillations we need to first calculate CP profiles for the selected observation. Let us use `henry` as the instance prediction of interest.

[TOMASZ: WHY NOT USING THE PRE-DEFINED DATA FRAME?]

```
henry <- data.frame(
  class = factor("1st", levels = c("1st", "2nd", "3rd", "deck crew", "engineering crew",
                                   "restaurant staff", "victualling crew")),
  gender = factor("male", levels = c("female", "male")),
  age = 8,
  sibsp = 0,
  parch = 0,
  fare = 72,
  embarked = factor("Southampton", levels = c("Belfast", "Cherbourg", "Queenstown", "Southampton"))
)
```

2D profiles are calculated by applyiing the `ceteris_paribus_2d()` function to the wrapper object. By default, all pairs of continuous explanatory variables are used, but one can limit number of variables for consideration through the `variables` argument. [TOMASZ: FACTORS?]

```
library("ingredients")
library("ggplot2")

wi_rf_2d <- ceteris_paribus_2d(explain_titanic_rf, observation = henry, variables = c("age", "sibsp", "parch"))
head(wi_rf_2d)
```

As a result, we obtain an object of class `ceteris_paribus_2d_explainer` with overloaded `print()` and `plot()` functions. We can use the latter function to obtain plots of the constructed 2D CP profilest.

[TOMASZ: LABELLING OF THE AXES COULD BE IMPROVED. IT IS UNCLEAR WHICH VARIABLES DEFINE THE Y- AND X AXES.]

```
plot(wi_rf_2d) +
  theme(legend.position = "right", legend.direction = "vertical") + ggtitle("Ceteris Paribus 2D Profiles")
```

The plot suggests that *age* and *sibsp* importantly influence the model response. [TOMASZ: WHY? WHICH FEATURE OF THE PLOTS DISTIGUISHES THIS PAIR FROM THE THREE OTHERS?]

[TOMASZ: WE SHOULD DISCUSS “HENRY” IN THE EXAMPLE SECTION. IN THE SNIPPETS, WE SHOULD SIMPLY SHOW THE UNDERLYING CODE.]

0.29 Variable attribution for linear models

0.29.1 Introduction

In this chapter we introduce the concept and the intuitions underlying ‘’variable attribution,” i.e., the decomposition of the difference between the single-instance and the average model predictions among the different explanatory variables. We can think about the following examples:

- Assume that we are interested in predicting the risk of heart attack based on person's age, sex, and smoking habits. A patient may want to know which factors have the highest impact on the his/her risk score.
- Consider a model for prediction of apartment prices. An investor may want to know how much of the predicted price may be attributed to, for instance, the location of an apartment.
- Consider a model for credit scoring. A customer may want to know if factors like gender, age, or number of children influence model predictions.

In each of those cases we want to attribute a part of the model prediction to a single explanatory variable. This can be done directly for linear models. Hence, in this chapter We focus on those models. The method can be easily extended to generalized linear models. Model-agnostic approaches will be presented in Chapters **??** and **??**.

0.29.2 Intuition

Assume a classical linear model for response Y with p explanatory variables collected in the vector $X = (X_1, X_2, \ldots, X_p)$ and coefficients $\beta = (\beta_0, \beta_1, .., \beta_p)$, where β_0 is the intercept. The prediction for Y at point $X = x = (x_1, x_2, \ldots, x_p)$ is given by the expected value of Y conditional on $X = x$. For a linear model, the expected value is given by the following linear combination:

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

We are interested in the contribution of the i-th explanatory variable to model prediction $f(x^*)$ for a single observation described by x^*. In this case, the contribution is equal to $x_i^*\beta_i$, because the i-th variable occurs only in this term. As it will become clear in the sequel, it is easier to interpret the variable's contribution if x_i is is centered by subtracting a constant $\hat{x}_i$ (usually, the mean of x_i). This leads the following, intuitive formula for the variable attribution:

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

0.29.3 Method

We want to calculate $v(f, x^*, i)$, which is the contribution of the i-th explanatory variable to the prediction of model $f()$ at point x^*. Assume that $E_Y(Y|x^*) \approx f(x^*)$, where $f(x^*)$ is the value of the model at x^*. A possible approach to define $v(f, x^*, i)$ is to measure how much the expected model response changes after conditioning on x_i^*:

$$v(f, x^*, i) = E_Y(Y|x^*) - E_{X_i}\{E_Y[Y|(x_1^*, \ldots, x_{i-1}^*, X_i, x_{i+1}^*, x_p^*)]\} \approx f(x^*) - E_{X_i}[f(x_{-i}^*)],$$

where x_{-i}^* indicates that variable X_i in vector x_{-i}^* is treated as random. For the classical linear model, if the explanatory variables are independent, $v(f, x^*, i)$ can be expressed as follows:

$$v(f, x^*, i) = f(x^*) - E_{X_i}[f(x_{-i}^*)] = \beta_0 + x_1^*\beta_1 + \ldots + x_p^*\beta_p - E_{X_i}[\beta_0 + x_1^*\beta_1 + \ldots + \beta_i X_i \ldots + x_p^*\beta_p] = \beta_i[x_i^* - E_{X_i}(X_i)].$$

In practice, given a dataset, the expected value of X_i can be estimated by the sample mean $\bar{x}_i$. This leads to

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

Note that the linear-model-based prediction may be re-expressed in the following way:

$$f(x^*) = [\beta_0 + \bar{x}_1\beta_1 + ... + \bar{x}_p\beta_p] + [(x_1^* - \bar{x}_1)\beta_1 + ... + (x_p^* - \bar{x}_p)\beta_p]$$

$$\equiv [average\ prediction] + \sum_{j=1}^{p} v(f, x^*, j).$$

Thus, the contributions of the explanatory variables are the differences between the model prediction for x^* and the average prediction.

** NOTE for careful readers **

Obviously, sample mean $\bar{x}_i$ is an estimator of the expected value $E_{X_i}(X_i)$, calculated using a dataset. For the sake of simplicity we do not emphasize these differences in the notation. Also, we ignore the fact that, in practice, we never know the model coefficients and we work with an estimated model.

Also, we assumed that the explanatory variables are independent, which may not be the case. We will return to this problem in Section **??**, when we will discuss interactions.

0.29.4 Example: Wine quality

Figure **??** shows the relation between alcohol and wine quality, based on the wine dataset (**?**). The linear model is

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

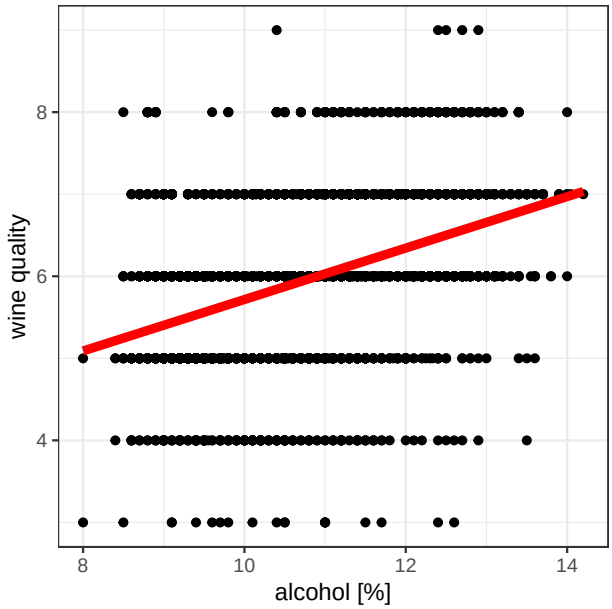


FIGURE 51 (fig:attribution1a)Relation between wine quality and concentration of alcohol assessed with linear model

The weakest wine in the dataset has 8% of alcohol, while the average alcohol concentration is 10.51%. Thus, the contribution of alcohol to the model prediction for the weakest wine is $0.3135 \cdot (8 - 10.51) = -0.786885$. This means that low concentration of alcohol for this wine (8%) decreses the predicted quality by -0.786885.

Note, that it would be misleading to use $x_i^* \beta_i = 0.3135 * 8 = 2.508$ as the alcohol contribution to the quality. The positive value of the product would not correrspond to the intuition that, in the presence of a positive relation, a smaller alcohol concentration should imply a lower quality of the wine.

0.29.5 Pros and Cons

The introduced values $v(f, x^*, i)$ do not depend on neither scale nor location of X_i; hence, it is easier to understand than, for instance, the standardized value of β_i. For the classical linear model, $v(f, x^*, i)$ is not an approximation and it is directly linked with the structure of a model. An obvious diasadvantage is that the definition of $v(f, x^*, i)$ is very much linear-model based. Also, it does not, in any way, reduce the model complexity; if model has 10 parameters then the prediction is decomposed into 10 numbers. Maybe these numbers are easier to understand, but dimensionality of model description has not changed.

0.29.6 Code snippets

In this section, we present an example of computing variable attributions using the HR dataset (see Section **??** for more details).

To calculate variable attributions for a particular point, first we have got to define this point:

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

Variable attributions for linear and generalized linear models may be directly extracted by applying the `predict()` function, with the argument `type = "terms"`, to an object containing results of fitting of a model. To illustrate the approach for logistic regression, we build a logistic regression model for the binary variable `status == "fired"` and extract the estimated model coefficients:

```
library("DALEX")
model_fired <- glm(status == "fired" ~ ., data = DALEX::HR, family = "binomial")
coef(model_fired)
```

```
##  (Intercept)   gendermale          age        hours   evaluation
##  5.737945729 -0.066803609 -0.001503314 -0.102021120 -0.425793369
##       salary
## -0.015740080
```

For the new observation, the predicted value of the logit of the probability of being fired is obtained by applying the `predict()` function:

```
as.vector(pred.inst <- predict(model_fired, dilbert)) # new pediction
```

```
## [1] 0.3858406
```

On the other hand, variable attributions can be obtained by applying the `predict()` function with the `type="terms"` argument:

```
(var.attr <- predict(model_fired, dilbert, type = "terms")) # attributions
```

```
##        gender         age     hours evaluation      salary
## 1 -0.03361889 -0.02660691 0.7555555  0.5547197 0.007287334
## attr(,"constant")
## [1] -0.8714962
```

The largest contributions to the prediction come from variables ''hours" and ''evaluation." Variables ''gender" and ''age" slightly decrease the predicted value. The sum of the attributions is equal to

```
sum(var.attr)
```

```
## [1] 1.257337
```

The attribute `constant` of object `var.attr` provides the ''average" prediction, i.e., the

predicted logit for an obsrvation defined by the means of the explanatory variables, as can be seen from the calculation below:

```
coef(model_fired)%*%c(1, mean((HR$gender=="male")), mean(HR$age), mean(HR$hours),
                      mean(HR$evaluation), mean(HR$salary))
```

```
##            [,1]
## [1,] -0.8714962
```

Adding the ''average" prediction to the sum of the variable attributions results in the new-observation prediction:

```
attributes(var.attr)$constant + sum(var.attr)
```

```
## [1] 0.3858406
```

Below we illustrate how to implement this approach with the `DALEX` package. Toward this end, functions `explain()` and `single_prediction()` can be used. Object `model_fired` stores the definition of the logistic-regression model used earlier in this section. The contents of object `attribution` correspond to the results obtained by using function `predict()`.

```
library("DALEX")

explainer_fired <- explain(model_fired,
                  data = HR,
                  y = HR$status == "fired",
                  label = "fired")

(attribution <- single_prediction(explainer_fired, dilbert))
```

```
##                    variable contribution variable_name variable_value
## 1               (Intercept) -0.871496150     Intercept              1
## hours        + hours = 42.3  0.755555494         hours           42.3
## evaluation + evaluation = 2  0.554719716    evaluation              2
## salary         + salary = 2  0.007287334        salary              2
## age            + age = 57.7 -0.026606908           age           57.7
## gender      + gender = male -0.033618893        gender           male
## 11          final_prognosis  0.385840593
##             cummulative sign position label
## 1           -0.8714962   -1        1 fired
## hours       -0.1159407    1        2 fired
## evaluation   0.4387791    1        3 fired
## salary       0.4460664    1        4 fired
## age          0.4194595   -1        5 fired
## gender       0.3858406   -1        6 fired
## 11           0.3858406    X        7 fired
```

After object `attribution` has been created, a plot presenting the variable attributions can be easily constructed:

```
plot(attribution)
```