Predictive Models: Visual Exploration, Explanation and Debugging

With examples in R and Python

Przemyslaw Biecek and Tomasz Burzykowski

2019-04-15

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 (O'Neil, 2016) 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 (Ross and Swetliz, 2018). *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 (Dastin, 2018). Or the COMPAS algorithm for predicting recidivism discriminates against race (Larson et al., 2016). 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 (Salzberg, 2014). 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 (GDPR, 2018) and the *„Right to Explanation"*, a civic right to be given an explanation for an output of the automated algorithm (Goodman and Flaxman, 2016). Some are already in use while new are being suggested (Casey et al., 2018), (Ruiz, 2018). 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 (O'Neil, 2016) 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 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:

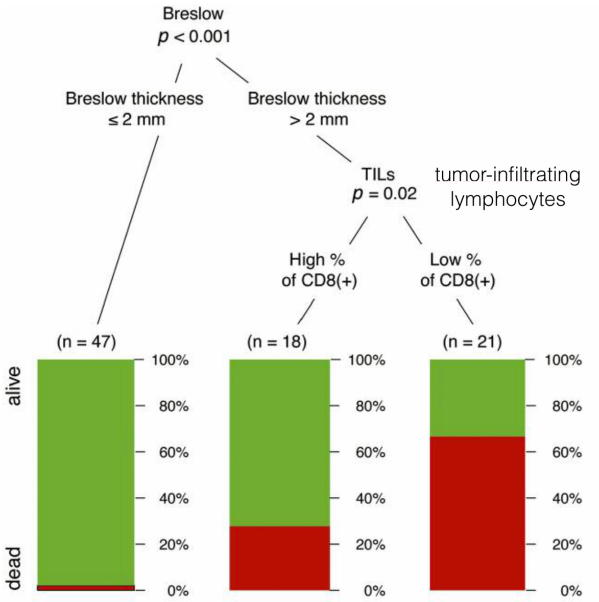
$$RelativeRisk = 1 + 3.6 * [Breslow > 2] - 2 * [TILs > 0]$$

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 (Breiman et al., 2018). Methods have been developed to extract information from the model structure about possible interactions (Paluszynska and Biecek, 2017b). Similar tools have been developed for other ensembles of trees, like xgboost models (Foster, 2018).
- Neural networks enjoy a large collection of dedicated model-explanation tools that use, for instance, the layer-wise relevance propagation technique (Bach et al., 2015), or saliency maps technique (Simonyan et al., 2013), 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 (Xie, 2018), created thanks to the amazing work of Yihui Xie.

0.2 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 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.

0.2.1 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, including some variable transformations, is also available 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)

FIGURE 1 Titanic sinking by Willy Stöwer

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"
```

0.2.1.1 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 value of 0.

[TOMASZ: FOR FARE, ONE COULD USE THE MEAN OF OBSERVED VALUES, AS FOR AGE. TAKING 0 CORRESPONDS TO "crew".]

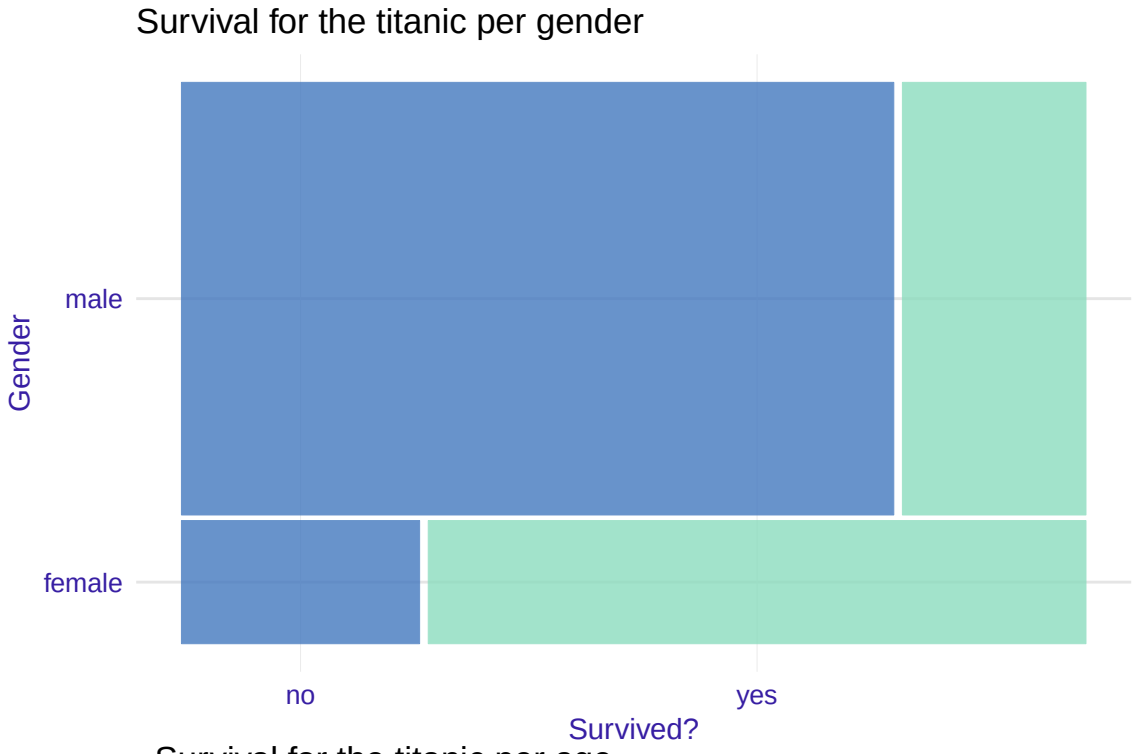
```
# missing country is replaced by "X"
titanic$country[is.na(titanic$country)] = "X"
# missing age is replaced by average (30)
titanic$age[is.na(titanic$age)] = 30
# missing fare, sibsp, parch are replaced by 0
titanic$fare[is.na(titanic$fare)] = 0
titanic$sibsp[is.na(titanic$sibsp)] = 0
titanic$parch[is.na(titanic$parch)] = 0
```

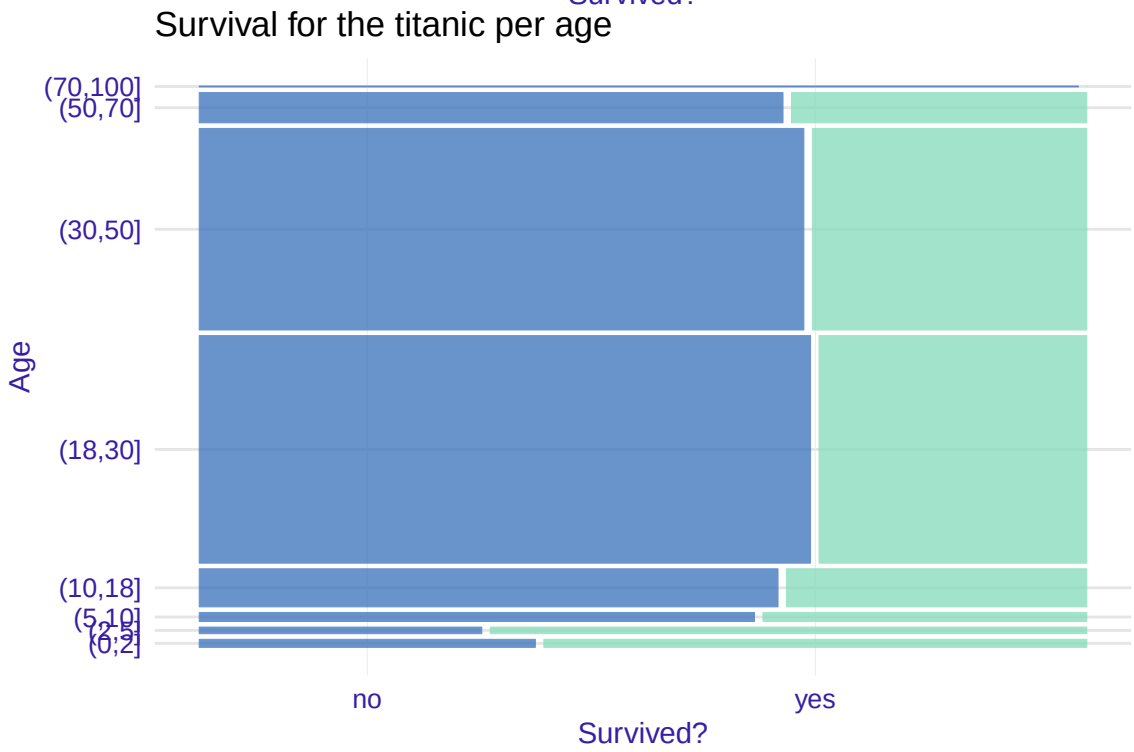
After imputing the missing values, we investigate the association between survival status and the other variables. Figures XXX-XXX present graphically the proportion non- and survivors for different levels of the other variables. The height of the bars (on the y-axis) reflects the marginal distribution (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.

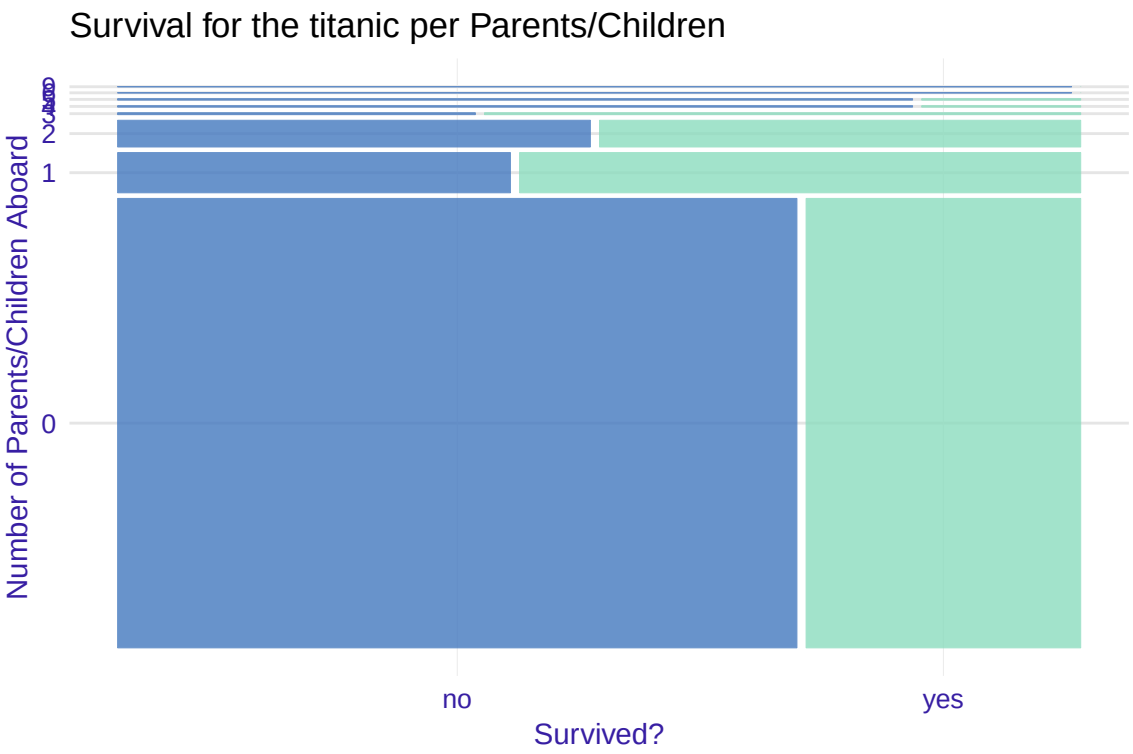
[TOMASZ: MAKE SEPARATE FIGURES SO THAT WE CAN REFER TO THEM. CHANGE THE ORDER, AS SUGGESTED IN THE R CODE. LABEL THE HORIZONTAL "SURVIVAL" AXIS WITH PROPORTIONS. IMPROVE LABELING OF THE Y-AXIS. CATEGORIZE FARE IN A SIMILAR WAY AS FOR AGE.]

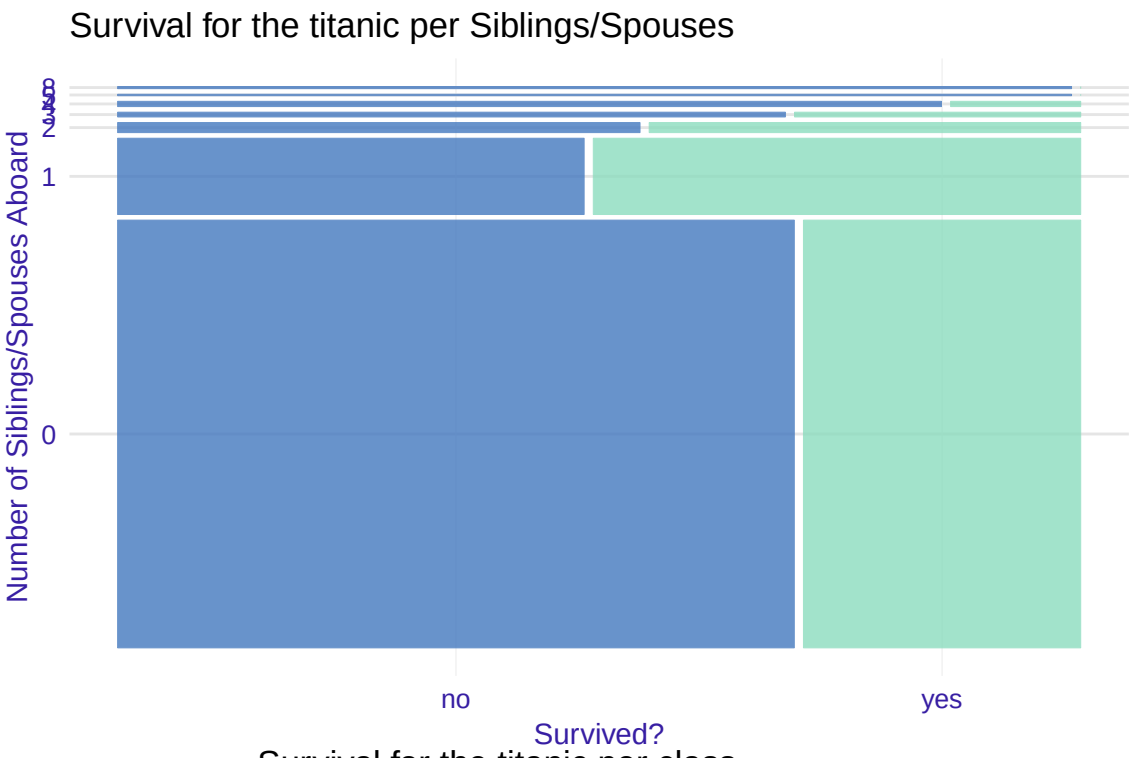
Figures XXX and XXX indicate that the proportion of survivors was larger for females and children below 5 years of age. This is most likely the result of the "women and children first principle that is often evoked in situations that require evacuation of persons whose life is in danger. The principle can, perhaps, partially explain the trend seen in Figures XXX and XXX, i.e., a higher proportion of survivors among those with 1-3 parents/children and 1-2 siblings/spouses aboard. Figure XXX 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 XXX shows that the proportion of survivors increased with the fare, which is consistent with the fact that the proportion

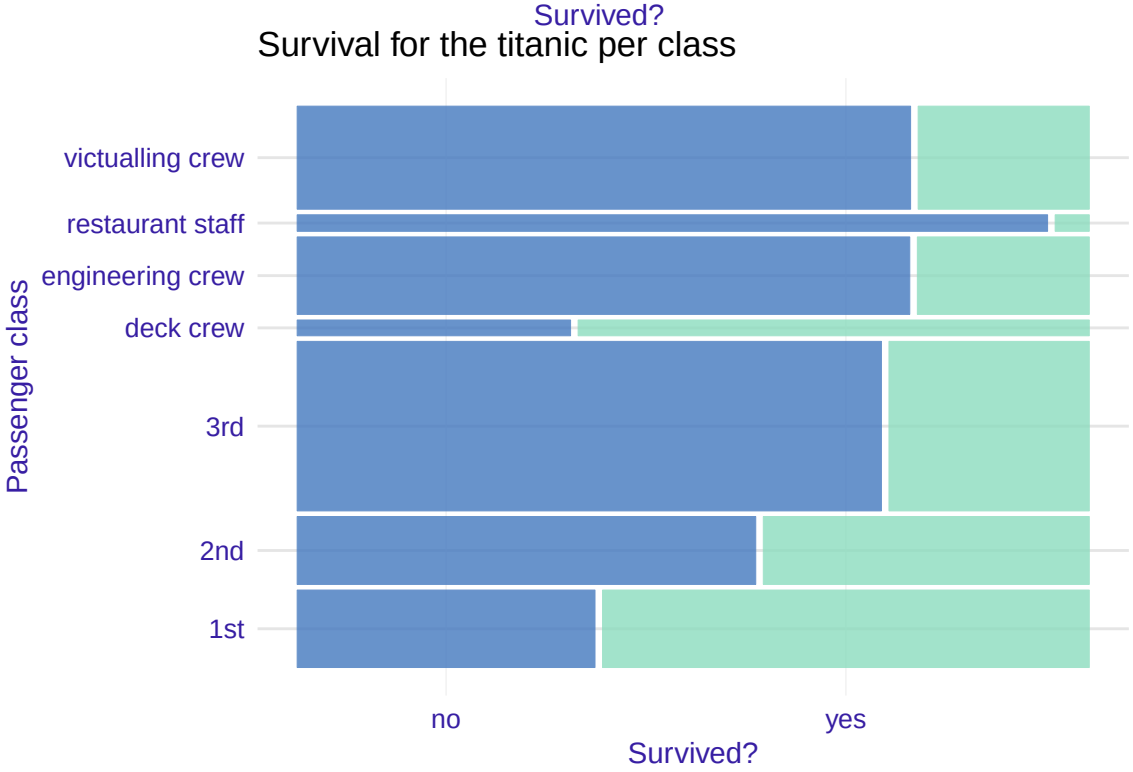
was higher for passengers travelling in the first and second class. Finally, Figures XXX and XXX do not suggest any noteworthy trends.

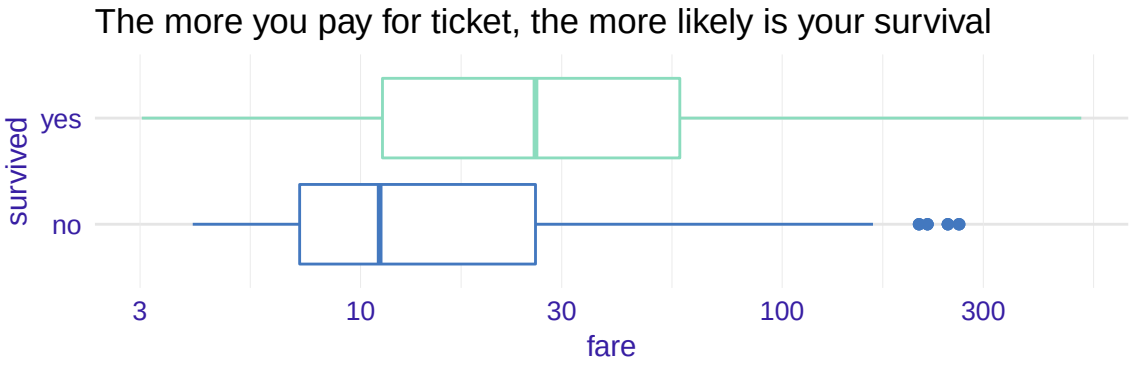


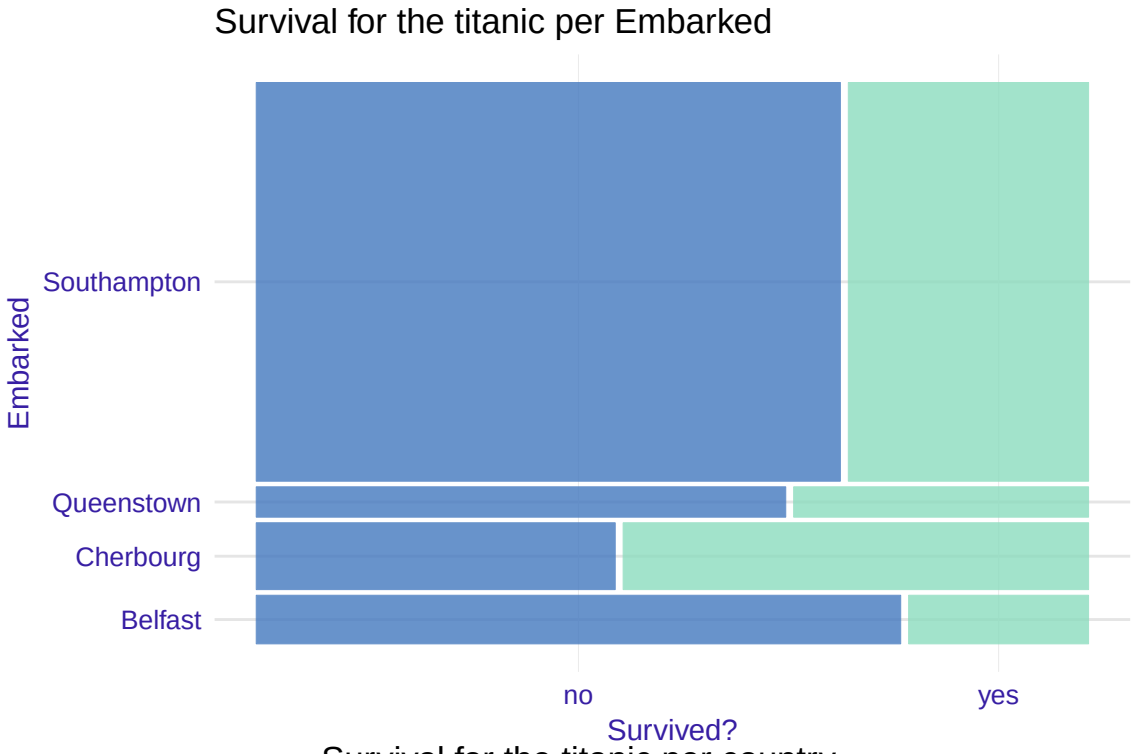


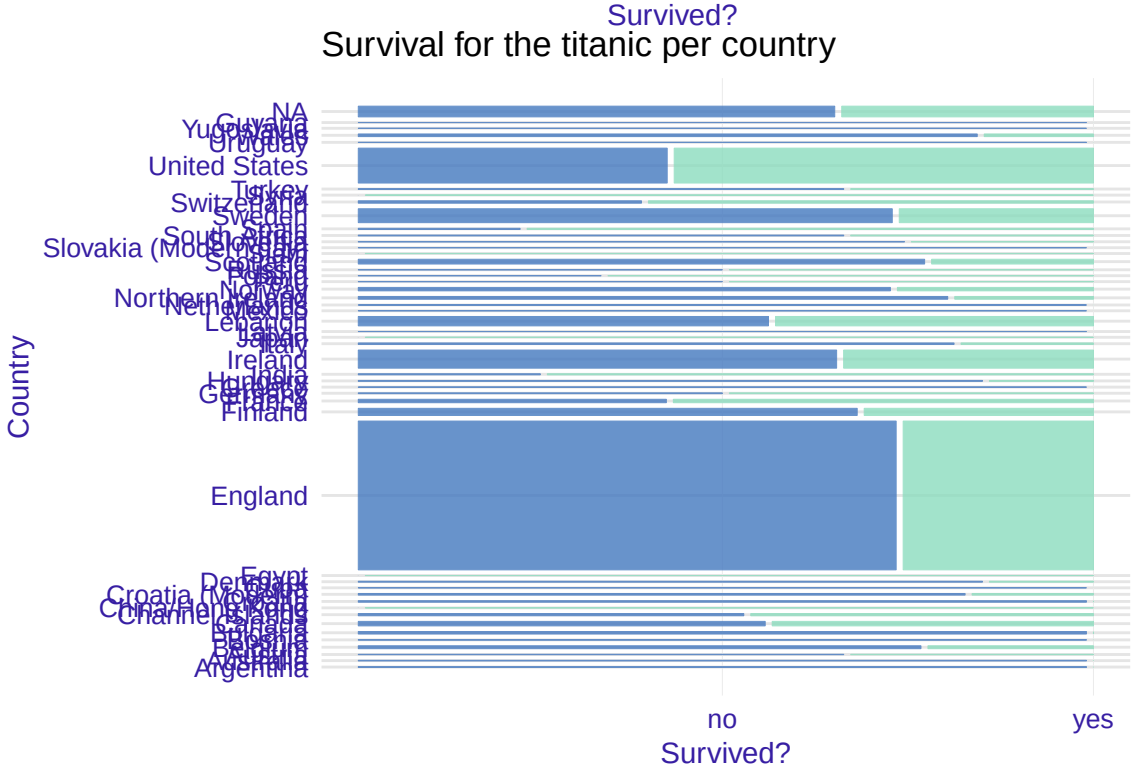

Survival for the titanic per Parents/Children
Number of Parents/Children Aboard
2
1
0
no
yes
Survived?


Survival for the titanic per Siblings/Spouses
Number of Siblings/Spouses Aboard
8
5
4
3
2
1
0
no
yes
Survived?


Survival for the titanic per class
Passenger class
victualling crew
restaurant staff
engineering crew
deck crew
3rd
2nd
1st
no
yes
Survived?


The more you pay for ticket, the more likely is your survival
survived
yes
no
3
10
30
100
300
fare





0.2.1.2 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 (Harrell Jr, 2018), to model the effect of age. [TOMASZ: IS THERE A REASON TO ASSUME A LINEAR EFFECT OF FARE?] The results of the model are stored in model-object `titanic_lmr_v6`, which will be used in subsequent chapters.

```
library("rms")
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
##  Obs            2207    LR chi2     752.73    R2       0.404    C       0.817
##   FALSE         1496    d.f.            17    g        1.648    Dxy     0.635
##   TRUE           711    Pr(> chi2) <0.0001    gr       5.195    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                4.5458 0.5460   8.33 <0.0001
##  gender=male             -2.7658 0.1587 -17.43 <0.0001
##  age                     -0.1186 0.0221  -5.37 <0.0001
##  age'                     0.6339 0.1629   3.89 <0.0001
##  age''                   -2.6621 0.7841  -3.39 0.0007
##  age'''                   2.8936 1.0131   2.86 0.0043
##  class=2nd               -1.1029 0.2486  -4.44 <0.0001
##  class=3rd               -2.0185 0.2466  -8.18 <0.0001
##  class=deck crew          1.1067 0.3468   3.19 0.0014
##  class=engineering crew  -0.9287 0.2615  -3.55 0.0004
##  class=restaurant staff  -3.1278 0.6571  -4.76 <0.0001
##  class=victualling crew  -1.0473 0.2558  -4.09 <0.0001
##  sibsp                   -0.4574 0.1012  -4.52 <0.0001
##  parch                   -0.0970 0.0992  -0.98 0.3282
##  fare                     0.0021 0.0020   1.05 0.2928
##  embarked=Cherbourg       0.7725 0.2844   2.72 0.0066
##  embarked=Queenstown      0.2653 0.3411   0.78 0.4368
##  embarked=Southampton     0.2287 0.2119   1.08 0.2805
##
```

0.2.1.3 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 [TOMASZ: REFERENCE?]. To fit the model, we apply the `randomForest()` function, with default settings, from the package with the same name (Liaw and Wiener, 2002).

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")
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: 18.31%
## Confusion matrix:
##       no yes class.error
## no  1399  97  0.06483957
## yes  307 404  0.43178622
```

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: 20.98%
## Confusion matrix:
##       no yes class.error
## no  1356 140  0.09358289
## yes  323 388  0.45428973
```

0.2.1.4 Gradient boosting

Finally, we consider the gradient-boosting model. The model is known for being able to grasp deep interactions between variables. [TOMASZ: WHAT ARE "DEEP INTERACTIONS"? REFERENCE?] 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 (Ridgeway, 2017). The results of the model are stored in model-object `titanic_gbm_v6`.

```
library("gbm")
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.
```

0.2.1.5 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 2nd class with no parents nor siblings with a ticket costing 72 pounds. First, we create a data frame `henry` that contains the data describing the passenger.

```
henry <- data.frame(
            class = factor("2nd", 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, henry, type = "fitted"))
```

```
##         1
## 0.4702145
```

The predicted probability is equal to 0.47.

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

```
(pred_rf = predict(titanic_rf_v6, henry, type = "prob"))
```

```
##      no   yes
## 1 0.654 0.346
## attr(,"class")
## [1] "matrix" "votes"
```

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

```
## [1] 0.4422346
```

As a result, we obtain the predicted probabilities of 0.35 and 0.44, 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.

[TOMASZ: GRADIENT-BOOSTING LEADS TO DIFFERENT PREDICTIONS AT EACH RUN. POSSIBLE TO "FREEZE" THE MODEL/PREDICTIONS? PERHAPS AT A RUN GIVING A DISTINCT PREDICTION? AT ONE POINT I GOT 0.6 FOR GBM, WHICH WAS MARKEDLY - AND INTERESTINGLY - DIFFERENT FROM LR AND RF.]

0.2.2 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.

The dataset is available in the `DALEX` package (Biecek, 2018b). 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

FIGURE 2 Warsaw skyscrapers by Artur Malinowski Flicker

- *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.

```
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
```

0.2.2.1 Data exploration

[TOMASZ: TO ADD, EVEN IF SHORT]

0.2.2.2 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:
## lm(formula = m2.price ~ ., data = apartments)
##
## Coefficients:
##         (Intercept)    construction.year              surface
##            5020.139               -0.229              -10.238
##                floor             no.rooms      districtBielany
##             -99.482              -37.730               17.214
##     districtMokotow       districtOchota        districtPraga
##             918.380              926.254              -37.105
## districtSrodmiescie        districtUrsus      districtUrsynow
##            2080.611               29.942              -18.865
##        districtWola     districtZoliborz
##             -16.891              889.973
```

0.2.2.3 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 (Liaw and Wiener, 2002).
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
```

0.2.2.4 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
```

```
library("randomForest")
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.

0.2.3 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 (Biecek, 2018b). 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 (veru good) [TOMASZ: CORRECT?]
- *salary*, the salary level, a numerical variable with values from 0 to 5 [TOMASZ: WHAT DOES 0 MEAN?]
- *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
```

0.2.3.1 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 [TOMASZ: REFERENCE]. We fit the model with the help of function `multinom` from package `nnet`. The function fits multinomial log-linear models by using the neural-networks approach. [TOMASZ: WHY THIS APPROACH?] 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")
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
```

0.2.3.2 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 (Liaw and Wiener, 2002). The results of the model are stored in model-object `HR_rf_v5`.

```
set.seed(59)
library("randomForest")
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: 27.31%
## Confusion matrix:
##           fired   ok promoted class.error
## fired      2274  384      197   0.2035026
## ok          525 1255      441   0.4349392
## promoted    201  395     2175   0.2150848
```

0.2.3.3 Model predictions

[TOMASZ: TO ADD]

0.2.4 List of models

In the previous sections we have built several predictive models for diffrent datasets. The models will be used in the rest of the book to illustrate the mdoel explanation methods and tools. For the ease of reference, we summarize the models in the table beloW. The binary model-objects can be donwloaded by using the attached `archivist` hooks (Biecek and Kosinski, 2017). [TOMASZ: THIS IS USEFUL FOR E-BOOK. HOW ABOUT THE PRINTED VERSION?]

Model name	Model generator	Dataset	Variables	Link to the model
`titanic_lmr_v6`	`rms:: lmr`	`DALEX:: titanic`	gender, age, class, sibsp, parch, fare, embarked	`archivist:: aread("pbiecek/models/f285c")`
`titanic_rf_v6`	`randomForest:: randomForest`	`DALEX:: titanic`	gender, age, class, sibsp, parch, fare, embarked	`archivist:: aread("pbiecek/models/92753")`
`titanic_rf_v3`	`randomForest:: randomForest`	`DALEX:: titanic`	gender, age, class	`archivist:: aread("pbiecek/models/bcd20")`
`titanic_gbm_v6`	`gbm:: gbm`	`DALEX:: titanic`	gender, age, class, sibsp, parch, fare, embarked	`archivist:: aread("pbiecek/models/2bdad")`
`apartments_lm_v5`	`stats:: lm`	`DALEX:: apartments`	construction .year, surface, floor, no.rooms, district	`archivist:: aread("pbiecek/models/55f19")`

Model name	Model generator	Dataset	Variables	Link to the model
`apartments_rf_v5`	`randomForest:: randomForest`	`DALEX:: apartments`	construction .year, surface, floor, no.rooms, district	`archivist:: aread("pbiecek/models/fe7a5")`
`HR_rf_v5`	`randomForest:: randomForest`	`DALEX:: HR`	gender, age, hours, evaluation, salary	`archivist:: aread("pbiecek/models/1ecfd")`
`HR_glm_v5`	`stats:: glm`	`DALEX:: HR`	gender, age, hours, evaluation, salary	`archivist:: aread("pbiecek/models/f0244")`

Instance-level explanation

0.3 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 4. 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^*.

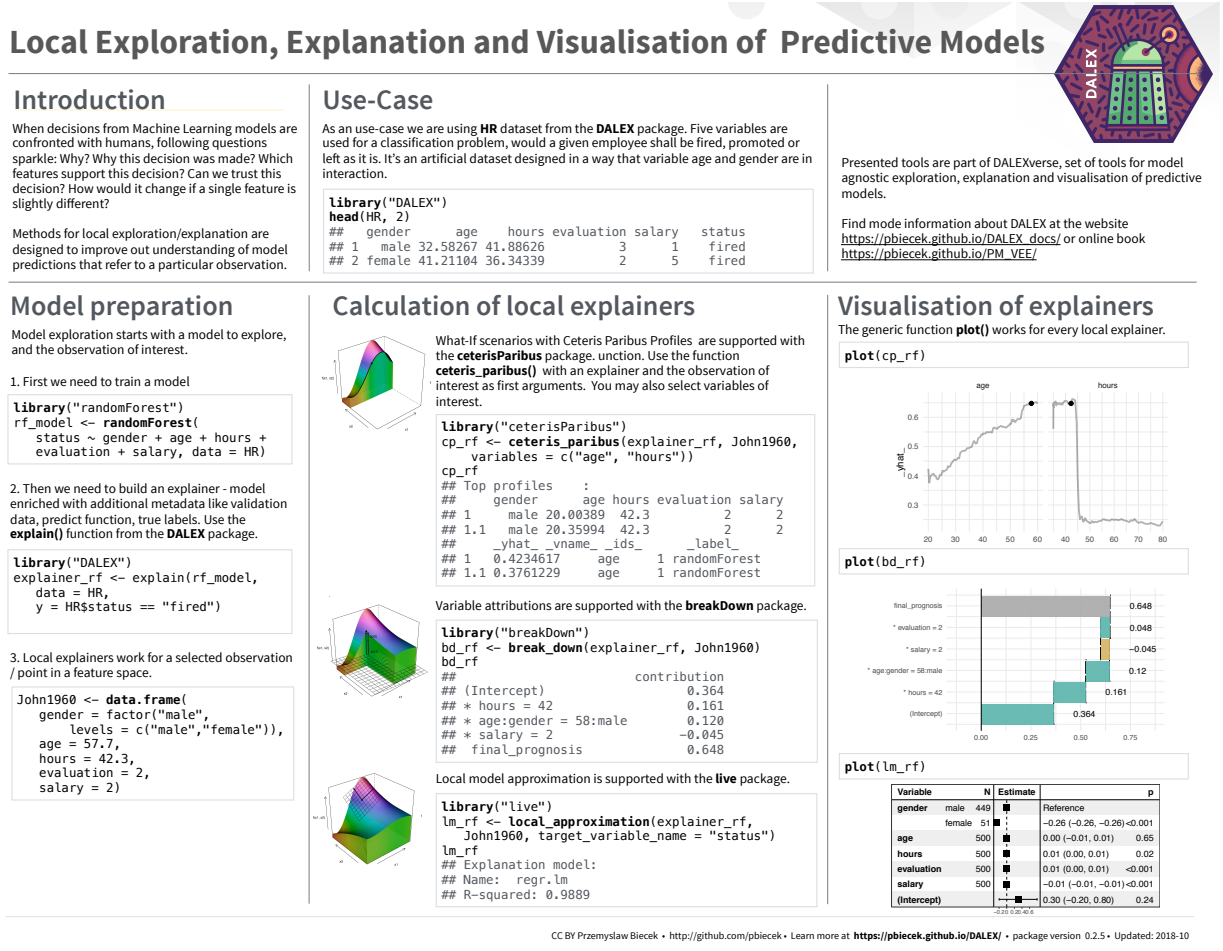


FIGURE 3 (fig:localDALEXsummary) Summary of three approaches to local model exploration and explanation.

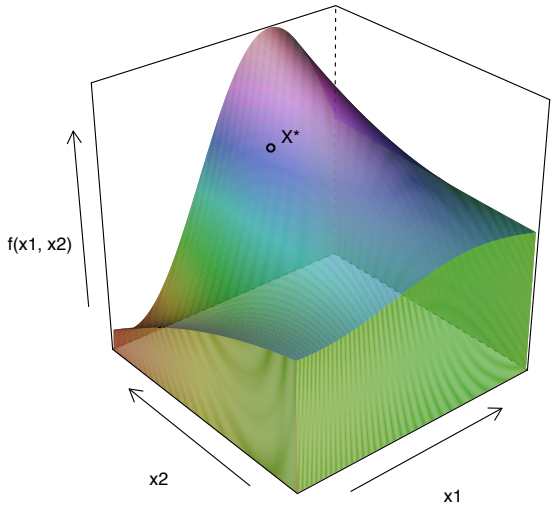


FIGURE 4 (fig:cutsSurfaceReady) Response surface for a model that is a function of two variables. We are interested in understanding the response of a model in a single point x*

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 0.4. An example is provided in panel A of Figure 5.
- Another approach is to analyze the curvature of the response surface (see Figure 4) 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 5. In Chapter 0.12 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" approach. An example is provided in panel C of Figure 5. In Chapter 0.8 we present two methods implementing this approach, sequential conditioning and average conditioning (also called Shapley values).

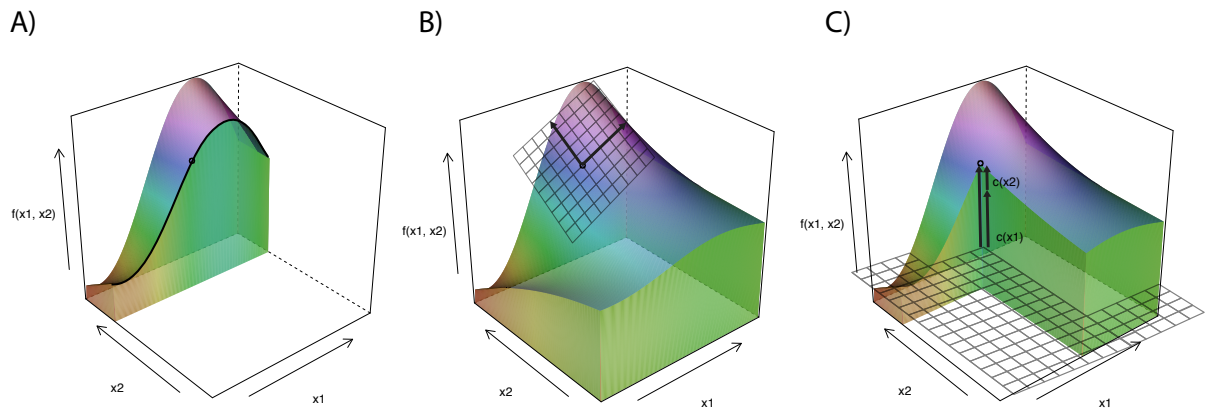


FIGURE 5 (fig:cutsTechnikiReady) Illustration of different approaches to instance-level explanation. Panel A presents a What-If analysis with Ceteris Paribus profiles. The profiles plot the model response as a function of a value of a single variable, while keeping the values of all other explanatory 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.

0.4 Ceteris Paribus Profiles - a tool for What-If analysis

0.4.1 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, asumming that effects of all other variables are unchanged. The main goal is to understand how changes in a single explanatory variable affects model predictions.

Explainers presented in this chaprter are linked to the second law introduced in Section 0.1.2, 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* (Goldstein et al., 2015a). It turns out 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.

0.4.2 Intuition

Panel A of Figure 6 presents a response surface for a `titanic_lmr_v6` model for two explanatory variables, *age* and *class*, from the *titanic* dataset (see Section 0.2.1). 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 that is

marked on the plot by a black dot. Ceteris-paribus (CP) profiles are one-dimensional profiles that examine the curvature across each dimension, i.e., for each variable. Panel B of Figure 6 presents the profiles corresponding to *age* and *class*. 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 Section 0.12). 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 intepret.

0.4.3 Method

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

[PBI: Do we need to assume that model is correct? first sentence may be removed]

Assume that $E_Y(Y|x^*) \approx f(x^*)$, where $f(x^*)$ is the value of the model at x^*. Note that x^* is a vector containing values for explanatory covariates. 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 resultinig from removing the j-th element from vector x^*. Moreover, let $x^{*|j} = z$ denotes 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 (approximate) expected value (prediction) of the model for Y on the value of j-th explanatory variable z, where z is taken to go through the range of values typical for the variable and values of all other variables in x^* are kept fixed at the values present in x^*.

For continuous explanatory variables a natural way to represent the CP function is to use a profile plot similar to the one presented in Figure 0.4.3. In the figure, the dot presents an instance prediction, i.e., prediction $f(x^*)$ for a single observation x^*. The curve shows how the prediction would change if the value of a particular explanatory variable is changed (in this case, "age"; see Section 0.2.3). It is worth oberving that the profile for logistic regression is smooth while for random forest model is expresses some variabilit. Moreover, for this instance (observation), the prediction would increase substantially if the value of the explanatory variable became lower than 20.

\begin{figure}

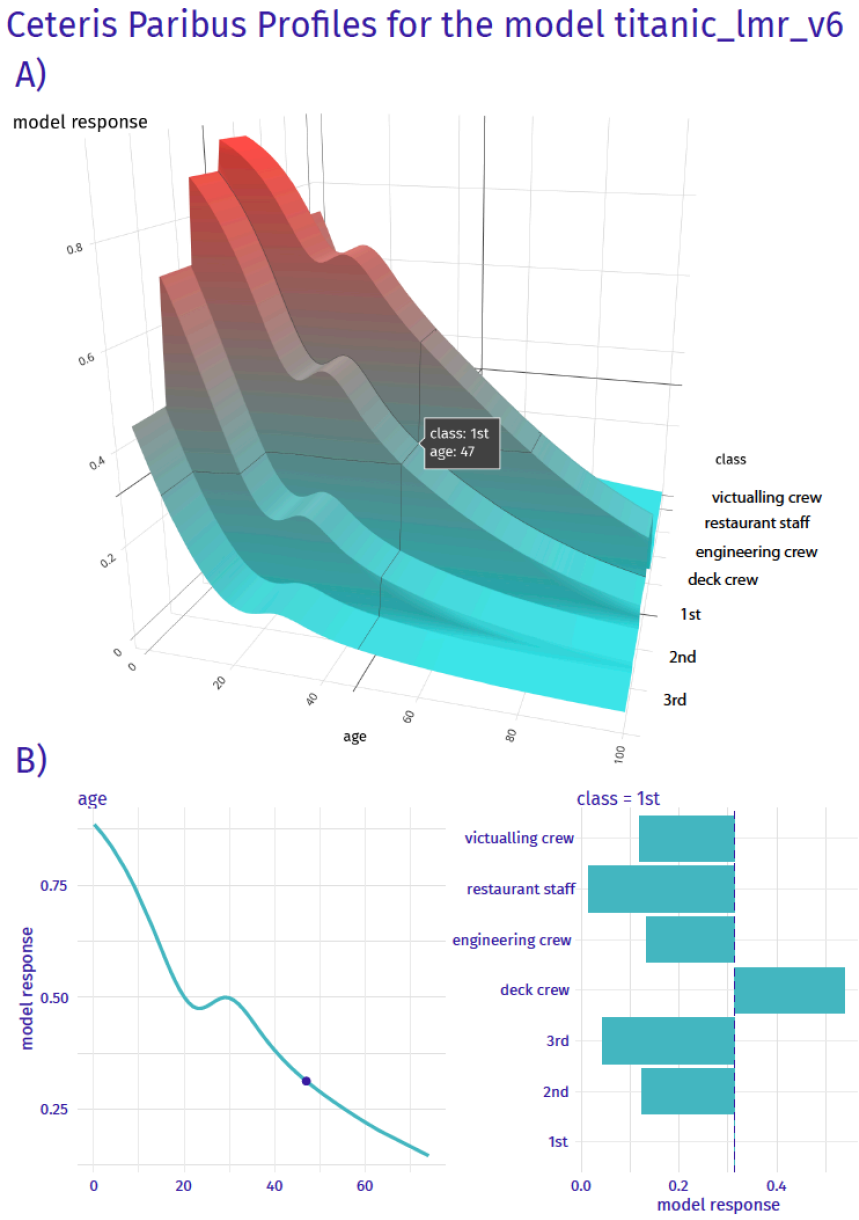
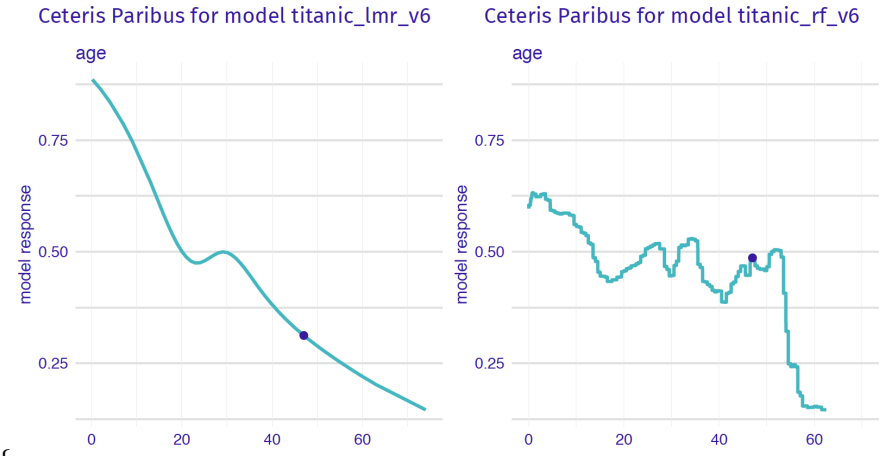


FIGURE 6 (fig:modelResponseCurveLine) A) Model response (prediction) surface. Ceteris-paribus (CP) profiles marked with black curves help to understand the curvature of the surface while changing only a single explanatory variable. B) CP profiles for individual variables, age (continuous) and class (categorical).



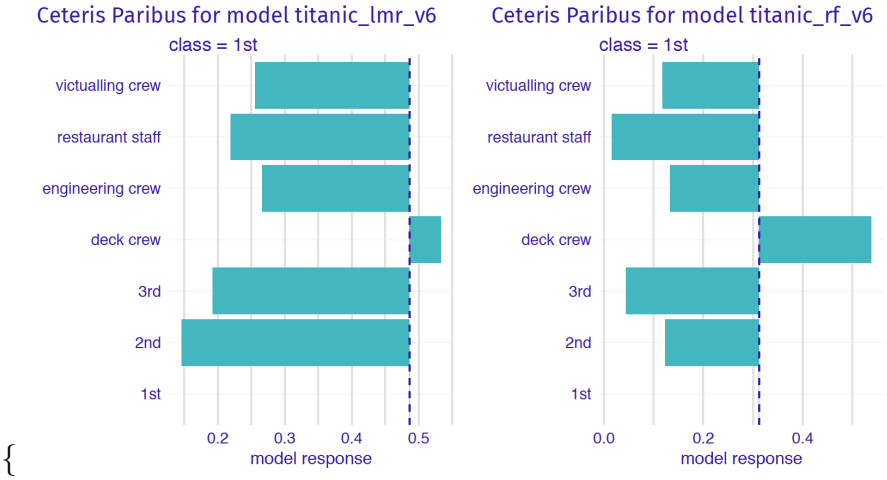
{

}

\caption{(fig:profileAgeRf) Ceteris-Paribus profile for two models, `titanic_lmr_v6` and `titanic_rf_v6` that predicts the probability of being rescued as a function of the age}
\end{figure}

For categorical explanatory variables a natural way to represent the CP function is to use barplot as the one presented in Figure 0.4.3.

\begin{figure}



{

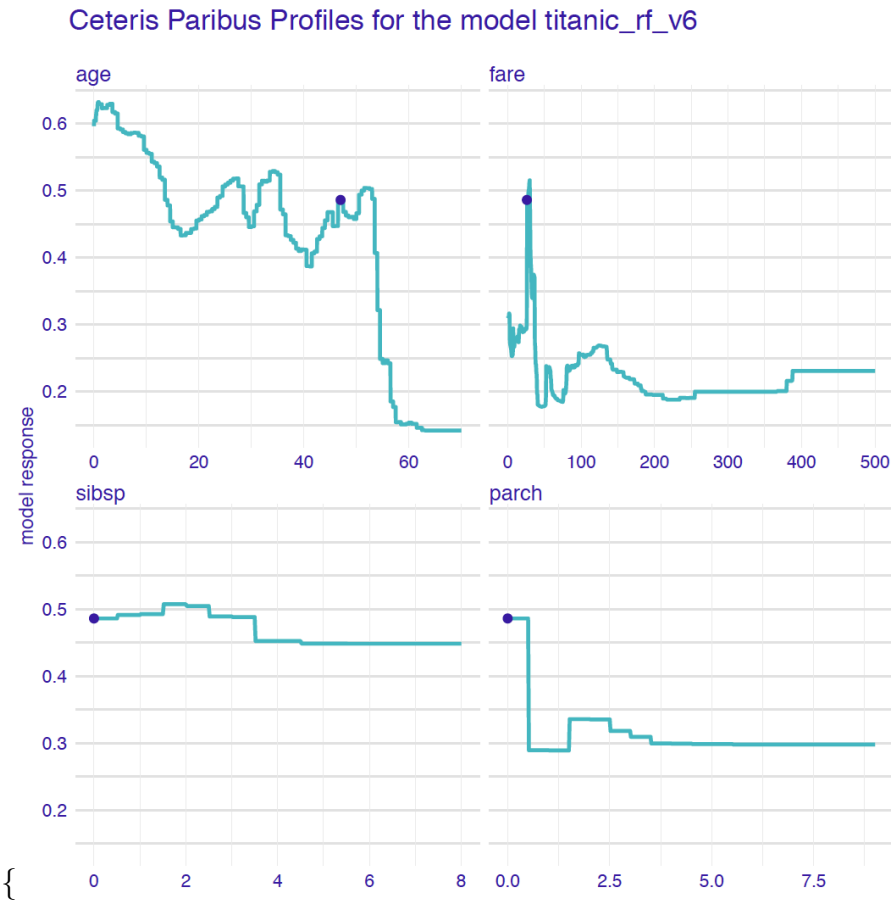
}

\caption{(fig:profileAgeRf2) Ceteris-Paribus profile for two models, `titanic_lmr_v6` and `titanic_rf_v6` that predicts the probability of being rescued as a function of the class}
\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 (Tufte, 1986). 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 0.4.3.

It helps if these panels are ordered so that the most important profiles are listed first. We discuss a method to assess importance of CP profiles in the next subsection.

\begin{figure}



{

}

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

0.4.4 Pros and cons

CP profiles offer a uniform, easy to comunicate 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 changing the other one. A special case are interactions, which require the use of 2D CP profiles that are more complex than the 1D ones. Also, in case of a model with hundreds or thousands of variables, the number of plots

to inspect may be daunting. Finally, visualization of CP profiles for factors (categorical explanatory variables) is not trivial, especially for factors with many nominal (unordered) categories (like, for example, a ZIP-code).

0.4.5 Code snippets for R

In this section we present key features of the R package `ingredients` (Biecek, 2019) 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` (O'Connell et al., 2017), `pdp` (Greenwell, 2017a), `ICEbox` (Goldstein et al., 2015b), `ALEPlot` (Apley, 2018a), `iml` (Molnar et al., 2018a).

In this section we use a random forest (Breiman et al., 2018) model `titanic_rf_v6` developed for the Titanic dataset (see Section 0.2.1). In particular, we deal with a binary classification problem - we want to predict the probability of survival for a selected passenger.

```
library("DALEX")
titanic <- na.omit(titanic)
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
##     age_c
## 1 (30,50]
## 2 (10,18]
```

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

CP profiles are calculated in four steps with the `ingredients` package.

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

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 (Biecek, 2018b). 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 use the training data as the validation dataset.

```
explain_titanic_rf <- explain(model = model_titanic_rf,
                              data = titanic[,-9],
                              y = titanic$survived == "yes",
                              label = "Random Forest v7")
```

2. Define the instance (observation) of interest.

CP profiles explore model around a single observation. Thus, in the exampe below, we define data frame `johny_d` with a single row. It 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. In particular, we compute the probability for each category of the dependent binary variable.

```
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
## 0.4511397
```

3. Calculate CP profiles

To obtain CP profiles, we use the `ceteris_paribus()` function. It requires the explainer-object and the instance data frame as arguments. By default, CP profiles are calculated for all numerical variables. To select a subset of variables, the `variables` argument can be used.

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_rf <- ceteris_paribus(explain_titanic_rf, johny_d,
                                 variables = c("age", "fare", "class", "gender"))
cp_titanic_rf
```

```
## Top profiles    :
##     class gender        age sibsp parch fare    embarked    _yhat_ _vname_
```

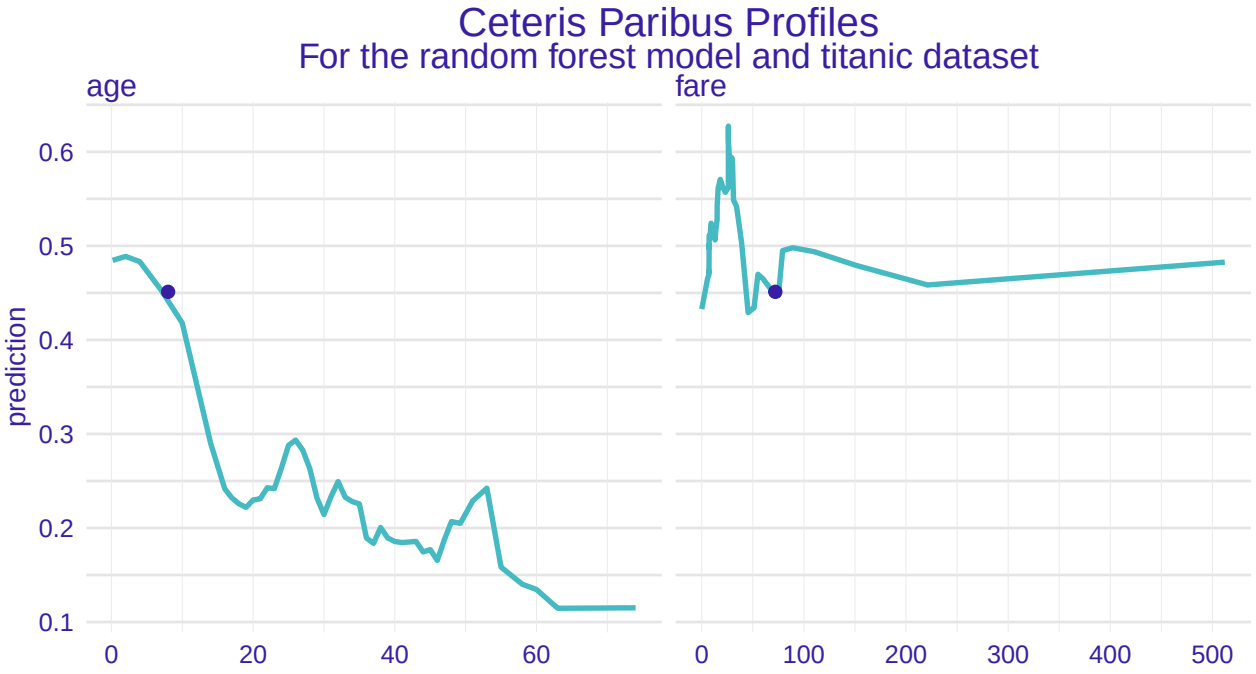
```
## 1     1st   male  0.1666667     0     0   72 Southampton 0.4845677     age
## 1.1   1st   male  2.0000000     0     0   72 Southampton 0.4888091     age
## 1.2   1st   male  4.0000000     0     0   72 Southampton 0.4832353     age
## 1.3   1st   male  7.0000000     0     0   72 Southampton 0.4531397     age
## 1.4   1st   male 10.0000000     0     0   72 Southampton 0.4179202     age
## 1.5   1st   male 14.0000000     0     0   72 Southampton 0.2901186     age
##     _ids_          _label_
## 1       1 Random Forest v7
## 1.1     1 Random Forest v7
## 1.2     1 Random Forest v7
## 1.3     1 Random Forest v7
## 1.4     1 Random Forest v7
## 1.5     1 Random Forest v7
##
##
## Top observations:
##   class gender age sibsp parch fare    embarked    _yhat_          _label_
## 1   1st   male   8     0     0   72 Southampton 0.4511397 Random Forest v7
##   _ids_
## 1     1
```

4. Plot CP profiles.

To obtain a graphical represenation of CP profiles, the generic `plot()` function can be applied to the data frame returend by the `ceteris_paribus()` function. It returns a `ggplot2` object that can be processed if needed.

The resulting plot can be enriched with additional data by adding functions `show_rugs` (adds rugs for the selected points), `show_observations` (adds observations), or `show_aggreagated_profiles` (see Section @ref{variableEngeneering}). All these functions can take additional arguments to modify size, color or linetype.

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



0.5 Ceteris Paribus Oscillations - a tool for local variable importance

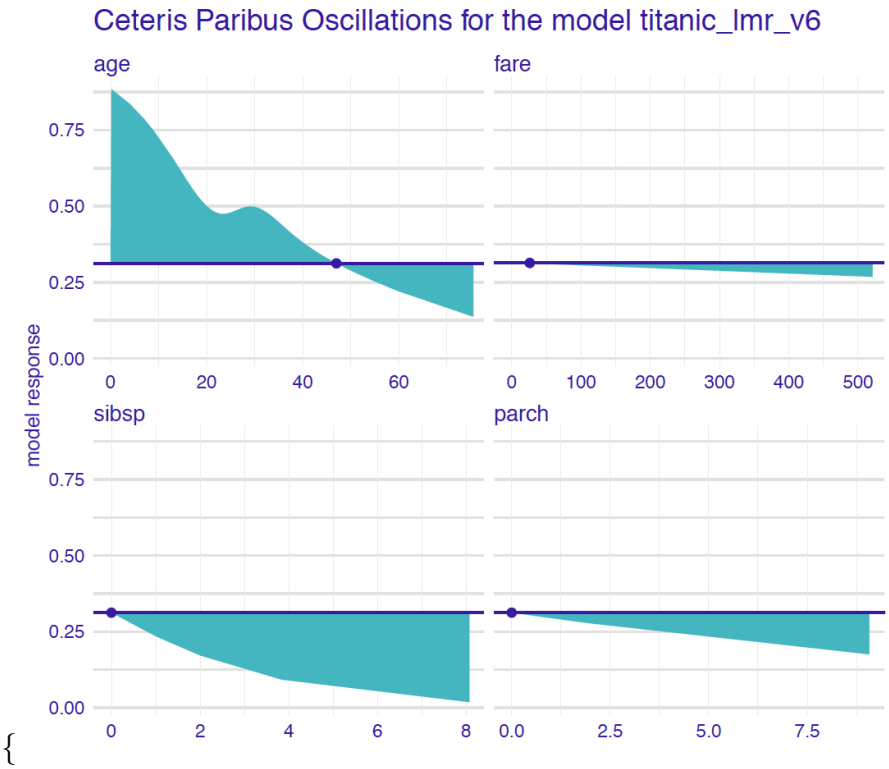
0.5.1 Introduction

Visual examination of Ceteris Paribus 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 profiles we need a measure that would summarize the impact of a selected variable on model's predictions. We will propose now a solution closely linked with CP profiles, but the issue is also discussed also in the next chapter.

0.5.2 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 0.5.2 illustrates the idea behind measuring oscillations. The larger the highlighted area, the more important is the variable.

\begin{figure}



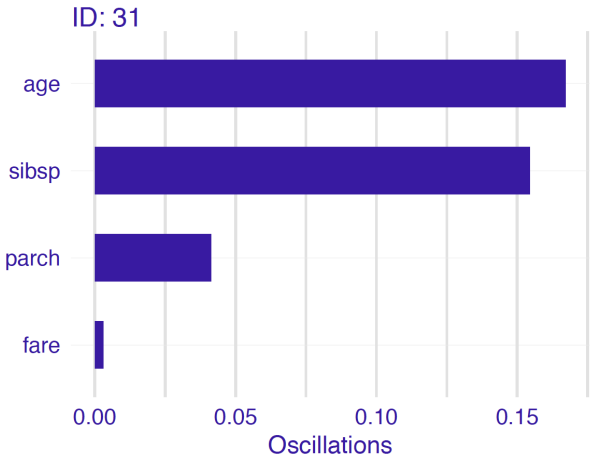
{

}

\caption{(fig:CPVIPprofiles) The value of the red are area summarizes CP oscillations and provides the average absolute deviations between the CP profile and the instance prediction. This example is for the `titanic_lmer_v6` model and the `titanic` dataset} \end{figure}

Figure 0.5.2 provides a plot of variable importance measures for different variables for the `titanic_lmer_v6` and the `titanic` dataset. The wider the interval, the larger the CP-profile oscillations for a particular explanatory variable. Thus, Figure 0.5.2 indicates that the most important variable for prediction for a selected observation is `age`, followed by `sibsp`.

\begin{figure}



{

}

\caption{(fig:CPVIP1) Variable-importance measures calculated for Ceteris-paribus oscillations for observation ID: 31. This example is for the `titanic_lmer_v6` model and the `titanic` dataset} \end{figure}

0.5.3 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 $x^* = (1, 0)$.

0.5.4 Pros and cons

By using oscillations of Ceteris Paribus profiles it is possible to select the most important variables for a single instance. The methodology is easy to extend to two or more variables. Oscillations are easy to interpret.

There are several issues related to the use of the CP oscillations. Two most serious are:

- if two variables are correlated, then Ceteris Paribus profiles may be misleading as they do not take correlations into account.
- local feature importance do not sum up to model prediction. In next sections we will introduce variable attribution measures that eliminate this problem.

0.5.5 Code snippets for R

In this section we present key features of the R package `ingredients` (Biecek, 2019) 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/`.

In this section we use a random forest (Breiman et al., 2018) model `titanic_rf_v6` developed for the Titanic dataset (see Section 0.2.1). In particular, we deal with a binary classification problem - we want to predict the probability of survival for a selected passenger.

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

model_titanic_rf <- randomForest(survived == "yes" ~ gender + age + class + embarked +
                                   fare + sibsp + parch,  data = titanic)
explain_titanic_rf <- explain(model = model_titanic_rf,
                              data = titanic[,-9],
                              y = titanic$survived == "yes",
                              label = "Random Forest v7")
```

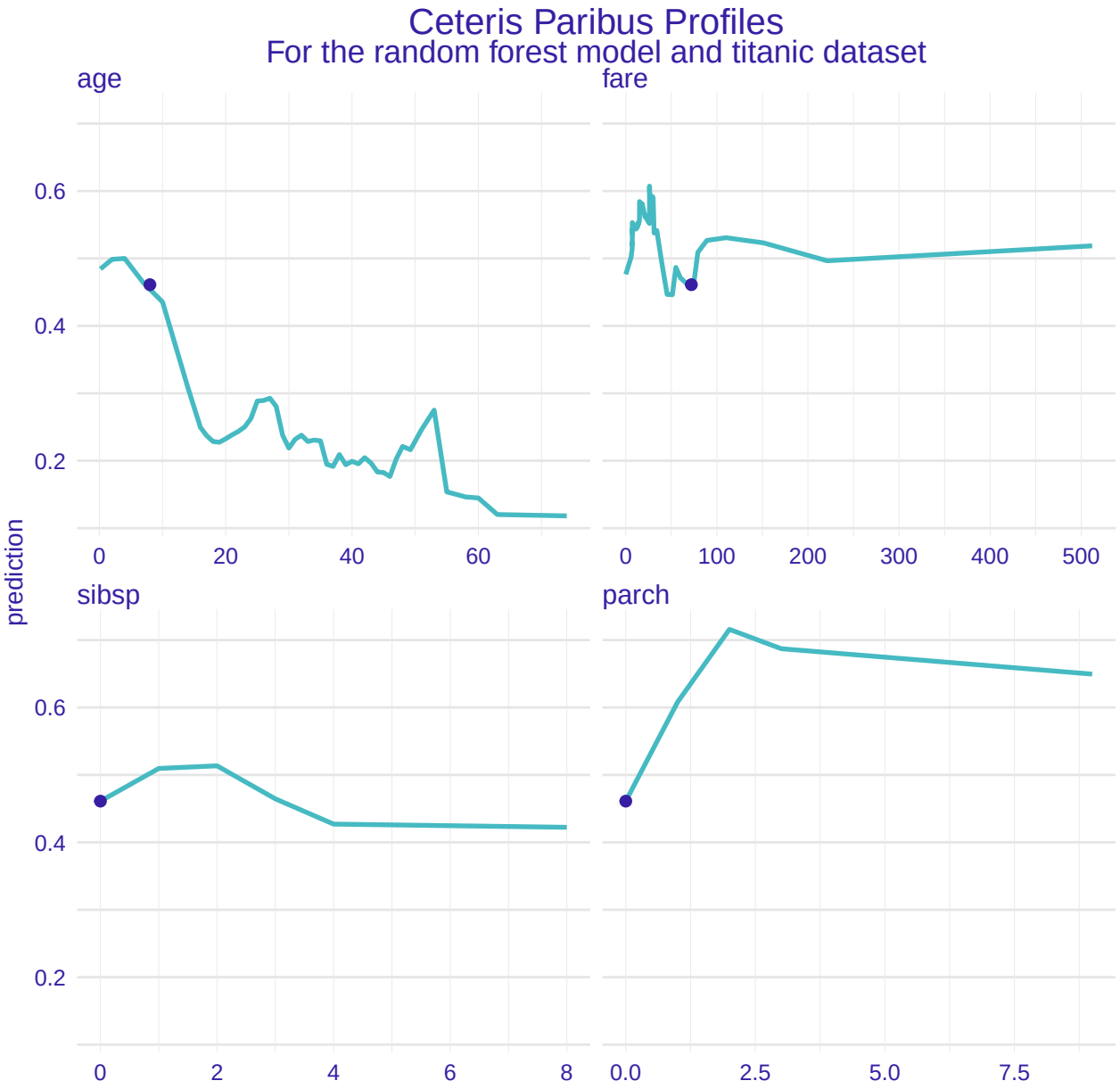
In order to calculate oscillations we need to first calculate Ceteris Paribus profiles for a selected observation.

Let us use the `johny_d` instance again.

```
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"))
)
```

Profiles are calculated with the `ceteris_paribus()` function.

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

cp_titanic_rf <- ceteris_paribus(explain_titanic_rf, johny_d,
                                 variables = c("age", "fare", "sibsp", "parch"))

plot(cp_titanic_rf) +
  show_observations(cp_titanic_rf, variables = c("age", "fare", "sibsp", "parch")) +
  ggtitle("Ceteris Paribus Profiles", "For the random forest model and titanic dataset")
```



For selected profiles we can calculate oscillations with the `calculate_oscillations` function.

```
cpo_titanic_rf <- calculate_oscillations(cp_titanic_rf)
cp_titanic_rf$`_ids_` = "Johny D"
cpo_titanic_rf
```

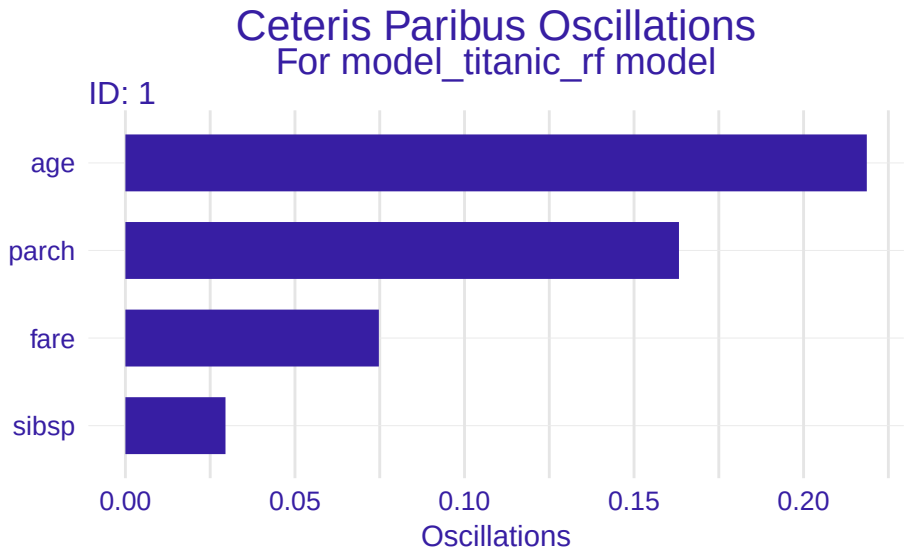
```
##   _vname_ _ids_ oscillations
## 1     age     1   0.21864350
## 4   parch     1   0.16324640
## 2    fare     1   0.07473998
## 3   sibsp     1   0.02951691
```

The `calculate_oscillations()` function returns an object of the class

`ceteris_paribus_oscillations`, which has a form of a data frame but has also overloaded `plot()` function.

Use it to plot local variable importance for a selected observation

```
plot(cpo_titanic_rf) +
  ggtitle("Ceteris Paribus Oscillations",
          "For model_titanic_rf model")
```



It looks like for Jony D. the two most important features are `age` and `parch`. Higher `age` of this passenger would significantly lower the chances of survival while higher `parch` would increase these odds.

0.6 Ceteris-Paribus 2D Profiles - a tool for pairwise interactions

0.6.1 Introduction

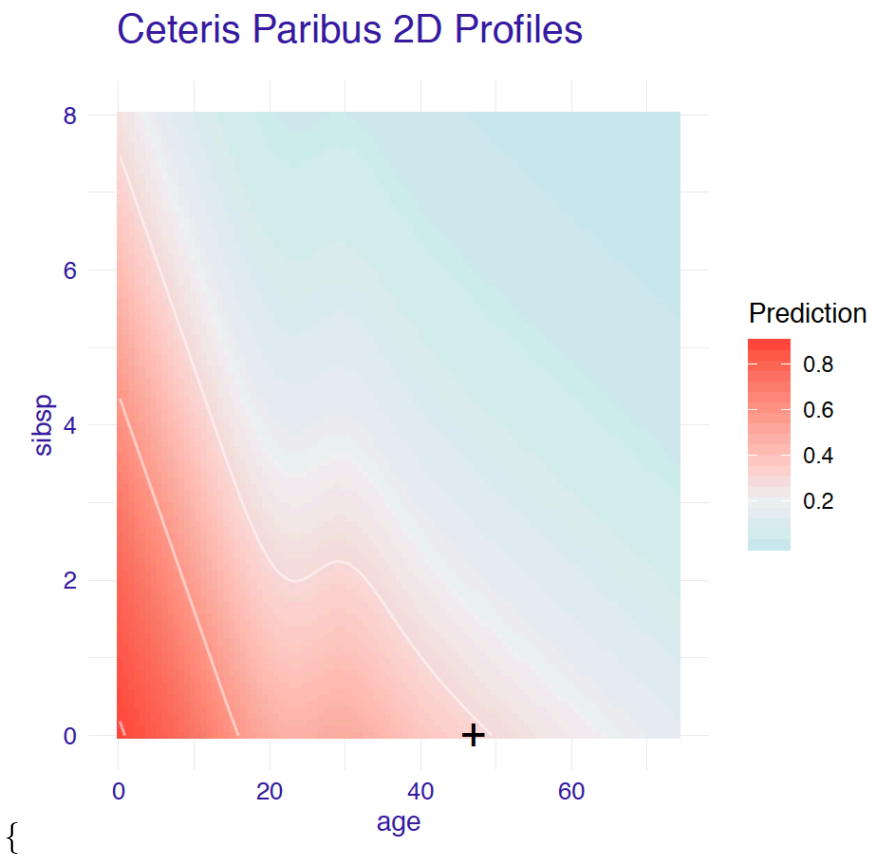
The definition of Ceteris Paribus profiles, given in Section 0.4, may be easily extended to two or more explanatory variables. Also, the definition of the variance importance measure $vip_j^{CP}(x^*)$ have a straightforward extension for a larger number of variables.

Such extension is useful to identify or visualize of pairwise interactions between variables.

0.6.2 Intuition

Figure 0.6.2 presents a response surface for a `titanic_lmr_v6` model for two explanatory variables, *age* and *sibsp*, from the *titanic* dataset (see Section 0.2.1). We are interested in the change of the model prediction induced by each of the variables.

\begin{figure}



{

}

\caption{(fig:profile2d) Ceteris-paribus profile for a pair of explanatory variables for a model `titanic_lmer_v6` for `age` and `sibsp` variables.} \end{figure}

0.6.3 Method

The definition of Ceteris Paribus profiles may be easily extended to two or more explanatory variables. A two-dimensional Ceteris Paribus profile for model f, explanatory variables j and k, and point x^* is defined as follows:

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

Ceteris Paribus 2D profile is a function that provides the dependence of the prediction of the model on the values of j-th and k-th explanatory variables z_1 and z_2, respectively, where z_1 and z_2 are taken to go through the range of values typical for the variables, and values of all other variables in x^* are kept fixed at the values present in x^*.

The corresponding variance importance measure would be defined as follows:

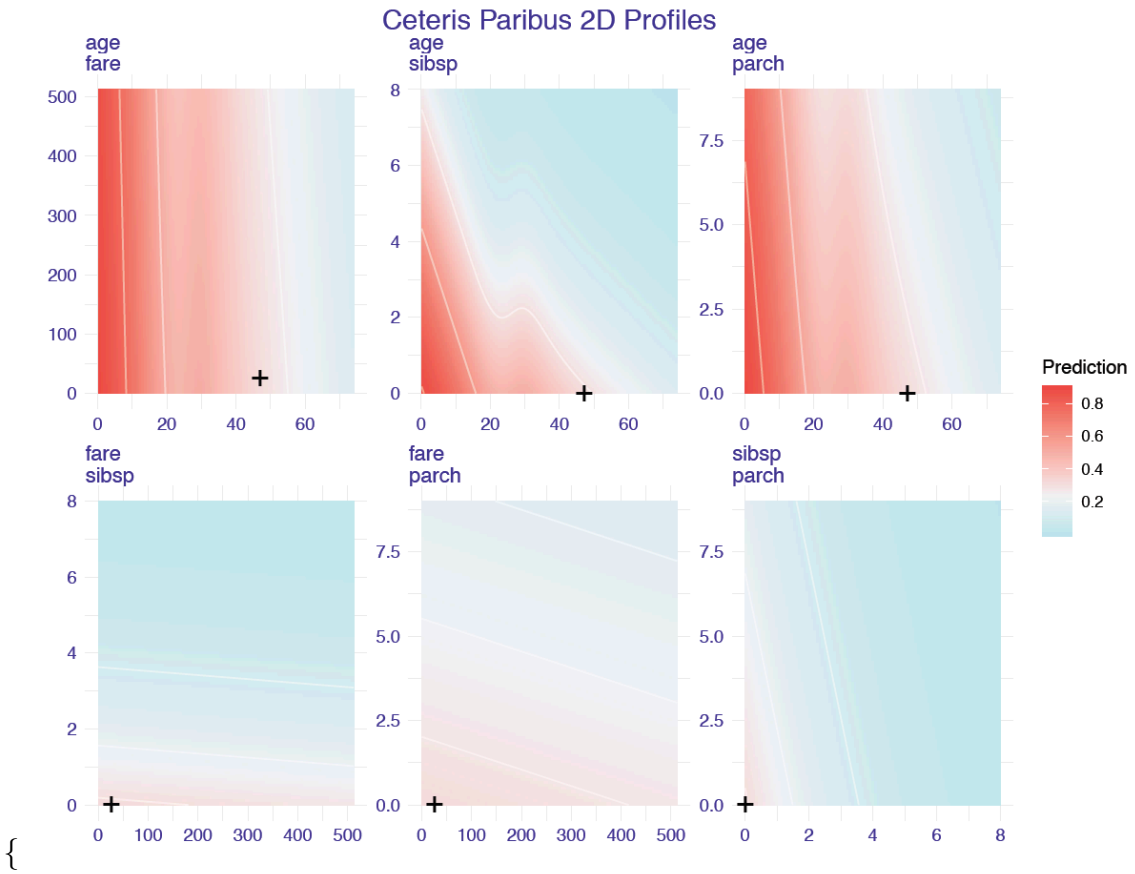
$$vip^{CP}_{j,k}(x^*) = \int_{\mathcal{R}}\int_{\mathcal{R}} |CP^{f,(j,k),x^*}(z_1,z_2) - f(x^*)| g^{j,k}(z_1,z_2) dz_1 dz_2 = E_{X_j,X_k}[|CP^{f,j,x^*}(X_j,X_k) - f(x^*)|],$$

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 two-dimensional (2D) CP profiles.

A natural way to visualize 2D CP profiles is to use a heat map for all pairs of explanatory variables as in Figure 0.6.3.

\begin{figure}



{

}

\caption{(fig:profile2dAll) Ceteris-paribus profile for all pairs of explanatory variables for a `titanic_lmer_v6` model. Black cross marks the coordinates of 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.

0.6.4 Pros and cons

Two-dimensional CP profiles can be used to identify the presence of pairwise interactions in a model.

But number of pairs may be large and 3d profiles are more difficult to read than 1d profiles.

0.6.5 Code snippets for R

In this section we present key features of the R package `ingredients` (Biecek, 2019) 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` (O'Connell et al., 2017) or `ICEbox` (Goldstein et al., 2015b).

In this section we use a random forest (Breiman et al., 2018) model `titanic_rf_v6` developed for the Titanic dataset (see Section 0.2.1). In particular, we deal with a binary classification problem - we want to predict the probability of survival for a selected passenger.

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

model_titanic_rf <- randomForest(survived == "yes" ~ gender + age + class + embarked +
                                   fare + sibsp + parch,  data = titanic)
explain_titanic_rf <- explain(model = model_titanic_rf,
                              data = titanic[,-9],
                              y = titanic$survived == "yes",
                              label = "Random Forest v7")
```

In order to calculate oscillations we need to first calculate Ceteris Paribus profiles for a selected observation.

Let us use the `johny_d` instance again.

```
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"))
)
```

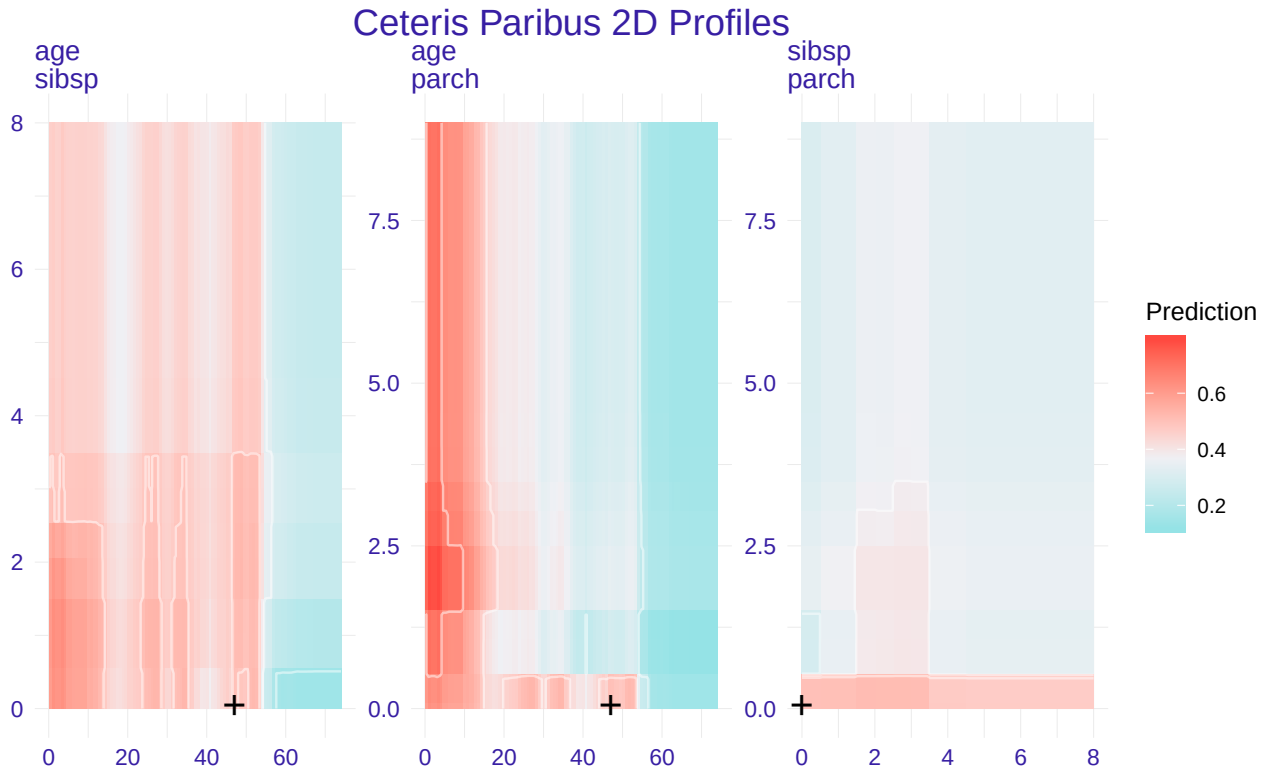
2D profiles are calculated with the `ceteris_paribus_2d()` function. By default all pairs between continuous variables are being plotted, but one can limit number of variables for consideration through the `variables` argument.

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

wi_rf_2d <- ceteris_paribus_2d(explain_titanic_rf, observation = titanic[31,], variables = c("age", "sibsp", "p
head(wi_rf_2d)
```

```
##          y_hat    new_x1 new_x2 vname1 vname2            label
## 31   0.5977772 0.1666667   0.00    age  sibsp Random Forest v7
## 31.1 0.5977772 0.1666667   0.08    age  sibsp Random Forest v7
## 31.2 0.5977772 0.1666667   0.16    age  sibsp Random Forest v7
## 31.3 0.5977772 0.1666667   0.24    age  sibsp Random Forest v7
## 31.4 0.5977772 0.1666667   0.32    age  sibsp Random Forest v7
## 31.5 0.5977772 0.1666667   0.40    age  sibsp Random Forest v7
```

Result is an object of the class `ceteris_paribus_2d_explainer` with overloaded `print()` and `plot()` function.

```
plot(wi_rf_2d) +
  theme(legend.position = "right", legend.direction = "vertical") + ggtitle("Ceteris Paribus 2D Profiles")
```



In the presented example both variables `age` and `sibsp` are important and influence the model response.

0.7 Local diagnostic with Ceteris-Paribus profiles

0.7.1 Introduction

What do you think, is it possible that model is good on average but terrible for some observations?

I think that most of us would agree. But how we can check if a single prediction is „unlucky"? In this chapter we present two techniques for local model diagnostic that are based on Ceteris Paribus Profiles.

It may happen that global performance of the model is good, while for some particular observations the fit is very bad. Local fidelity helps to understand how good is the model fit at a particular observation. In this section we show how to use Ceteris Paribus profiles to validate local model fidelity.

The idea behind fidelity plots is to select a number of observatons ("neighbors") from the validation dataset that are closest to the 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 neighbours, we may add residuals to the plot to evaluate the local fit of the model.

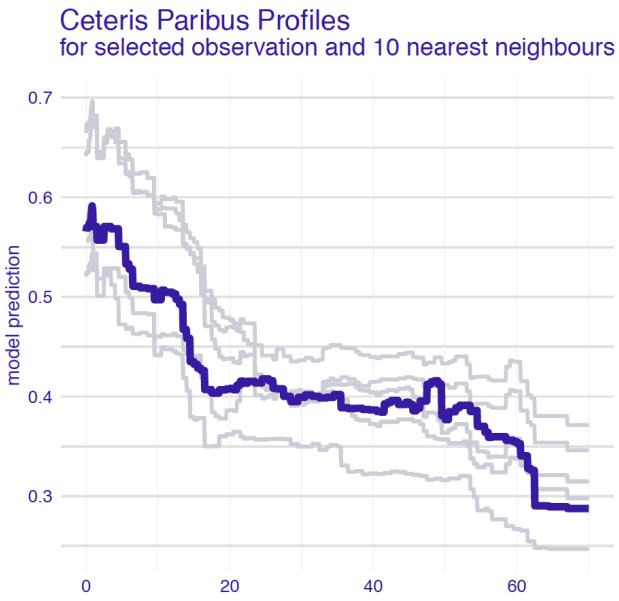
0.7.2 Intuition

We want to diagnose model behavior around a single instance prediction.

One approach to local model validation is to examine how model behaves for points from training data. See for example profiles presented in Figure 0.7.2. Ceteris Paribus Profiles are almost parallel and close to each other what suggests that model is stable around the point of interest.

It looks like small changes in the model input (represented by nearest neighbors) do not influence model response in an unpredicted way.

\begin{figure}



{

}

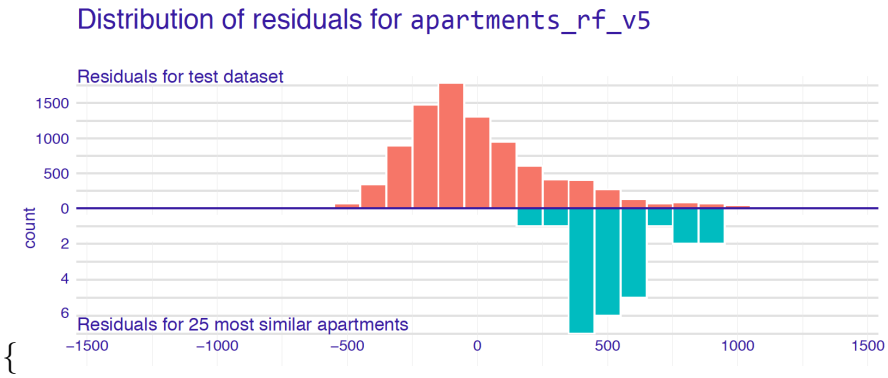
\caption{(fig:profileWith10NN) Ceteris-paribus profiles for a selected instance (dark violet line) and 10 nearest neighbors (light grey lines). Note that these profiles are almost parallel and close to each other what suggests the stability of the model. This example is based on the `titanic_rf_b6` model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

Once we consider nearest neighbors we can also look closer on the model fit around a specific point of interest. See for example the Figure 0.7.2. The back-to-back histogram shows distribution of residuals for the whole validation data and selected neighbors.

Distribution of residuals for the whole validation dataset is rather symmetric and centered around 0. Residuals for neighbors are centered on 500, what means that the average model bias around the point of interest is close to 500.

In the figure we showed an example in which residuals for nearest neighbors are larger than average residuals. Thus the local model fit is smaller than the global model fit.

\begin{figure}



{

}

\caption{(fig:profileBack2BackHist) Back-to-back histograms for residuals calculated with the `apartments_rf_v5` model on `apartments` data. Upper panel correspond to residuals calculated for all observations in the validation dataset while the bottom panel correspond to residuals calculated for nearest neighbors. Their residuals are higher than average model residuals.} \end{figure}

0.7.3 Method

Local diagnostic is based on three elements.

- identification of nearest neighbors,
- calculation and visualization of Ceteris Paribus profiles for selected neighbors,
- analysis of residuals for nearest neighbors.

Let us discuss them point by point.

0.7.3.1 Nearest neighbors

How many nearest neighbors shall we choose and what metric can we use?

Answer for both questions is of course *it depends*.

- The smaller the number of neighbors is the more local is the analysis. But very small numbers lead to larger variability of the diagnosis. The default value in the `select_neighbours()` function is `n = 20`.
- The distance measure if very important. The higher the dimension the more crucial is this choice. For tabular data we additionally need to face a problem of different distributions of different features. The default choice in the `select_neighbours()` function is to use `distance = gower_dist()` function from the `gower` package (van der Loo, 2017). The Gower distance is an average from normalized distances $d_k()$ calculated for particular features.

$$d_{gower}(x^i, x^j) = \frac{1}{p}\sum_{k=1}^{p} d_k(x^i_k, x^j_k).$$

* Distance can be calculated based on selected variables. In the `select_neighbours()` function one can specify them in the `variables` argument.

0.7.3.2 Profiles for neighbors

Once nearest neighbors are identified we can compare Ceteris Paribus profiles for selected variables.

If the number of features is large, then we can easily end up with bunch of very similar small plots with spaghetti profiles. Better strategy would be to focus only on profiles for the k-most important features. In this chapter we show profiles only for a single variable. This makes examples easier to read without reducing the generality.

For better readability we show only profiles for continuous variables.

0.7.3.3 Residuals

Ceteris Paribus profiles that assess model stability. In addition we can add examination of the local residuals that assess local model fit.

Residuals for observation x^i are here defined as the difference between the observed y^i and predicted x^i, that is

$$r^i = y^i - f(x^i).$$

0.7.3.4 Local Fidelity Plot

Both residuals and Ceteris Paribus Profiles for nearest neighbors can be presented in a single plot, called Local Fidelity Plot. See an example in the Figure 7.

Such plots are complex as they convey lots of information. Yet it is not that hard to learn how to read this information.

The Figure 7 shows meaning of every element of the plot. For a single instance, here the passenger Johnny D, we can read how large are residuals for its neighbor, whatever residuals are biased or symmetric, if Ceteris Profiles are stable and parallel.

0.7.4 Pros and cons

Local diagnostic plots may be very helpful to validate if

- model is additive locally (CP profiles are parallel),

FIGURE 7 (fig:localFidelityPlots) Elements of Local Fidelity Plots. Vertical intervals correspond to residuals, the smaller the more accurate is the model. Residuals are color coded, so we can easier recognized if they are biased or not, here there are balanced. Ceteris Paribus Profiles for neighbors are marked with grey lines. Stable model will have profiles close to each other; additive model will have parallel lines.

- model is stable locally (CP profiles are close to each other),
- local fit is good (residuals are small and symmetric).

The drawback is that such plots are quite complex and lack objective measures for quality of fit. As most diagnostic tools it's suited for exploratory analysis.

0.7.5 Code snippets for R

In this section we show how to use an R package `ingredients` (**?**) for the Local Fidelity Plots. More details and examples can be found at `https://modeloriented.github.io/ingredients/`.

In this section we use a random forest (Breiman et al., 2018) model `titanic_rf_v6` developed for the Titanic dataset (see Section @ref{model_titanic_rf}). In particular, we deal with a binary classification problem - we want to predict the probability of survival for a selected passenger.

```
library("DALEX")
titanic <- na.omit(titanic)
library("randomForest")
titanic_rf_v6 <- randomForest(survived == "yes" ~ gender + age + class + embarked +
                                    fare + sibsp + parch,  data = titanic)

explain_titanic_rf <- explain(model = titanic_rf_v6,
                              data = titanic[,-9],
                              y = titanic$survived == "yes",
                              label = "Random Forest v7")
```

Local diagnostic is specified for an instance. Let us define a new observation - 8 years old Johnny D, male passenger of the 1st class.

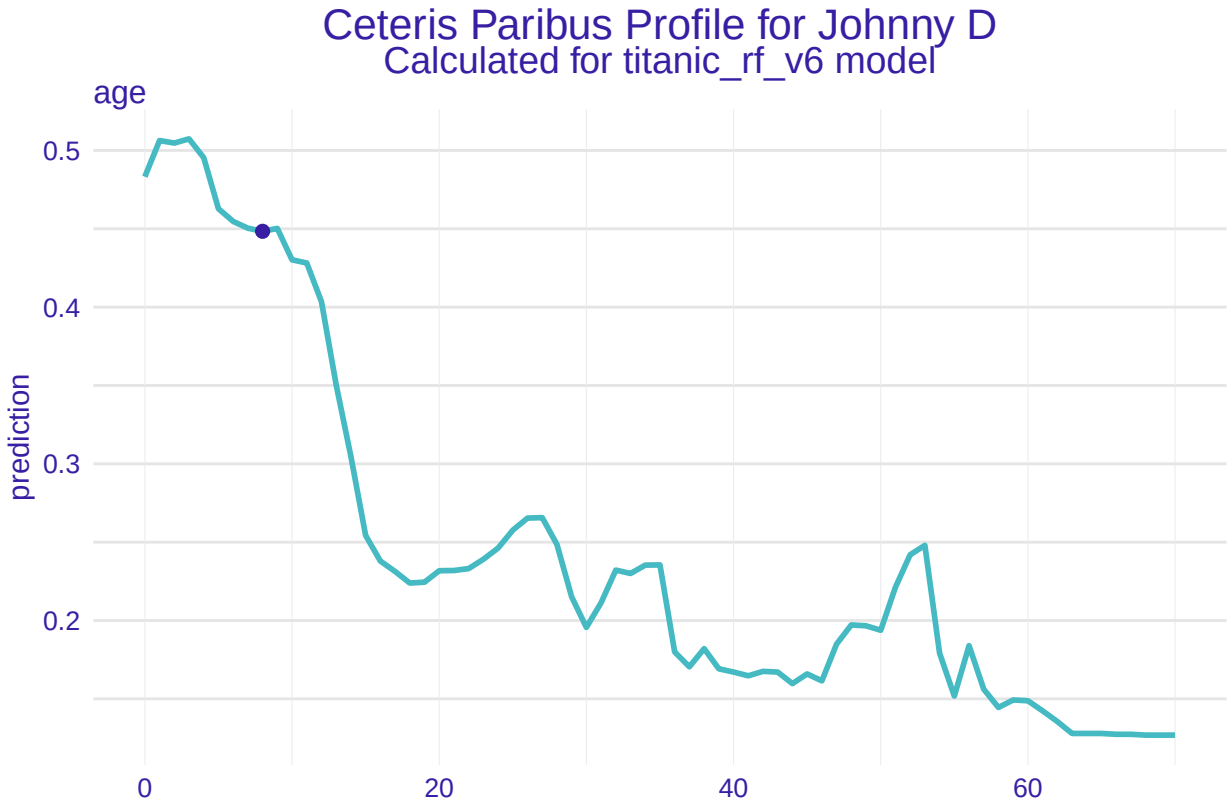
```
johny_d <- data.frame(
  class = factor("1st", levels = c("1st", "2nd", "3rd", "deck crew", "engineering crew",
                                   "restaurant staff", "victualling crew")),
  gender = factor("male", levels = c("female", "male")),
  age = 8,
  sibsp = 0,
  parch = 0,
  fare = 72,
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

0.7.5.1 Start with Ceteris Paribus Profile

First we will plot a Ceteri Paribus Profiles for the observation of interese.

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

```
cp_johny <- ceteris_paribus(explain_titanic_rf, johny_d, variable_splits = list(age = 0:70))
plot(cp_johny) +
  show_observations(cp_johny, variables = "age") +
  ggtitle("Ceteris Paribus Profile for Johnny D","Calculated for titanic_rf_v6 model")
```

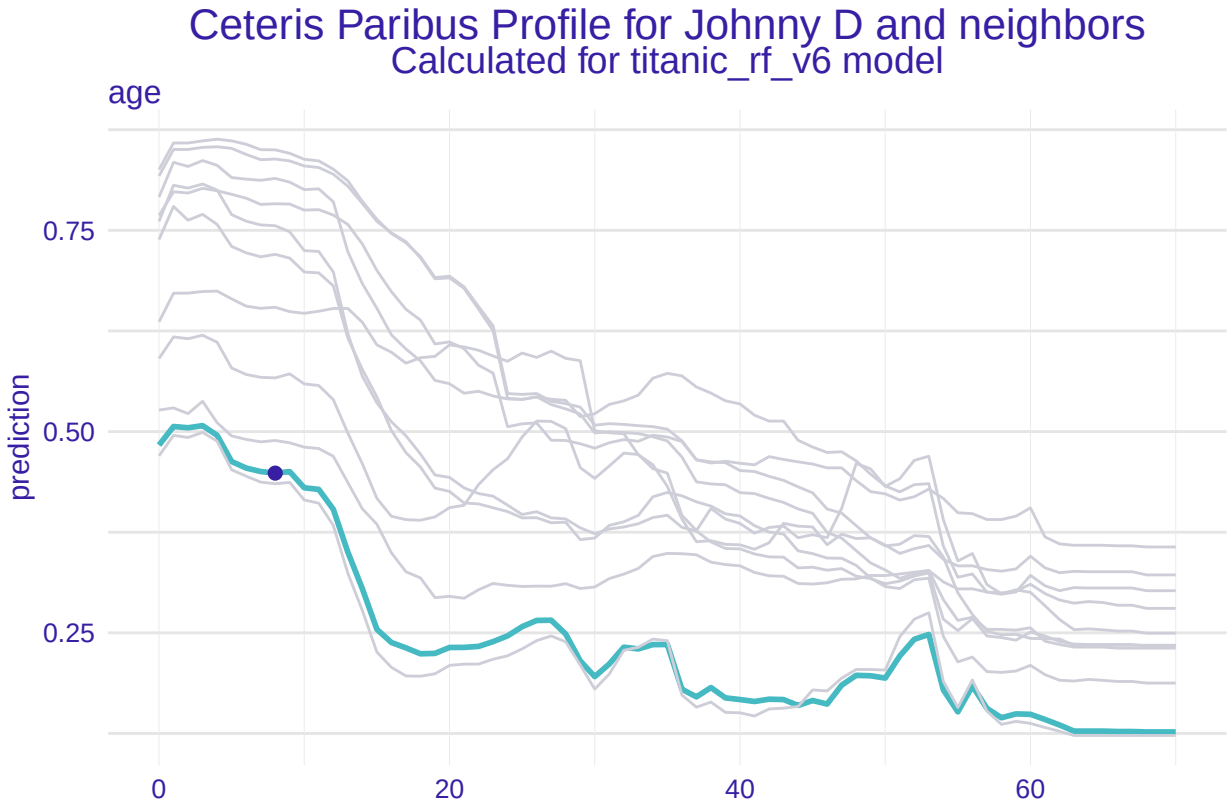


In the model chances of survival diminish with age. Johnny D is lucky to have chances higher than 50%.

0.7.5.2 Add profiles for neighbors

Second we need to identify neighbors and plot their profiles.

```
johny_neighbors <- select_neighbours(titanic, johny_d, n = 10, variables = c("age", "class", "gender"))

cp_johny_neighbors <- ceteris_paribus(explain_titanic_rf, johny_neighbors,
                                      y = johny_neighbors$survived == "yes",
                                      variable_splits = list(age = 0:70))

plot(cp_johny) +
  show_profiles(cp_johny_neighbors, color = '#ceced9') +
  show_observations(cp_johny, variables = "age") +
  ggtitle("Ceteris Paribus Profile for Johnny D and neighbors","Calculated for titanic_rf_v6 model")
```

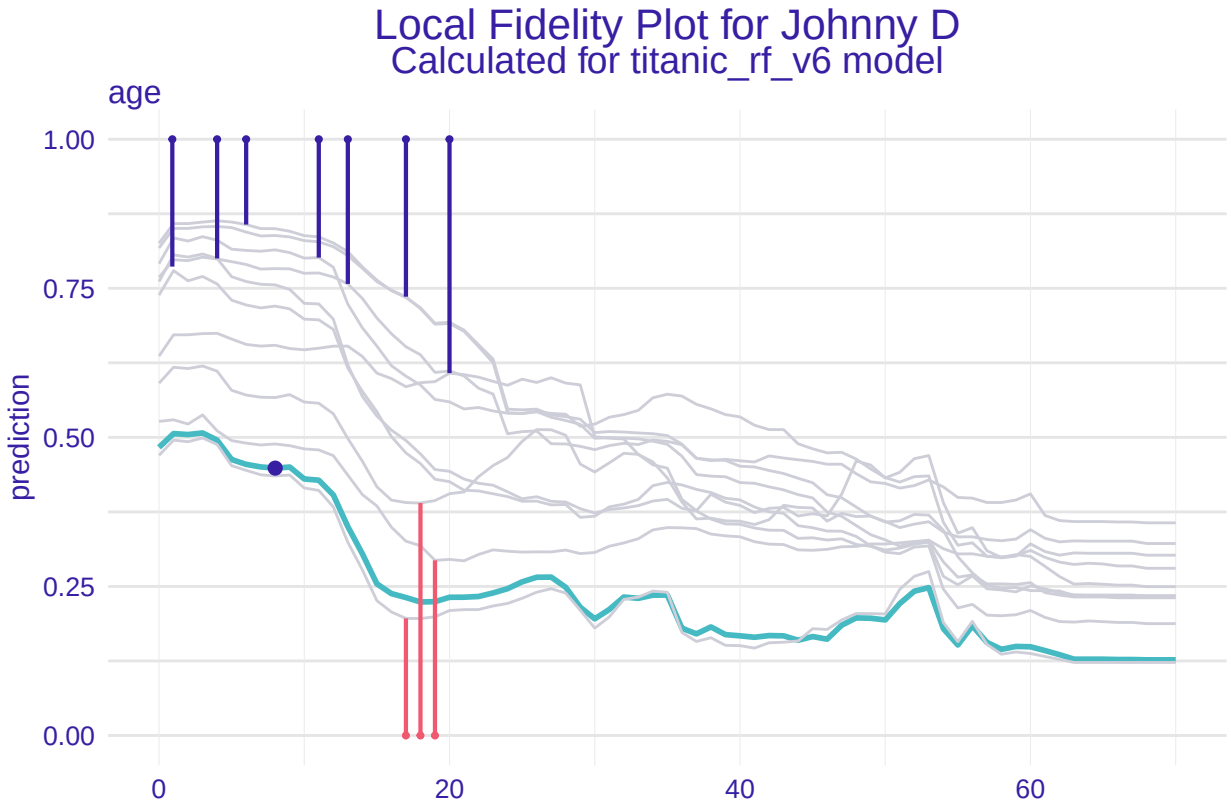


Almost all profiles behave in a similar way. The relation between age and model response is on average decreasing for almost every neighbor.

0.7.5.3 Add residuals for neighbors

And the last step is to add lines for residuals.

```
plot(cp_johny) +
  show_profiles(cp_johny_neighbors, color = '#ceced9') +
  show_observations(cp_johny, variables = "age") +
  show_residuals(cp_johny_neighbors, variables = "age") +
  ggtitle("Local Fidelity Plot for Johnny D","Calculated for titanic_rf_v6 model")
```



Added residuals works increase the trust in this model. Neighbors that in fact died have lower profiles while neighbors that survived have predictions higher than 50%.

0.8 Variable attribution for linear models

0.8.1 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 0.9 and 0.11.

0.8.2 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:

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

0.8.3 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 [TOMASZ: INSERT REF], when we will discuss interactions.

0.8.4 Example: Wine quality

Figure 8 shows the relation between alcohol and wine quality, based on the wine dataset (Cortez et al., 2009). The linear model is

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

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 corrrespond to the intuition that, in the presence of a positive relation, a smaller alcohol concentration should imply a lower quality of the wine.

0.8.5 Pros and Cons

The proposed definition of $v(f, x^*, i)$ is linked neither with scale nor location of X_i; hence, it is easier to understand than, for instance, the standardized value of β_i. [TOMASZ: I AM NOT SURE ABOUT THIS ARGUMENT. THE DEFINITION OF v() INVOLVES CENTERING.] 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; it only presents model coefficients in a different way. [TOMASZ: I AM NOT SURE ABOUT THIS ARGUMENT. MODEL COMPLEXITY IS WHAT IT IS. HOW CAN WE REDUCE IT?]

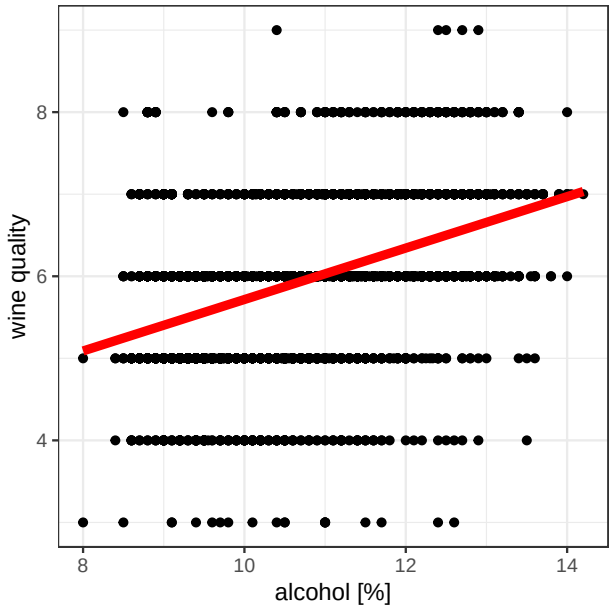


FIGURE 8 (fig:attribution1a)Relation between wine quality and concentration of alcohol assessed with linear model

0.8.6 Code snippets

In this section, we present an example of computing variable attributions using the `HR` dataset (see Section 0.26.1 for more details).

To calculate variable attributions for a particular point, first we have got to define this point:

```
new_observation <- 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 = 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, new_observation)) # 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, new_observation, 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()`.

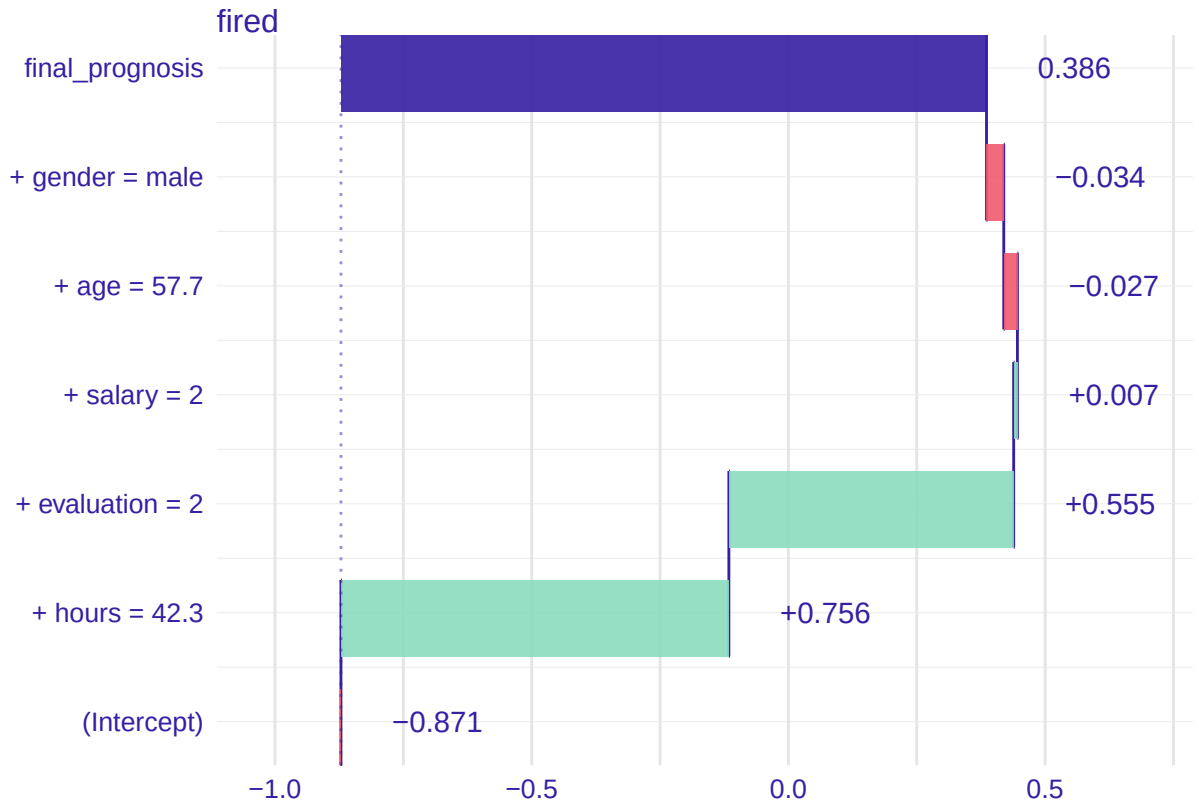
```
library("DALEX")

explainer_fired <- explain(model_fired,
                  data = HR,
```

```
                y = HR$status == "fired",
                label = "fired")

(attribution <- single_prediction(explainer_fired, new_observation))
```

```
##                    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)
```



0.9 Variable attributions

In the Section 0.8 we introduced a method for calculation of variable attributions for linear models. This method is accurate, based directly on the structure of the model. But for most popular machine learning models we cannot assume that they are linear nor even additive.

0.9.1 Intuition

For any model we may repeat the intuition presented in Section 0.8 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 original dataset. The red dot stands for average is is an estimate of expencted 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 `surface` fixed to value `35`. The

next for corresponds to average model prediction with variables `surface` set to `35` and `floor` set to `1`, and so on. The top 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 repaced by the x_i^*. But finaly we are not interestes 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 yellow bar. They sum up to final model prediction, which is denoted by a grey bar in this example.

0.9.2 Method

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

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

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

where *baseline* stands for average model response.

Note that the equation above may be rewritten as

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

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

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

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

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

See for example Figure 10. In the first ordering the contribution of variable `age` is calculated as 0.01, while in the second the contribution is calculated as 0.13. Such differences are related to the lack of additivness of the model $f()$. Propositions presented in next two sections present different solutions for this problem.

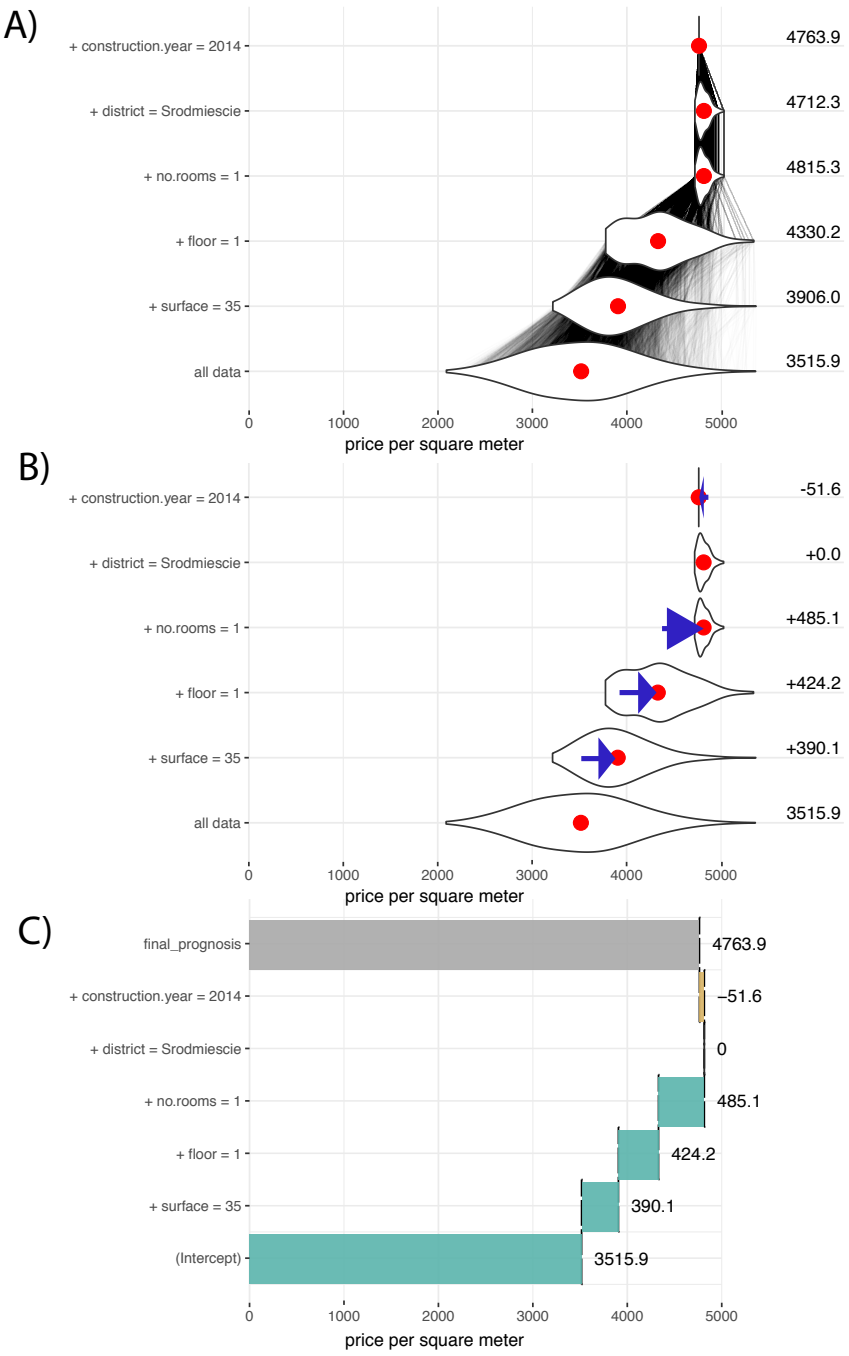


FIGURE 9 (fig:BDPrice4) Break Down Plots show how variables move the model prediction from population average to the model prognosis for a single observation. A) The last row shows distribution of model predictions. Next rows show conditional distributions, every row a new variable is added to conditioning. The first row shows model prediction for a single point. Red dots stand for averages. B) Blue arrows shows how the average conditional response change, these values are variables contributions. C) Only variable contributions are presented.

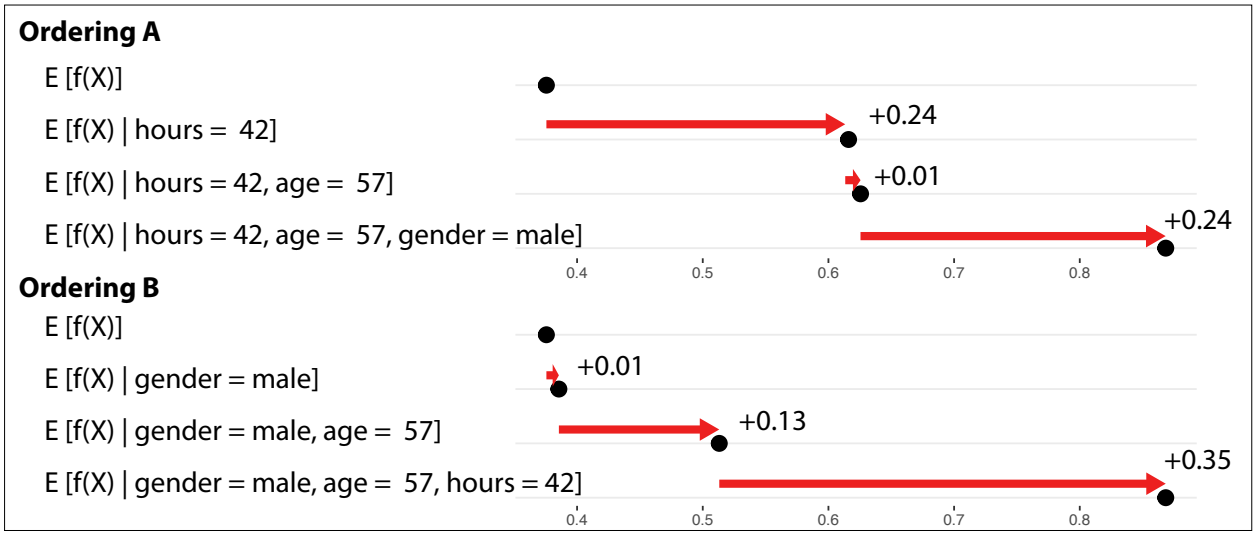


FIGURE 10 (fig:ordering) Two different paths between average model prediction and the model prediction for a selected observation. Black dots stand for conditional average, red arrows stands for changes between conditional averages.

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

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: Hire or Fire?

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

	Ei f(X)	scorei
hours	0.616200	0.230614
salary	0.225528	0.160058
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 `breakDown2` package for R (Biecek, 2018a). This package covers all features presented in this chapter. It is available on CRAN and GitHub. Find more examples at the website of this package `https://pbiecek.github.io/breakDown2/`.

Model preparation

In this section we will present an example based on the `HR` dataset and Random Forest model (Breiman et al., 2018). See the Section 0.26.1 for more details.

```
library("DALEX2")
library("randomForest")
model <- randomForest(status ~ gender + age + hours + evaluation + salary, data = HR)
model
```

```
##
## Call:
##  randomForest(formula = status ~ gender + age + hours + evaluation +      salary, data = HR)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 27.42%
## Confusion matrix:
##           fired   ok promoted class.error
## fired      2268  395      192   0.2056042
## ok          526 1255      440   0.4349392
## promoted    205  394     2172   0.2161674
```

Model exploration with the `breakDown2` 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 `DALEX2` package (Biecek, 2018b).

```
explainer_rf <- explain(model,
                 data = HR,
                 y = HR$status)
```

2. Select an observation of interest.

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

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

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

```
##   fired  ok promoted
## 1 0.798 0.2    0.002
## attr(,"class")
## [1] "matrix" "votes"
```

3. Calculate Break Down decomposition

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

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

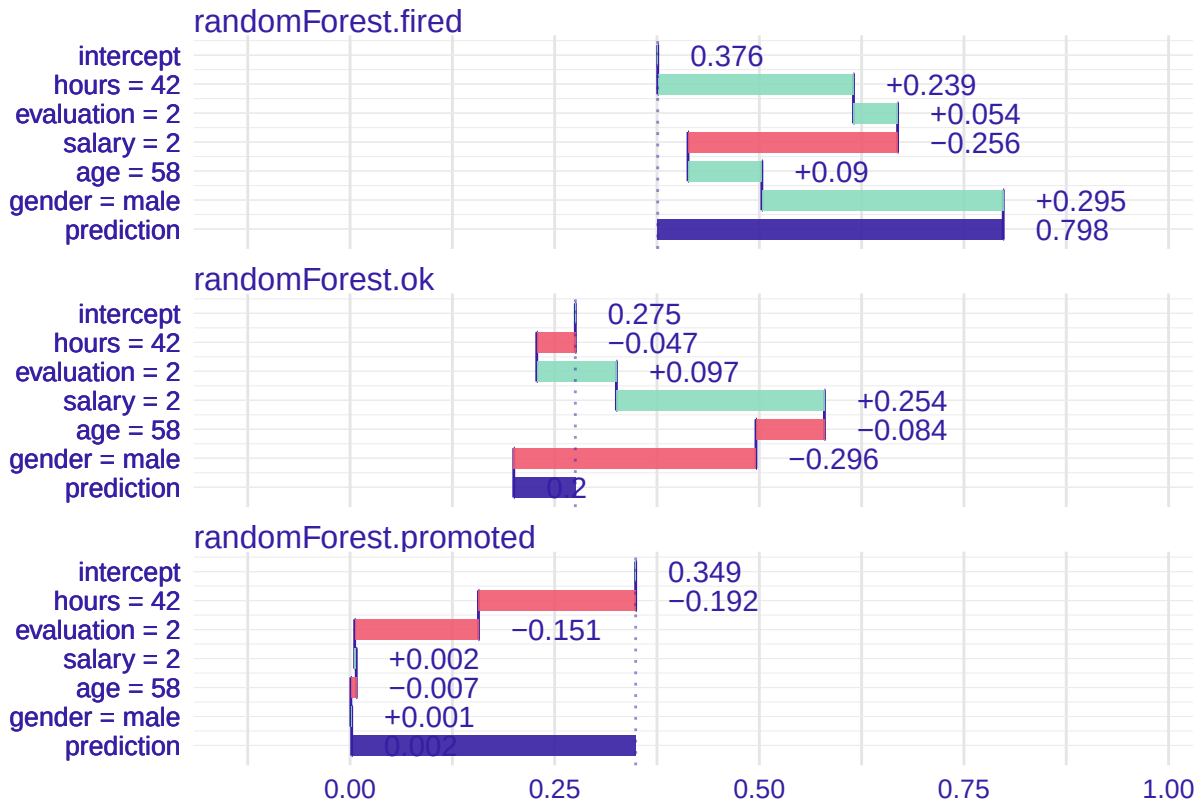
```
library("iBreakDown")
bd_rf <- local_attributions(explainer_rf,
                  new_observation,
                  keep_distributions = TRUE)

bd_rf
```

```
##                                     contribution
## randomForest.fired: intercept              0.376
## randomForest.fired: hours = 42             0.239
## randomForest.fired: evaluation = 2         0.054
## randomForest.fired: salary = 2            -0.256
## randomForest.fired: age = 58               0.090
## randomForest.fired: gender = male          0.295
## randomForest.fired: prediction             0.798
## randomForest.ok: intercept                 0.275
## randomForest.ok: hours = 42               -0.047
## randomForest.ok: evaluation = 2            0.097
## randomForest.ok: salary = 2                0.254
## randomForest.ok: age = 58                 -0.084
## randomForest.ok: gender = male            -0.296
## randomForest.ok: prediction                0.200
## randomForest.promoted: intercept           0.349
## randomForest.promoted: hours = 42         -0.192
## randomForest.promoted: evaluation = 2     -0.151
## randomForest.promoted: salary = 2          0.002
## randomForest.promoted: age = 58           -0.007
```

```
## randomForest.promoted: gender = male          0.001
## randomForest.promoted: prediction             0.002
```

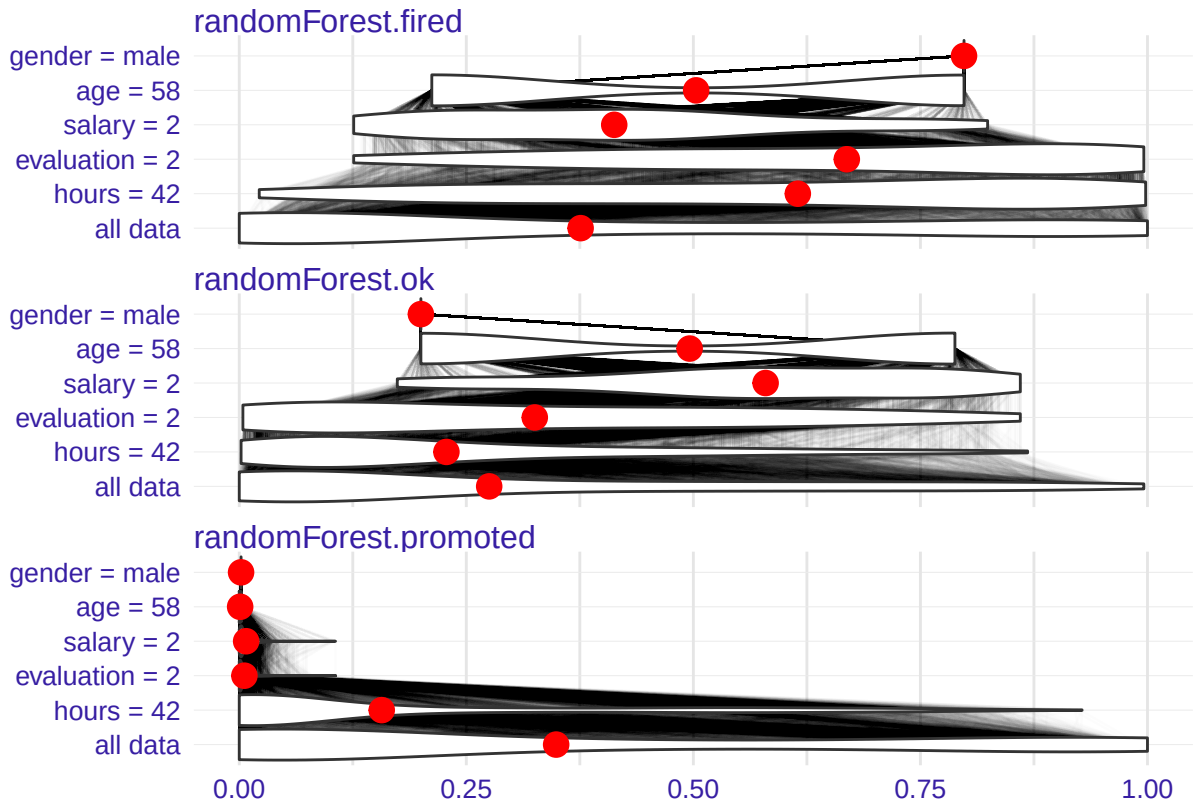
The generic `plot()` function creates a Break Down plots.

```
plot(bd_rf)
```



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

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



0.10 Variable attribution with interactions

In the Section 0.9 we presented model agnostic approach for additive decomposition of a model prediction for a single observation.

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

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

0.10.1 Intuition

The key idea here is to identify interactions between variables. This can be done in two steps.

1. First we determine variable contributions for each variable independently.
2. Second, we calculate effect for pair of variables. If this effect is different than the sum of consecutive variables then it may be an interaction.

TODO: easy example for interaction

0.10.2 Method

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

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

For a single variable

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

For pairs of variables

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

Note that this is equivalent to

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

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

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

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

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

0.10.3 Example: Hire or Fire?

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

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

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

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

0.10.4 Break Down Plots

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

0.10.5 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

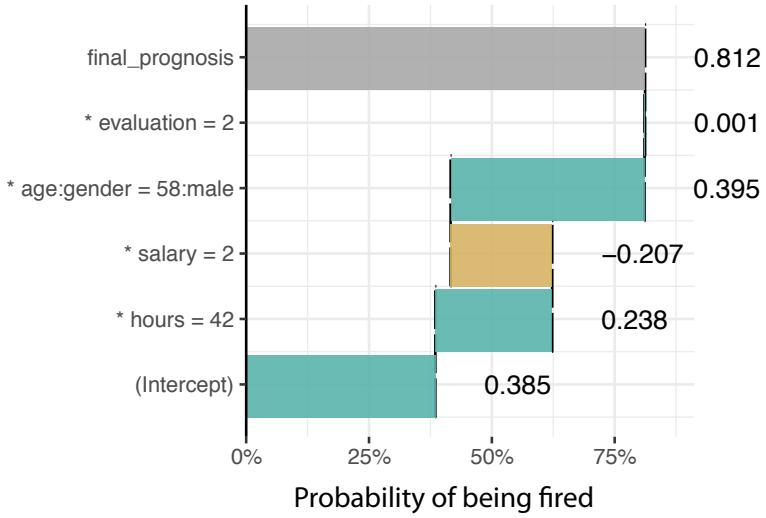


FIGURE 11 (fig:bdInter1) Break Down Plot for variable attrbution with interactions

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

Cons

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

0.10.6 Code snippets for R

The algorithm for Break Down for Interactions is also implemented in the `local_interactions` function from `breakDown2` package.

Model preparation

First a model needs to be trained.

```
library("DALEX2")
library("randomForest")
model <- randomForest(status ~ gender + age + hours + evaluation + salary, data = HR)
model
```

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

```
##
##         OOB estimate of  error rate: 27.22%
## Confusion matrix:
##           fired   ok promoted class.error
## fired     2283  377      195   0.2003503
## ok         535 1240      446   0.4416929
## promoted   205  378     2188   0.2103934
```

Model exploration with the `breakDown2` package is performed in three steps.

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

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

```
explainer_rf <- explain(model,
                  data = HR,
                  y = HR$status)
```

2. Select an observation of interest.

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

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

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

```
##   fired    ok promoted
## 1  0.78 0.218    0.002
## attr(,"class")
## [1] "matrix" "votes"
```

3. Calculate Break Down decomposition

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

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

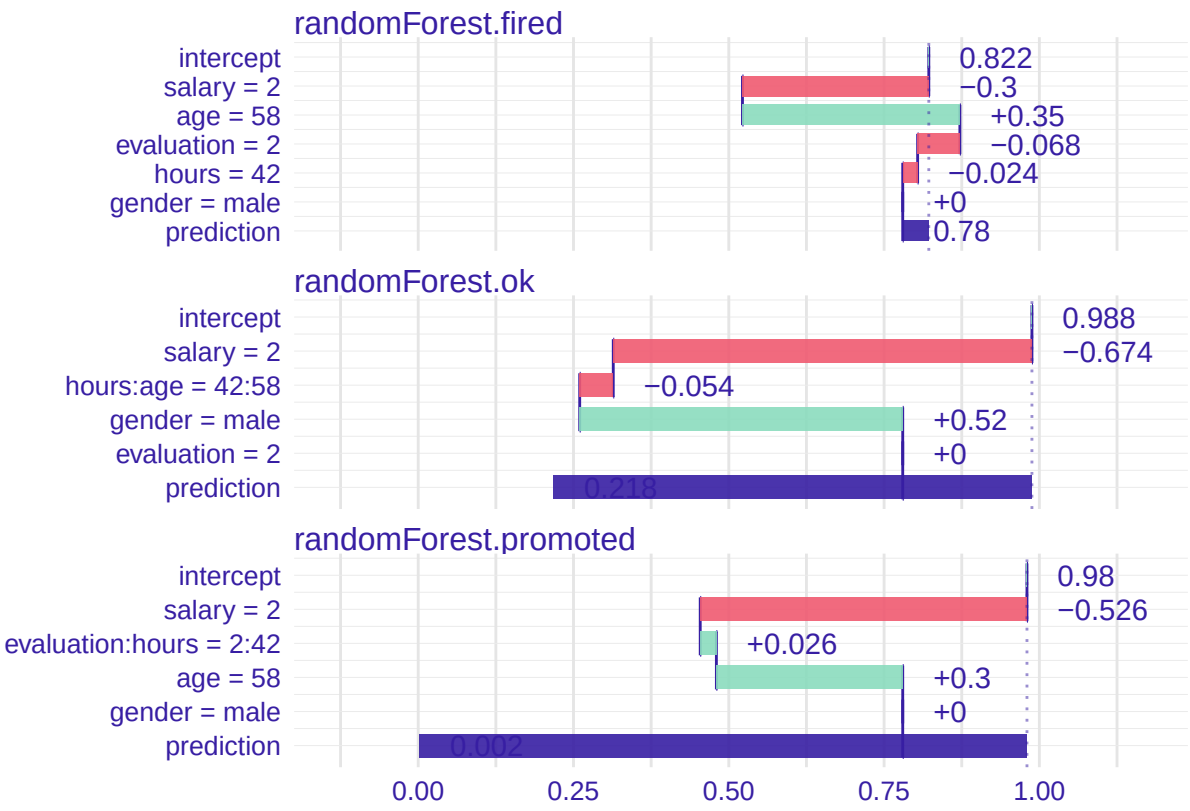
```
library("iBreakDown")
bd_rf <- local_interactions(explainer_rf,
                  new_observation)

bd_rf
```

```
##                                          contribution
```

```
## randomForest.fired: intercept                          0.822
## randomForest.fired: salary = 2                        -0.300
## randomForest.fired: age = 58                           0.350
## randomForest.fired: evaluation = 2                    -0.068
## randomForest.fired: hours = 42                        -0.024
## randomForest.fired: gender = male                      0.000
## randomForest.fired: prediction                         0.780
## randomForest.ok: intercept                             0.988
## randomForest.ok: salary = 2                           -0.674
## randomForest.ok: hours:age = 42:58                    -0.054
## randomForest.ok: gender = male                         0.520
## randomForest.ok: evaluation = 2                        0.000
## randomForest.ok: prediction                            0.218
## randomForest.promoted: intercept                       0.980
## randomForest.promoted: salary = 2                     -0.526
## randomForest.promoted: evaluation:hours = 2:42         0.026
## randomForest.promoted: age = 58                        0.300
## randomForest.promoted: gender = male                   0.000
## randomForest.promoted: prediction                      0.002
```

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

```
plot(bd_rf)
```



0.11 Average variable attributions

In the Section **??** we show the problem related to the ordering of variables. In the Section 0.9 we show an approach in which the ordering was determined based on single step assessment of variable importance.

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

This method is motivated with results in cooperative game theory and was first introduced in (Štrumbelj and Kononenko, 2014). Wide adoption of this method comes with a NIPS 2017 paper (Lundberg and Lee, 2017) and python library SHAP `https://github.com/slundberg/shap`. Authors of the SHAP method introduced also efficient algorithm for tree-based models, see (Lundberg et al., 2018).

0.11.1 Intuition

Since in sequential attribution effect depends on the ordering. Here the idea is to average across all possible orderings.

TODO: a nice example

0.11.2 Method

The name *Shapley Values* comes from the solution in cooperative game theory attributed to Lloyd Shapley. The original problem was to assess how important is each player to the overall cooperation, and what payoff can he or she reasonably expect from the coalition? (Shapley, 1953)

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

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

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

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

Time complexity of this method is $O(2^p)$ where p stands for the number of variables. Such complexity makes this method impractical for most cases. Fortunately it is enough to assess this value. (Štrumbelj and Kononenko, 2014) proposed to use sampling. (Lundberg et al., 2018) proposed fast implementations for tree based ensembles.

Properties

Shaply values have (as a single unique solution) following properties

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

$$f(x^*) = \sum_i v(f, x^*, i)$$

- Missingness, if simplified (add to notation) input is 0, then it's impact is also 0

$$x^*_i = 0 implies v(f, x^*, i) = 0$$

- Consistency, if a new model g is larger for model f then it's attributions are larger than attributions for f.

0.11.3 Example: Hire or Fire?

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.
- (Lundberg and Lee, 2017) shows that this method unifies different approaches to additive features attribution, like DeepLIFT, Layer-Wise Relevance Propagation, LIME.
- There is efficient implementation available for Python.
- (Lundberg and Lee, 2017) shows more desired properties of this method, like symmetry or additivity.

Cons

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

Note that fully additive model solutions presented in sections **??**, 0.9 and 0.11 lead to same variable contributions.

0.11.5 Code snippets for R

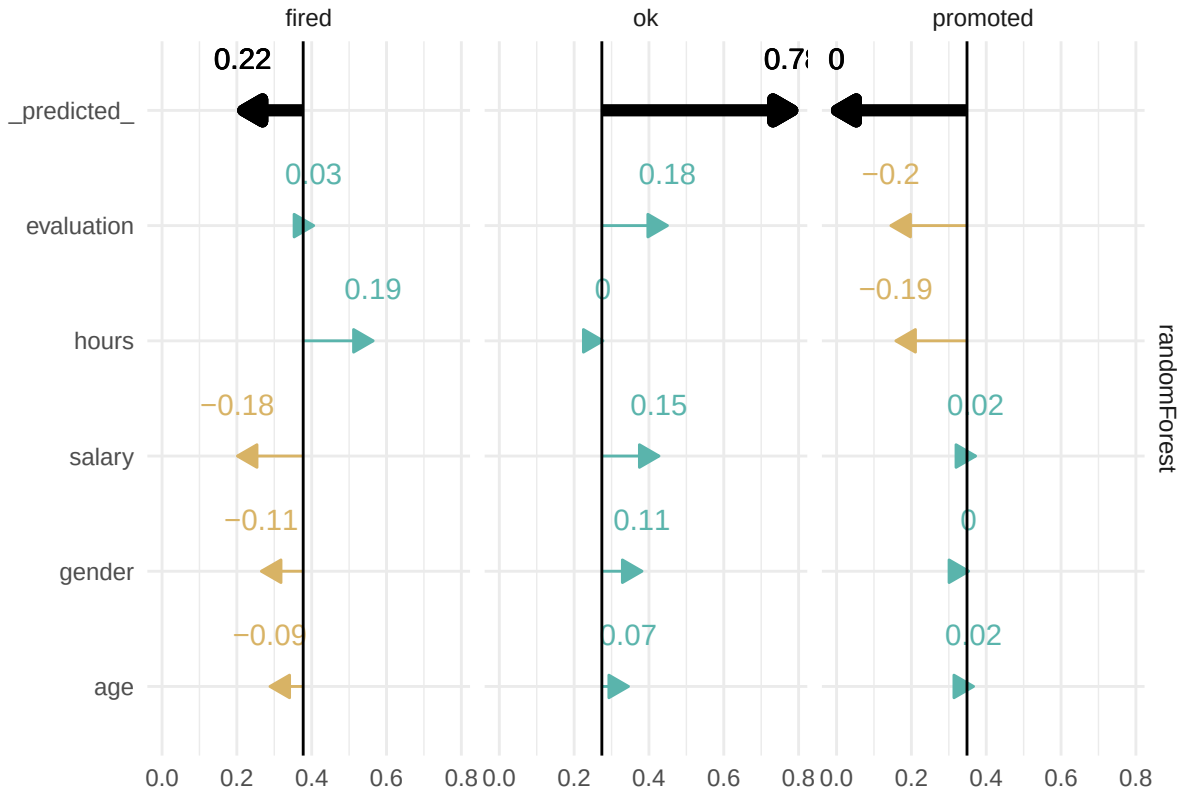
In this section we will present an example based on the HR dataset and Random Forest model (Breiman et al., 2018). See the Section 0.26.1 for more details.

```
library("DALEX")
library("randomForest")
set.seed(123)
model_rf <- randomForest(status ~ gender + age + hours + evaluation + salary, data = HR)
```

First, we use a shapper package - a wrapper over SHAP python package.

```
library("shapper")
Y_train <- HR$status
x_train <- HR[ , -6]
x_train$gender <- as.numeric(x_train$gender)
model_rfs <- randomForest(x = x_train, y = Y_train)

p_fun <- function(x, data){
  predict(x, newdata = data, type = "prob")
}

new_observation <- data.frame(gender = 1,
                      age = 57.7,
                      hours = 42.3,
                      evaluation = 2,
                      salary = 2)

x <- individual_variable_effect(x = model_rfs, data = x_train, predict_function = p_fun,
                                new_observation = new_observation)

#plot(x)
library("ggplot2")

x$`_vname_` <- reorder(x$`_vname_`, x$`_attribution_`, function(z) -sum(abs(z)))
levels(x$`_vname_`) <- paste(sapply(1:6, substr, x="        ", start=1), levels(x$`_vname_`))

ggplot(x, aes(x=`_vname_`, xend=`_vname_`,
              yend = `_yhat_mean_`, y = `_yhat_mean_` + `_attribution_`,
              color=`_sign_`)) +
  geom_segment(arrow = arrow(length=unit(0.30,"cm"), ends="first", type = "closed")) +
  geom_text(aes(label = round(`_attribution_`, 2)), nudge_x = 0.45) +
  geom_segment(aes(x = "_predicted_",xend = "_predicted_",
                   y = `_yhat_`, yend = `_yhat_mean_`), size = 2, color="black",
               arrow = arrow(length=unit(0.30,"cm"), ends="first", type = "closed")) +
  geom_text(aes(x = "_predicted_",
```

```
              y = `_yhat_`, label = round(`_yhat_`, 2)), nudge_x = 0.45, color="black") +
  geom_hline(aes(yintercept = `_yhat_mean_`)) +
  facet_grid(`_label_`~`_ylevel_`) +
  scale_color_manual(values =  c(`-` = "#d8b365", `0` = "#f5f5f5", `+` = "#5ab4ac",
                                 X = "darkgrey")) +
  coord_flip() + theme_minimal() + theme(legend.position="none") + xlab("") + ylab("")
```



Here we use the `iml` package, see more examples in (Molnar et al., 2018b).

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

Explanations for a new observation.

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

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

And the plot with Shapley attributions.

```
plot(shapley)
```

See more examples for `iml` package in the (Molnar, 2018b) book.

0.12 Local approximations with white-box model

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

Variable attribution methods, that were presented in the Section 0.9 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 0.4 we introduced Ceteris Paribus tool for such what-if analysis. But the limitation of ceteris Paribus pltos is that they explore changes along single dimension or pairs of dimensions.

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

The most known method from this class if LIME (Local Interpretable Model-Agnostic Explanations), introduced in the paper *Why Should I Trust You?: Explaining the Predictions of Any Classifier* (Ribeiro et al., 2016). This methods and it's clones are now implemented in various R and python packages, see for example (Pedersen and Benesty, 2018), (Staniak and Biecek, 2018) or (Molnar, 2018a).

0.12.1 Intuition

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

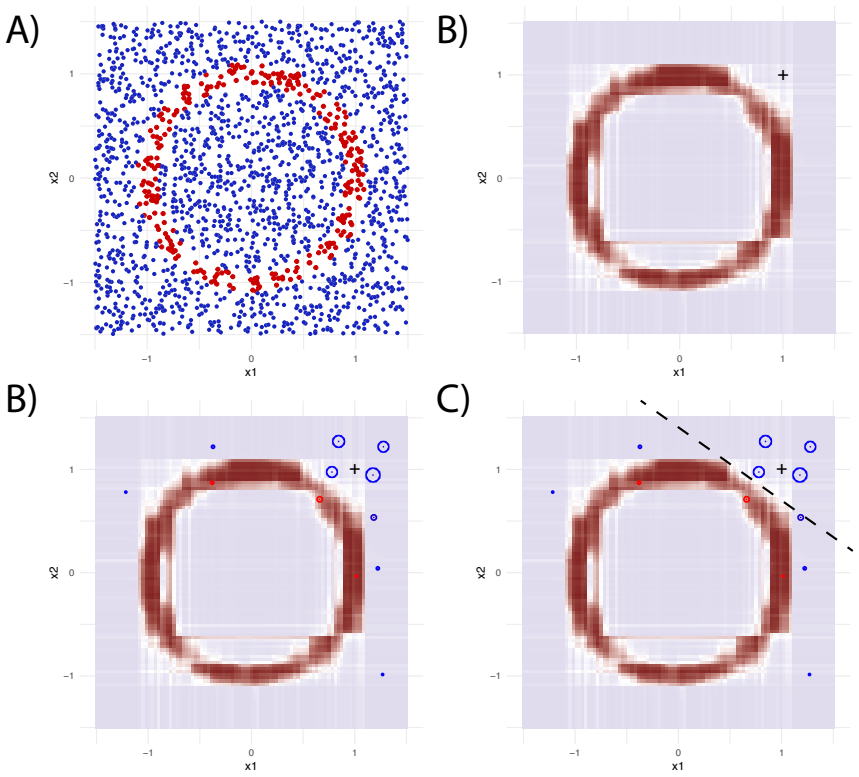


FIGURE 12 (fig:LIME1) A schematic idea behind local model approximations. Panel A shows training data, colors correspond to classess. Panel B showhs results fom the Random Forest model, whis is where the algorithm starts. Panel C shows new data sampled around the point of interest. Their color correspond to model response. Panel D shows fitted linear model that approximated the random forest model around point of interest

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.

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

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

The algorithm is composed from three steps:

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

Identification of interpretable data representations

For image data, single pixel is not an interpretable feature. In this step the input space of the model is transformed to input space that is easier to understand for human. The image

may be decomposed into parts and represented as presence/absence of some part of an image.

Local sampling around the point of interest

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

Fitting a white box model in this neighbouhood

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

0.12.3 Example: Hire or Fire?

0.12.4 Pros and cons

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

Pros

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

Cons

- For continuous variables and tabular data it is not that easy to find interpretable representations
- The black-box model approximated the data and the white box model approximates the black box model. We do not have control over the quality of local fit of the white box model, thus the surrogate model may be misleading.
- Due to the *curse of dimensionality*, for high dimensional space points are sparse.

0.12.5 Code snippets for R

In this section we present example application of `lime` (Pedersen and Benesty, 2018) and `live` (Staniak and Biecek, 2018) packages. Note that this method is also implemented in `iml` (Molnar, 2018a) and other packages. These pacakages differ in some details and also results in different explanations.

Model preparation

In this section we will present examples based on the `HR` dataset. See the Section 0.26.1 for more details.

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

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

The problem here is to predict average price for square meter for an apartment. Let's build a random forest model with `randomForest` package (Breiman et al., 2018).

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

```
##
## Call:
##  randomForest(formula = status ~ gender + age + hours + evaluation +      salary, data = HR)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 27.49%
## Confusion matrix:
##          fired   ok promoted class.error
## fired     2270  384      201   0.2049037
## ok         528 1244      449   0.4398919
## promoted   206  389     2176   0.2147239
```

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

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

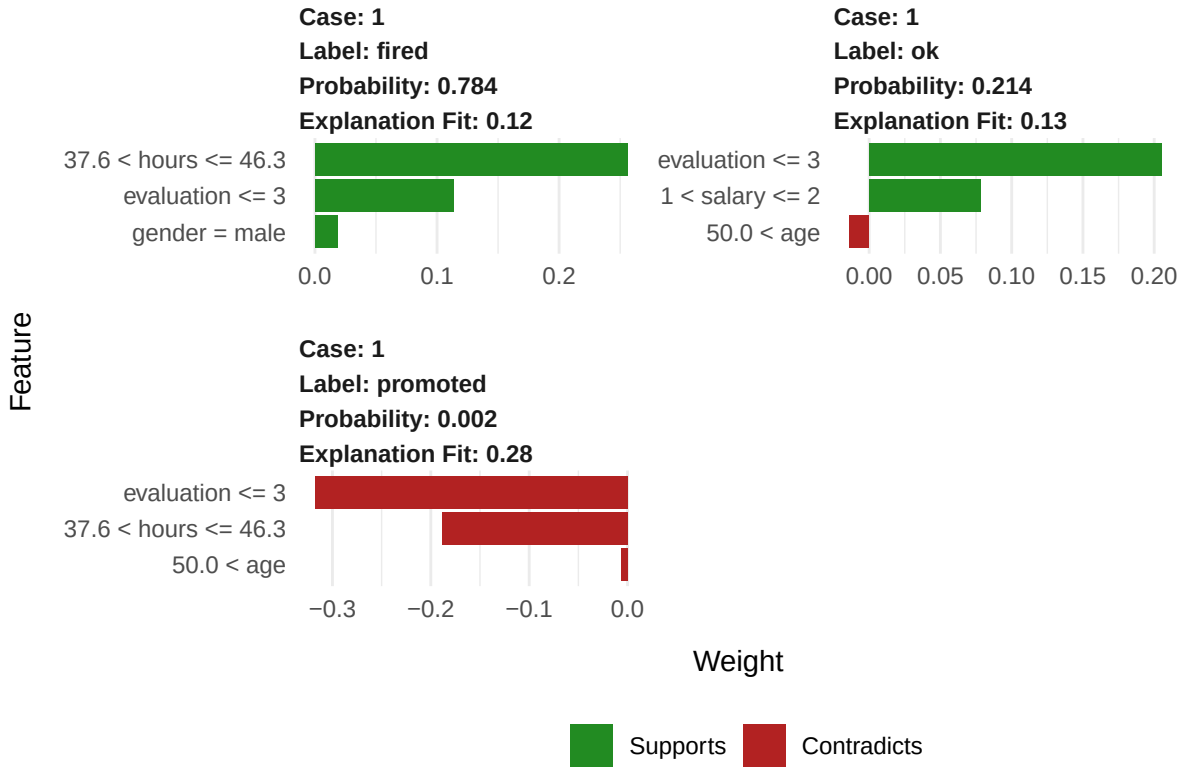
```
##   fired    ok promoted
## 1 0.784 0.214    0.002
## attr(,"class")
## [1] "matrix" "votes"
```

0.12.5.1 The lime pacakge

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

```
##       model_type case    label label_prob  model_r2 model_intercept
## 1 classification    1    fired      0.784 0.1229568       0.2549494
## 2 classification    1    fired      0.784 0.1229568       0.2549494
## 3 classification    1    fired      0.784 0.1229568       0.2549494
## 4 classification    1       ok      0.214 0.1301129       0.1884677
## 5 classification    1       ok      0.214 0.1301129       0.1884677
## 6 classification    1       ok      0.214 0.1301129       0.1884677
## 7 classification    1 promoted      0.002 0.2803287       0.5167543
## 8 classification    1 promoted      0.002 0.2803287       0.5167543
## 9 classification    1 promoted      0.002 0.2803287       0.5167543
##   model_prediction    feature feature_value feature_weight
## 1       0.64320321     gender           2.0    0.018638406
## 2       0.64320321      hours          42.3    0.256064763
## 3       0.64320321 evaluation           2.0    0.113550640
## 4       0.45858087        age          57.7   -0.013851520
## 5       0.45858087 evaluation           2.0    0.205469213
## 6       0.45858087     salary           2.0    0.078495471
## 7       0.00344001        age          57.7   -0.006718524
## 8       0.00344001 evaluation           2.0   -0.317770612
## 9       0.00344001      hours          42.3   -0.188825120
##           feature_desc                      data          prediction
## 1        gender = male 2.0, 57.7, 42.3, 2.0, 2.0 0.784, 0.214, 0.002
## 2 37.6 < hours <= 46.3 2.0, 57.7, 42.3, 2.0, 2.0 0.784, 0.214, 0.002
## 3      evaluation <= 3 2.0, 57.7, 42.3, 2.0, 2.0 0.784, 0.214, 0.002
## 4           50.0 < age 2.0, 57.7, 42.3, 2.0, 2.0 0.784, 0.214, 0.002
## 5      evaluation <= 3 2.0, 57.7, 42.3, 2.0, 2.0 0.784, 0.214, 0.002
## 6      1 < salary <= 2 2.0, 57.7, 42.3, 2.0, 2.0 0.784, 0.214, 0.002
## 7           50.0 < age 2.0, 57.7, 42.3, 2.0, 2.0 0.784, 0.214, 0.002
## 8      evaluation <= 3 2.0, 57.7, 42.3, 2.0, 2.0 0.784, 0.214, 0.002
## 9 37.6 < hours <= 46.3 2.0, 57.7, 42.3, 2.0, 2.0 0.784, 0.214, 0.002
```

```
plot_features(explanations)
```



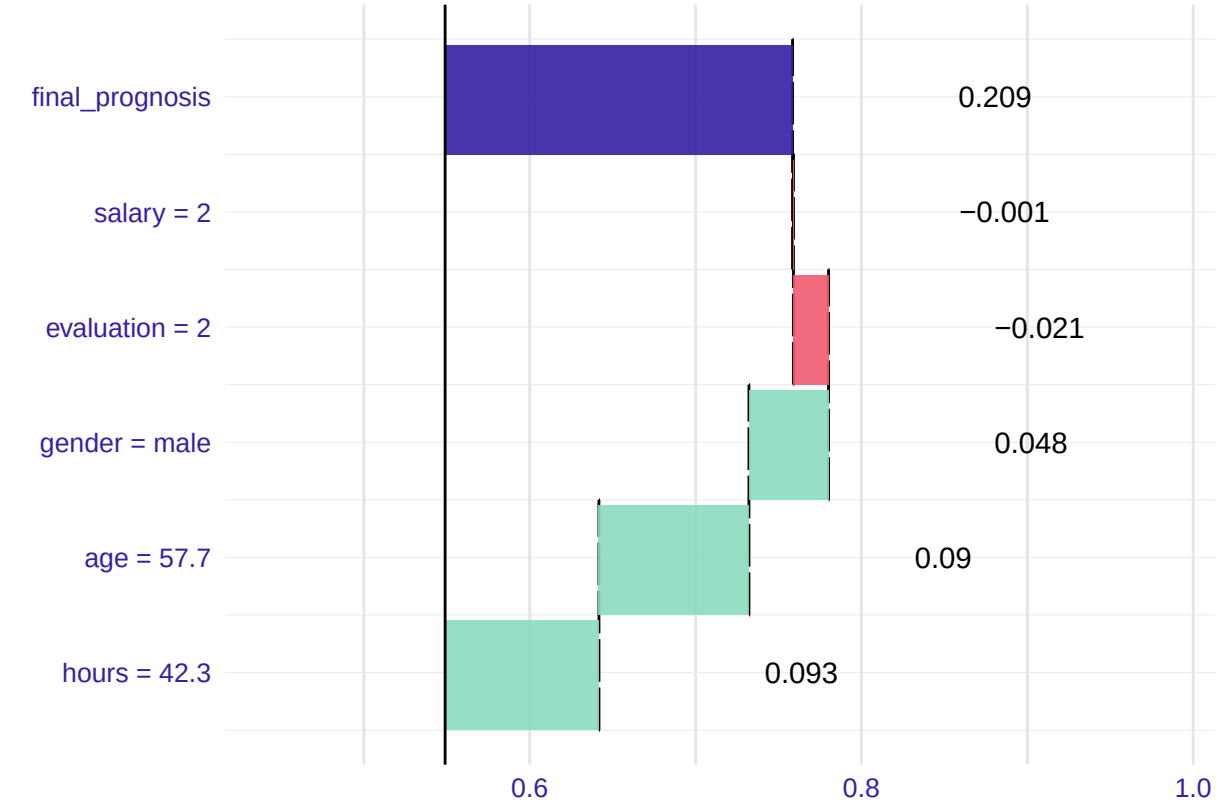
0.12.5.2 The live package

```
library("live")

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

local_model <- local_approximation(explainer_rf_fired, new_observation,
                    target_variable_name = "status", n_new_obs = 500)

local_model
```

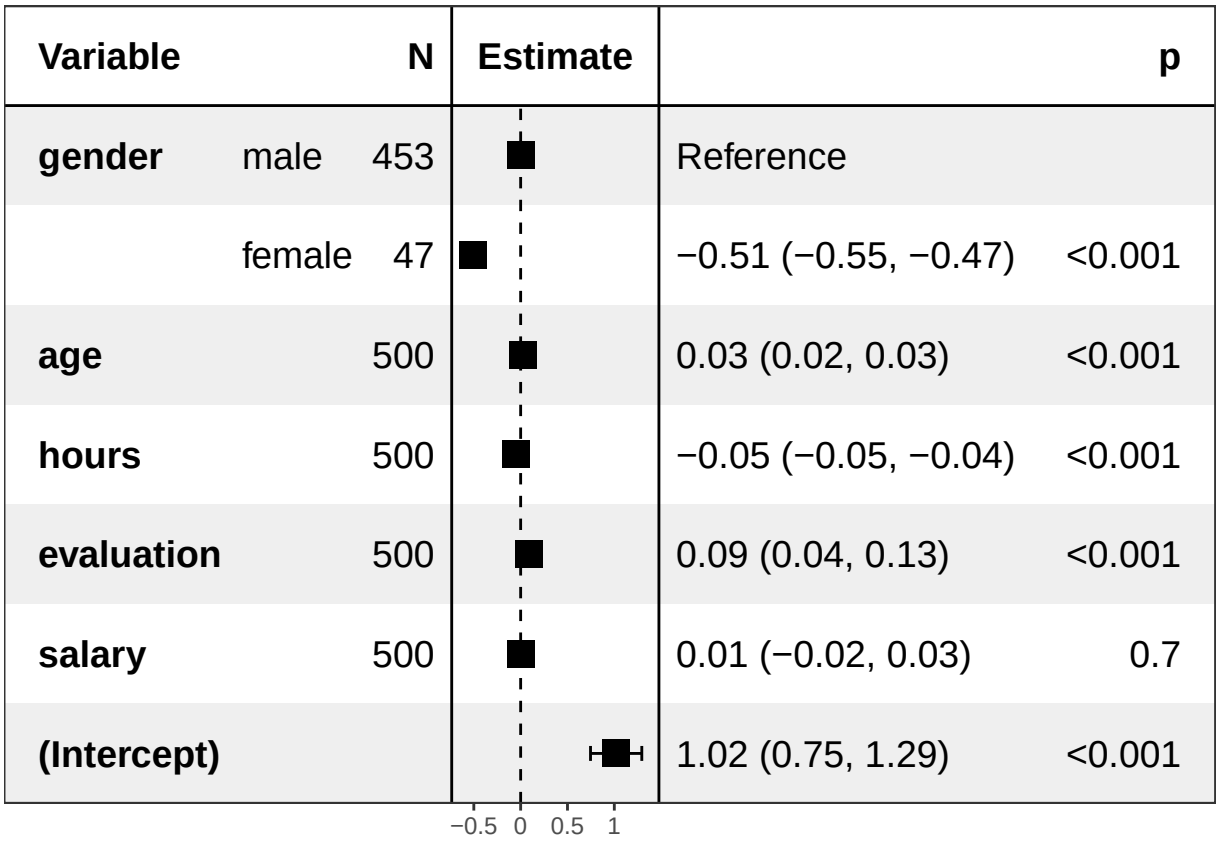
```
## Dataset:
##  Observations:  500
##  Variables:  6
##  Response variable:  status
## Explanation model:
##  Name:  regr.lm
##  Variable selection wasn't performed
##  Weights present in the explanation model
```

```
##  R-squared: 0.7596
```

```
plot(local_model)
```




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



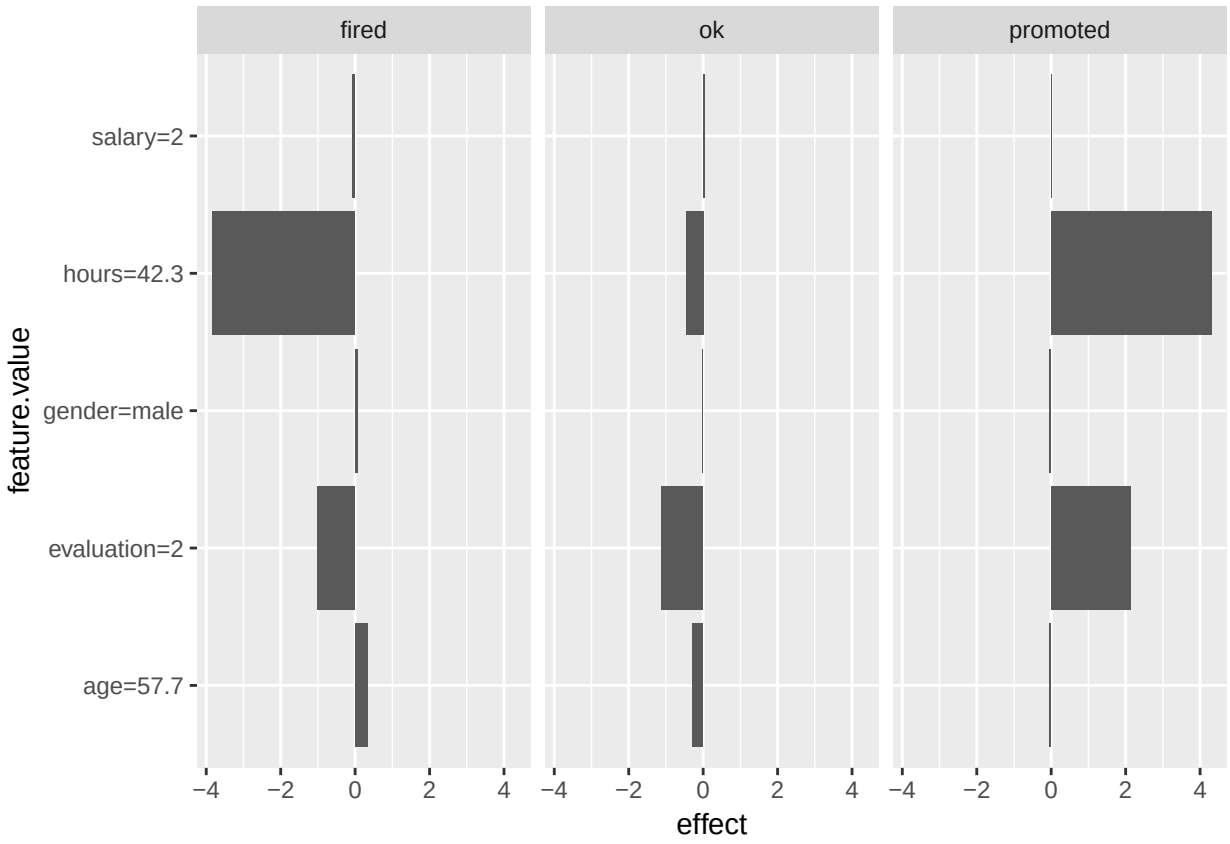
0.12.5.3 The iml package

```
library("iml")

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

```
## Interpretation method:  LocalModel
##
##
## Analysed predictor:
## Prediction task: unknown
##
##
## Analysed data:
## Sampling from data.frame with 7847 rows and 5 columns.
##
## Head of results:
##           beta x.recoded      effect x.original     feature feature.value
## 1  0.062176640       1.0  0.06217664       male gender=male   gender=male
## 2  0.005789257      57.7  0.33404015       57.7          age      age=57.7
```

```
## 3 -0.090772953      42.3 -3.83969593       42.3       hours    hours=42.3
## 4 -0.506977522       2.0 -1.01395504          2  evaluation  evaluation=2
## 5 -0.031598533       2.0 -0.06319707          2      salary      salary=2
## 6 -0.015944921       1.0 -0.01594492       male gender=male   gender=male
##   .class
## 1  fired
## 2  fired
## 3  fired
## 4  fired
## 5  fired
## 6     ok
```

```
plot(white_box)
```



0.13 Comparision of prediction level explainers

TODO

compare pros and cons of different techniques

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'? (Edwards and Veale, 2018)

TODO: Sparse model approximation / variable selection / feature ranking

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 0.1.2 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 (Foster, 2017) for gradient boosting or (Paluszynska and Biecek, 2017a) 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 (Fisher et al., 2018). 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 (Breiman et al., 2018). 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_c
## 1 (30,50]
## 2 (10,18]
## 3 (10,18]
## 4 (30,50]
## 5 (10,18]
## 6 (18,30]
```

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

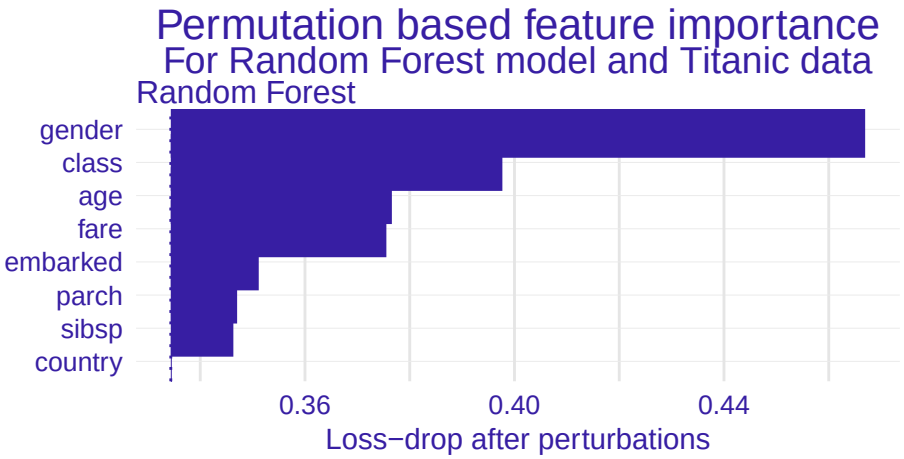


FIGURE 13 Feature importance. Each interval presents the difference between original model performance (left end) and the performance on a dataset with a single feature perturbed

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. 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.

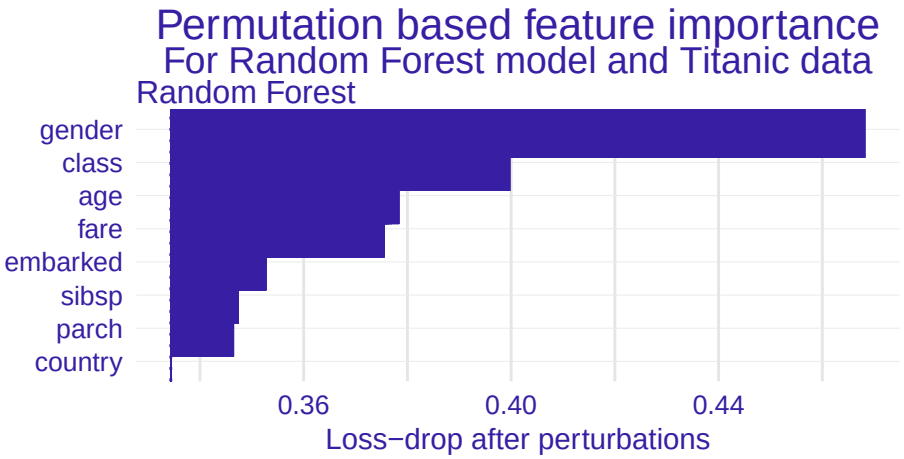


FIGURE 14 Feature importance for 10 replication of feature importance assessment

```
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
)
```

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

Let's calculate feature importance

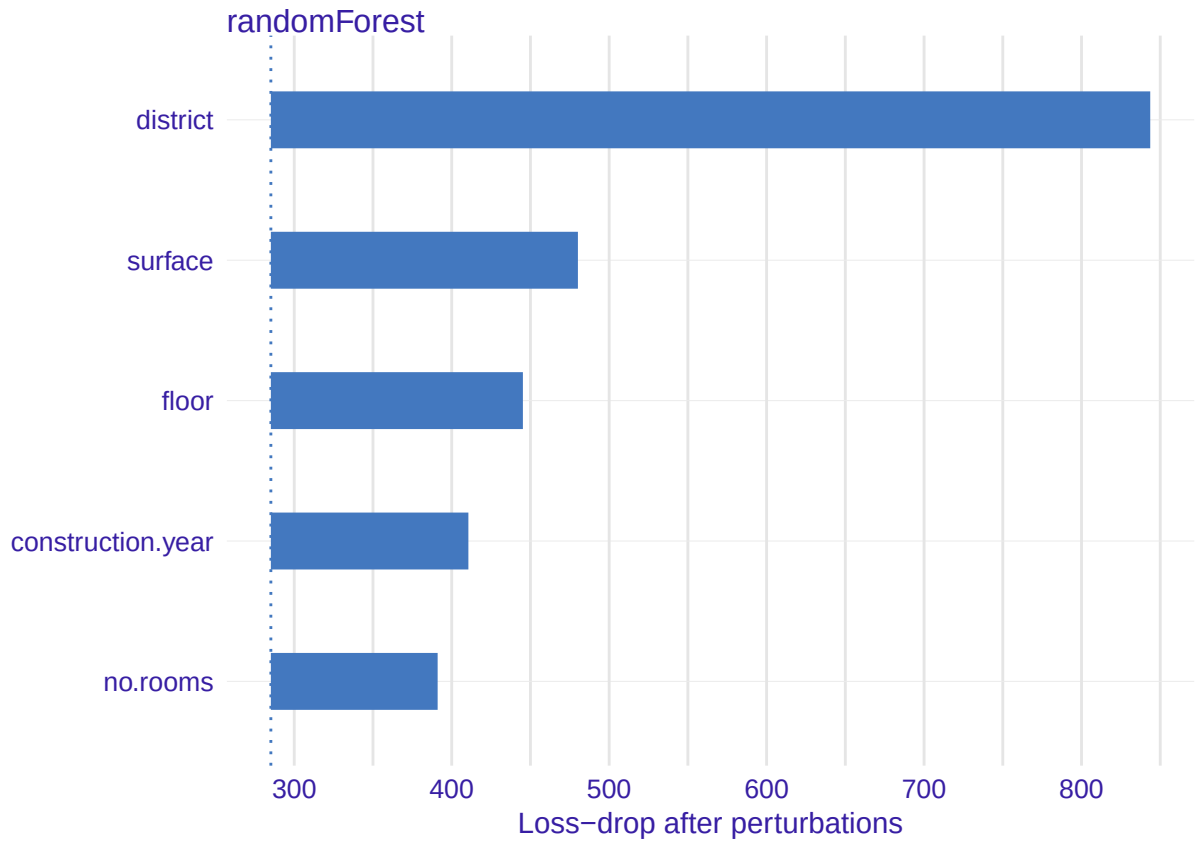
FIGURE 15 Feature importance for random forest, gradient boosting and logistic regression models

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

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 (Breiman et al., 2018) (elastic but biased), Support Vector Machines model (Meyer et al., 2017) (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_     293.2729 randomForest
## 2          no.rooms     389.4526 randomForest
## 3 construction.year     416.1154 randomForest
## 4             floor     453.9195 randomForest
## 5           surface     480.4062 randomForest
## 6          district     867.7050 randomForest
## 7        _baseline_    1116.2616 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
```

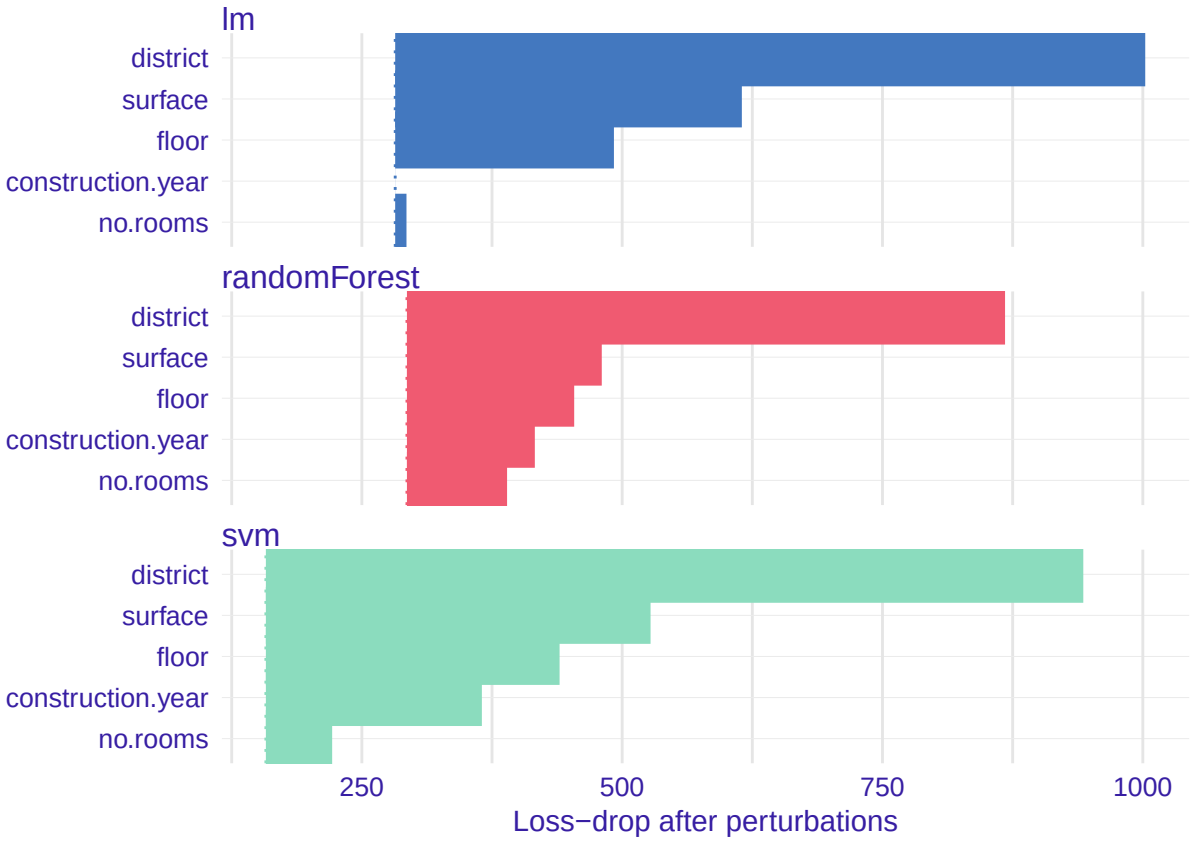
```
##            variable dropout_loss label
## 1      _full_model_     157.7938   svm
## 2          no.rooms     221.4595   svm
## 3 construction.year     365.2600   svm
## 4             floor     439.8724   svm
## 5           surface     527.2598   svm
## 6          district     942.8512   svm
## 7        _baseline_    1203.7571   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 (Gosiewska et al., 2019). 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 (Friedman, 2000). 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.

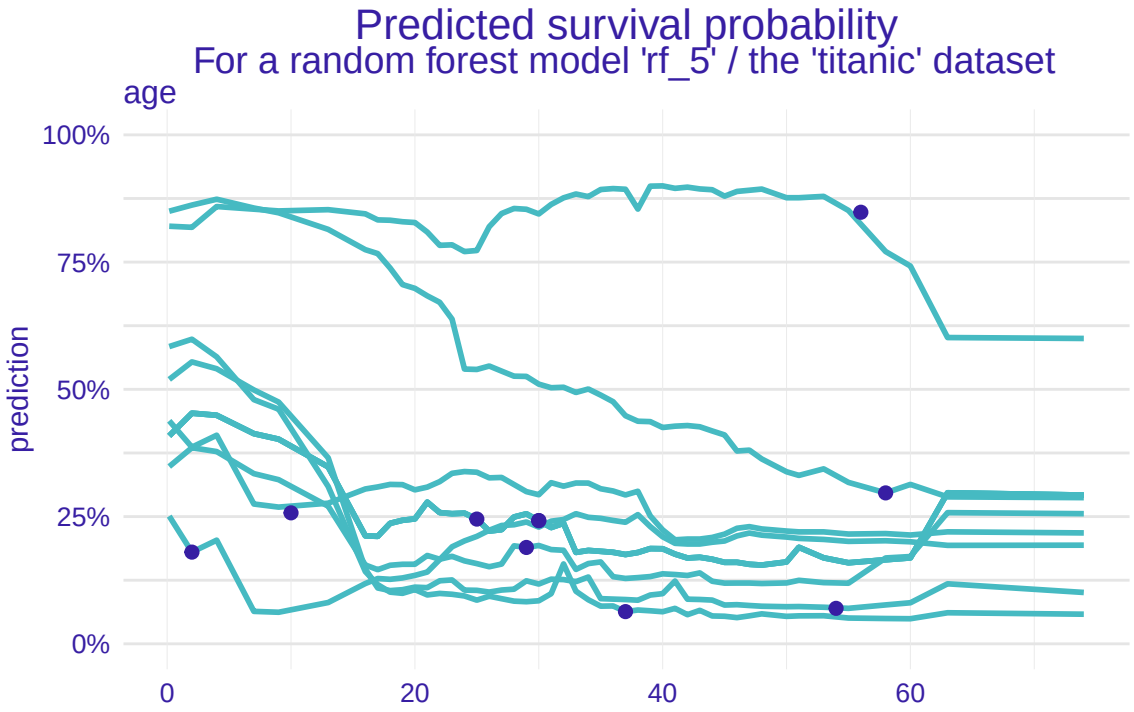


FIGURE 16 (#fig:pdp_part_1A)Ceteris Paribus profiles for 10 passangers and the random forest model

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 0.4.

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.

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

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 (Greenwell, 2017b), `ALEPlots` package (Apley, 2018b) or `iml` (Molnar et al., 2018c) 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")
```

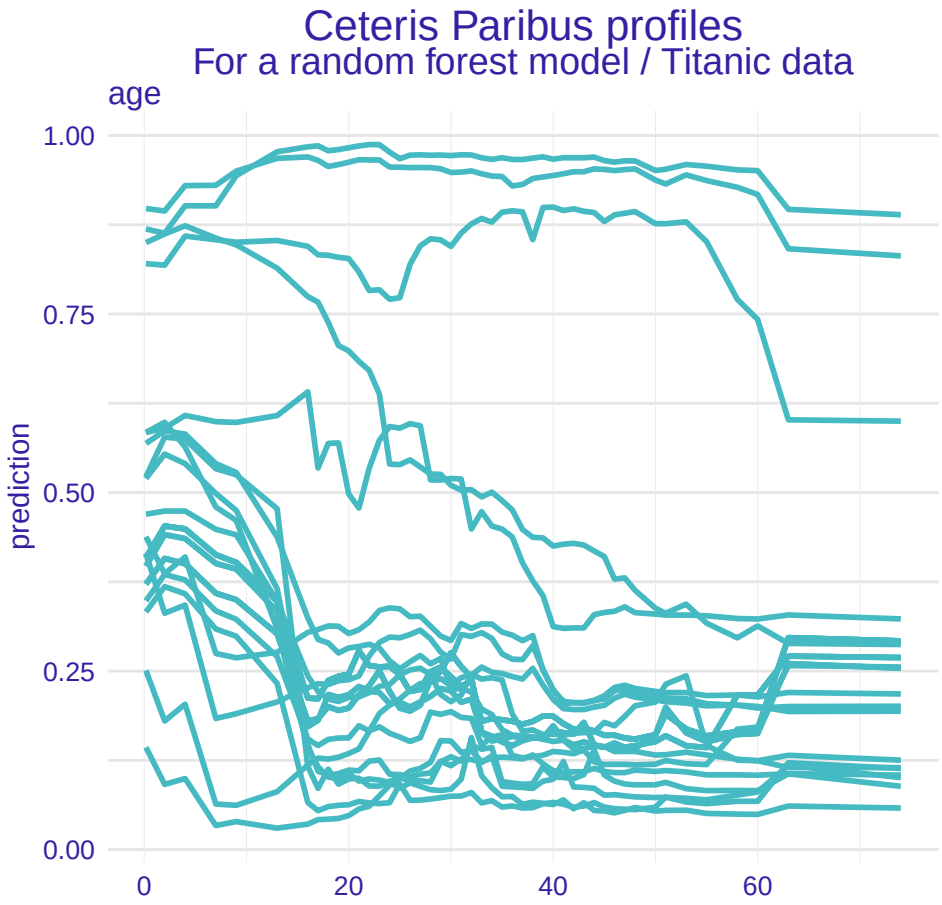
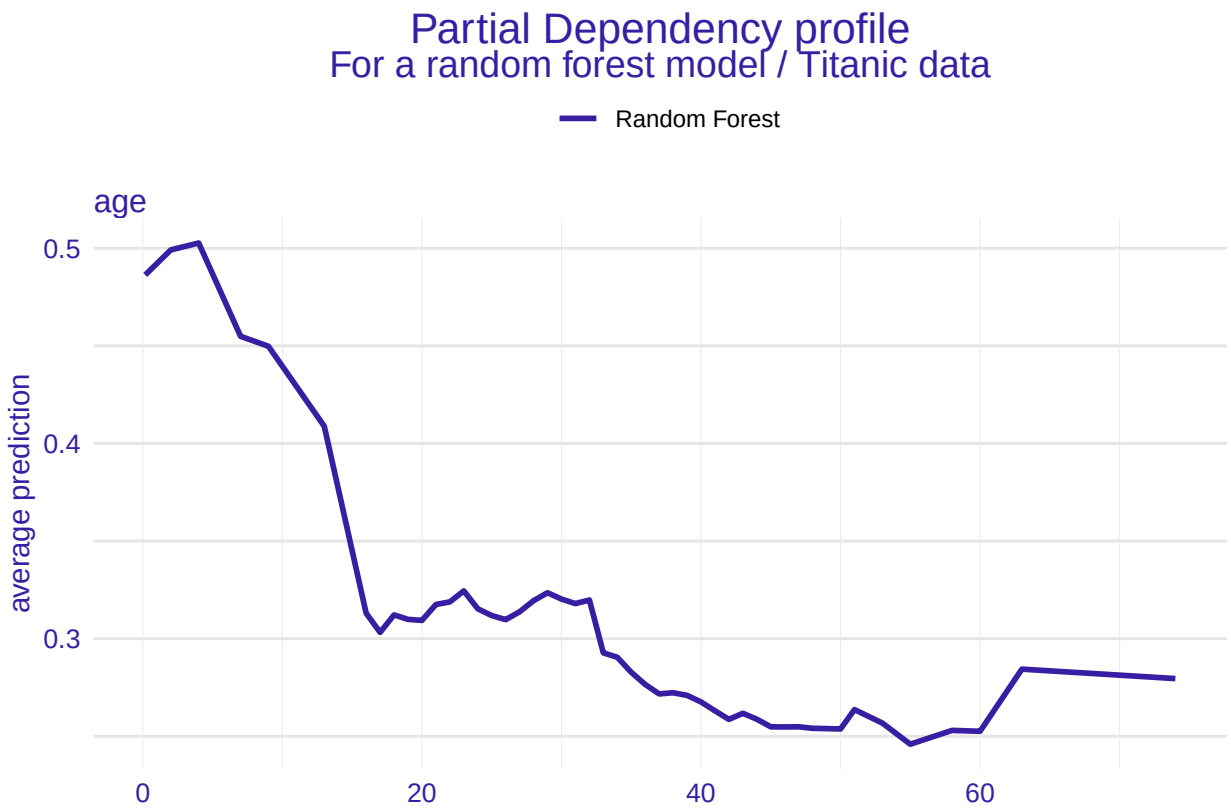


FIGURE 17 (#fig:pdp_part_1)Ceteris Paribus profiles for 100 observations, the age variable and the random forest model



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

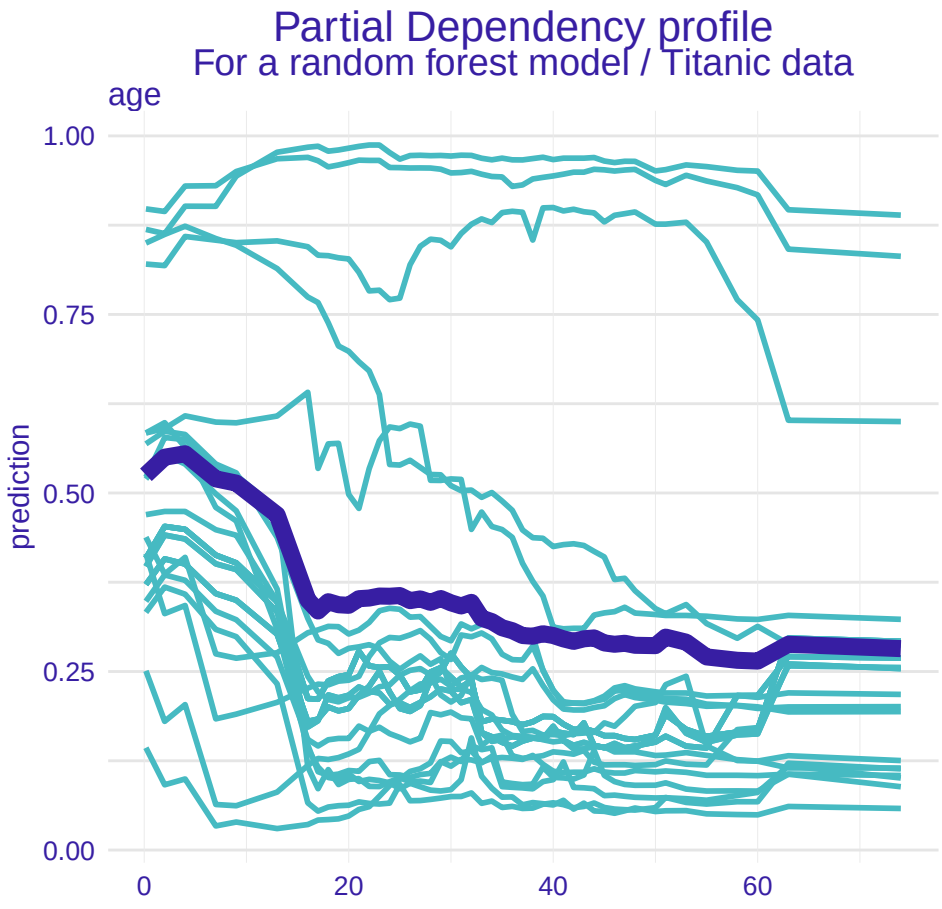


FIGURE 18 (#fig:pdp_part_2)Partial Dependency profile as an average for 100 observations

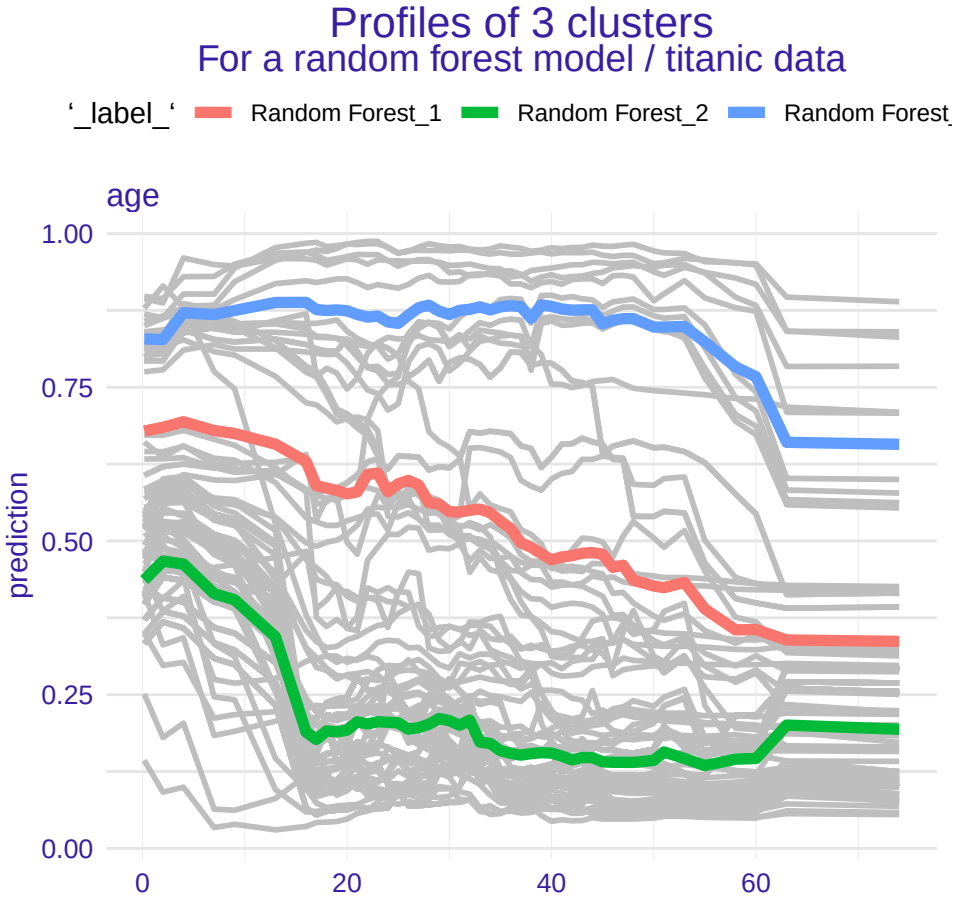


FIGURE 19 (#fig:pdp_part_4)Cluster profiles for 3 clusters over 100 Ceteris Paribus profiles

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.

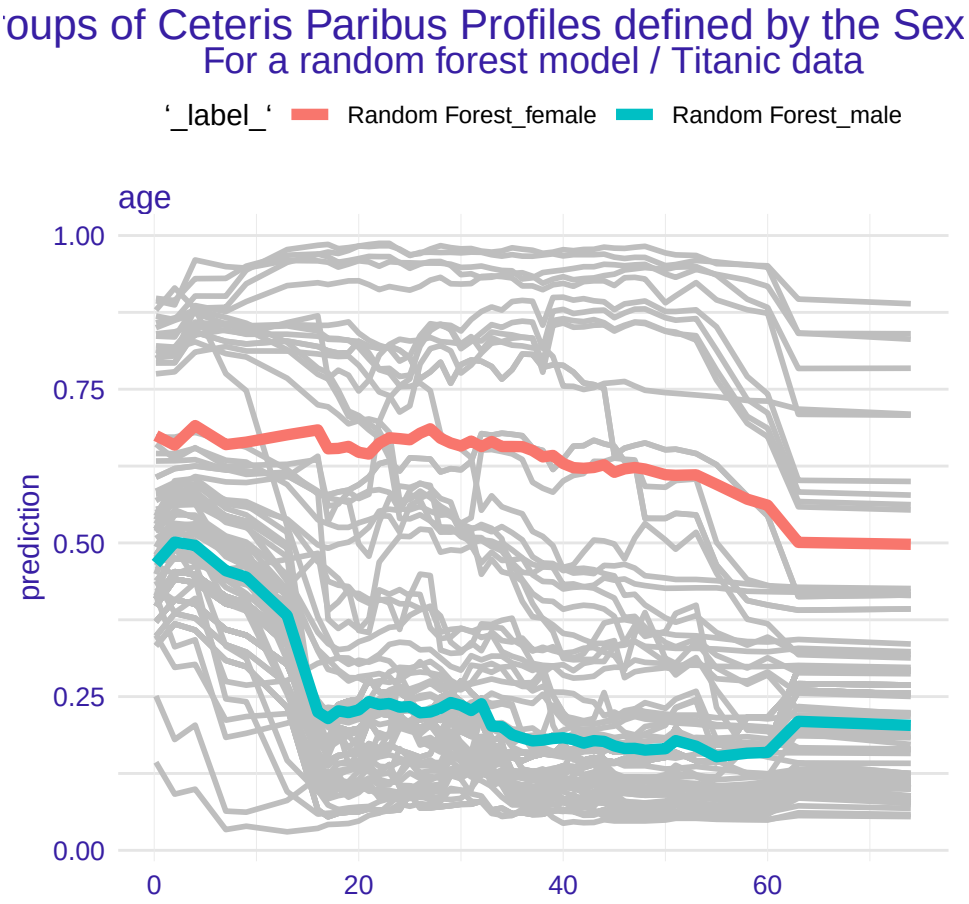


FIGURE 20 (#fig:pdp_part_5)Grouped profiles with respect to the gender variable

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 (Apley, 2018b) 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}.

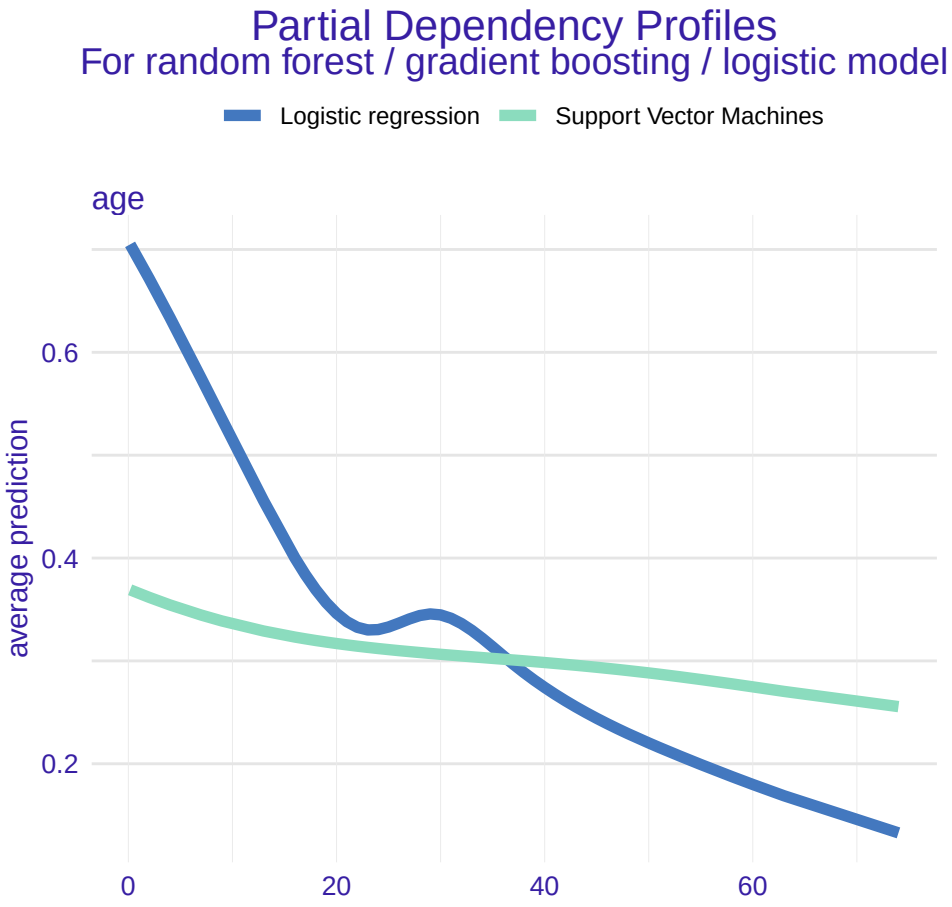


FIGURE 21 (#fig:pdp_part_7)Comparison on three predictive models with different structures.

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

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

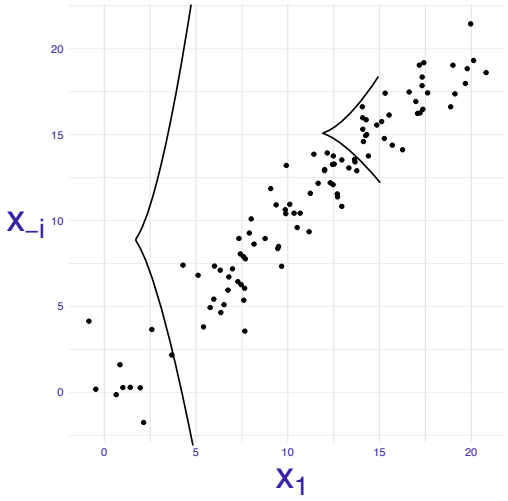


FIGURE 22 (fig:accumulatedCor)

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

As it is justified in (Apley, 2018b), 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 (Apley, 2018b) paper.

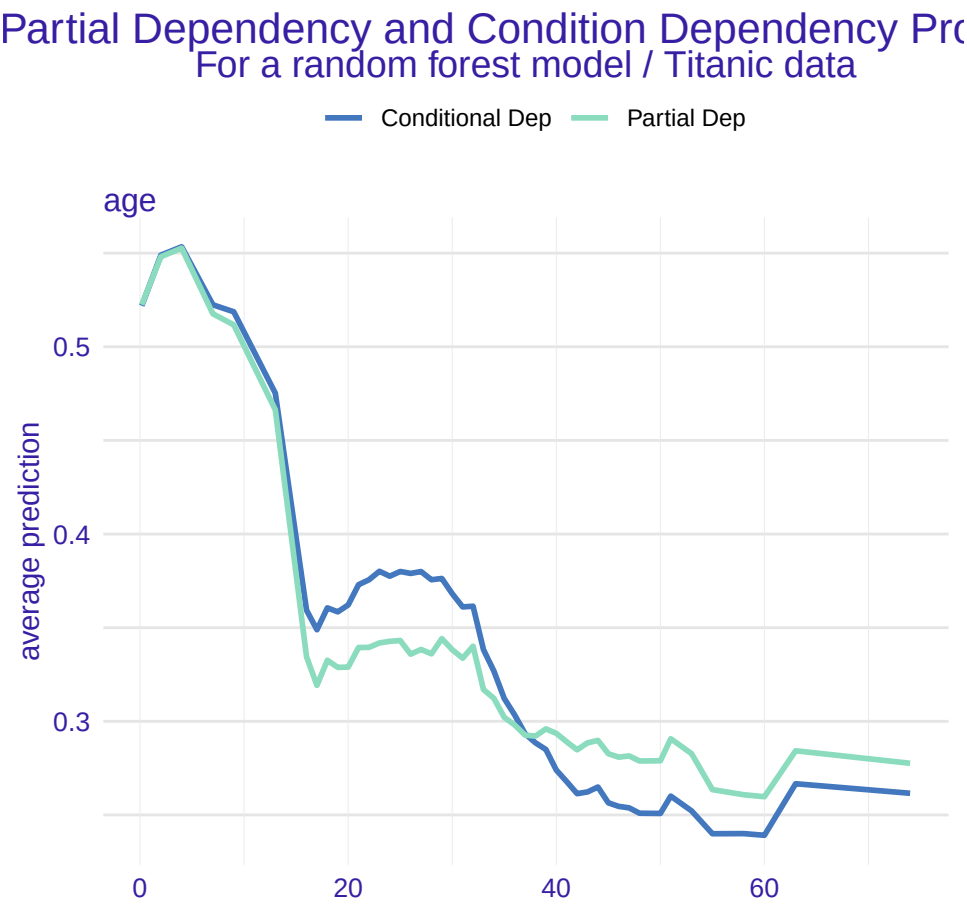


FIGURE 23 (#fig:mp_part_1)Conditional Dependency Profile for 100 observations

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

(Demšar and Bosnić, 2018)

(Greenwell, 2017c) (Puri et al., 2017)

(Sitko et al., 2018)

(Strobl et al., 2007) (Strobl et al., 2008) - variable importance

(Fisher et al., 2018)

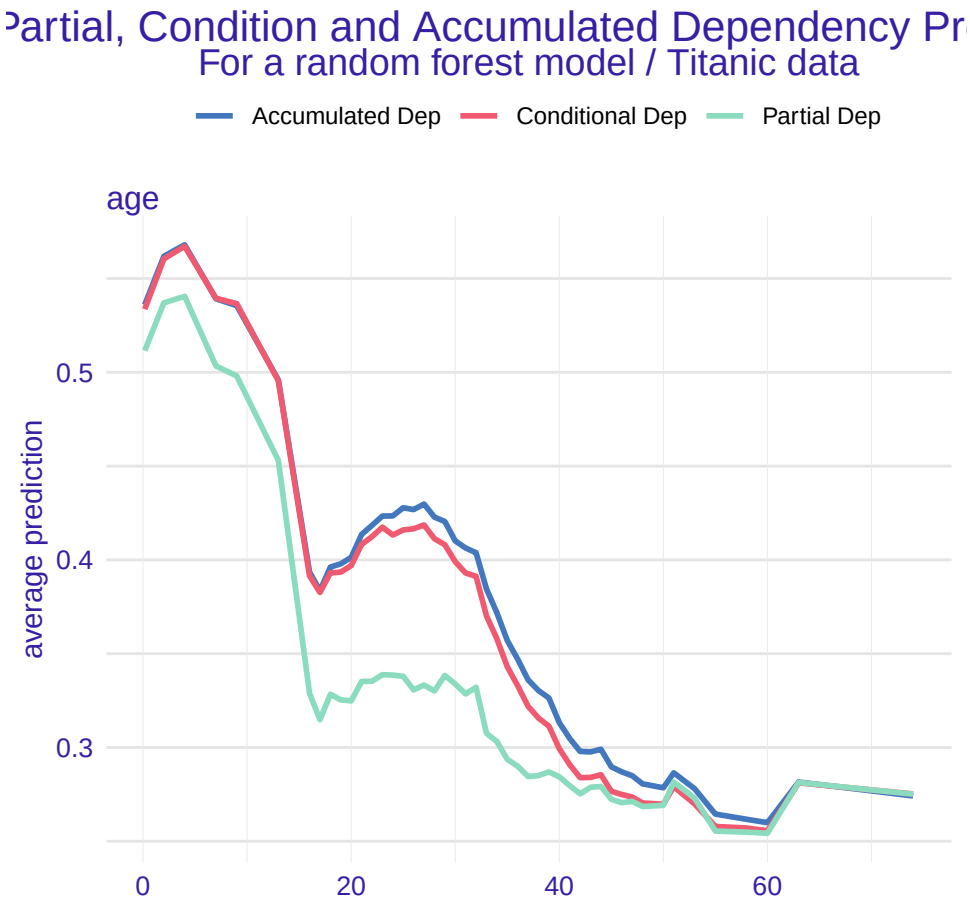


FIGURE 24 (#fig:ale_part_1)Accumulated Local Effects for 100 observations

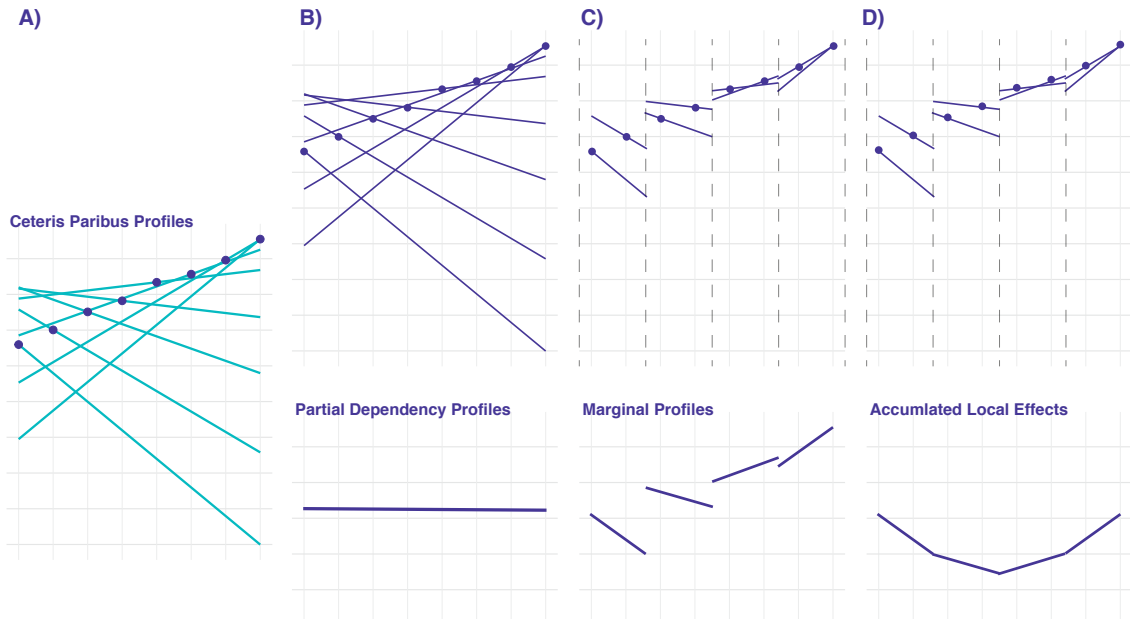


FIGURE 25 (fig:accumulatedLocalEffects) Differences between Partial Dependency, Marginal and Accumulated Local Effects profiles. Panel A) shows Ceteris Paribus Profiles for 8 points. Panel B) shows Partial Dependency profiles, i.e. an average out of these profiles. Panel C shows Marginal profiles, i.e. 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

(Paluszynska and Biecek, 2017b) (Goldstein et al., 2017) (Apley, 2018b)

(Tatarynowicz et al., 2018)

0.23 Performance Diagnostic

Goal: how good is the model, which is better

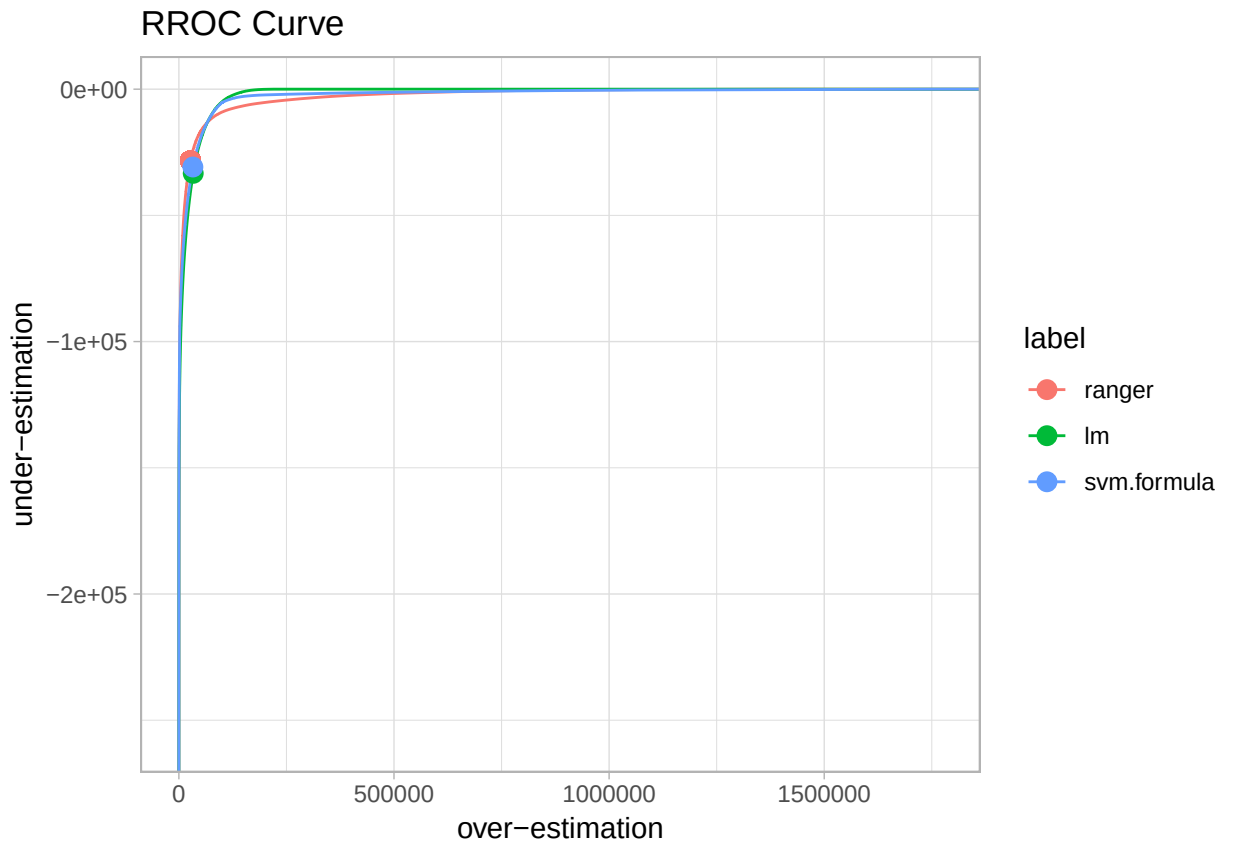
(Piltaver et al., 2016) how good is the tree explainer

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

(Gosiewska and Biecek, 2018)

```
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    2.8
```

```
# 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    3.6
```

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()`.

Bibliography

Apley, D. (2018a). *ALEPlot: Accumulated Local Effects (ALE) Plots and Partial Dependence (PD) Plots*. R package version 1.1.

Apley, D. (2018b). *ALEPlot: Accumulated Local Effects (ALE) Plots and Partial Dependence (PD) Plots*. R package version 1.1.

Bach, S., Binder, A., Montavon, G., Klauschen, F., Müller, K.-R., and Samek, W. (2015). On Pixel-Wise Explanations for Non-Linear Classifier Decisions by Layer-Wise Relevance Propagation. *PLOS ONE*, 10(7):e0130140.

Biecek, P. (2018a). *breakDown: Model Agnostic Explainers for Individual Predictions*. R package version 0.1.6.

Biecek, P. (2018b). *DALEX: Descriptive mAchine Learning EXplanations*. R package version 0.2.4.

Biecek, P. (2019). *ingredients: Effects and Importances of Model Ingredients*. R package version 0.3.2.

Biecek, P. and Kosinski, M. (2017). archivist: An R package for managing, recording and restoring data analysis results. *Journal of Statistical Software*, 82(11):1–28.

Breiman, L., Cutler, A., Liaw, A., and Wiener, M. (2018). *randomForest: Breiman and Cutler's Random Forests for Classification and Regression*. R package version 4.6-14.

Casey, B., Farhangi, A., and Vogl, R. (2018). Rethinking explainable machines: The gdpr's 'right to explanation' debate and the rise of algorithmic audits in enterprise. *Berkeley Technology Law Journal*.

Cortez, P., Cerdeira, A., Almeida, F., Matos, T., and Reis, J. (2009). Modeling wine preferences by data mining from physicochemical properties. *Decision Support Systems*, 47(4):547–553.

Dastin, J. (2018). Amazon scraps secret ai recruiting tool that showed bias against women.

Demšar, J. and Bosnić, Z. (2018). Detecting concept drift in data streams using model explanation. *Expert Systems with Applications*, 92:546–559.

Edwards, L. and Veale, M. (2018). Enslaving the algorithm: From a "right to an explanation" to a "right to better decisions"? *IEEE Security and Privacy*, 16(3):46–54.

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

Fisher, A., Rudin, C., and Dominici, F. (2018). Model Class Reliance: Variable Importance Measures for any Machine Learning Model Class, from the "Rashomon" Perspective. *ArXiv e-prints*.

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

Foster, D. (2018). *xgboostExplainer: XGBoost Model Explainer*. R package version 0.1.

Friedman, J. H. (2000). Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29:1189–1232.

GDPR (2018). The eu general data protection regulation (gdpr) is the most important change in data privacy regulation in 20 years.

Goldstein, A., Kapelner, A., and Bleich, J. (2017). *ICEbox: Individual Conditional Expectation Plot Toolbox*. R package version 1.1.2.

Goldstein, A., Kapelner, A., Bleich, J., and Pitkin, E. (2015a). Peeking inside the black box: Visualizing statistical learning with plots of individual conditional expectation. *Journal of Computational and Graphical Statistics*, 24(1):44–65.

Goldstein, A., Kapelner, A., Bleich, J., and Pitkin, E. (2015b). Peeking inside the black box: Visualizing statistical learning with plots of individual conditional expectation. *Journal of Computational and Graphical Statistics*, 24(1):44–65.

Goodman, B. and Flaxman, S. (2016). European union regulations on algorithmic decision-making and a "right to explanation". *Arxiv*.

Gosiewska, A. and Biecek, P. (2018). *auditor: Model Audit - Verification, Validation, and Error Analysis*. R package version 0.2.1.

Gosiewska, A., Gacek, A., Lubon, P., and Biecek, P. (2019). Safe ml: Surrogate assisted feature extraction for model learning.

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

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

Greenwell, B. M. (2017c). pdp: An R Package for Constructing Partial Dependence Plots. *The R Journal*, 9(1):421–436.

Harrell Jr, F. E. (2018). *rms: Regression Modeling Strategies*. R package version 5.1-2.

Larson, J., Mattu, S., Kirchner, L., and Angwin, J. (2016). How we analyzed the compas recidivism algorithm.

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

Lundberg, S. M., Erion, G. G., and Lee, S. (2018). Consistent individualized feature attribution for tree ensembles. *CoRR*, abs/1802.03888.

Lundberg, S. M. and Lee, S.-I. (2017). A unified approach to interpreting model predictions. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30*, pages 4765–4774. Curran Associates, Inc.

Meyer, D., Dimitriadou, E., Hornik, K., Weingessel, A., and Leisch, F. (2017). *e1071: Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien*. R package version 1.6-8.

Molnar, C. (2018a). *iml: Interpretable Machine Learning*. R package version 0.7.0.

Molnar, C. (2018b). *Interpretable Machine Learning*. https://christophm.github.io/interpretable-ml-book/. https://christophm.github.io/interpretable-ml-book/.

Molnar, C., Bischl, B., and Casalicchio, G. (2018a). iml: An r package for interpretable machine learning. *JOSS*, 3(26):786.

Molnar, C., Bischl, B., and Casalicchio, G. (2018b). iml: An r package for interpretable machine learning. *JOSS*, 3(26):786.

Molnar, C., Bischl, B., and Casalicchio, G. (2018c). iml: An r package for interpretable machine learning. *JOSS*, 3(26):786.

O'Connell, M., Hurley, C., and Domijan, K. (2017). Conditional visualization for statistical models: An introduction to the condvis package in r. *Journal of Statistical Software, Articles*, 81(5):1–20.

O'Neil, C. (2016). *Weapons of Math Destruction: How Big Data Increases Inequality and Threatens Democracy*. Crown Publishing Group, New York, NY, USA.

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

Paluszynska, A. and Biecek, P. (2017b). *randomForestExplainer: Explaining and Visualizing Random Forests in Terms of Variable Importance*. R package version 0.9.

Pedersen, T. L. and Benesty, M. (2018). *lime: Local Interpretable Model-Agnostic Explanations*. R package version 0.4.0.

Piltaver, R., Luštrek, M., Gams, M., and Martinčić-Ipšić, S. (2016). What makes classification trees comprehensible? *Expert Systems with Applications*, 62:333 – 346.

Puri, N., Gupta, P., Agarwal, P., Verma, S., and Krishnamurthy, B. (2017). MAGIX: model agnostic globally interpretable explanations. *CoRR*, abs/1706.07160.

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

Ridgeway, G. (2017). *gbm: Generalized Boosted Regression Models*. R package version 2.1.3.

Ross, C. and Swetliz, I. (2018). Ibm's watson supercomputer recommended 'unsafe and incorrect' cancer treatments, internal documents show.

Ruiz, J. (2018). Machine learning and the right to explanation in gdpr.

Salzberg, S. (2014). Why google flu is a failure.

Shapley, L. S. (1953). A value for n-person games. In Kuhn, H. W. and Tucker, A. W., editors, *Contributions to the Theory of Games II*, pages 307–317. Princeton University Press, Princeton.

Simonyan, K., Vedaldi, A., and Zisserman, A. (2013). Deep inside convolutional networks: Visualising image classification models and saliency maps. *CoRR*, abs/1312.6034.

Sitko, A., Grudziąż, A., and Biecek, P. (2018). *factorMerger: The Merging Path Plot*. R package version 0.3.6.

Staniak, M. and Biecek, P. (2018). *live: Local Interpretable (Model-Agnostic) Visual Explanations*. R package version 1.5.7.

Strobl, C., Boulesteix, A.-L., Kneib, T., Augustin, T., and Zeileis, A. (2008). Conditional variable importance for random forests. *BMC Bioinformatics*, 9(1):307.

Strobl, C., Boulesteix, A.-L., Zeileis, A., and Hothorn, T. (2007). Bias in random forest variable importance measures: Illustrations, sources and a solution. *BMC Bioinformatics*, 8(1):25.

Štrumbelj, E. and Kononenko, I. (2014). Explaining prediction models and individual predictions with feature contributions. *Knowledge and Information Systems*, 41(3):647–665.

Tatarynowicz, M., Romaszko, K., and Urbański, M. (2018). *modelDown: Make Static HTML Website for Predictive Models*. R package version 0.1.1.

Tufte, E. R. (1986). *The Visual Display of Quantitative Information*. Graphics Press, Cheshire, CT, USA.

van der Loo, M. (2017). *gower: Gower's Distance*. R package version 0.1.2.

Xie, Y. (2018). *bookdown: Authoring Books and Technical Documents with R Markdown*. R package version 0.7.