Predictive Models: Visualisation, Exploration and Explanation

With examples in R and Python

Przemyslaw Biecek and Tomasz Burzykowski

2019-03-05

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.

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:

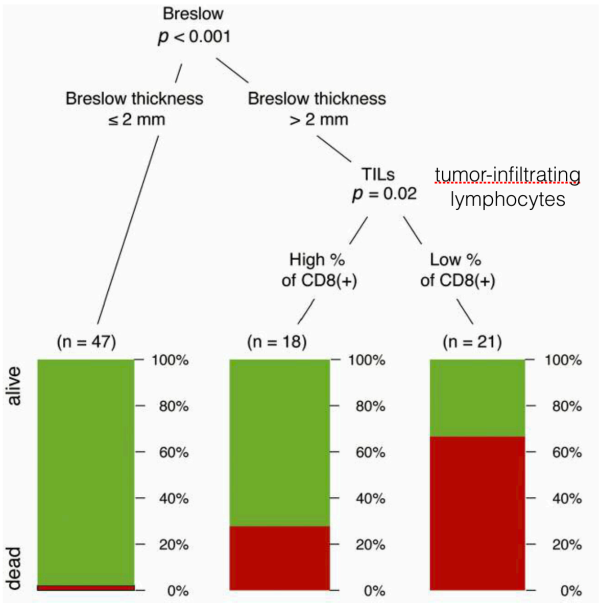
$$RelativeRisk = 1 + 3.6 * [Breslow > 2] - 2 * [TILs > 0]$$

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 techniques introduced in this book on three datasets:

- *Sinking of the RMS Titanic* as an example of binary classification
- *Apartment prices* as an example of regression model
- *Hire or Fire* as an example of multi-class classification and interactions

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. Thanks to projects like *Encyclopedia titanica* `https://www.encyclopedia-titanica.org/` we have a very rich and precise data about passengers. This dataset is available in the `titanic` dataset.

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

```
##   PassengerId Survived Pclass
```

FIGURE 1 Titanic sinking by Willy Stöwer

```
## 1           1        0      3
## 2           2        1      1
##                                                  Name    Sex Age SibSp
## 1                             Braund, Mr. Owen Harris   male  22     1
## 2 Cumings, Mrs. John Bradley (Florence Briggs Thayer) female  38     1
##   Parch    Ticket    Fare Cabin Embarked
## 1     0 A/5 21171  7.2500              S
## 2     0  PC 17599 71.2833   C85        C
```

0.2.1.1 What are the chances for survival on the Titanic?

Feature of interest is the binary variable `Survived`. Let's build some predictive models for this variable.

First we need to do some data preprocessing. Columns with characters are converted to factors and rows with missing data are removed.

```
titanic_small <- titanic_train[,c("Survived", "Pclass", "Sex", "Age", "SibSp", "Parch", "Fare", "Embarked"
titanic_small$Survived <- factor(titanic_small$Survived)
titanic_small$Sex <- factor(titanic_small$Sex)
titanic_small$Embarked <- factor(titanic_small$Embarked)
titanic_small <- na.omit(titanic_small)
```

0.2.1.2 Data exploration

It is always a good idea to do data exploration before modeling. But since this book is focused on model exploration we will spend only a few lines on data exploration part. And we will limit ourselves to two-variable summaries for each variable.

TODO: PBi make it readable

```
table(titanic_small$Survived , titanic_small$Pclass)
```

```
##
##       1   2   3
##   0  64  90 270
##   1 122  83  85
```

```
table(titanic_small$Survived , titanic_small$Sex)
```

```
##
##     female male
##   0     64  360
##   1    197   93
```

```
table(titanic_small$Survived , titanic_small$Parch)
```

```
##
##       0   1   2   3   4   5   6
##   0 335  49  29   2   4   4   1
##   1 186  61  39   3   0   1   0
```

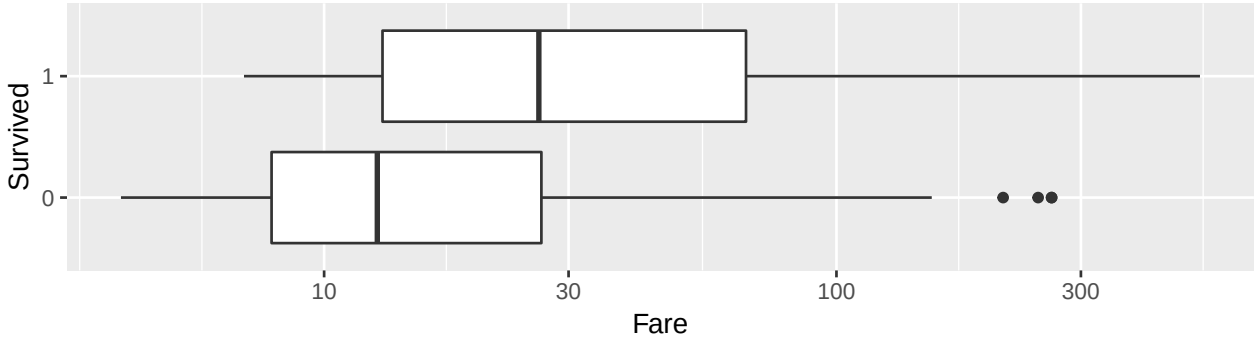
```
table(titanic_small$Survived , titanic_small$SibSp)
```

```
##
##       0   1   2   3   4   5
##   0 296  86  14   8  15   5
##   1 175  97  11   4   3   0
```

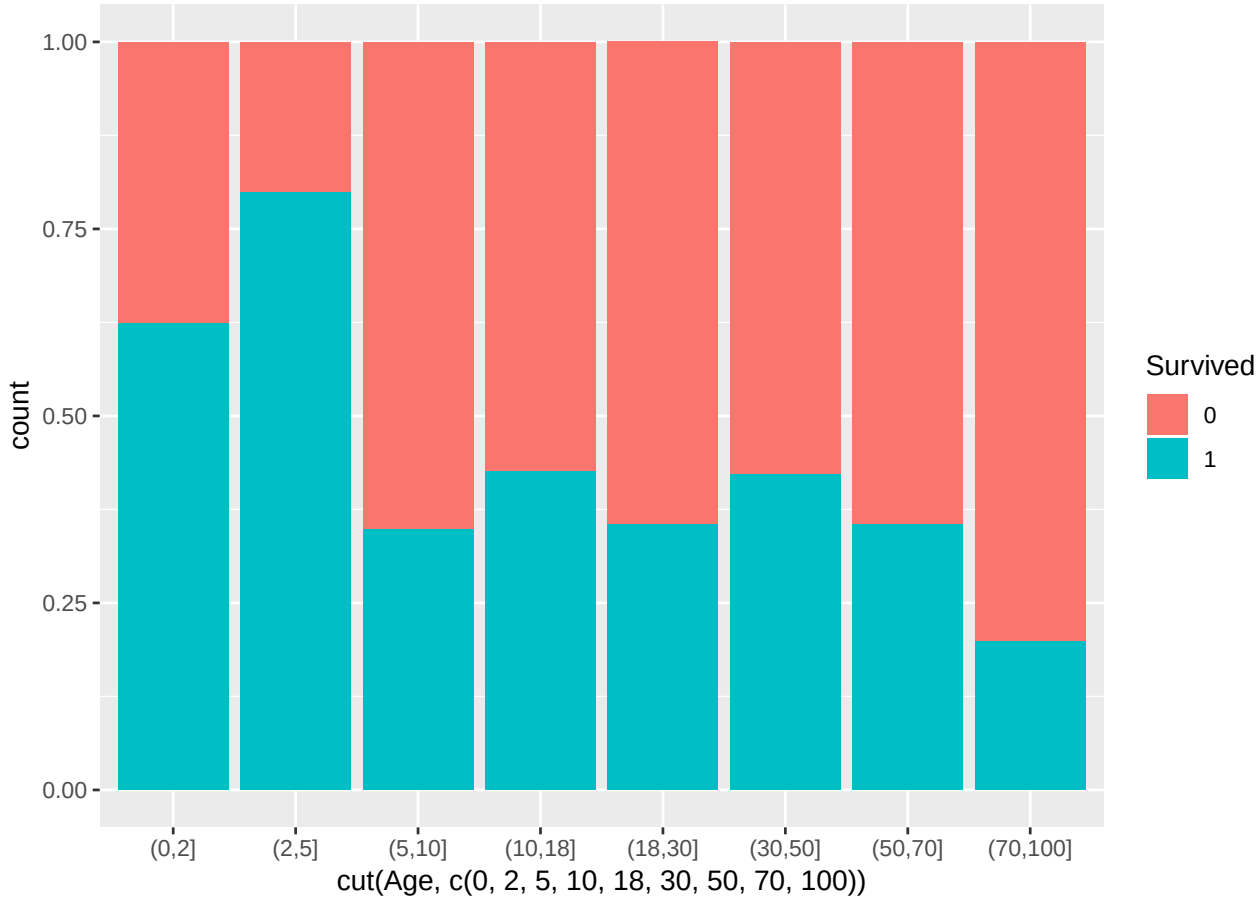
```
table(titanic_small$Survived , titanic_small$Embarked)
```

```
##
##            C   Q   S
##   0   0  51  20 353
##   1   2  79   8 201
```

```
library("ggplot2")
ggplot(titanic_small, aes(Survived, Fare)) +
  geom_boxplot() + coord_flip() + scale_y_log10()
```




```
ggplot(titanic_small, aes(fill = Survived, x=cut(Age,c(0,2,5,10,18,30,50,70,100)))) +
  geom_bar(position = "fill")
```



0.2.1.3 Logistic regression is always a good choice

The feature of interest `survival` is binary, thus a natural choice is a logistic regression. Most of predictive features are categorical except age.

There is no reason to expect a linear relation between age and odds of survival, thus for age we will use linear tail-restricted cubic splines available in the `rcs()` function in the `rms` package (Harrell Jr, 2018).

```
library("rms")
lmr_model <- lrm(Survived == "1" ~ Pclass + Sex + rcs(Age) + SibSp +
                   Parch + Fare + Embarked, titanic_small)
lmr_model
```

```
## Logistic Regression Model
##
##  lrm(formula = Survived == "1" ~ Pclass + Sex + rcs(Age) + SibSp +
##      Parch + Fare + Embarked, data = titanic_small)
##
##                       Model Likelihood     Discrimination    Rank Discrim.
##                          Ratio Test           Indexes           Indexes
##  Obs          714    LR chi2     353.20    R2       0.527    C       0.874
##   FALSE       424    d.f.            12    g        2.192    Dxy     0.748
##   TRUE        290    Pr(> chi2) <0.0001    gr       8.957    gamma   0.749
##  max |deriv| 0.1                          gp       0.363    tau-a   0.361
##                                           Brier    0.134
##
##              Coef    S.E.    Wald Z Pr(>|Z|)
##  Intercept  12.1530 26.3889   0.46 0.6451
##  Pclass     -1.0822  0.1693  -6.39 <0.0001
##  Sex=male   -2.7848  0.2317 -12.02 <0.0001
##  Age        -0.2382  0.0472  -5.04 <0.0001
##  Age'        0.7961  0.2138   3.72 0.0002
##  Age''      -4.3833  1.5123  -2.90 0.0038
##  Age'''      5.1859  2.2693   2.29 0.0223
##  SibSp      -0.5292  0.1449  -3.65 0.0003
##  Parch      -0.1990  0.1344  -1.48 0.1387
##  Fare        0.0032  0.0029   1.11 0.2670
##  Embarked=C -4.6054 26.3806  -0.17 0.8614
##  Embarked=Q -5.3341 26.3853  -0.20 0.8398
##  Embarked=S -4.9968 26.3802  -0.19 0.8498
##
```

0.2.1.4 Random Forest to the rescue

In addition to a logistic regression we will use a random forest model with default settings. Random forest is known for good performance, is able to grasp low-level variable interactions and is quite stable.

Here we are using the `randomForest` package (Liaw and Wiener, 2002).

```
library("randomForest")
rf_model <- randomForest(Survived ~ Pclass + Sex + Age + SibSp +
                           Parch + Fare + Embarked,
                           data = titanic_small)
rf_model
```

```
##
```

```
## Call:
##  randomForest(formula = Survived ~ Pclass + Sex + Age + SibSp +     Parch + Fare + Embarked, data = titanic_sma
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 18.63%
## Confusion matrix:
##     0   1 class.error
## 0 382  42   0.0990566
## 1  91 199   0.3137931
```

0.2.1.5 Gradient boosting for interactions

Last model that we will train on this dataset is the gradient boosting model. This family of models is known for being able to grasp deep interactions between variables.

Here we are using the implementation from the `gbm` package (Ridgeway, 2017).

```
library("gbm")
gbm_model <- gbm(Survived == "1" ~ Pclass + Sex + Age + SibSp +
                    Parch + Fare + Embarked, data = titanic_small, n.trees = 15000)
```

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

```
gbm_model
```

```
## gbm(formula = Survived == "1" ~ Pclass + Sex + Age + SibSp +
##     Parch + Fare + Embarked, data = titanic_small, 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.
```

0.2.1.6 Model predictions

Having all three models let's see what are odds of surviving for a 2-years old boy that travels in the 3rd class with 1 parent and 3 siblings.

```
henry <- data.frame(
            Pclass = 1,
            Sex = factor("male", levels = c("female", "male")),
            Age = 8,
            SibSp = 0,
            Parch = 0,
            Fare = 72,
            Embarked = factor("C", levels = c("","C","Q","S"))
)
```

Logistic regression model says 88.3% for survival.

```
predict(lmr_model, henry, type = "fitted")
```

```
##         1
## 0.8836101
```

Random forest model says 53.2% for survival.

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

```
##       0     1
## 1 0.444 0.556
## attr(,"class")
## [1] "matrix" "votes"
```

Gradient boosting model says 53.2% for survival.

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

```
## [1] 0.8337385
```

Three different opinions. Which one should we trust? Tools introduced in following sections will help to understand how these models are different.

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 3. We will use it as an illustration of key ideas. The plot provides an information about the values of the function in the vicinity of point x^*.

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

- One approach is to investigate how the model prediction changes if the value of a single explanatory variable changes. It is useful in the so-called „What-If" scenarios. In particular, we can construct plots presenting the change of model-based predictions in function of a single variable. Such plots are usually called Ceteris Paribus profiles. They are presented in Chapter 0.9. An example is provided in panel A of Figure 4.
- Another approach is to analyze the curvature of the response surface (see Figure 3) around the point of interest x^*. Treating the model as a function, we are interested in the local behavior of this function

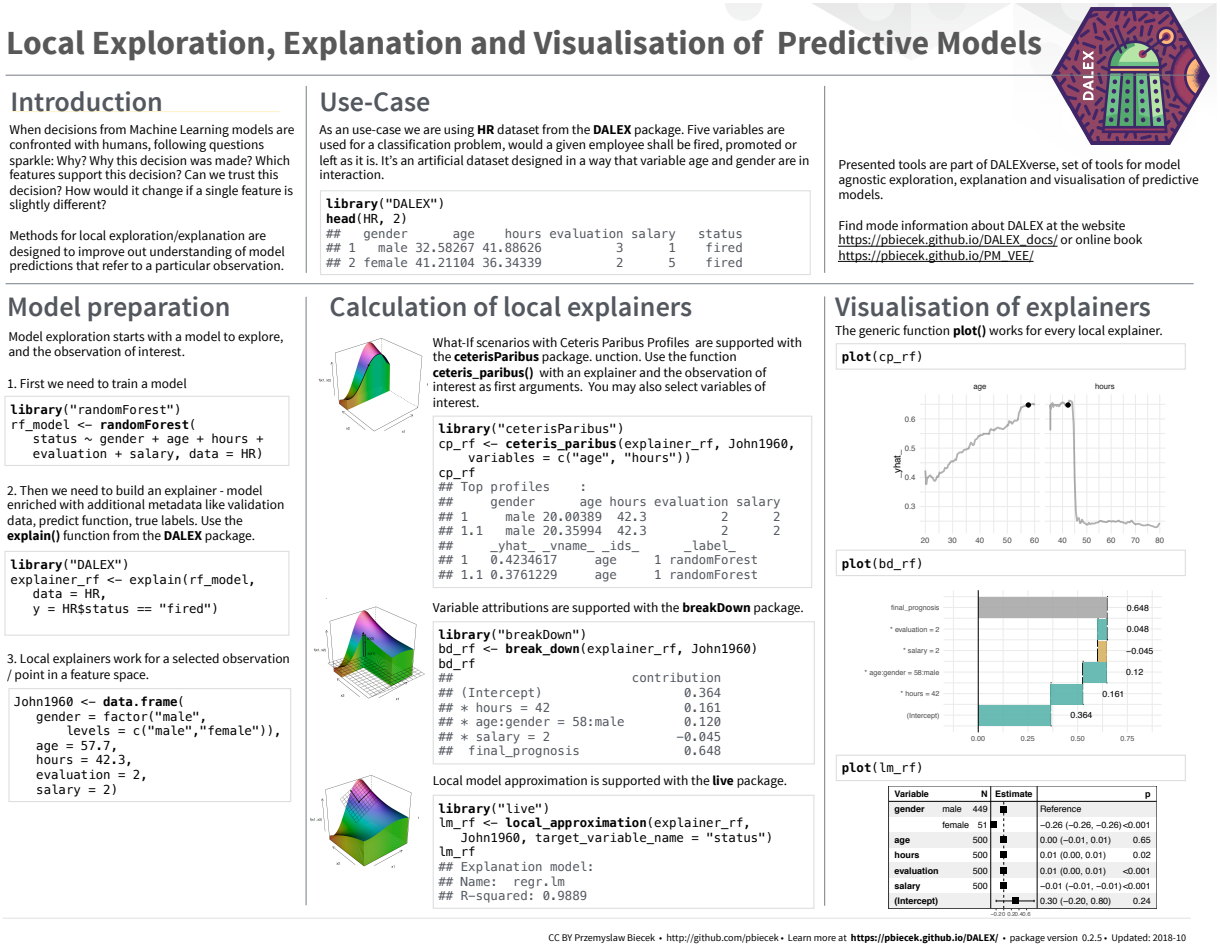


FIGURE 2 (fig:localDALEXsummary) Summary of three approaches to local model exploration and explanation.

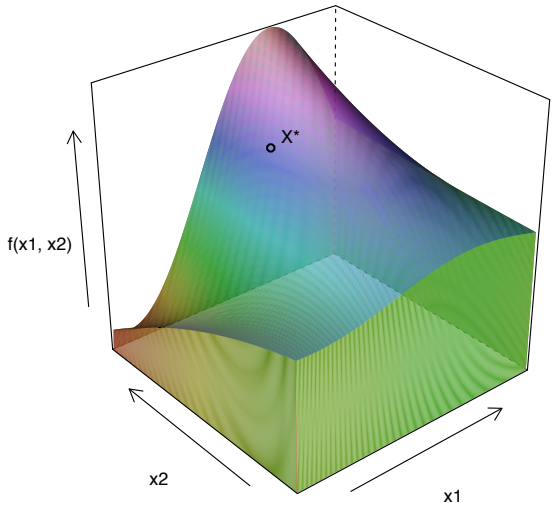


FIGURE 3 (fig:cutsSurfaceReady) Response surface for a model that is a function of two variables. We are interested in understanding the response of a model in a single point x*

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 4. In Chapter 0.8 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 4. In Chapter 0.4 we present two methods implementing this approach, sequential conditioning and average conditioning (also called Shapley values).

0.4 Variable attribution for linear models

0.4.1 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.

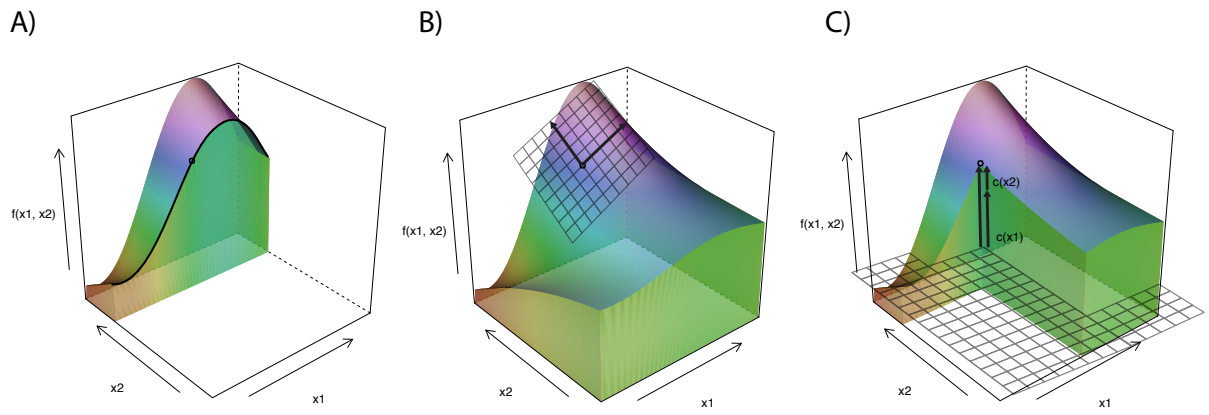


FIGURE 4 (fig:cutsTechnikiReady) Illustration of different approaches to instance-level explanation. Panel A presents a What-If analysis with Ceteris Paribus profiles. The profiles plot the model response as a function of a value of a single variable, while keeping the values of all other explanatory 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.

- 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 0.5 and 0.7.

0.4.2 Intuition

Assume a linear model 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 is given by the following linear combination:

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

We are interested in the contribution of variable x_i on model prediction $f(x^*)$ for a single observation described by x^*. In this case, the contribution is related to $x_i^*\beta_i$, as variable x_i 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:

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

0.4.3 Method

We want to calculate $v(f, x^*, i)$, which is the contribution of variable x_i on prediction of model $f()$ in point x^*.

Geneal approach for calculation of variable attributions would be to measure how much the expected model response would change after conditioning on $x_i = x_i^*$.

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

For linear models, if coordinates of x are independent, this is equivalent of

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

Expected value can be estimated as averages, and this leads to

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

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

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

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

where

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

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

** NOTE for careful readers **

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

Also we assumed that coordinated of x are independent, which may not be the case. We will return to this problem later, during the discussion related to interactions.

0.4.4 Example: Wine quality

It may be a surprise, that the attribution for variable x_i is not the $\beta_i x_i$. To understand this, consider following example.

Figure 5 shows the relation between alcohol and wine quality, based on the wine dataset (Cortez et al., 2009). The corresponding linear model is

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

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

Note, that it would be confusing to use $x_i\beta_i$ as alcohol contribution on quality would be $0.3135 * 8 = 2.508$. This would not reflect the intuition that for positive relation, the smaller is the alcohol concentration the lower should be the quality of wine.

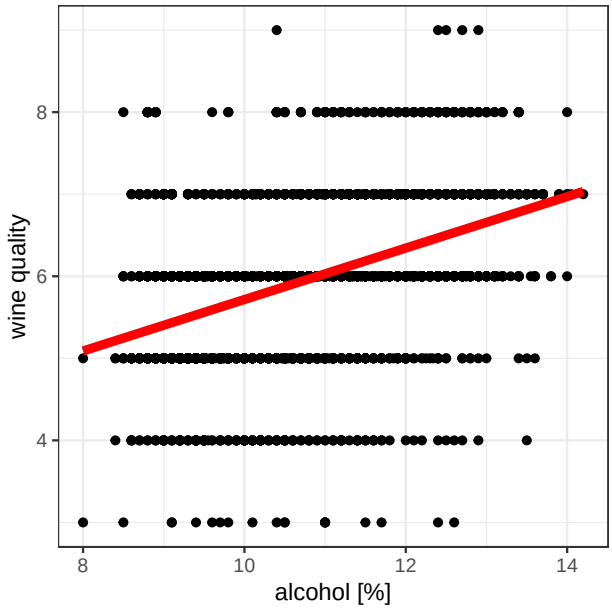


FIGURE 5 (fig:attribution1)Relation between wine quality and concentration of alcohol assessed with linear model

0.4.5 Pros and Cons

Here we summarise pros and cons of this approach.

Pros

- Presented variable attribution for linear model is not an approximation, it is directly linked with the structure of a model.
- It is easier to understand attributions that are not linked with scale nor location of x_i as the standard β_i are.

Cons

- It works only for linear models.
- This do not reduce model complexity. Just present model coefficients in a different way.

0.4.6 Code snippets

Variable attributions for linear models may be directly extracted from the `predict()` function for linear models.

In this section we will present an example for logistic regression based on the `HR` dataset. See the Section 0.18.1 for more details.

First we build a logistic regression model for binary variable `status == "fired"`. Here are fitted 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
```

We want to calculate variable attributions for a particular point. Here we define this point.

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

For linear and generalized linear models we may specify argument `type = "terms"` that extracts variable contributions.

```
predict(model_fired, new_observation, type = "terms")
```

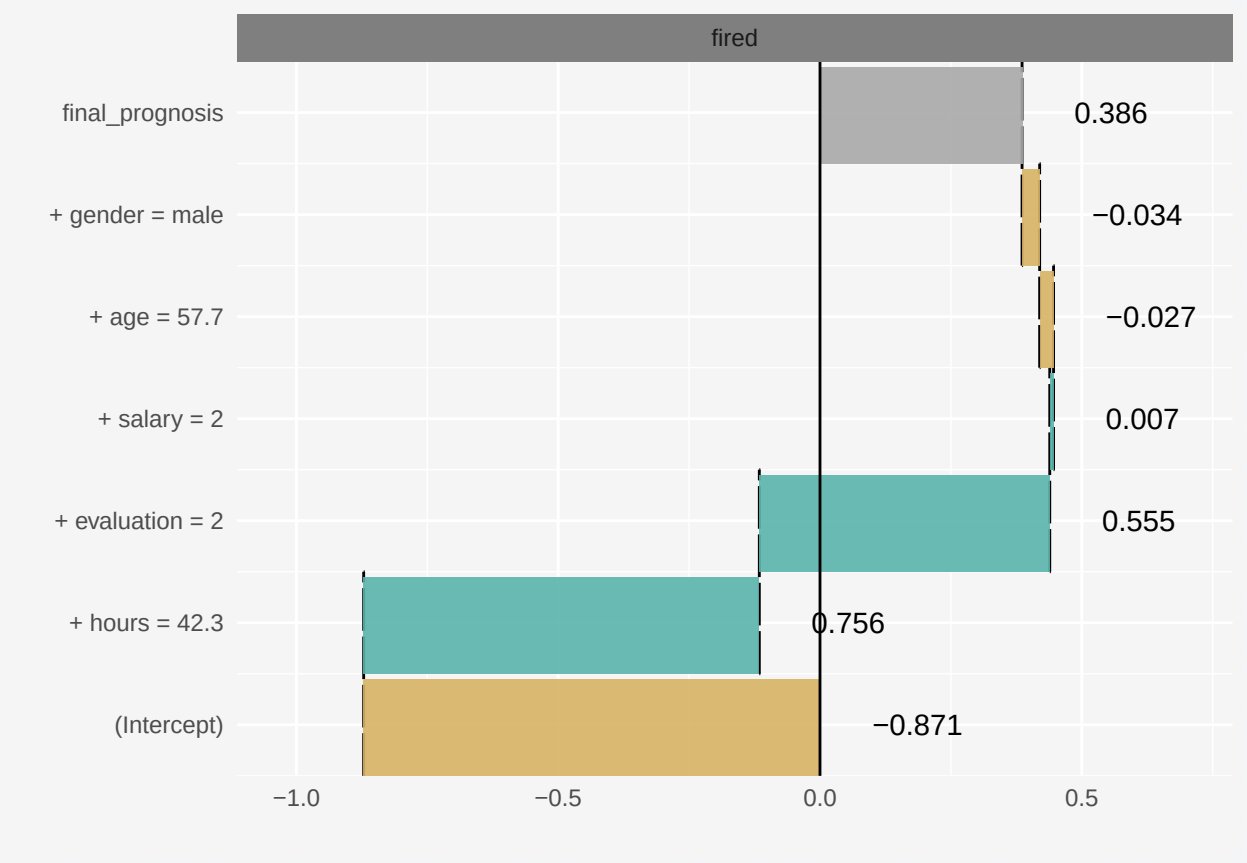
```
##        gender         age     hours evaluation      salary
## 1 -0.03361889 -0.02660691 0.7555555  0.5547197 0.007287334
## attr(,"constant")
## [1] -0.8714962
```

Below we show how to do this with the `DALEX` package. Additionaly we may easily plot contributions.

```
library("DALEX")

explainer_fired <- explain(model_fired,
                 data = HR,
                 y = HR$status == "fired",
                 label = "fired")

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

```
##                    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
```

```
plot(attribution)
```



0.5 Variable attributions

In the Section 0.4 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.

0.5.1 Intuition

For any model we may repeat the intuition presented in Section 0.4 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.

0.5.2 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 7. 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.

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.

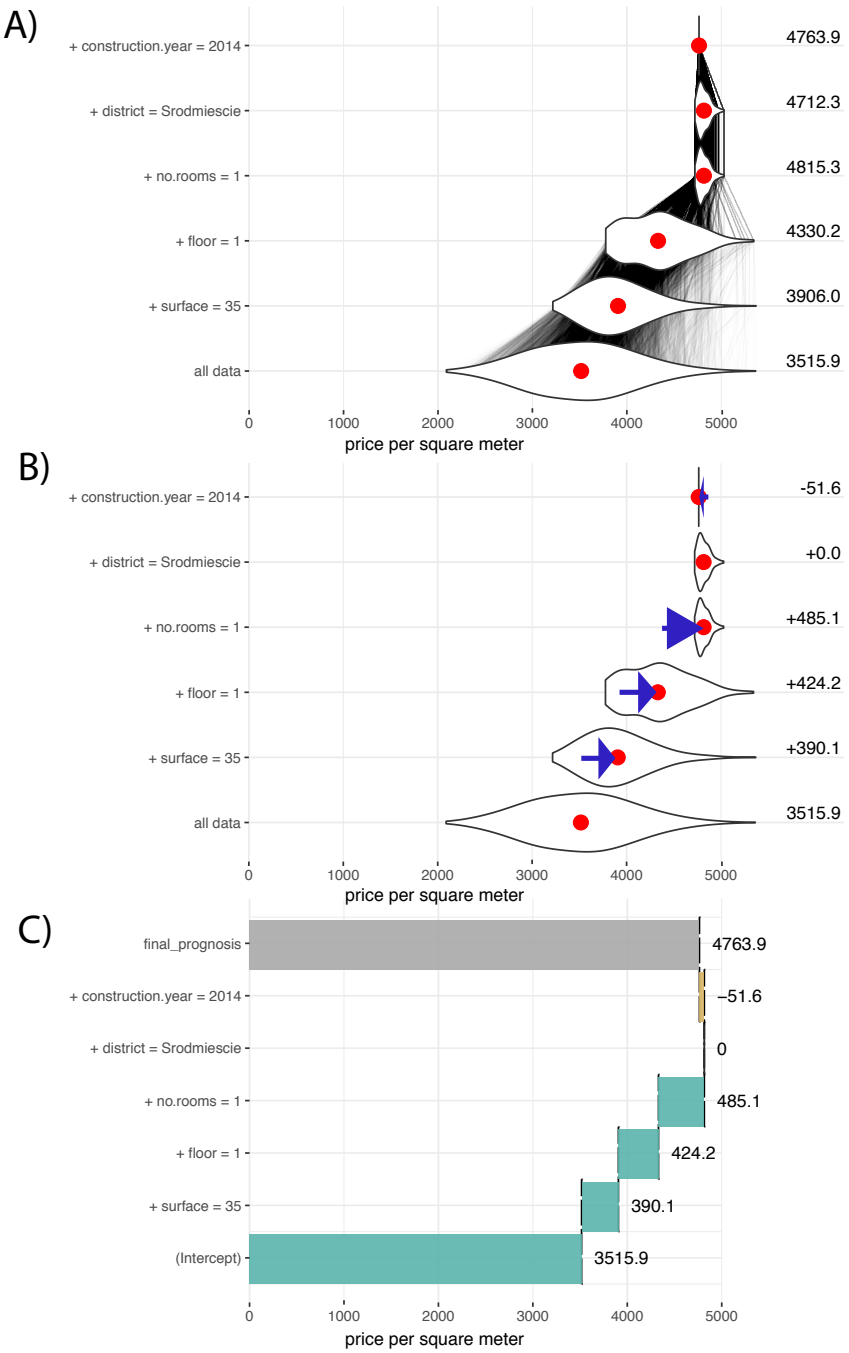


FIGURE 6 (fig:BDPrice4) Break Down Plots show how variables move the model prediction from population average to the model prognosis for a single observation. A) The last row shows distribution of model predictions. Next rows show conditional distributions, every row a new variable is added to conditioning. 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.

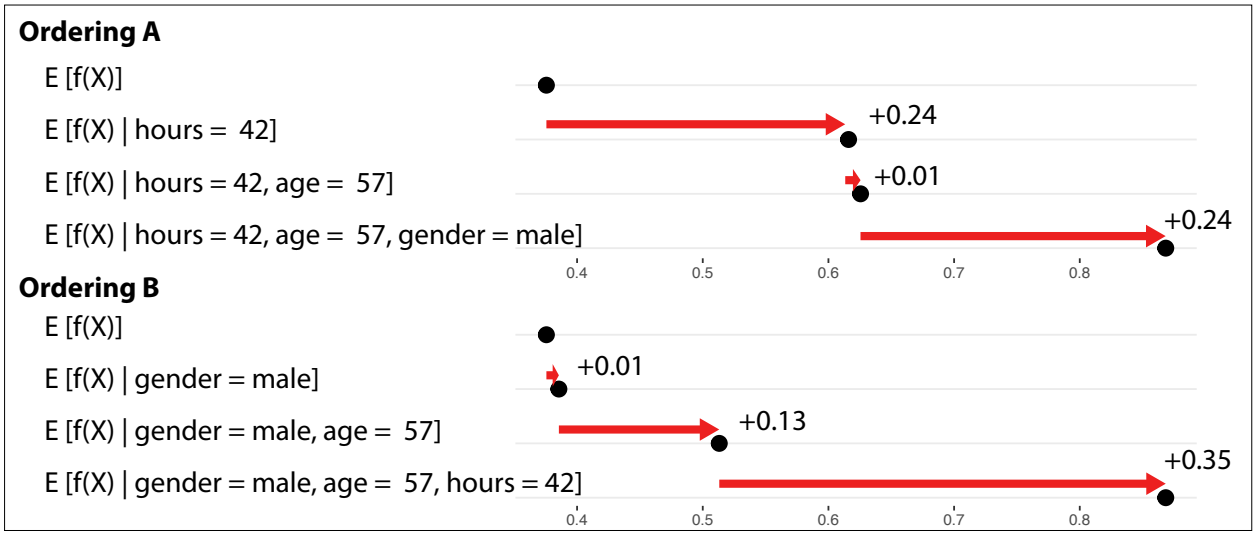


FIGURE 7 (fig:ordering) Two different paths between average model prediction and the model prediction for a selected observation. Black dots stand for conditional average, red arrows stands for changes between conditional averages.

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

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 $O(p)$.

0.5.3 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

0.5.4 Pros and cons

0.5.5 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 0.18.1 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: 27.32%
## Confusion matrix:
##           fired   ok promoted class.error
## fired     2272  386      197   0.2042032
## ok         524 1256      441   0.4344890
## promoted   199  397     2175   0.2150848
```

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, 2018c).

```
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  0.79 0.208    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.

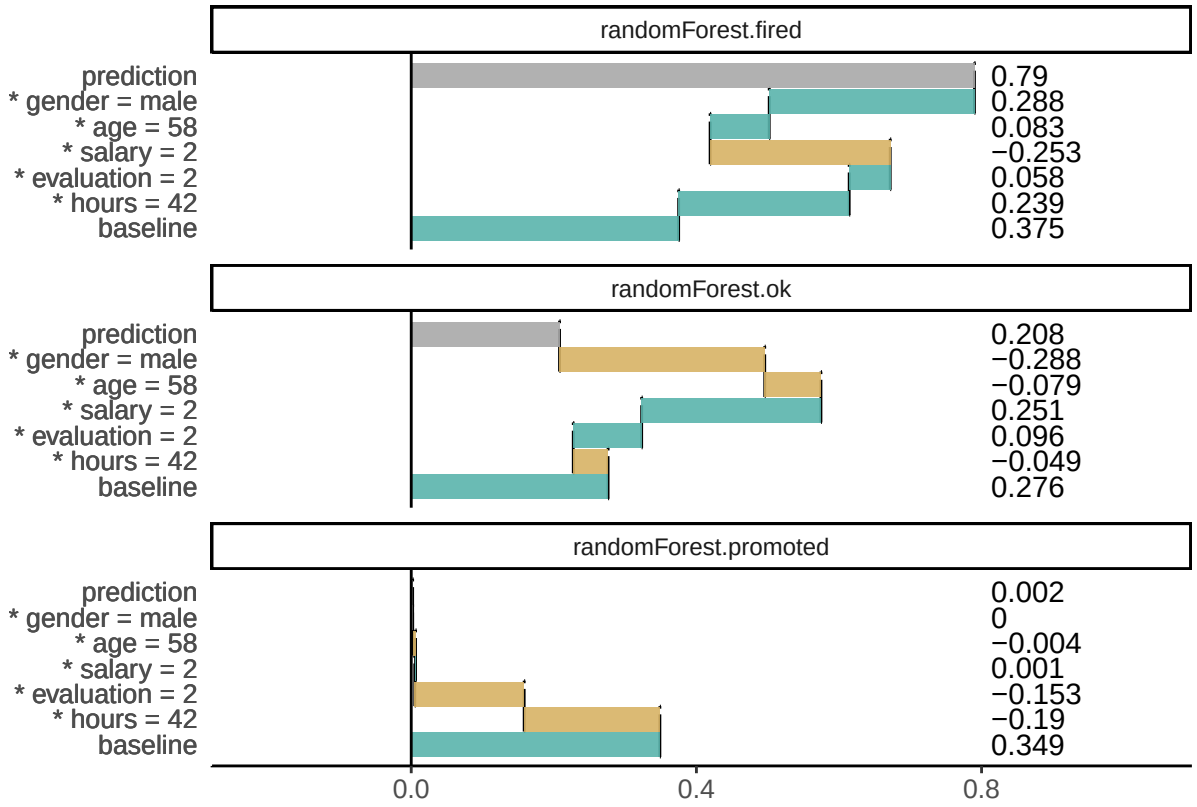
The result from `local_attributions()` function is a data frame with variable attributions.

```
library("breakDown2")
bd_rf <- local_attributions(explainer_rf,
                 new_observation,
                 keep_distributions = TRUE)

bd_rf
```

```
##                                        contribution
## randomForest.fired: baseline                 0.375
## randomForest.fired: * hours = 42             0.239
## randomForest.fired: * evaluation = 2         0.058
## randomForest.fired: * salary = 2            -0.253
## randomForest.fired: * age = 58               0.083
## randomForest.fired: * gender = male          0.288
## randomForest.fired: prediction               0.790
## randomForest.ok: baseline                    0.276
## randomForest.ok: * hours = 42               -0.049
## randomForest.ok: * evaluation = 2            0.096
## randomForest.ok: * salary = 2                0.251
## randomForest.ok: * age = 58                 -0.079
## randomForest.ok: * gender = male            -0.288
## randomForest.ok: prediction                  0.208
## randomForest.promoted: baseline              0.349
## randomForest.promoted: * hours = 42         -0.190
## randomForest.promoted: * evaluation = 2     -0.153
## randomForest.promoted: * salary = 2          0.001
## randomForest.promoted: * age = 58           -0.004
## randomForest.promoted: * gender = male       0.000
## randomForest.promoted: prediction            0.002
## baseline:  0
```

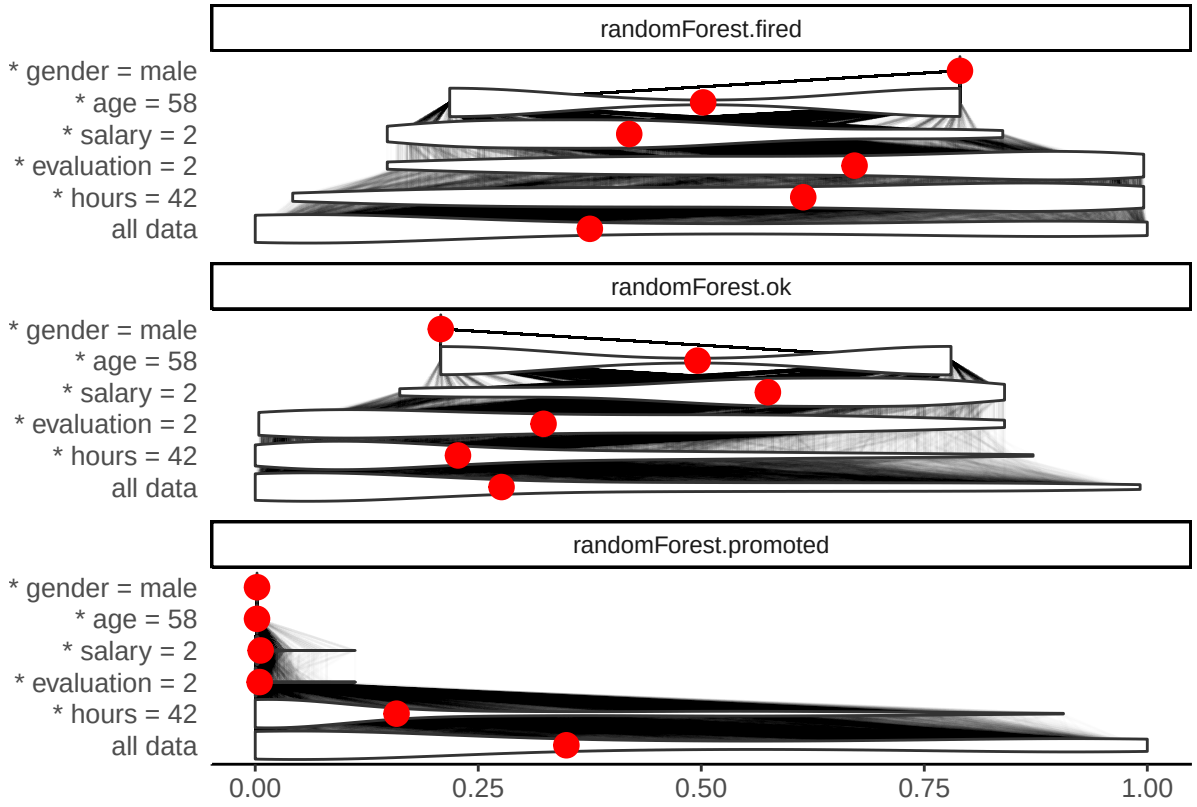
The generic `plot()` function creates a Break Down plots.

```
plot(bd_rf)
```



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

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



0.6 Variable attribution with interactions

In the Section 0.5 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.

0.6.1 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

0.6.2 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

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

Note that this is equivalent to

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

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

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

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

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

0.6.3 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

0.6.4 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 **??**.

0.6.5 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

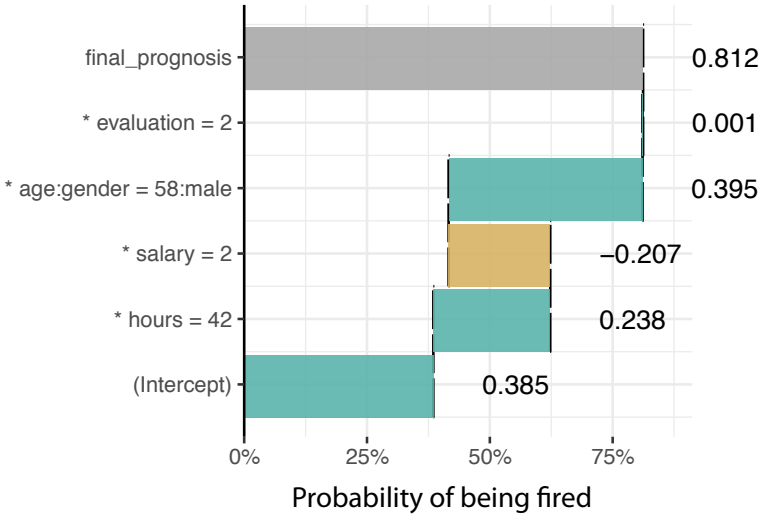


FIGURE 8 (fig:bdInter1) Break Down Plot for variable attrbution with interactions

- 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 $score_1$.

0.6.6 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: 27.37%
## Confusion matrix:
##          fired   ok promoted class.error
## fired     2263  390      202   0.2073555
## ok         534 1252      435   0.4362900
```

```
## promoted   195  392     2184   0.2118369
```

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 0.818 0.18    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.

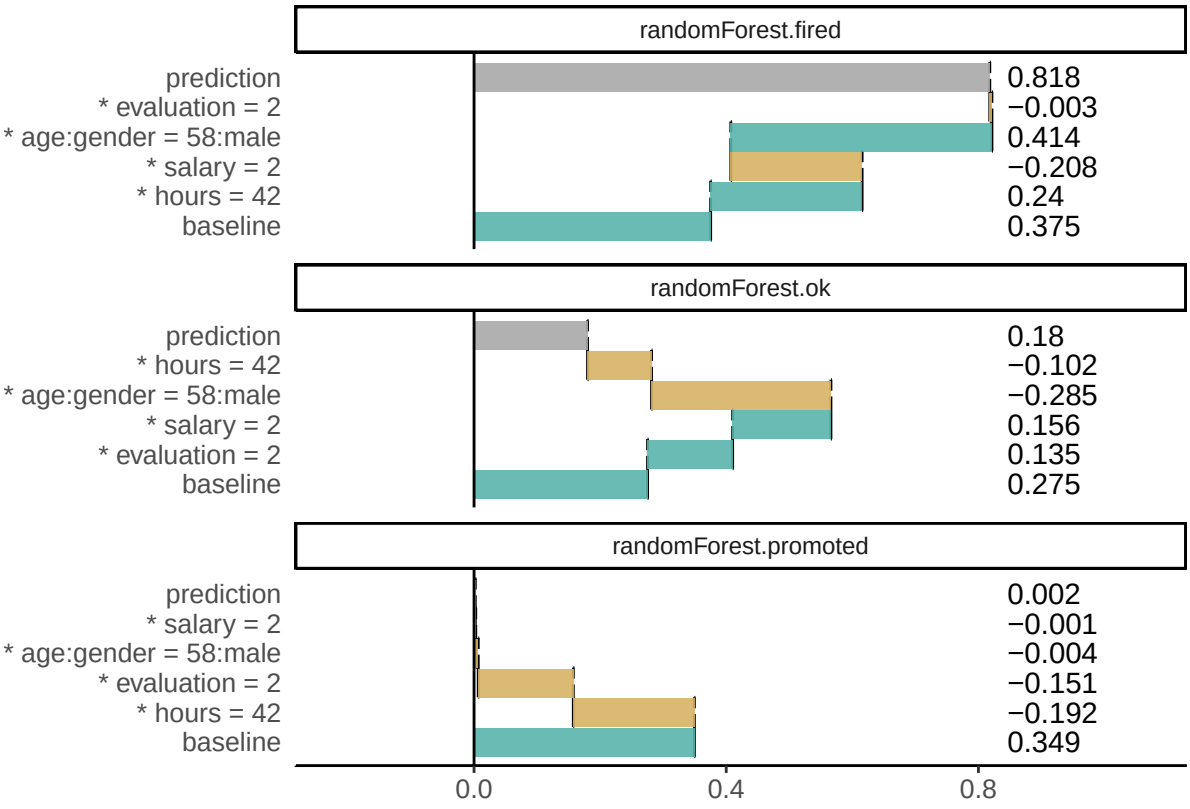
```
library("breakDown2")
bd_rf <- local_interactions(explainer_rf,
                  new_observation)

bd_rf
```

```
##                                                contribution
## randomForest.fired: baseline                          0.375
## randomForest.fired: * hours = 42                      0.240
## randomForest.fired: * salary = 2                     -0.208
## randomForest.fired: * age:gender = 58:male            0.414
## randomForest.fired: * evaluation = 2                 -0.003
## randomForest.fired: prediction                        0.818
## randomForest.ok: baseline                             0.275
## randomForest.ok: * evaluation = 2                     0.135
## randomForest.ok: * salary = 2                         0.156
## randomForest.ok: * age:gender = 58:male              -0.285
## randomForest.ok: * hours = 42                        -0.102
## randomForest.ok: prediction                           0.180
## randomForest.promoted: baseline                       0.349
## randomForest.promoted: * hours = 42                  -0.192
## randomForest.promoted: * evaluation = 2              -0.151
```

```
## randomForest.promoted: * age:gender = 58:male       -0.004
## randomForest.promoted: * salary = 2                 -0.001
## randomForest.promoted: prediction                    0.002
## baseline:  0
```

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

```
plot(bd_rf)
```



0.7 Average variable attributions

In the Section **??** we show the problem related to the ordering of variables. In the Section 0.5 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).

0.7.1 Intuition

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

TODO: a nice example

0.7.2 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.

0.7.3 Example: Hire or Fire?

0.7.4 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 **??**, 0.5 and 0.7 lead to same variable contributions.

0.7.5 Code snippets for R

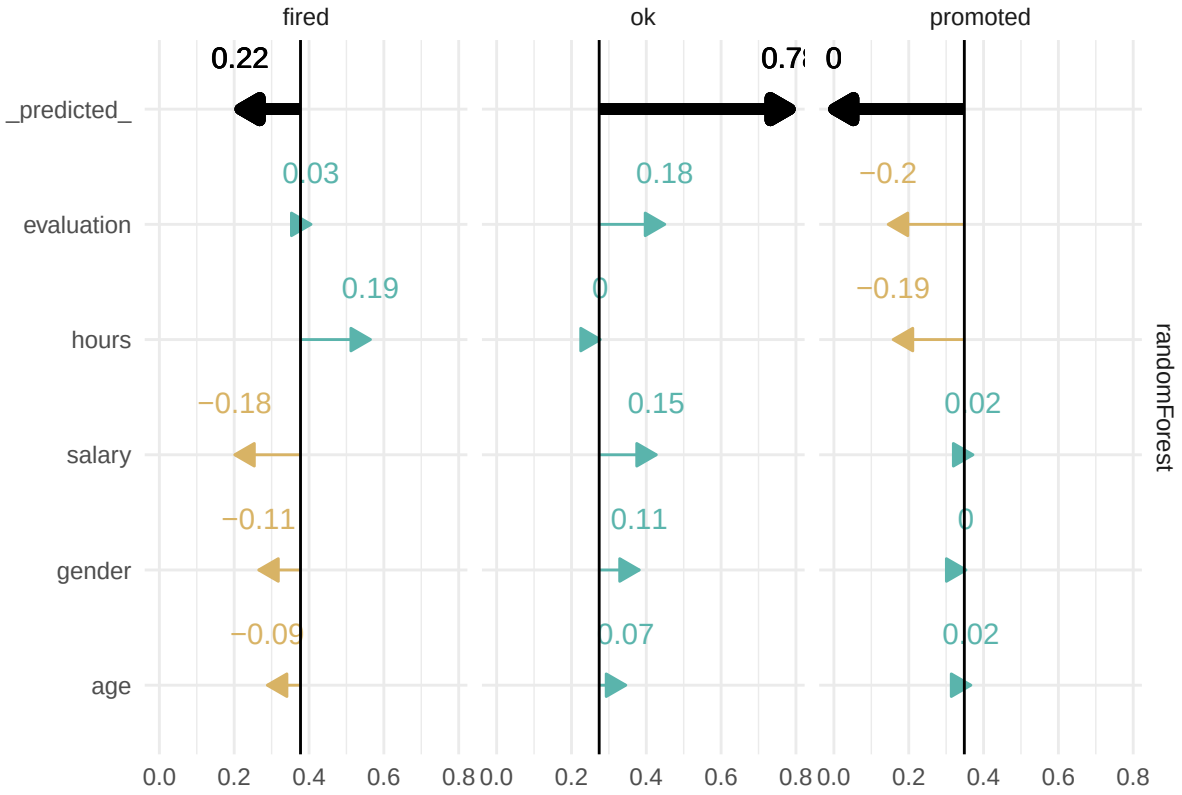
In this section we will present an example based on the `HR` dataset and Random Forest model (Breiman et al., 2018). See the Section 0.18.1 for more details.

```
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., 2018).

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

Explanations for a new observation.

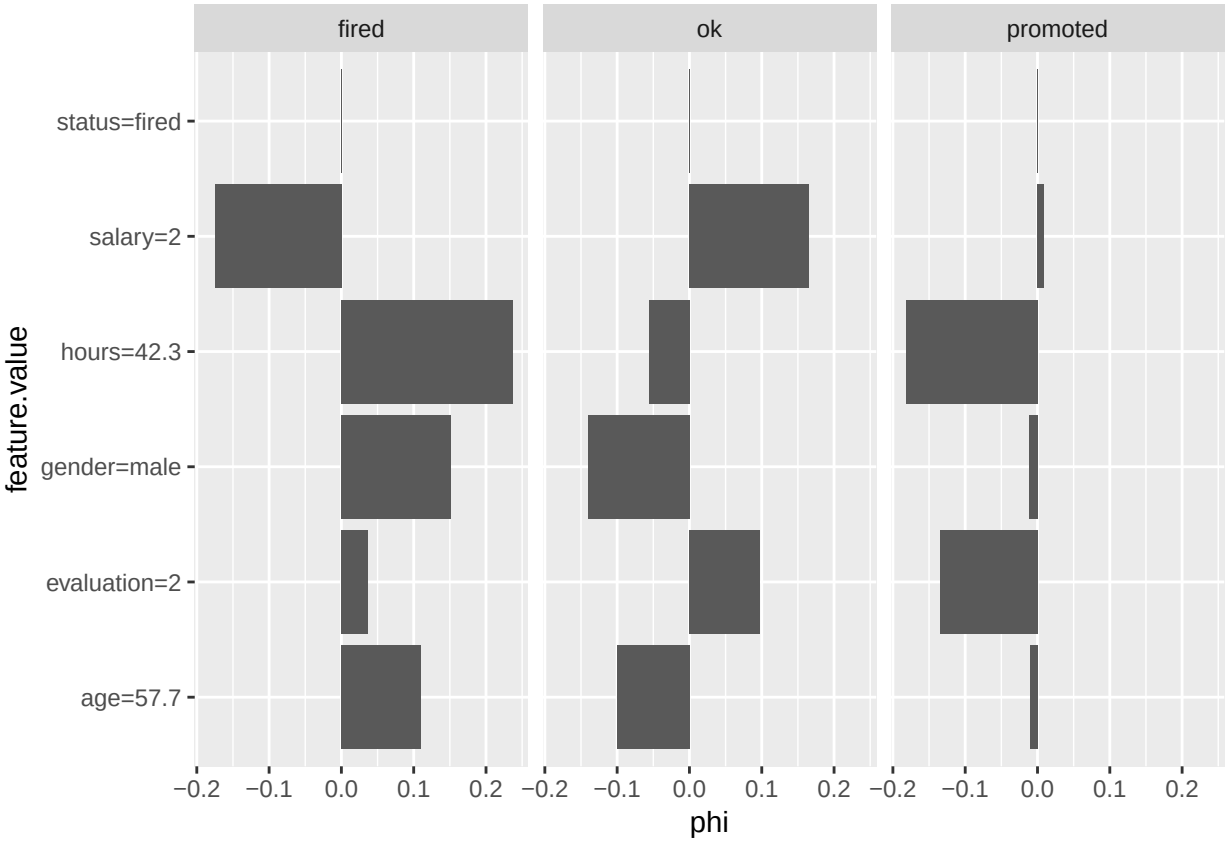
```
new_observation <- data.frame(gender = factor("male", levels = c("male", "female")),
                      age = 57.7,
                      hours = 42.3,
                      evaluation = 2,
                      salary = 2,
                      status = factor("fired"))

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

```
## Interpretation method:  Shapley
```

```
## Predicted value: 0.768000, Average prediction: 0.375522 (diff = 0.392478) Predicted value: 0.232000, Average
##
## Analysed predictor:
## Prediction task: unknown
##
##
## Analysed data:
## Sampling from data.frame with 7847 rows and 6 columns.
##
## Head of results:
##      feature class      phi    phi.var feature.value
## 1     gender fired  0.15184 0.06525201   gender=male
## 2        age fired  0.10980 0.07939374      age=57.7
## 3      hours fired  0.23736 0.11281486    hours=42.3
## 4 evaluation fired  0.03640 0.01645164  evaluation=2
## 5     salary fired -0.17504 0.04931632      salary=2
## 6     status fired  0.00000 0.00000000  status=fired
```

And the plot with Shapley attributions.

```
plot(shapley)
```



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

0.8 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 0.5 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 0.9 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).

0.8.1 Intuition

0.8.2 Method

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

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

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

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

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.

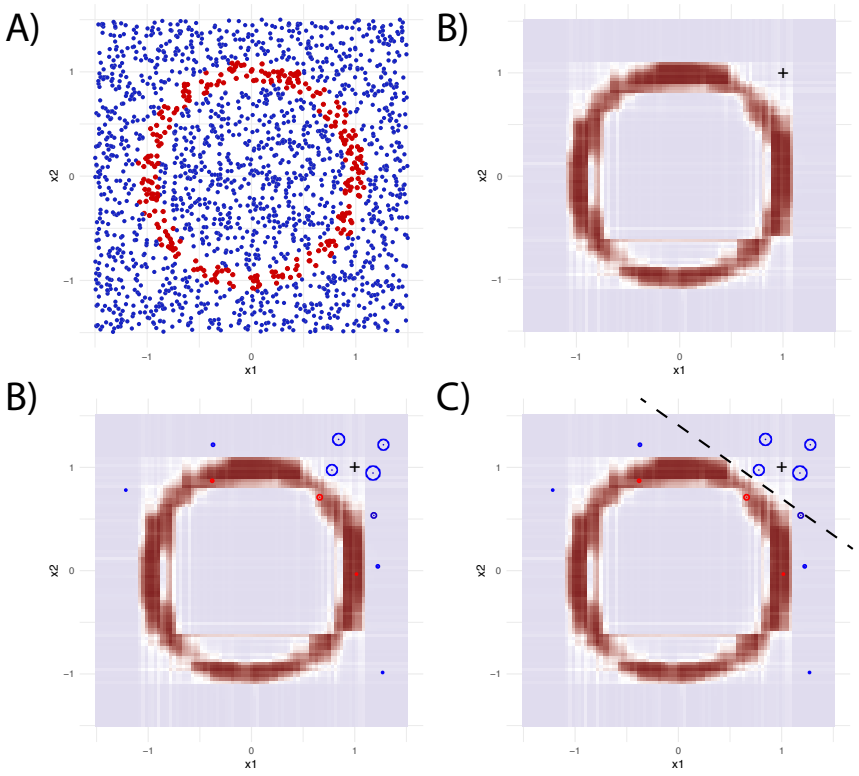


FIGURE 9 (fig:LIME1) A schematic idea behind local model approximations. Panel A shows training data, colors correspond to classess. Panel B showhs results fom the Random Forest model, whis is where the algorithm starts. Panel C shows new data sampled around the point of interest. Their color correspond to model response. Panel D shows fitted linear model that approximated the random forest model around point of interest

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.

0.8.3 Example: Hire or Fire?

0.8.4 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.

0.8.5 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 0.18.1 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: 27.4%
## Confusion matrix:
##          fired   ok promoted class.error
## fired     2274  380      201   0.2035026
## ok         539 1235      447   0.4439442
## promoted   199  384     2188   0.2103934
```

```
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 0.806 0.192    0.002
## attr(,"class")
## [1] "matrix" "votes"
```

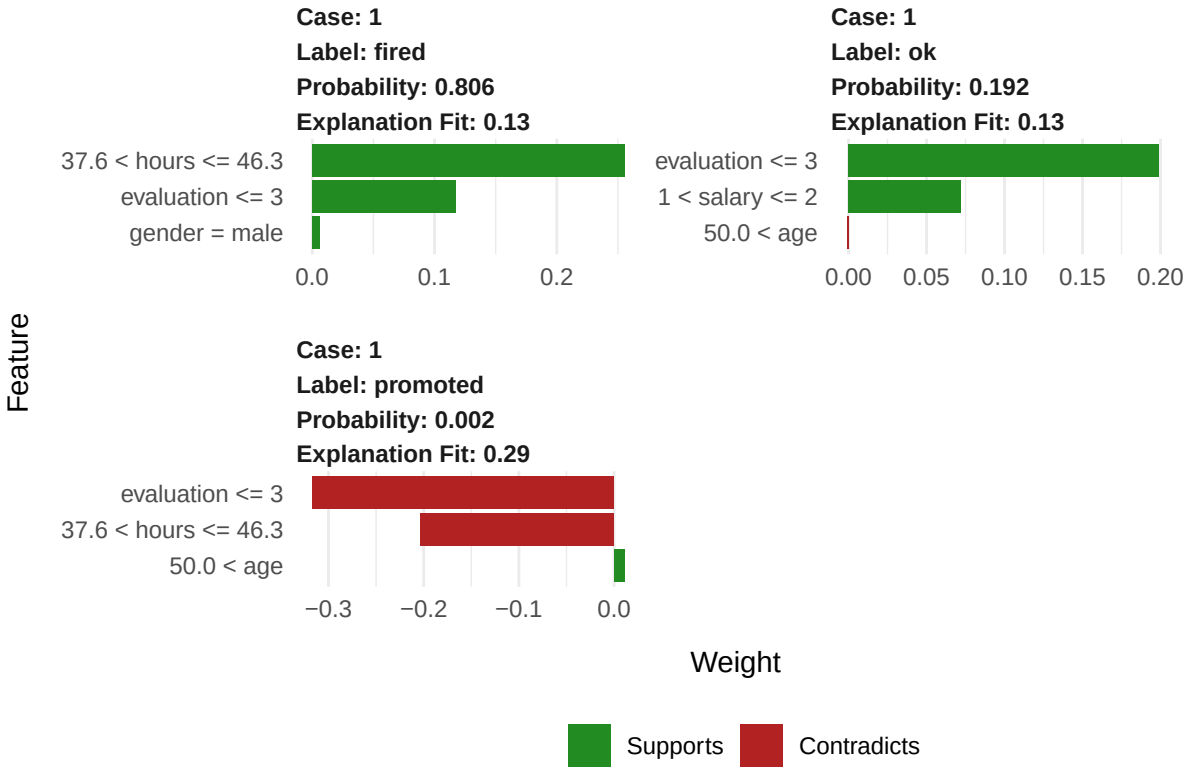
0.8.5.1 The lime pacakge

```
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      0.806 0.1300379       0.2535829
## 2 classification    1    fired      0.806 0.1300379       0.2535829
## 3 classification    1    fired      0.806 0.1300379       0.2535829
## 4 classification    1       ok      0.192 0.1251273       0.1892168
## 5 classification    1       ok      0.192 0.1251273       0.1892168
## 6 classification    1       ok      0.192 0.1251273       0.1892168
## 7 classification    1 promoted      0.002 0.2917896       0.5234322
## 8 classification    1 promoted      0.002 0.2917896       0.5234322
## 9 classification    1 promoted      0.002 0.2917896       0.5234322
##   model_prediction    feature feature_value feature_weight
## 1       0.63245342     gender           2.0    0.005909490
## 2       0.63245342      hours          42.3    0.255709364
## 3       0.63245342 evaluation           2.0    0.117251637
## 4       0.45914034        age          57.7   -0.001159746
```

```
## 5       0.45914034 evaluation           2.0    0.199215613
## 6       0.45914034     salary           2.0    0.071867712
## 7       0.01471303        age          57.7    0.011368221
## 8       0.01471303 evaluation           2.0   -0.317045218
## 9       0.01471303      hours          42.3   -0.203042163
##           feature_desc                        data          prediction
## 1        gender = male 2.0, 57.7, 42.3, 2.0, 2.0 0.806, 0.192, 0.002
## 2 37.6 < hours <= 46.3 2.0, 57.7, 42.3, 2.0, 2.0 0.806, 0.192, 0.002
## 3      evaluation <= 3 2.0, 57.7, 42.3, 2.0, 2.0 0.806, 0.192, 0.002
## 4           50.0 < age 2.0, 57.7, 42.3, 2.0, 2.0 0.806, 0.192, 0.002
## 5      evaluation <= 3 2.0, 57.7, 42.3, 2.0, 2.0 0.806, 0.192, 0.002
## 6      1 < salary <= 2 2.0, 57.7, 42.3, 2.0, 2.0 0.806, 0.192, 0.002
## 7           50.0 < age 2.0, 57.7, 42.3, 2.0, 2.0 0.806, 0.192, 0.002
## 8      evaluation <= 3 2.0, 57.7, 42.3, 2.0, 2.0 0.806, 0.192, 0.002
## 9 37.6 < hours <= 46.3 2.0, 57.7, 42.3, 2.0, 2.0 0.806, 0.192, 0.002
```

```
plot_features(explanations)
```



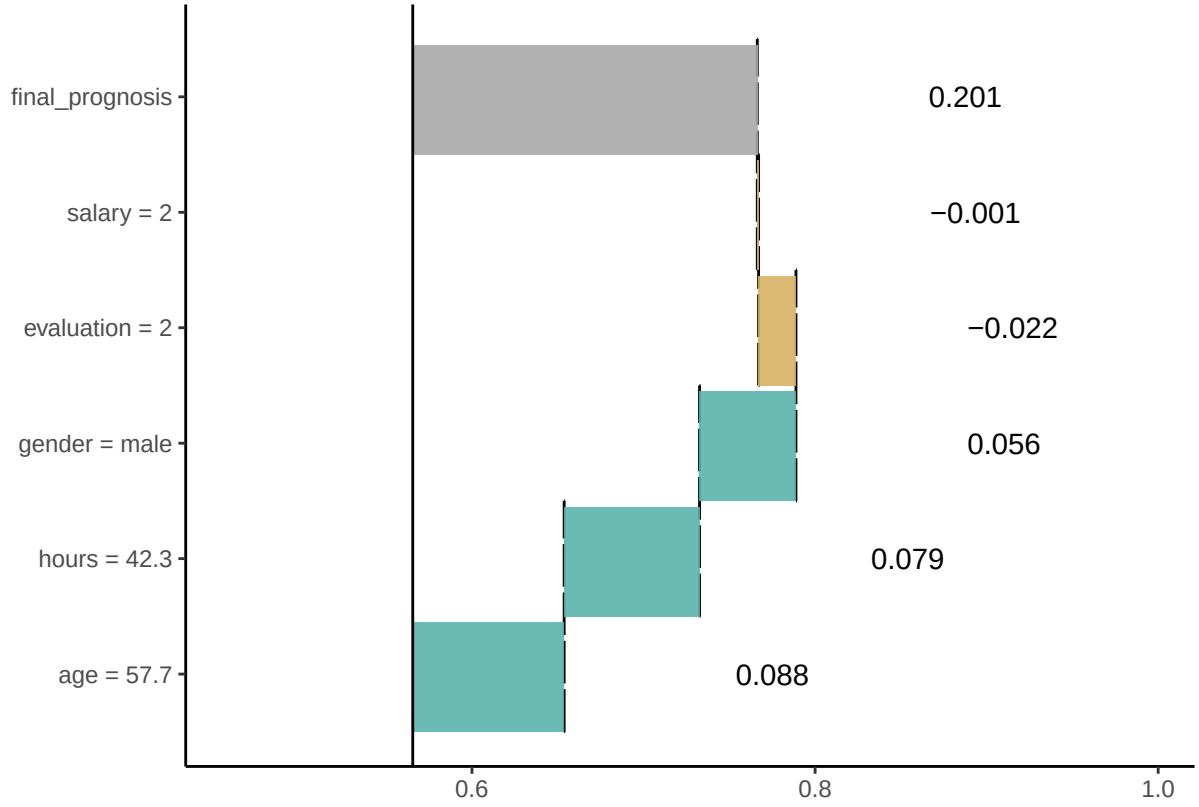
0.8.5.2 The live package

```
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: 0.7339
```

```
plot(local_model)
```




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

Variable		N	Estimate	p
gender	male	446		Reference
	female	54		−0.52 (−0.56, −0.48) <0.001
age		500		0.02 (0.02, 0.03) <0.001
hours		500		−0.05 (−0.06, −0.05) <0.001
evaluation		500		0.09 (0.03, 0.15) 0.002
salary		500		0.00 (−0.02, 0.03) 0.700
(Intercept)				1.45 (1.10, 1.79) <0.001

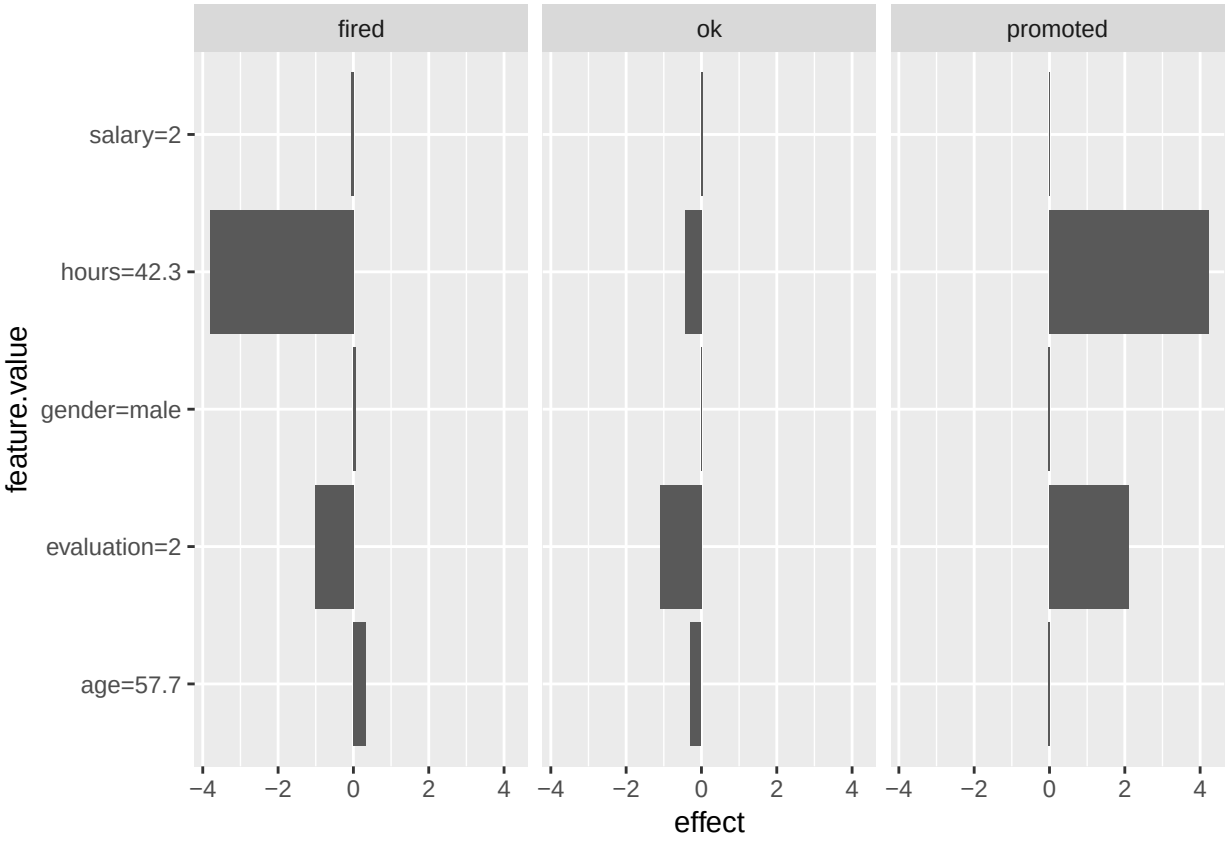
0.8.5.3 The iml package

```
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  0.070451023       1.0  0.07045102       male gender=male   gender=male
## 2  0.005839379      57.7  0.33693215       57.7         age      age=57.7
## 3 -0.090103309      42.3 -3.81136995       42.3       hours    hours=42.3
## 4 -0.508252954       2.0 -1.01650591          2  evaluation  evaluation=2
## 5 -0.032191570       2.0 -0.06438314          2      salary      salary=2
## 6 -0.018323731       1.0 -0.01832373       male gender=male   gender=male
```

```
##   .class
## 1  fired
## 2  fired
## 3  fired
## 4  fired
## 5  fired
## 6     ok
```

```
plot(white_box)
```



0.9 What-If analysis with the Ceteris Paribus Principle

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

Presented explainers are linked with the second law introduced in Section 0.1.2, i.e. law for prediction's speculations. This is why these explainers are also known as *What-If model analysis* or *Individual Conditional EXpectations* (Goldstein et al., 2015). It turns out that it is easier to understand how blacx-box model is working if we can play with it by changing variable by variable.

0.9.1 Introduction

Ceteris paribus is a Latin phrase meaning "other things held constant" or "all else unchanged". Using this principle we examine input variable per variable separatly, asumming that effects of all other variables are unchanged. See Figure 10

Similar to the LIME method introduced in the section 0.8, Ceteris Paribus profiles examine curvature of a model response function. The difference between these two methods that LIME approximates the model curvature with a simpler white-box model that is easier to present. Usually the LIME model is sparse, thus our attention may be limited to smaller number of dimensions. In contrary, the CP plots show conditional model response for every variable. In the last subsection we discuss pros and cons of this approach.

0.9.2 Intuition

0.9.3 Method

0.9.3.1 1D profiles

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

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

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

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

A natural way to visualise CP profiles is to use a profile plot as in Figure 11.

Figure 11 shows an example of Ceteris Paribus profile. The black dot stands for prediction for a single observation. Grey line show how the model response would change if in this single observation coordinate `hours` will be changed to selected value. One thing that we can read is that the model response is not smooth and there is some variability along the profile. Second thing is that for this particular observation the model response would drop significantly if the variable `hours` will be higher than 45.

Since in the example dataset we are struggling with model for three classes, one can plot CP profiles for each class in the same panel. See an example in the Figure 12.

Usually model input consist many variables, then it is beneficial to show more variables at the same time. The easiest way to do so is to plot consecutive variables on separate panels. See an example in Figure 13.

0.9.3.2 Profile oscillations

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

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

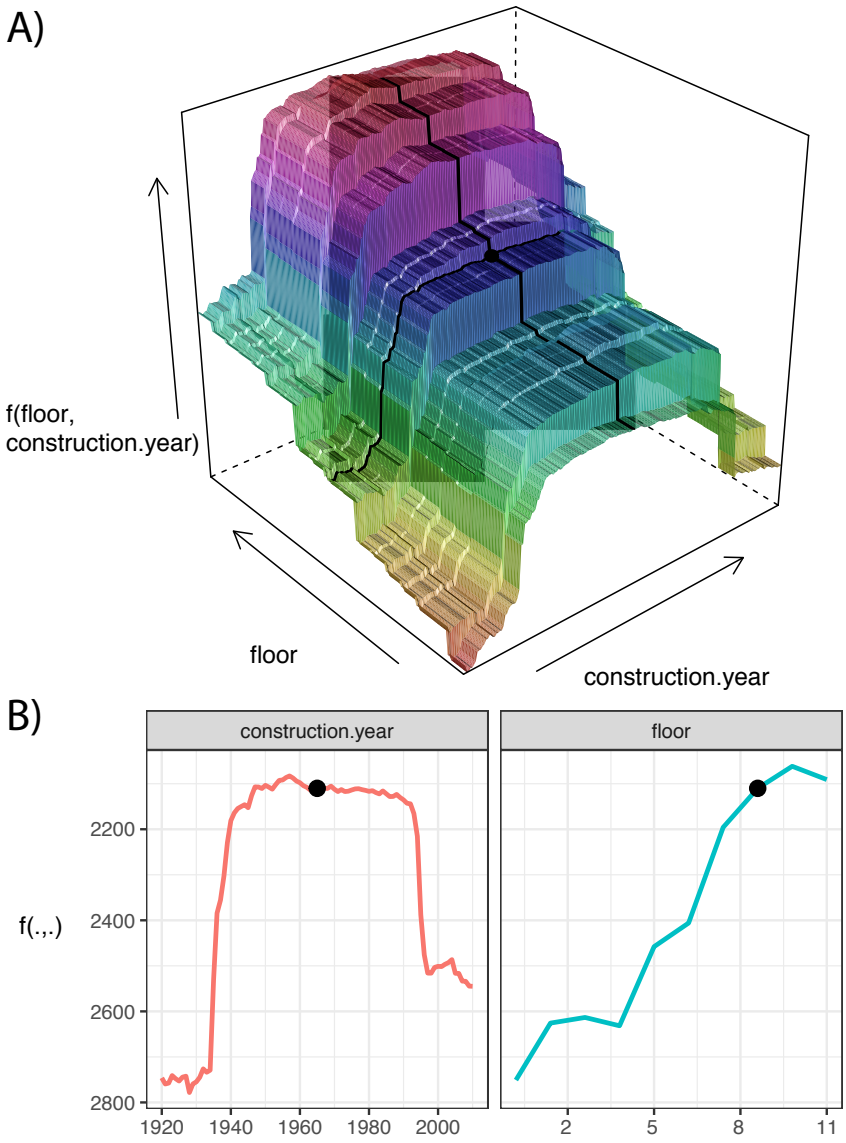


FIGURE 10 (fig:modelResponseCurveLine) A) Model response surface. Ceteris Paribus profiles marked with black curves helps to understand the curvature of the model response by updating only a single variable. B) CP profiles are individual conditional model responses

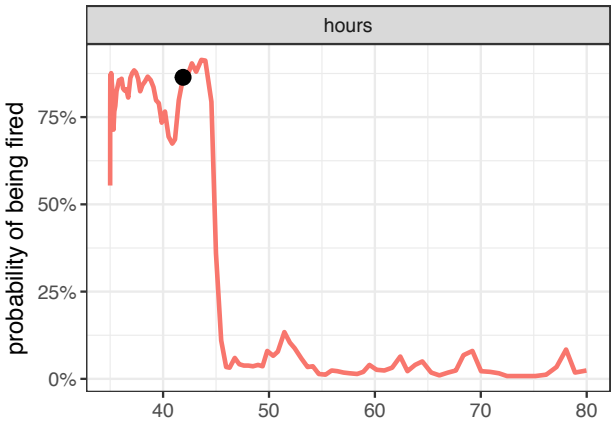


FIGURE 11 (fig:HRCPHiredHours) Ceteris Paribus profile for Random Forest model that assess the probability of being fired in call center as a function of average number of working hours

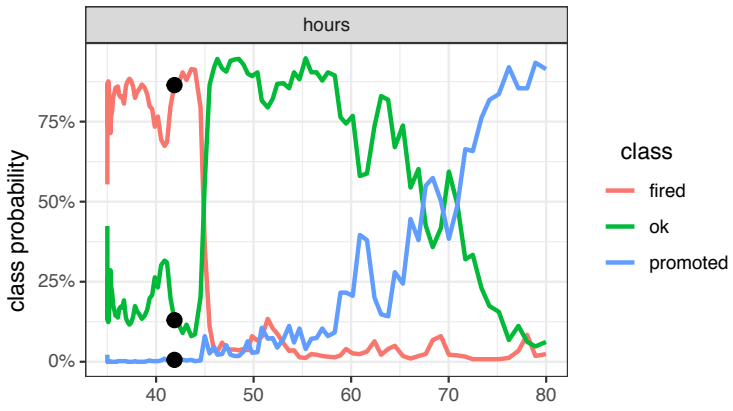


FIGURE 12 (fig:HRCPAllHours) Ceteris Paribus profiles for three classess predicted by the Random Forest model as a function of average number of working hours

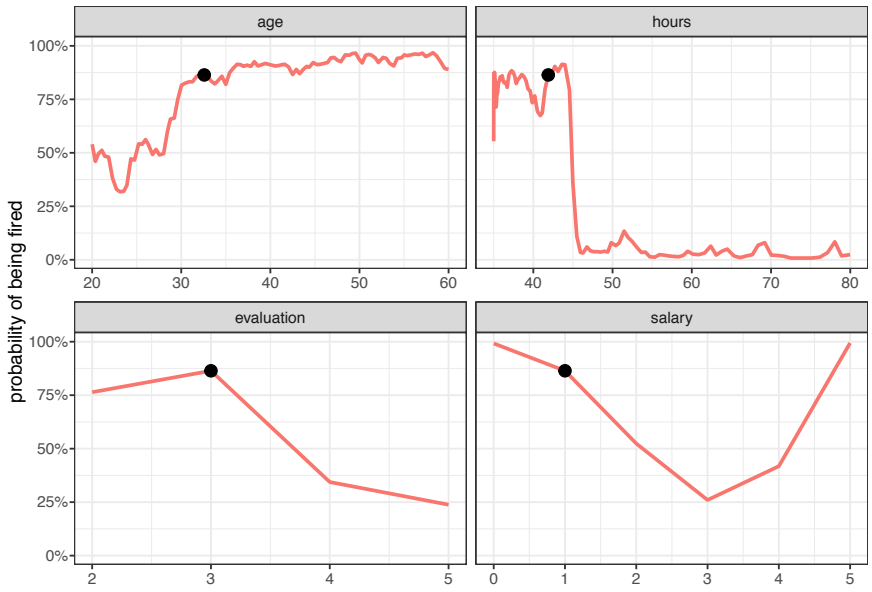


FIGURE 13 (fig:HRCPFiredAll) Ceteris Paribus profiles for all continuous variables

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

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

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

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

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

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

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

Figure 14 shows the idea behind measuring oscillations. The larger the highlighted area the more important is the variable.

Figure 15 summarizes variable oscillations. Such visuals help to quickly grasp how large are model oscillations around a specific point.

NOTE

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

For example, consider a model

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

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

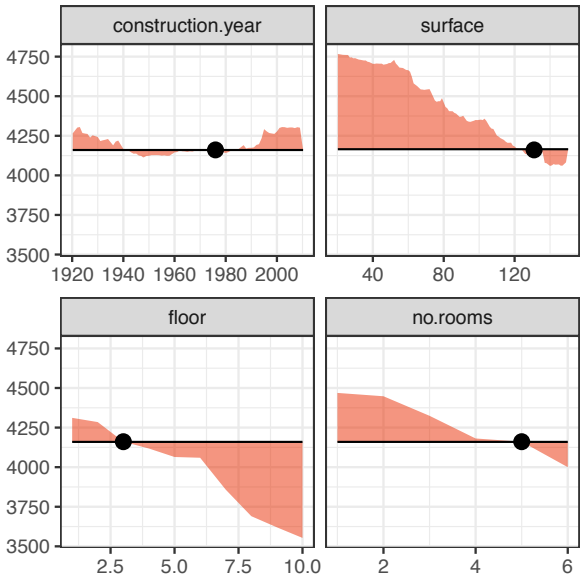


FIGURE 14 (fig:CPVIPprofiles) CP oscillations are average deviations between CP profiles and the model response

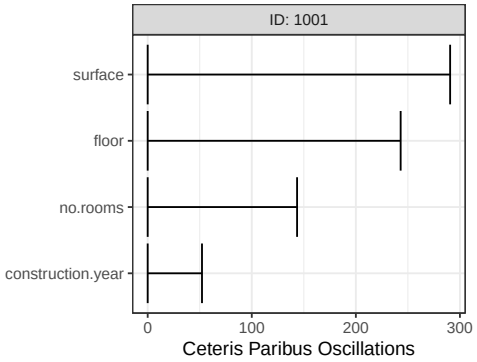


FIGURE 15 (fig:CPVIP1) Variable importance plots calculated for Ceteris Paribus profiles for observation ID: 1001

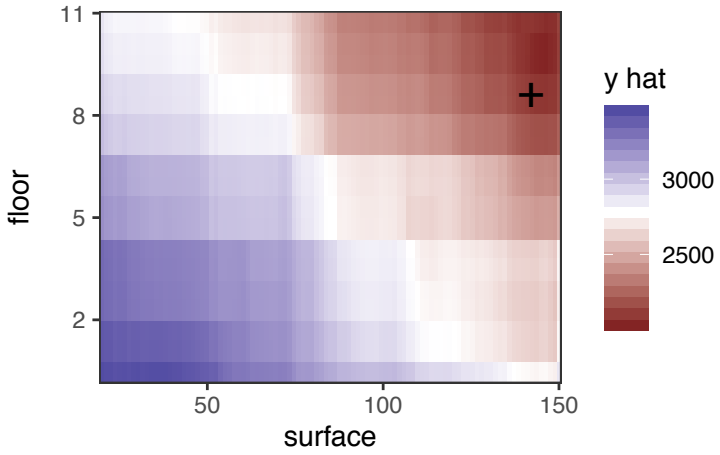


FIGURE 16 (fig:CP2Dsurflor) Ceteris Paribus plot for a pair of variales. Black cross marks coordinated for the observation of interest. Presented model estimates price of an appartment

From the global perspective both variables are equally important.

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

0.9.3.3 2D profiles

The definition of ceteris paribus profiles given in section 0.9.3.1 may be easily extended to two and more variables. Also definition of CP oscillations 0.9.3.2 have straight forward generalization for larger number of dimensions. Such generalisations are usefull when model is non additive. Presence of pairwise interactions may be detected with 2D Ceteris Paribus plots.

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

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

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

A natural way to visualise 2D CP profiles is to use a level plot as in Figure 16.

If number of variables is small or moderate thein it is possible to present all pairs of variables. See an example in Figure 17.

0.9.4 Local model fidelity

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

How does it work?

The idea behind fidelity plots is to select some number of points from the validation dataset that are closes

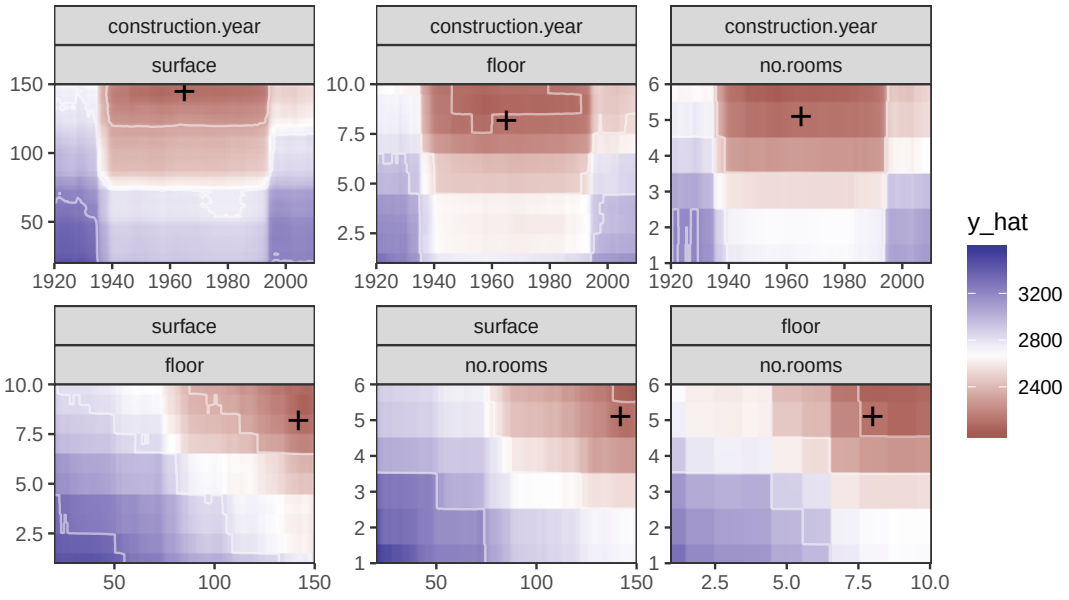


FIGURE 17 (fig:CP2Dall) Ceteris Paribus plot for all pairs of variales.

to the point of interest. It's a similar approach as in k nearest neighbours. Then for these neighbours we may plot Ceteris Paribus Profiles and check how stable they are.

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

An example fidelity plot is presented in Figure 18. Black line shows the CP profiles for the point of interest, while grey lines show CP profiles for neihgbors. Red intervals stand for residuals and in this example it looks like residuals for neighbours are all negative. Thus maybe model is biased around the point of interest.

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

See Figure 19 for an example. Here residuals for neighbors are shifted towards highest values. This suggests that the model response is biased around the observation of interest.

TODO: diagnostic score: average quantaile of neighbours.

0.9.5 Example

0.9.6 Pros and cons

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

Pros

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

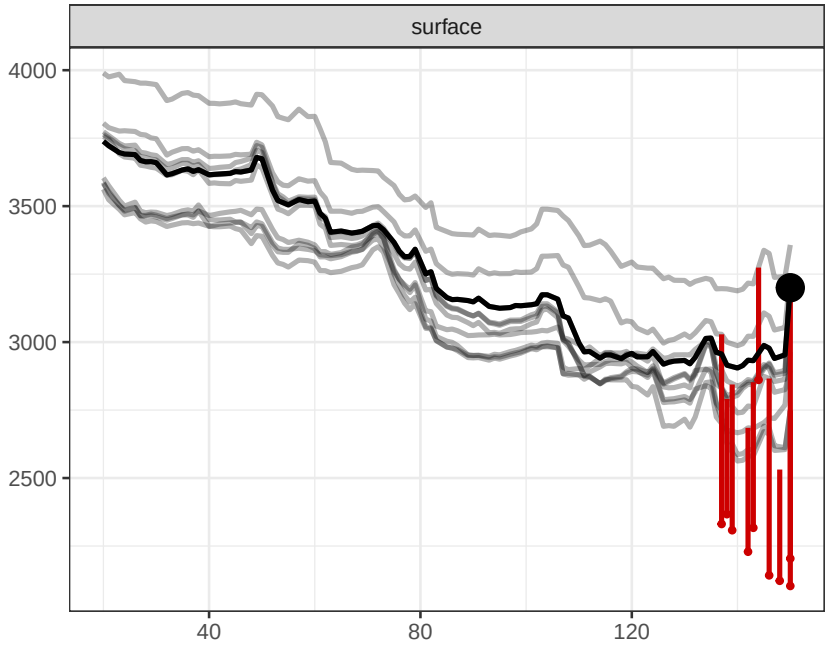


FIGURE 18 (fig:CPfidelity1) Local fidelity plots. Black line shows the CP profile for the point of interest. Grey lines show CP profiles for nearest neighbors. Red intervals correspond to residuals. Each red interval starts in a model prediction for a selected neighbor and ends in its true value of target variable.

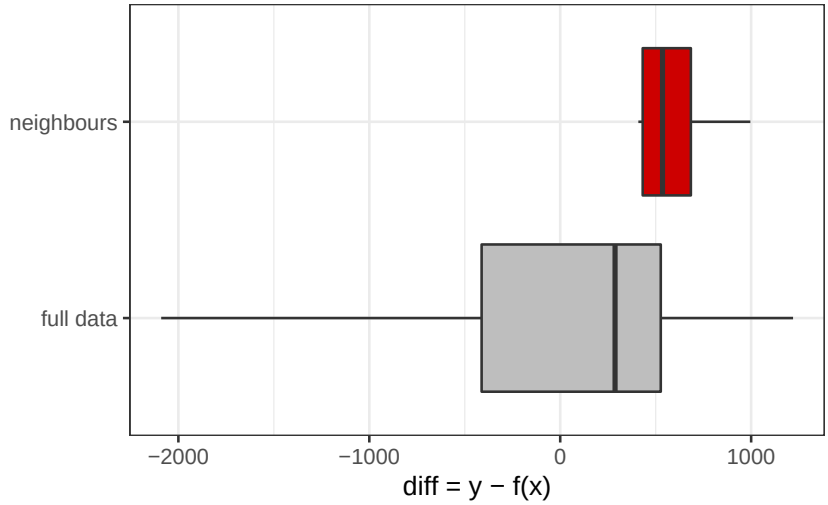


FIGURE 19 (fig:CPfidelityBoxplot) Distribution of residuals for whole validation data (grey boxplot) and for selected closes 15 neighbors (red boxplot).

Cons

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

0.9.7 Code snippets for R

In this section we present key features of the `ceterisParibus` package for R (Biecek, 2018b). This package covers all features presented in this chapter. It is available on CRAN and GitHub. Find more examples at the website of this package `https://pbiecek.github.io/ceterisParibus/`.

A very interesting tool for moedl explorartion with similar principle is implemented in the `condvis` package (O'Connell et al., 2017).

Model preparation

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

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

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

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

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

```
##
## Call:
##  randomForest(formula = m2.price ~ construction.year + surface +     floor + no.rooms, data = apartments)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 1
##
##           Mean of squared residuals: 485330.8
##                     % Var explained: 40.9
```

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

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

Since all other functions work in a model agnostic fashion, first we need to define a wrapper around the model. Here we are using the `explain()` function from `DALEX` package (Biecek, 2018c).

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

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

2. Define point of interest.

Certeris Paribus profiles explore model around a single point.

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

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

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

```
##        1
## 2341.367
```

3. Calculate CP profiles

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

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

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

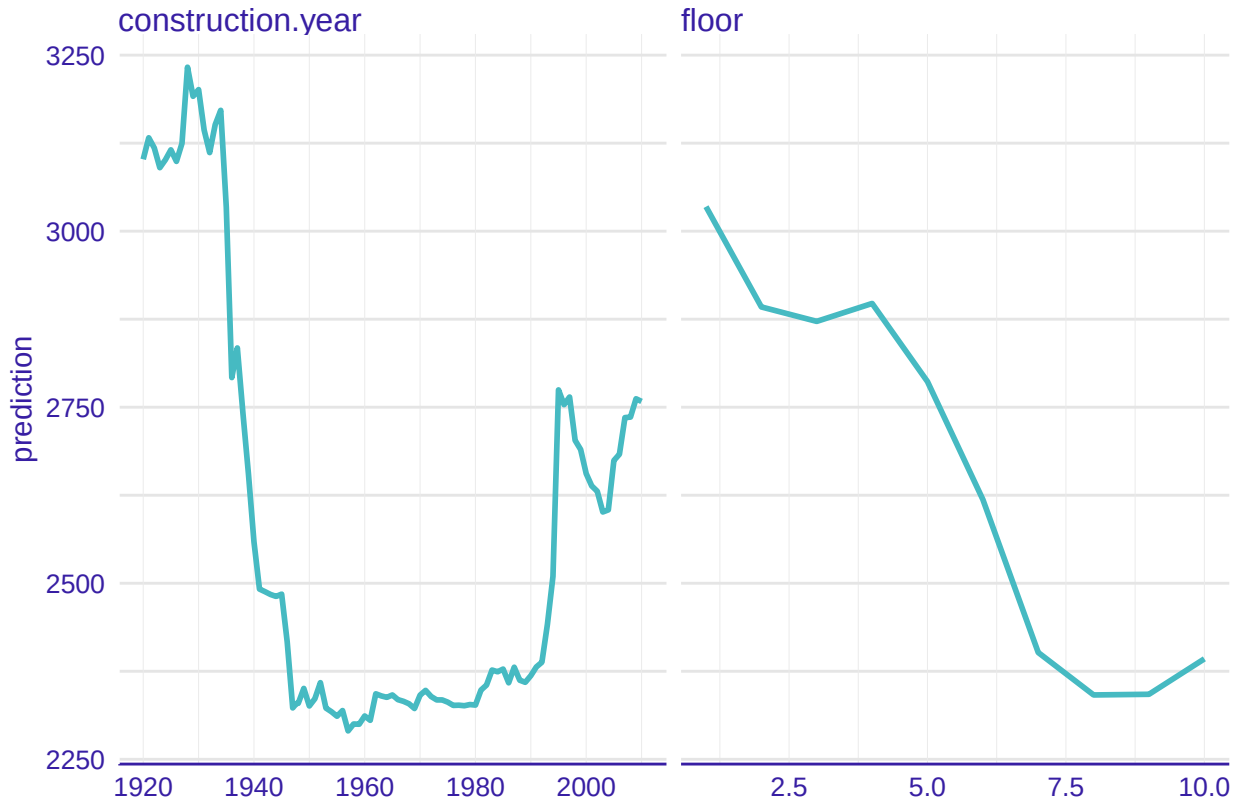
```
## Top profiles    :
##     construction.year no.rooms surface floor   _yhat_           _vname_
## 1                1920        5     142     8 3102.043 construction.year
## 1.1              1921        5     142     8 3132.637 construction.year
## 1.2              1922        5     142     8 3118.195 construction.year
## 1.3              1923        5     142     8 3090.137 construction.year
## 1.4              1924        5     142     8 3101.309 construction.year
## 1.5              1925        5     142     8 3115.526 construction.year
##     _ids_      _label_
## 1       1 randomForest
## 1.1     1 randomForest
## 1.2     1 randomForest
## 1.3     1 randomForest
## 1.4     1 randomForest
## 1.5     1 randomForest
##
##
## Top observations:
```

```
##   construction.year no.rooms surface floor   _yhat_      _label_ _ids_
## 1              1965        5     142     8 2341.367 randomForest     1
```

4. Plot CP profiles.

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

```
plot(cp_rf)
```



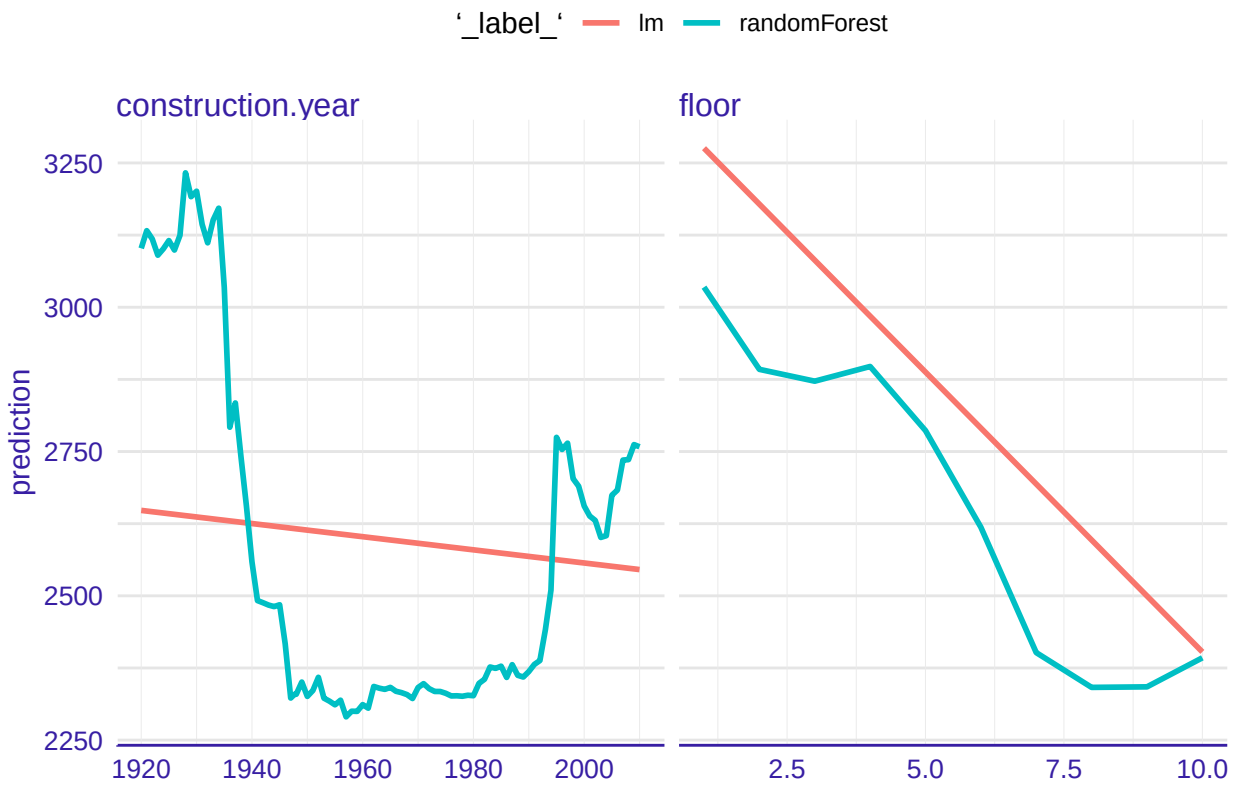
One of very useful features of `ceterisParibus` explainers is that profiles for two or more models may be superimposed in a single plot. This helps in model comparisons.

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

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

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

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



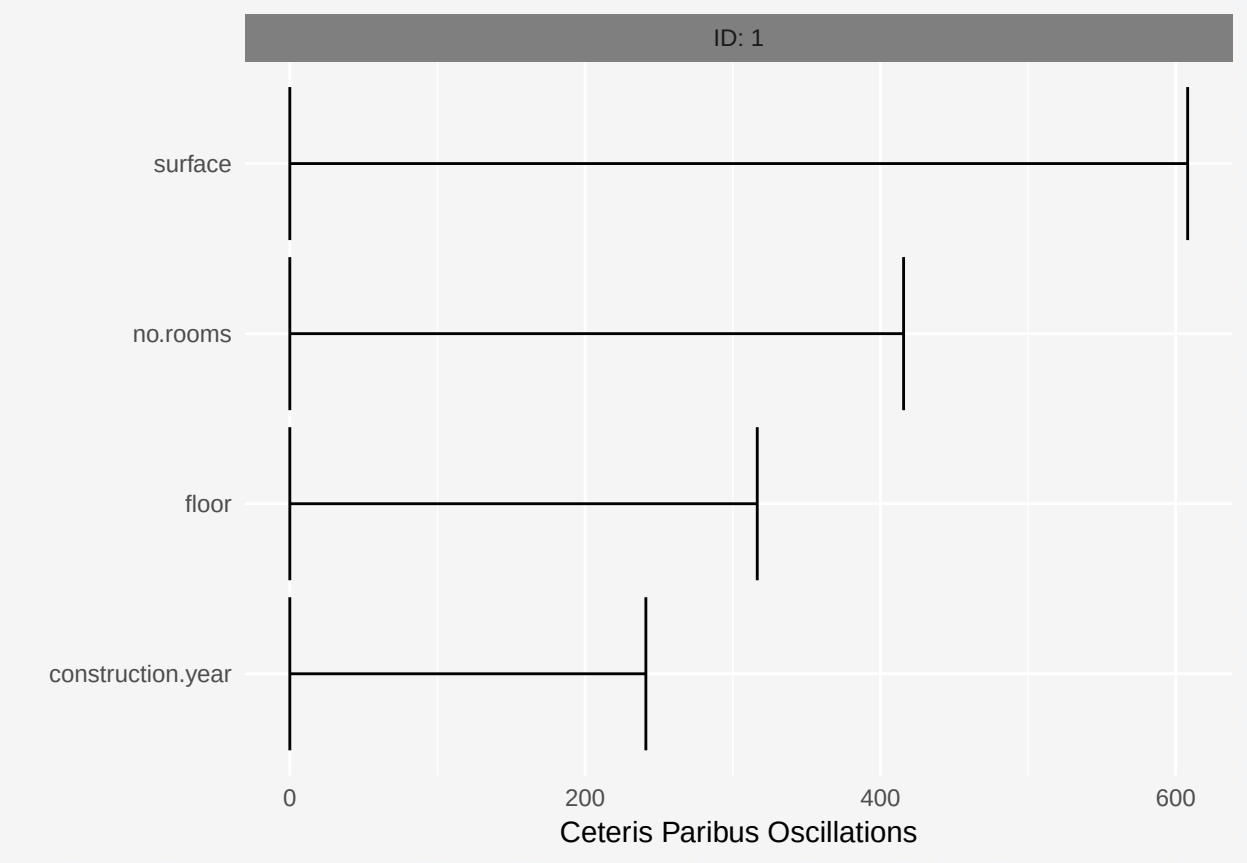
Oscillations

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

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

```
##              _vname_ _ids_ oscillations
## 3            surface     1     608.0858
## 2          no.rooms      1     415.7131
## 4              floor     1     316.5819
## 1 construction.year      1     241.1452
```

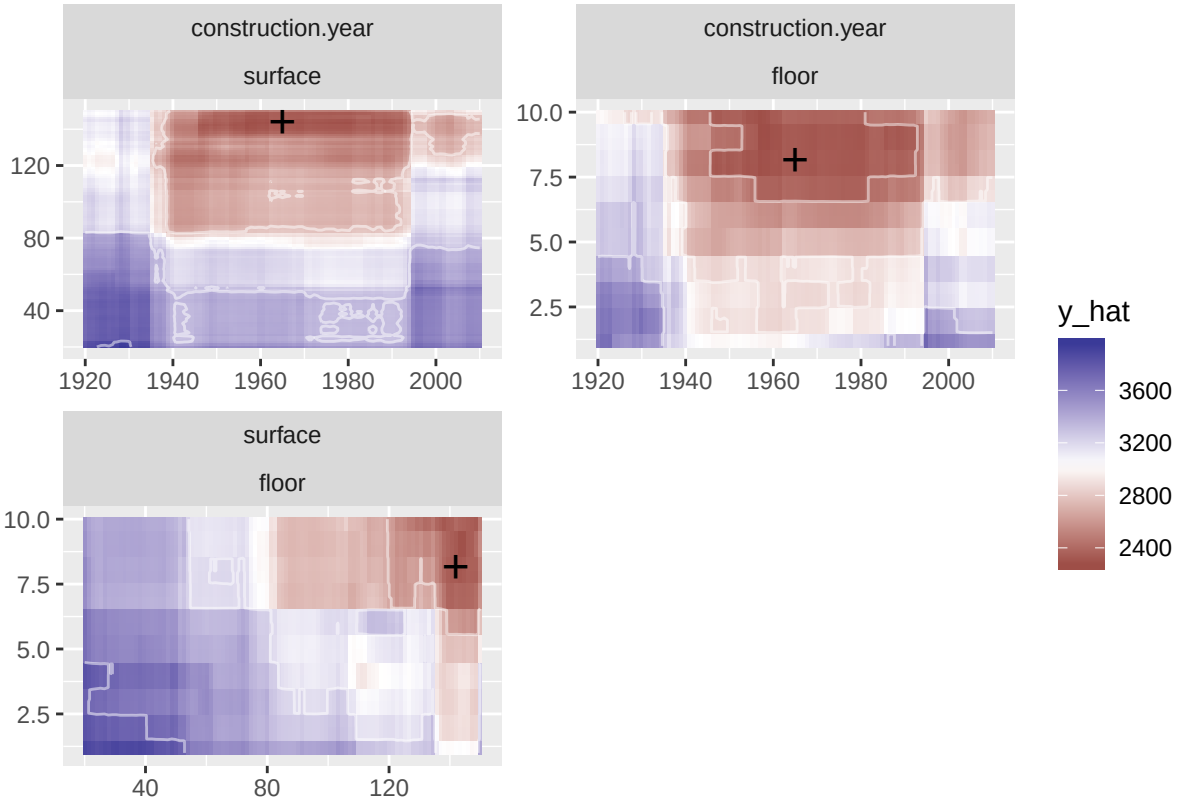
```
plot(co_rf_all)
```



2D Ceteris Paribus profiles

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

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



0.10 Comparision of prediction level explainers

TODO

compare pros and cons of different techniques

0.10.1 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

0.11 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).

0.11.1 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?

0.11.2 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.

0.12 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.

0.12.1 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.

0.12.2 Example: Titanic

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

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_small)
```

```
##   Survived Pclass    Sex Age SibSp Parch    Fare Embarked
## 1        0      3   male  22     1     0  7.2500        S
## 2        1      1 female  38     1     0 71.2833        C
## 3        1      3 female  26     0     0  7.9250        S
## 4        1      1 female  35     1     0 53.1000        S
## 5        0      3   male  35     0     0  8.0500        S
## 7        0      1   male  54     0     0 51.8625        S
```

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

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

```
library("ingredients")
fi_rf <- feature_importance(explainer_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(explainer_rf), simplify = FALSE)
do.call(plot, fi_rf10) + ggtitle("Permutation based feature importance", "For Random Forest model and Tita
```

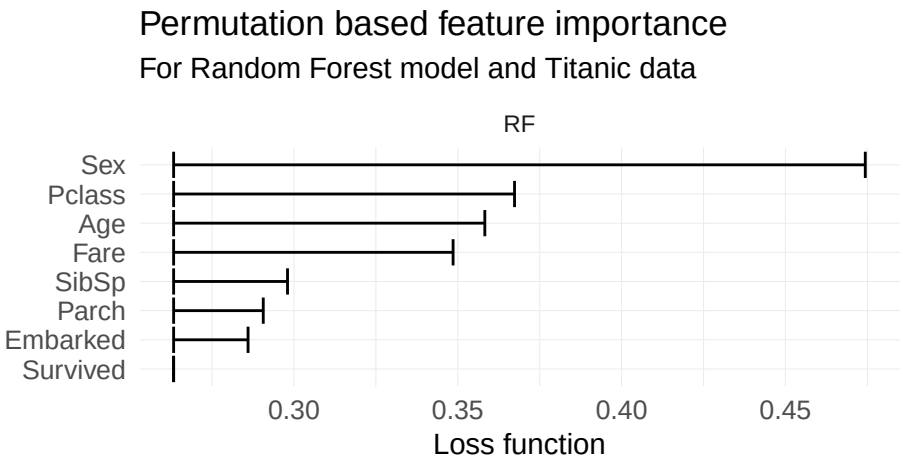


FIGURE 20 Feature importance. Each interval presents the difference between original model performance (left end) and the performance on a dataset with a single feature perturbed

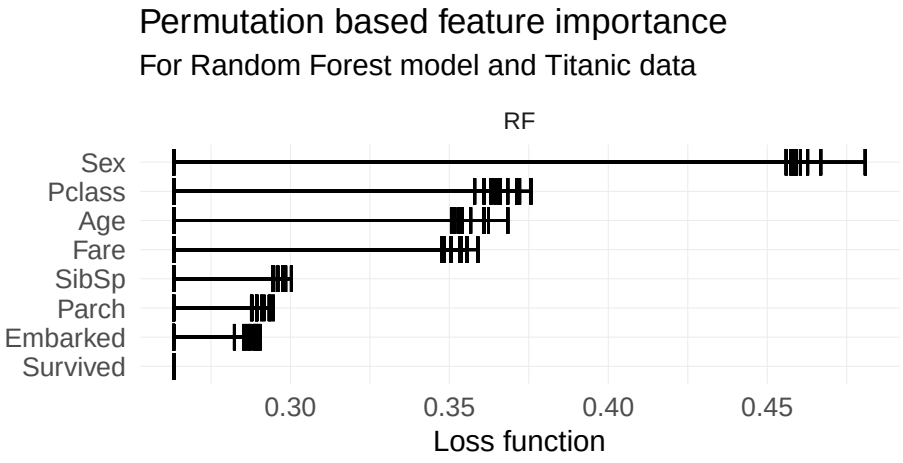


FIGURE 21 Feature importance for 10 replication of feature importance assessment

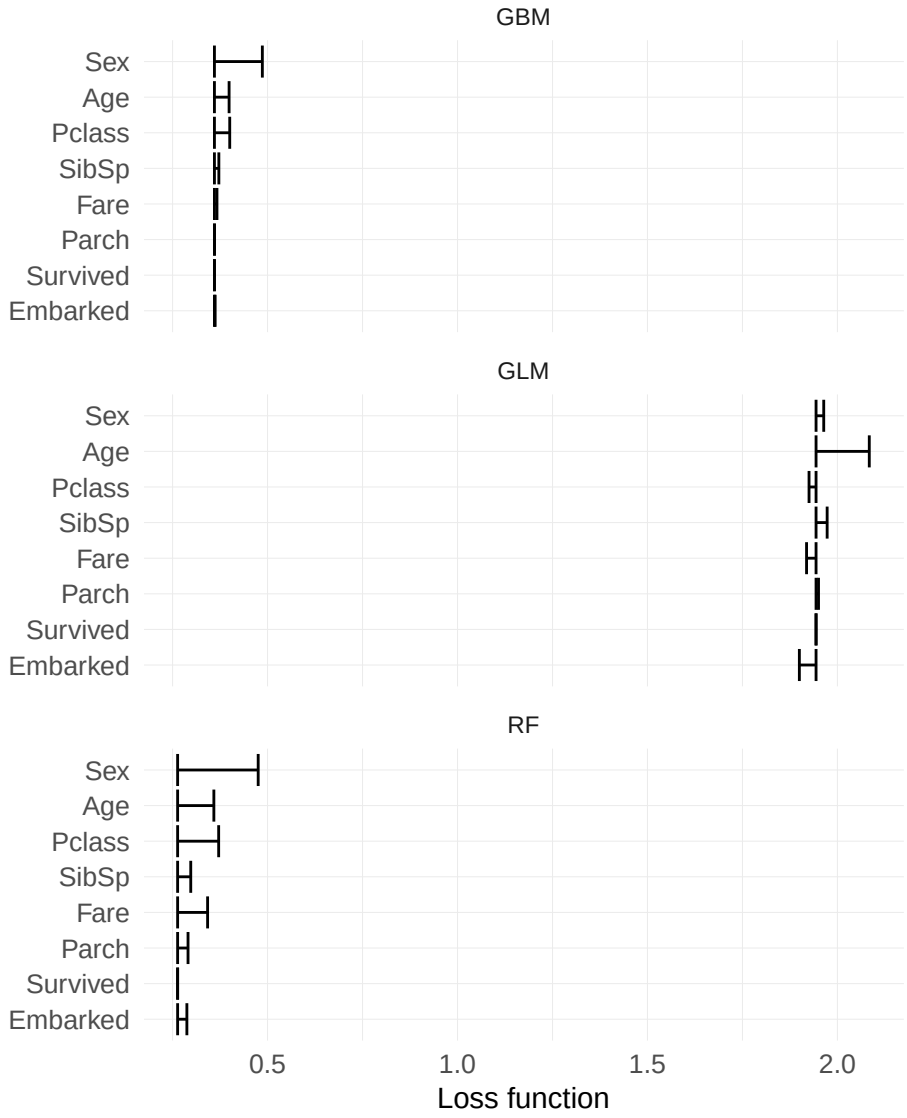


FIGURE 22 Feature importance for random forest, gradient boosting and logistic regression models

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.

```
fi_rf <- feature_importance(explainer_rf)
fi_gbm <- feature_importance(explainer_gbm)
fi_glm <- feature_importance(explainer_glm)

plot(fi_rf, fi_gbm, fi_glm)
```

0.12.3 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
```

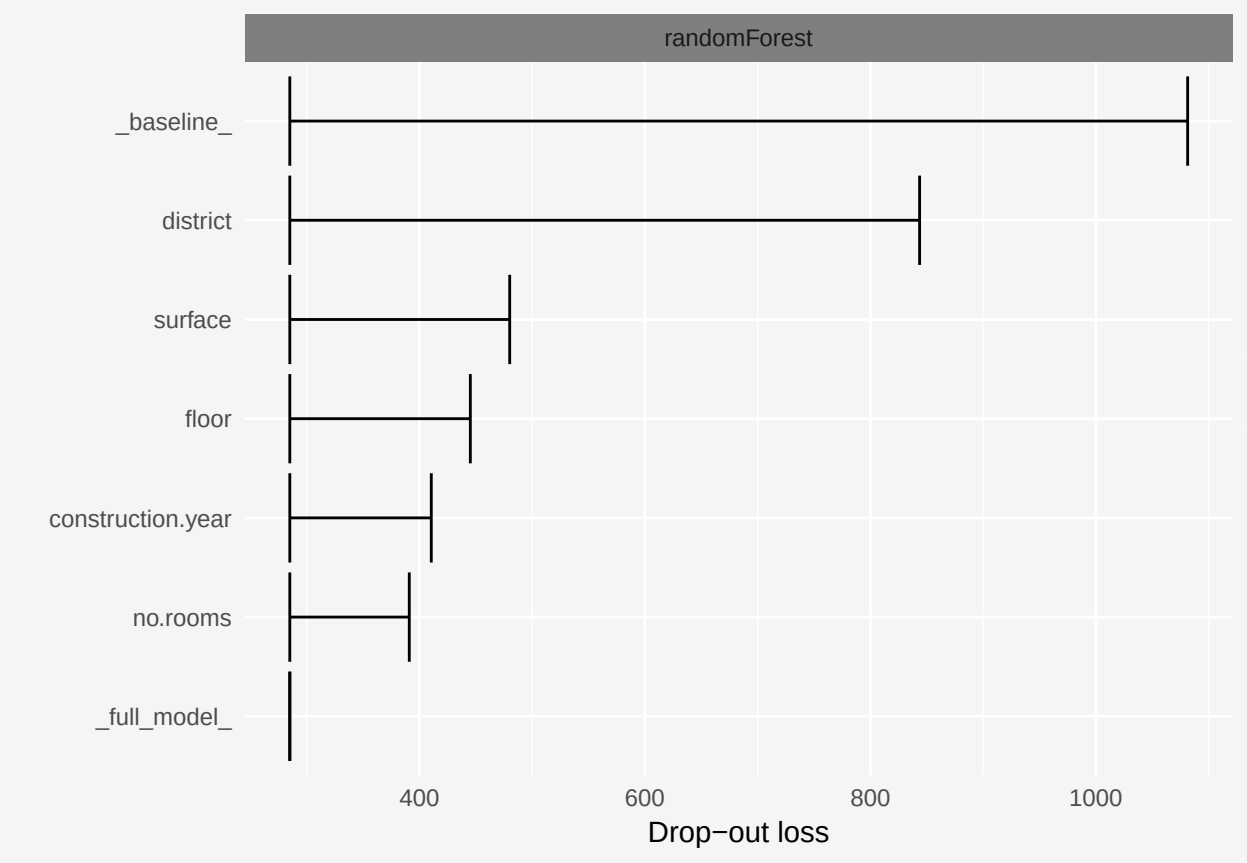
Let's calculate feature importance

```
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)
```



0.12.4 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
```

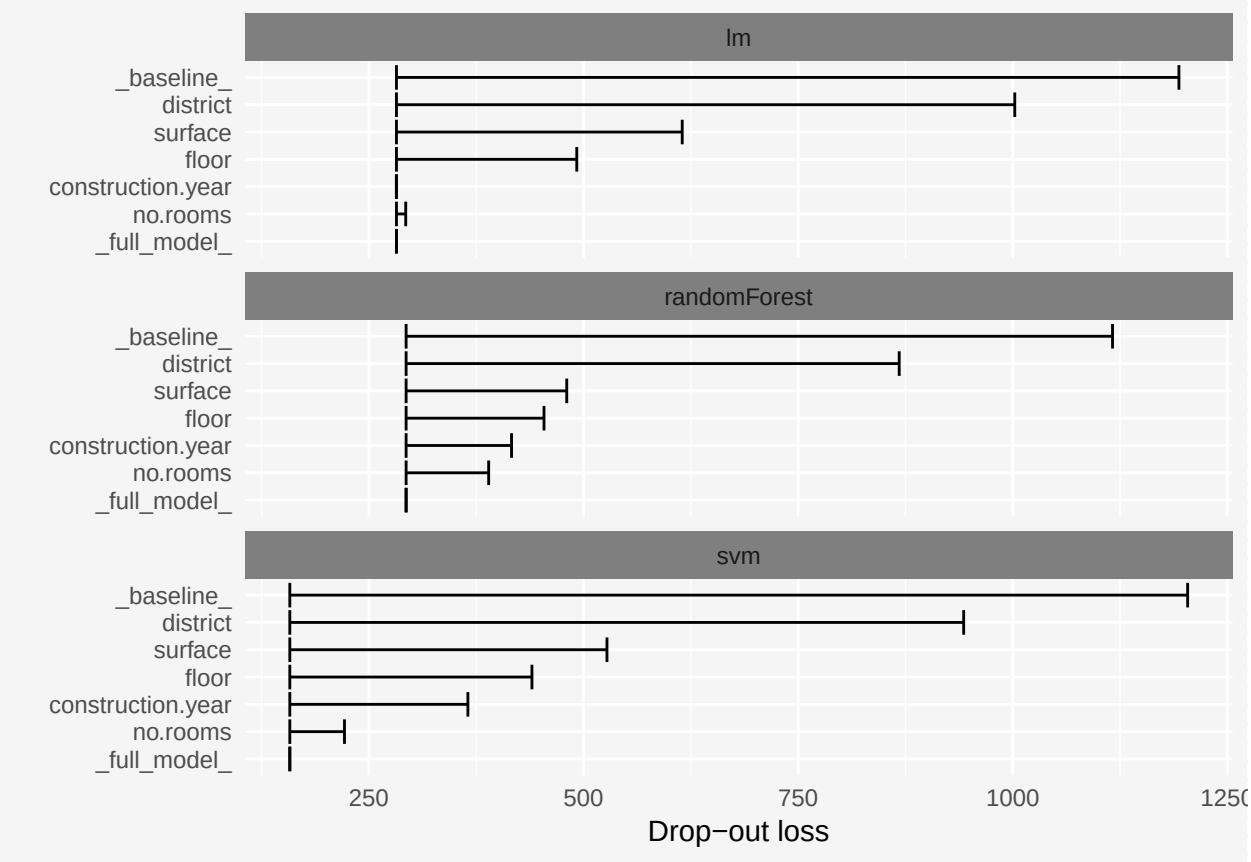
```
##            variable dropout_loss label
## 1      _full_model_     157.7938   svm
## 2          no.rooms     221.4595   svm
## 3 construction.year     365.2600   svm
## 4             floor     439.8724   svm
## 5           surface     527.2598   svm
## 6          district     942.8512   svm
## 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.

0.12.5 Level frequency

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

0.13 Feature effects

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. Surrogate training is a procedure in which 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.
- Understanding how the model utilize 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 assumptions may be testes
- Understanding of a link between target and the feature may increase our domain knowledge.
- Comparison of feature effects between different models may help us to understand how different models handle particular features.

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

0.13.1 Partial Dependency Plots

One of the first and the most popular tools for model understanding are Partial Dependence Plots (sometimes named Partial Dependence Profiles) (Friedman, 2000).

PDP was introduced by Friedman in 2000 in the paper devoted to Gradient Boosting Machines (GBM) - new type of complex yet effective models. For many years PDP as sleeping beauties stay in the shadow of the boosting. It has changed in recent years.

General idea is to show how the expected model response behaves as a function of a selected feature. Here the word „expected" means averaged over the population. We can think about them as about an average from Ceteris Paribus Profiles introduced in @ref{ceterisParibus}.

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

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.

```
library("ingredients")
selected_passangers <- select_sample(titanic_small, n = 100)
cp_rf <- ceteris_paribus(explainer_rf, selected_passangers)
```

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

In the most common formulation Partial Dependency Plots are expected values for CP profiles (see `pdp` package (Greenwell, 2017a)).

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

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

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

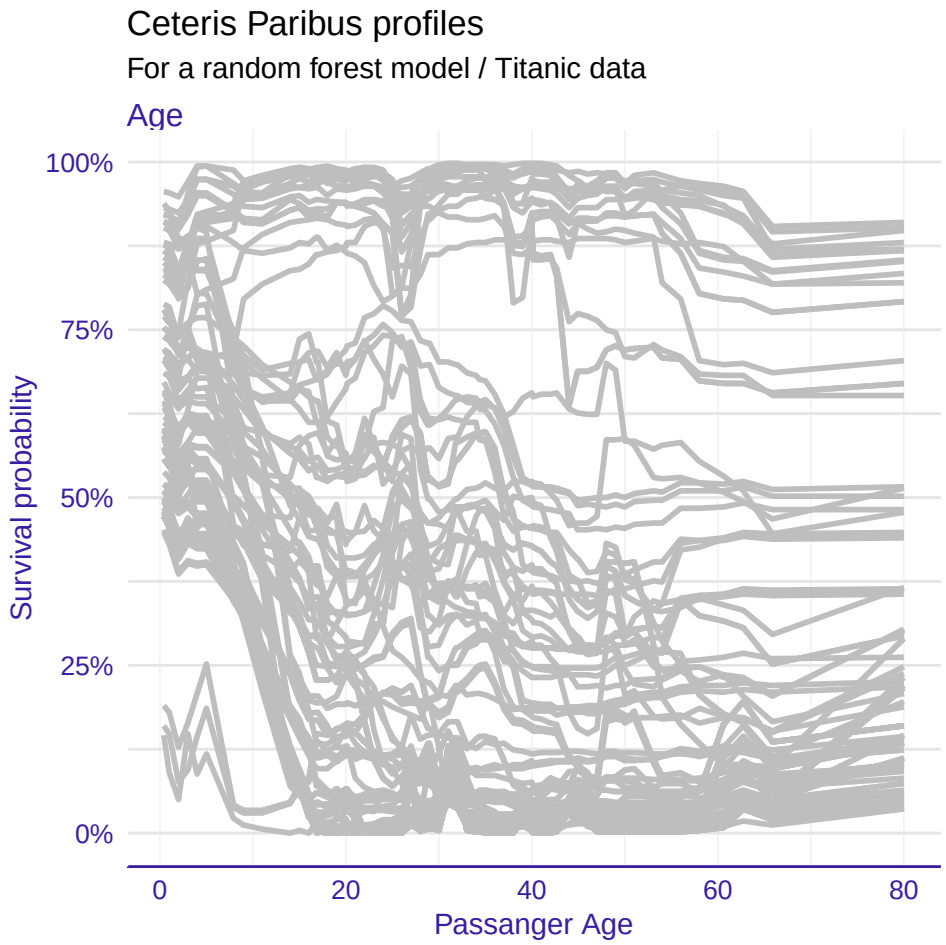


FIGURE 23 (#fig:pdp_part_1)Ceteris Paribus profiles for 100 observations, the Age variable and the random forest model

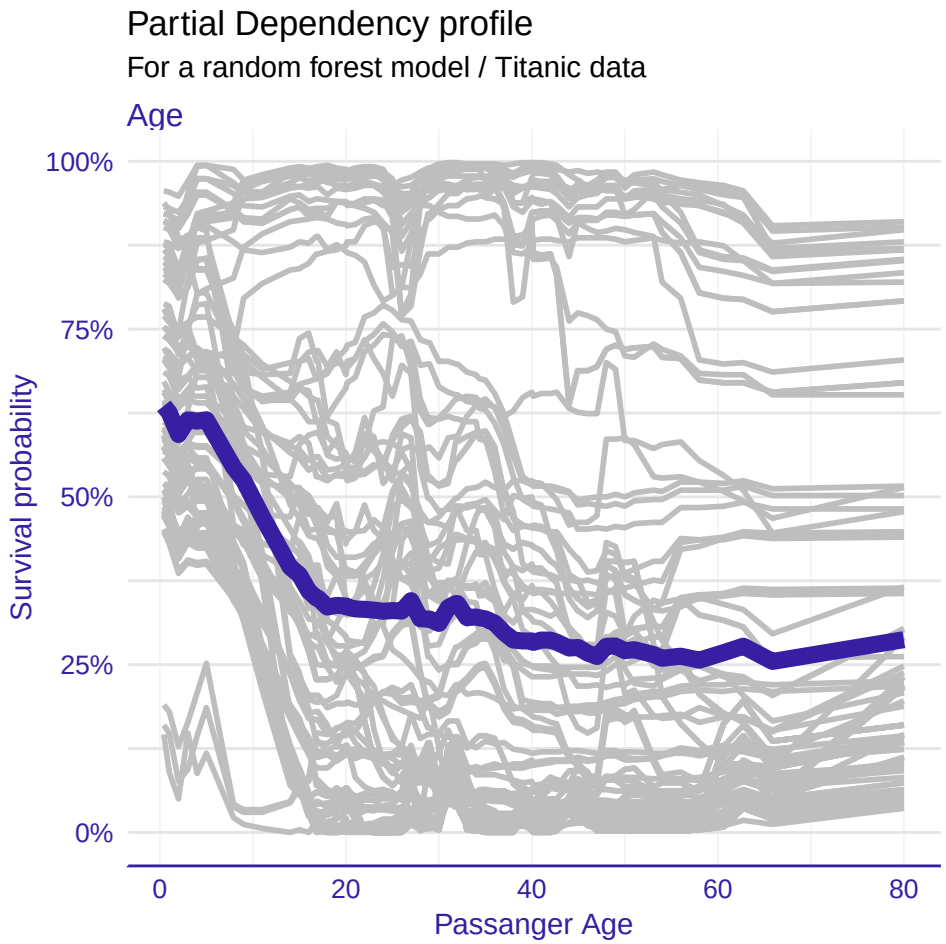


FIGURE 24 (#fig:pdp_part_2)Partial Dependency profile as an average for 100 observations

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

```
library("ingredients")
selected_passangers <- select_sample(titanic_small, n = 100)
cp_rf <- ceteris_paribus(explainer_rf, selected_passangers)
pdp_rf <- aggregate_profiles(cp_rf, selected_variables = "Age")
```

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.

0.13.1.1 Interactions and 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 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.

```
library("ingredients")
selected_passangers <- select_sample(titanic_small, n = 100)
cp_rf <- ceteris_paribus(explainer_rf, selected_passangers)
clust_rf <- cluster_profiles(cp_rf, k = 3, selected_variables = "Age")
```

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.

0.13.1.2 Groups of 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_small, n = 100)
cp_rf <- ceteris_paribus(explainer_rf, selected_passangers)
pdp_Sex_rf <- aggregate_profiles(cp_rf, selected_variables = "Age",
                groups = "Sex")
```

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.

0.13.1.3 Model comparisons with Partial Dependency Plots

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.

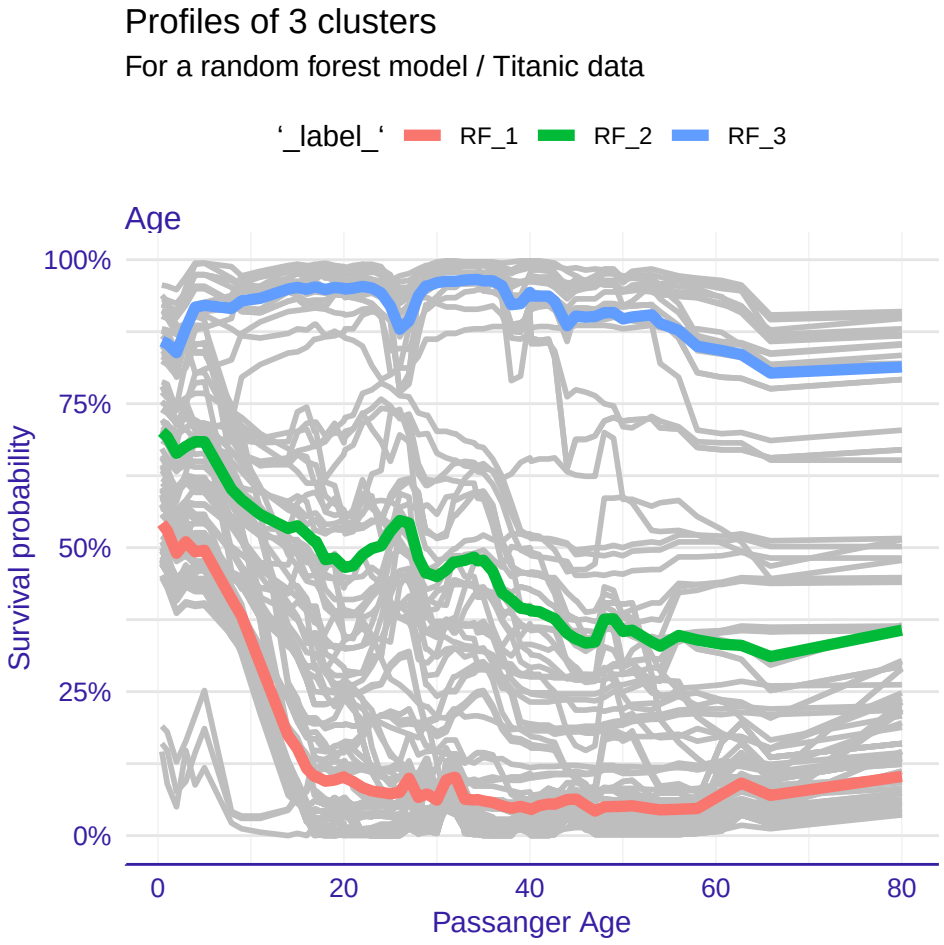


FIGURE 25 (#fig:pdp_part_4)Cluster profiles for 3 clusters over 100 Ceteris Paribus profiles

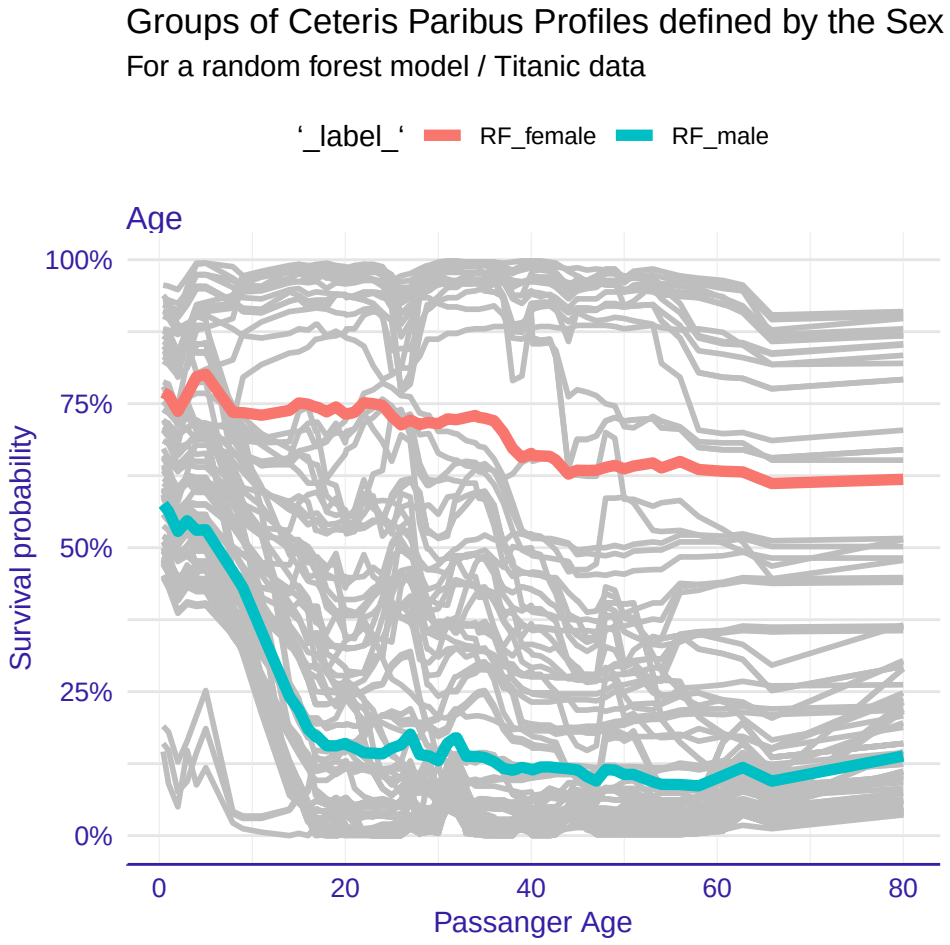


FIGURE 26 (#fig:pdp_part_5)Grouped profiles with respect to the Sex variable

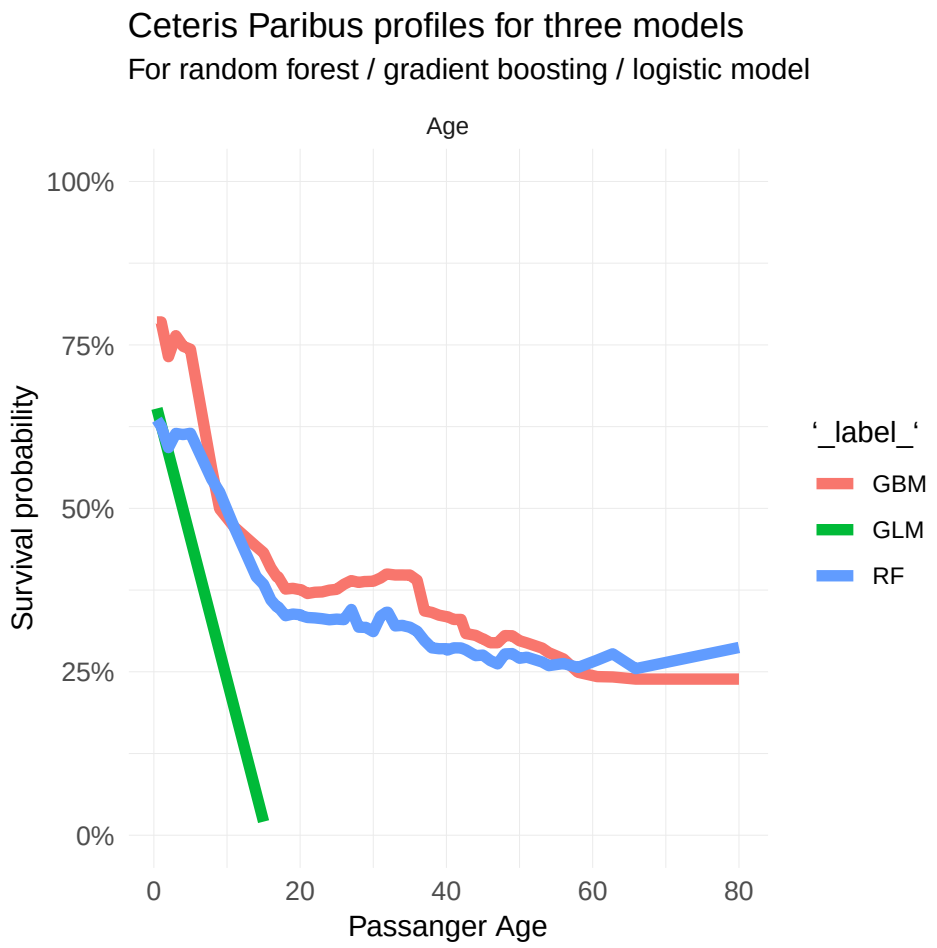


FIGURE 27 (#fig:pdp_part_7)Comparison on three predictive models with different structures.

- *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.

```
library("ingredients")
plot(pdp_rf, pdp_glm, pdp_gbm, selected_variables = "Age", color = "_label_")
```

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.

0.13.1.4 Correlation between features

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 assumptions of CPs. Profiles are created based on assumption that it makes sense to change variable x^i independently from all other variables x^{-i}.

But this may not have sense at all. 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 $x^{number.of.rooms}$.

There are several attempts to fix this problem. One of the most known are Accumulated Local Effects Plots (ALEPlots) (Apley, 2018) or Local Conditional Expectation Profiles (LCE) [TODO: referencja do pracy Rafala]?

0.13.2 Merging Path Plots

(Demšar and Bosnić, 2018)

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

(Sitko et al., 2018)

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

(Fisher et al., 2018)

Beware Default Random Forest Importances

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

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

(Sitko et al., 2018)

```
library(factorMerger)
```

0.14 Other topics

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

(Tatarynowicz et al., 2018)

0.15 Performance Diagnostic

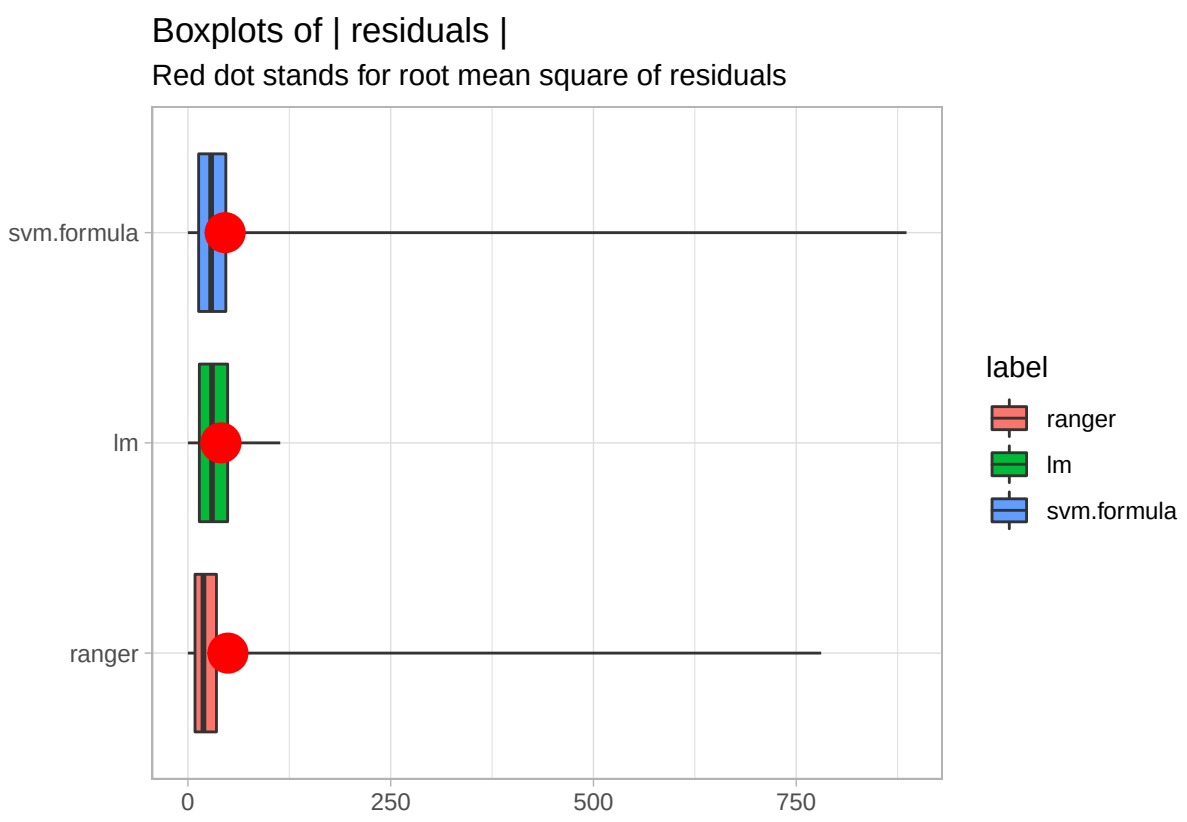
Goal: how good is the model, which is better

Model selection

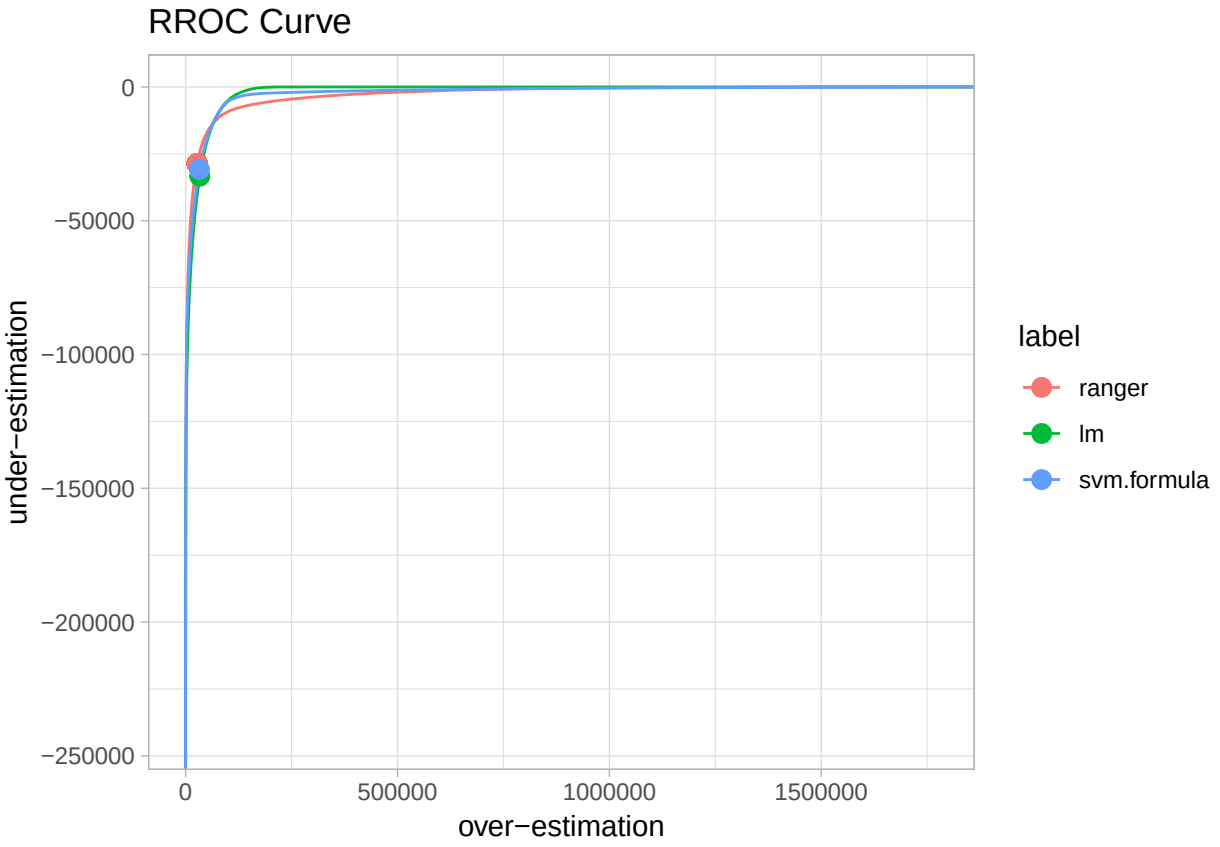
- ROC / RROC / LIFT

```
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)
```



0.16 Residual Diagnostic

Goal: verify if model is ok

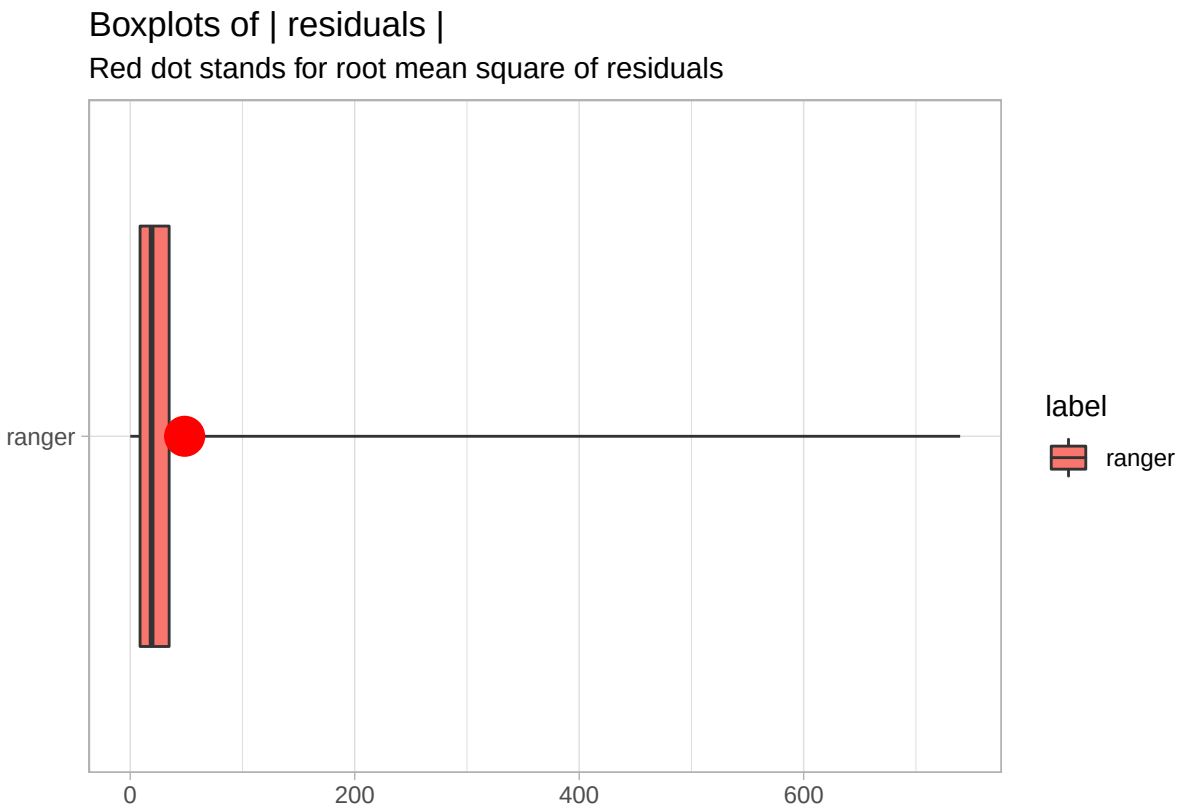
(Gosiewska and Biecek, 2018)

```
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)
```

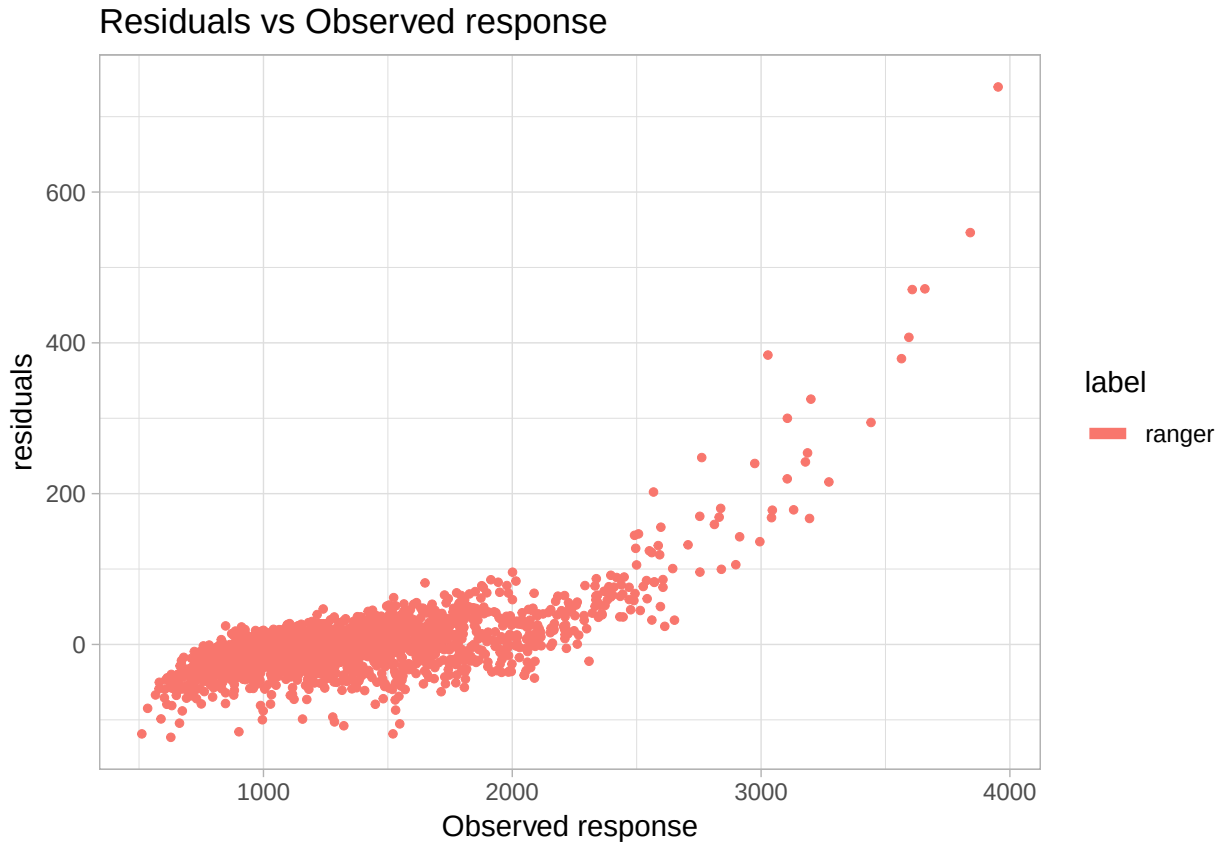
```
##   -----------------------------------------------
##    Checks for autocorrelation
##   -----------------------------------------------
##     Model name:  ranger
##     Autocorrelation in target:     +0.01
##     Autocorrelation in residuals:  +0.01
##   -----------------------------------------------
##    Checks for outliers
```

```
##   ---------------------------------------------
##     Model name:  ranger
##     Shift > 1:  20 ( 1 %)
##     Shift > 2:  14 ( 0.7 %)
##     Top lowest standardised residuals:
##      -2.5534 (1888), -2.4594 (920), -2.4588 (1388), -2.4019 (1508), -2.2377 (1829)
##     Top highest standardised residuals:
##      15.242 (1914), 11.256 (1745), 9.7175 (1532), 9.6979 (1111), 8.391 (1051)
##   ---------------------------------------------
##    Checks for trend in residuals
##   ---------------------------------------------
##     Model name:  ranger
##     Standardised loess fit:  +76.75     ***
```

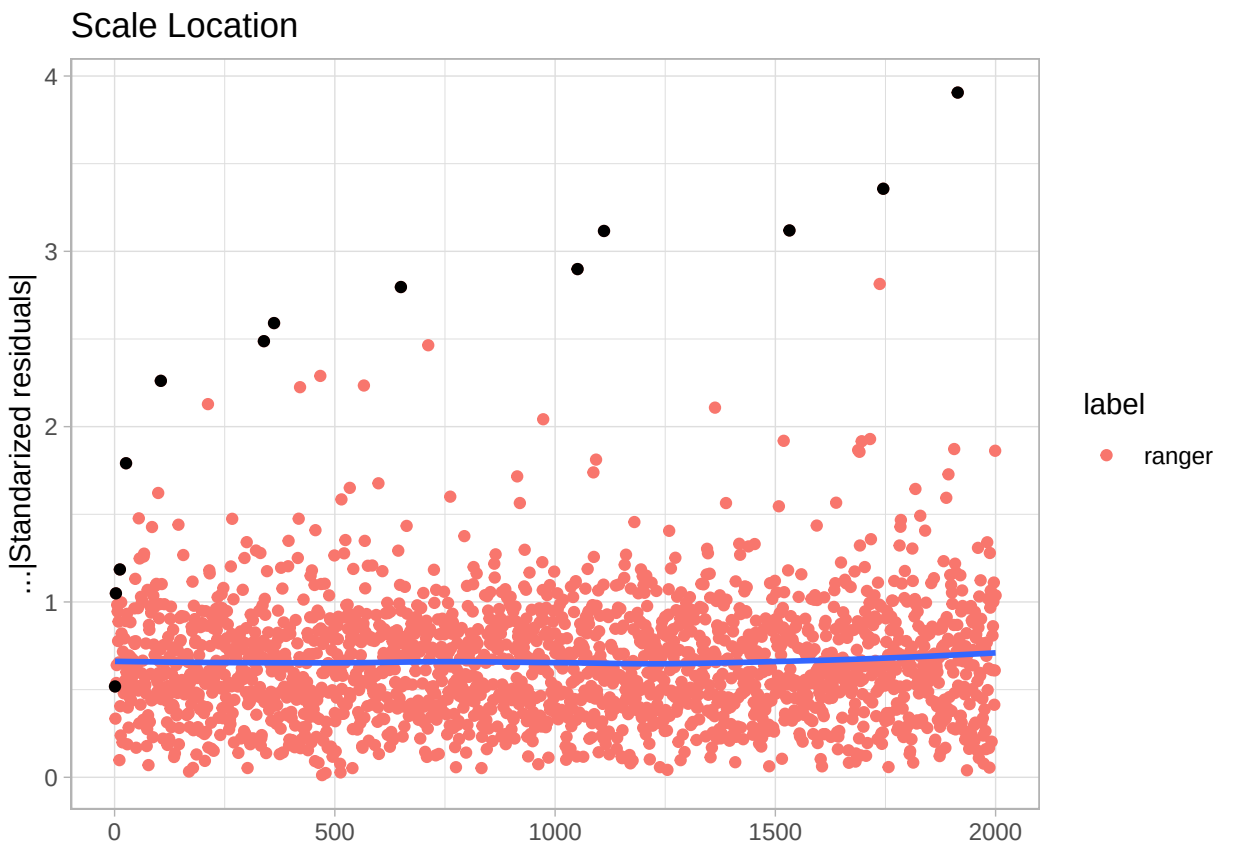
```
plotResidualBoxplot(rf_au)
```



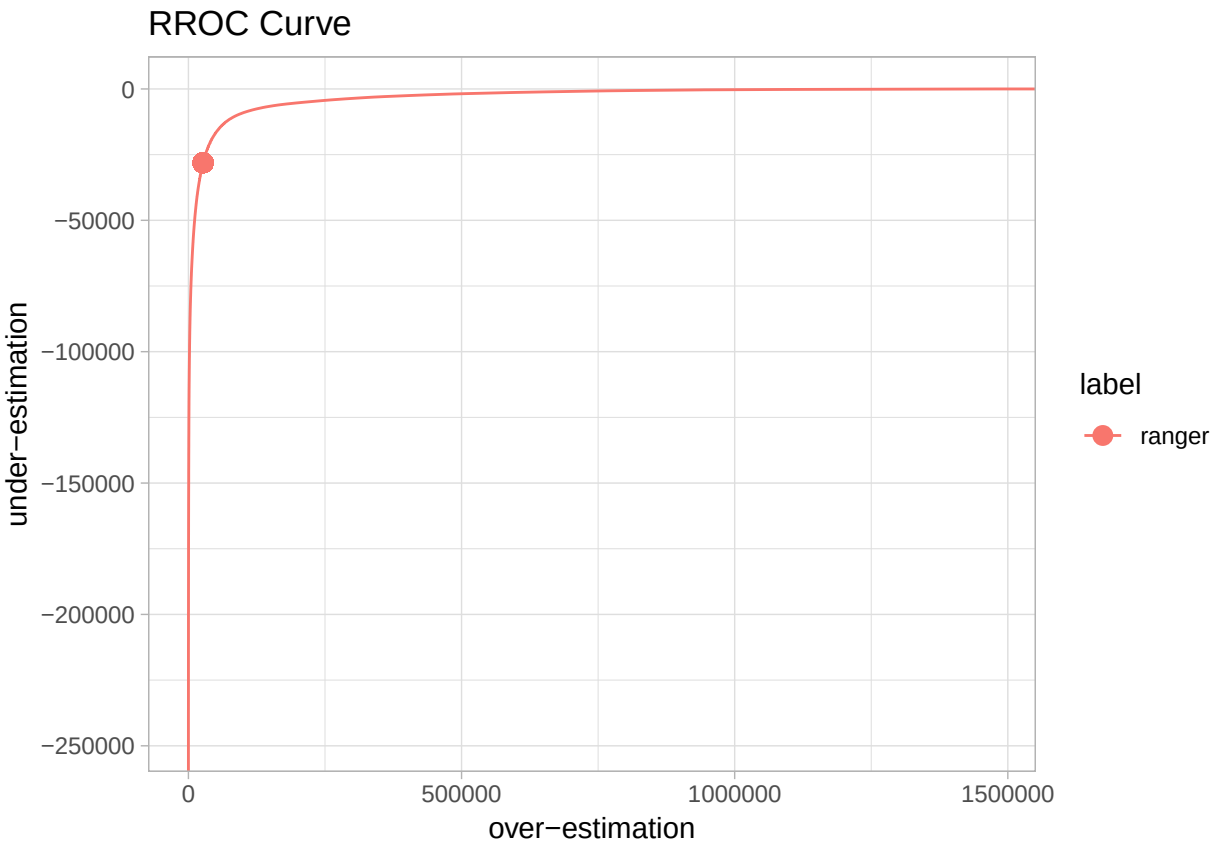

```
plotResidual(rf_au, variable = "Observed response")
```



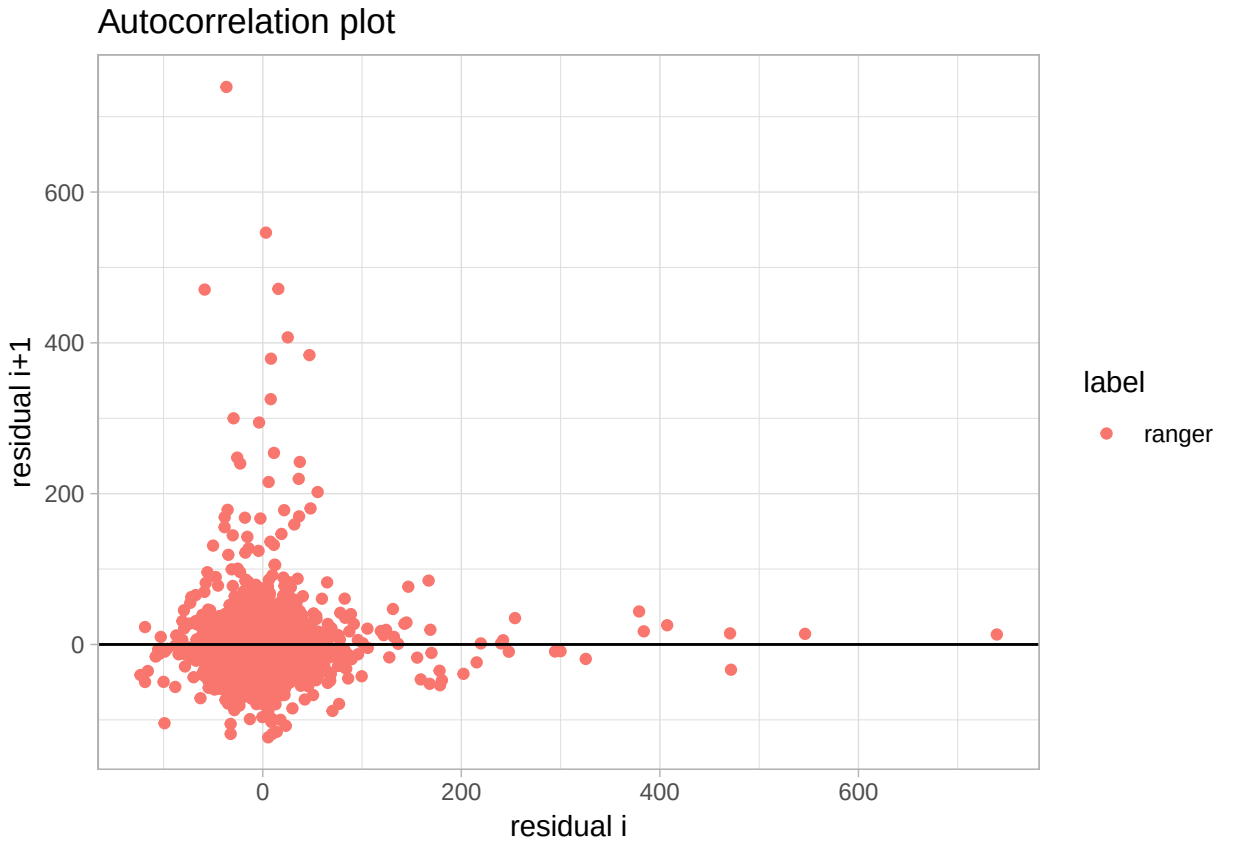

```
plotScaleLocation(rf_au)
```




```
plotRROC(rf_au)
```




```
plotAutocorrelation(rf_au)
```



0.17 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.

0.17.1 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.

0.17.2 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.

0.17.3 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
```

0.17.4 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.

0.17.5 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    3.4
```

0.17.6 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 0.13.1. 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}.

0.17.7 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.

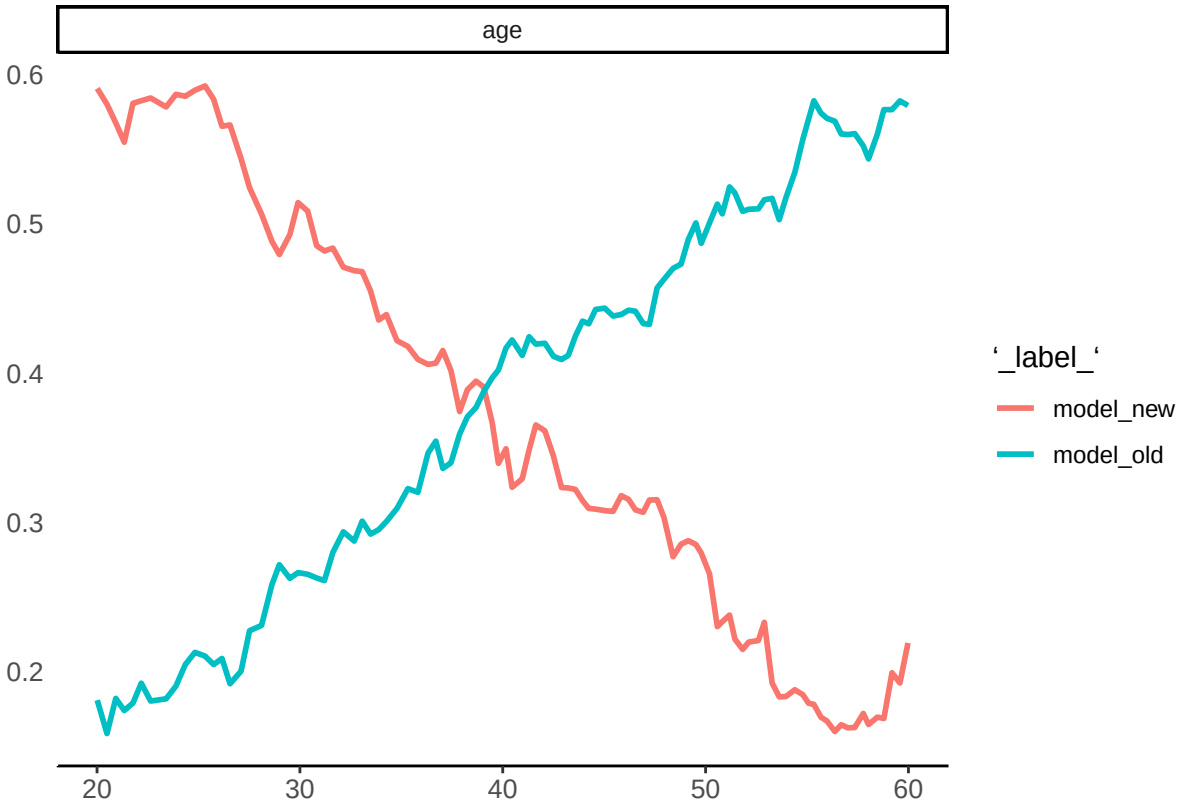
But, because of the interaction of gender and age, models created on these two datasets are different.

```
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)
```

```
##                    Variable    Shift  Scaled
##    ---------------------------------------------
##                      gender     0.00     0.9
##                         age     0.26    54.7  ***
##                       hours     0.04     8.3
##                  evaluation     0.01     2.6
##                      salary     0.02     3.4
##                      status     0.00     0.9
```

```
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,
     selected_variables = "age", aggregate_profiles = mean,
     show_observations = FALSE, color = "_label_", alpha = 1)
```



Appendixes

0.18 Data Sets

0.18.1 Hire or Fire? HR in Call Center

In this chapter we present an artificial dataset from Human Resources department in a Call Center.

The dataset is available in the `DALEX` package (Biecek, 2018c). Each row corresponds to a single employee in a call center. Features like gender, age, average number of working hours per week, grade from the last evaluation and level of salary are used as predictive features.

The goal here is to first build a model, that will guess when to fire and when to promote an employer, so it's a classification problem with three classes.

Why we need such model? We want to have objective decisions. That will not be subject to personal preferences of a manager. But is it possible to have an objective model? Would it be fair or it will just replicate some unfairness?

We will use this example to show how to use prediction level explainers to better understand how the model works for selected cases.

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

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

In this book we are focused on model exploration rather than model building, thus for sake ok simplicity we will use two default models created with random forest (Breiman et al., 2018) and generalized linear model (Ripley, 2016).

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

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

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

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

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

The dataset is available in the `DALEX` package (Biecek, 2018c). Each row corresponds to a single apartment. Features like surface, number of rooms, district or floor are used as predictive features.

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

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

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

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

```
##
## Call:
##  randomForest(formula = m2.price ~ construction.year + surface +     floor + no.rooms + district, data = apart
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 1
##
##           Mean of squared residuals: 82079.37
##                     % Var explained: 90.01
```

And a linear model.

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

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

0.19 Packages

0.19.1 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. (2018). *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). *ceterisParibus: Ceteris Paribus Profiles*. R package version 0.3.1.

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

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.

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.

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.

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. (2015). Peeking inside the black box: Visualizing statistical learning with plots of individual conditional expectation. *Journal of Computational and Graphical Statistics*, 24(1):44–65.

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

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.

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

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. (2018). 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.

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.

Ripley, B. (2016). *nnet: Feed-Forward Neural Networks and Multinomial Log-Linear Models*. R package version 7.3-12.

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.

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