Warsaw University of Technology

FACULTY OF
MATHEMATICS AND INFORMATION SCIENCE

Master's diploma thesis

in the field of study Mathematics
and specialisation Mathematical Statistics and Data Analysis

Surrogate-assisted feature extraction: an R package for extraction of interpretable features based on explainers for complex predictive models.

Anna Gierlak

student record book number 270733

thesis supervisor
dr hab. inż. Przemysław Biecek, prof. PW

WARSAW 2019

....

supervisor's signature author's signature

Abstract

Surrogate-assisted feature extraction: an R package for extraction of interpretable features based on explainers for complex predictive models

Nowadays, complex machine learning models, such as deep neural networks or boosting trees, are being applied in various areas of life. They are characterised by high predictive power but, unfortunately, they also have a huge complexity. That results in decrease of interpretability, which in turn is associated with reduced confidence in decisions the models take. Therefore, these days strong emphasis is placed on developing tools that allow to build models of both high accuracy and explainability.

This paper describes the methodology that uses elastic black-box models to create new features which then, used in a process of fitting another, simpler model, may significantly improve its performance. The methods for transforming original features, both continuous and categorical ones, are presented. Furthermore, an approach to feature selection and the results for a few chosen examples are illustrated.

The implementation of discussed techniques may be found in the **rSAFE** package for R.

Keywords: predictive models, machine learning, interpretability, explainability, automatic feature extraction, R

Streszczenie

Budowa interpretowalnych cech wspomagana przez wyjaśnienia dla złożonych modeli predykcyjnych

W dzisiejszych czasach skomplikowane algorytmy uczenia maszynowego, takie jak głębokie sieci neuronowe czy komitety drzew, znajdują szerokie zastosowania w różnych dziedzinach życia. Cechują się one wysoką jakością predykcyjną, jednak ich ogromny poziom skomplikowania pociąga za sobą niestety spadek interpretowalności. A to z kolei wiąże się z obniżonym zaufaniem w stosunku do podejmowanych przez nich decyzji. Dlatego w ostatnich dniach duży nacisk kładziony jest na rozwijanie narzędzi pozwalających na budowę modeli zarówno o wysokiej efektywności, jak i wytłumaczalności.

W niniejszej pracy opisana została metodyka wykorzystująca elastyczne modele czarnej skrzynki do generowania nowych zmiennych, które użyte następnie w procesie budowania innego, prostszego modelu mogą znacząco poprawić jego zdolność predykcyjną. Przedstawiono tutaj sposoby transformacji oryginalnych zmiennych, zarówno ciągłych, jak i dyskretnych. Opisano również metody wyboru zestawu zmiennych oraz zaprezentowano wyniki uzyskane dla kilku wybranych przykładów.

Implementacja omawianych metod znajduje się w pakiecie **rSAFE** napisanym w języku R.

Słowa kluczowe: modele predykcyjne, uczenie maszynowe, interpretowalność, wyjaśnialność, automatyczna ekstrakcja cech, R

Warsaw, 24.09.2019

Declaration

I hereby declare that the thesis entitled „Surrogate-assisted feature extraction: an R package for extraction of interpretable features based on explainers for complex predictive models", submitted for the Master degree, supervised by dr hab. inż. Przemysław Biecek, prof. PW, is entirely my original work apart from the recognized reference.

.....

Contents

Introduction

Nowadays, machine learning models have various applications in many areas of our everyday life. Since they have been started to use also in high-risk environments, such as those concerning human health and safety, it is crucial that we fully understand how they work and thus know in which cases they may fail.

Unfortunately, great forecasting abilities (meaning that errors of model predictions are rare or small on average) are usually associated with complex models, such as deep neural networks (Goodfellow et al., 2016)[16] or boosting trees (Chen and Guestrin, 2016)[11]. Their decisions are hard to comprehend by a human and therefore our trust in these models remains on low level. Recently, great emphasis has been placed on the interpretability requirement (Goodman and Flaxman, 2016)[17] and for this reason plenty of methods for explaining predictions of complicated models have been proposed. They differ widely when it comes to principles of operation, e.g. LIME method (Tulio Ribeiro et al., 2016)[42] uses local approximations with an interpretable model, SHAP method (Lundberg and Lee, 2017)[32] focuses on determining a contribution of each feature to the prediction.

In this paper, a Surrogate Assisted Feature Extraction (SAFE) method is presented. It uses a complex model with high accuracy to perform feature engineering that may be later applied to a simpler model, improving its performance. This first complex model is treated as a surrogate and does not have to be explainable itself, since its only task is to provide a good representation of data. The SAFE methodology is model agnostic, meaning it can be used for a wide range of predictive models. Therefore, the first complex model may be chosen so that is as accurate as possible (to provide valuable transformations of features) and the second one so that it meets the requirements arising from interpretability issue and other guideline.

This paper is organized as follows. Chapter 1 contains some fundamental concepts and methodology related to predictive analysis. All datasets used throughout the paper are described in chapter 2. Chapter 3 covers all the details about the SAFE algorithm. The **rSAFE** package containing its implementation and useful functionalities is presented in chapter 4. The examples of application of the **rSAFE** package may be found in chapter 5 and final conclusions are included in chapter 6.

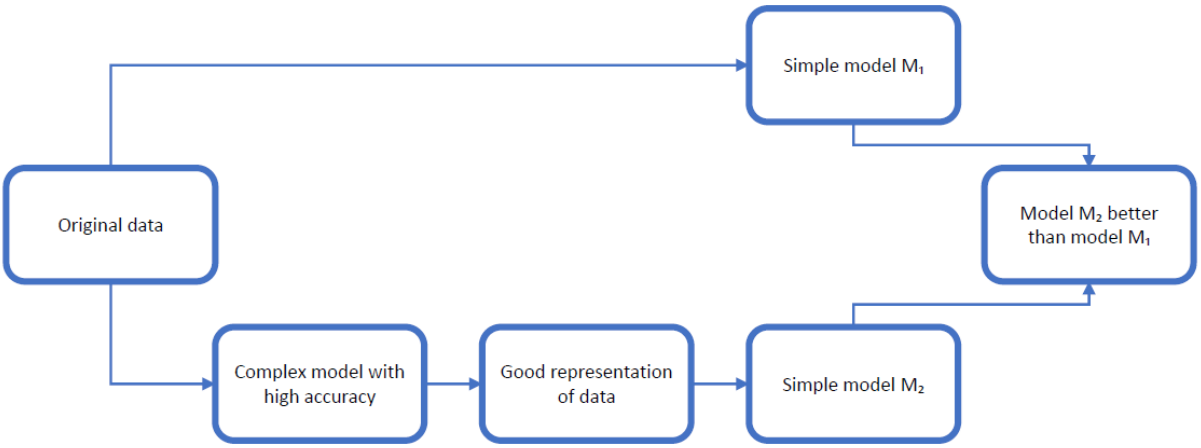


Figure 0.1: The idea of SAFE method. Instead of training a simple model on original data, we first use a complex model to extract valuable feature transformations. Then, a simple model is fitted using transformed variables.

1. Basic information

In this chapter, we present some fundamental concepts and methods related to predictive analysis.

1.1. Predictive models

Predictive modelling is a multistage process (Biecek, 2019)[8] of constructing an algorithm (a model) that would be able to make accurate predictions based on provided data. This paper focuses on supervised machine learning which refers to tasks where datasets contain not only features, but also output values to predict. Depending on the nature of the target two different types of prediction problems can be identified:

- regression - in case of numerical outputs,
- classification - in case of categorical (factor) outputs.

There are plenty of models of different kinds for both regression and classification tasks - the workflow related to the modelling process, however, remains the same and is presented in figure 1.1.

Below, there is some basic terminology that is being used throughout the paper.
Let $x^{(1)}, x^{(2)}, \ldots, x^{(n)}$ be the observations in the given dataset. Each of them is a p-dimensional vector consisting of the corresponding feature values and $x_j^{(i)}$ denotes the value of the j-th variable associated with the i-th instance. Hence, a design matrix X is as follows:

$$X = \begin{pmatrix} x_1^{(1)} & x_2^{(1)} & \cdots & x_p^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \cdots & x_p^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{(n)} & x_2^{(n)} & \cdots & x_p^{(n)} \end{pmatrix}.$$

On the other hand, if we want to refer to all features except j-th one we simply write $x_{-j}^{(i)}$.
A vector of outputs for all observations in the dataset is denoted as $Y = (y^{(1)}, y^{(2)}, \ldots, y^{(n)})$.

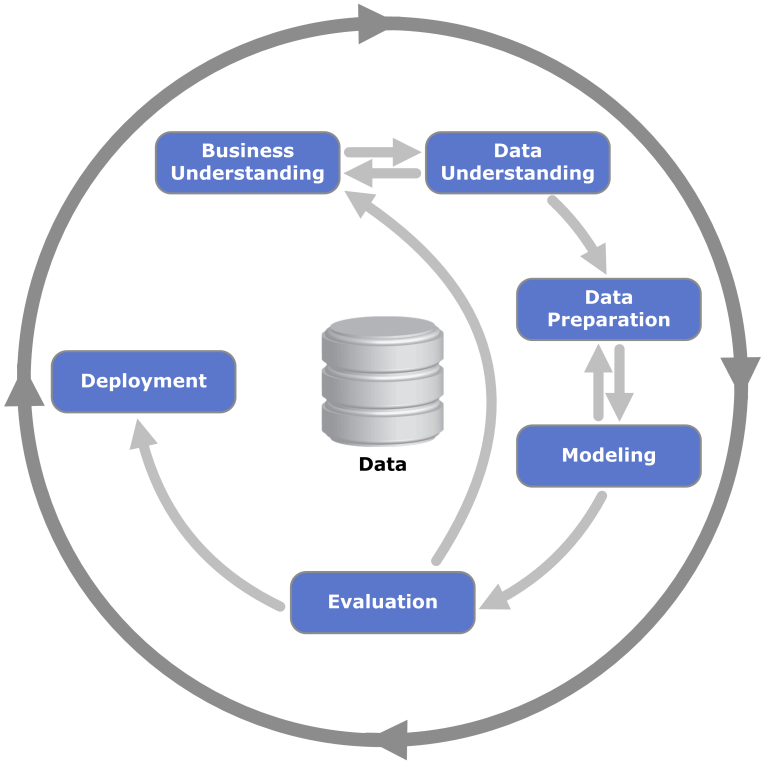


Figure 1.1: Diagram for CRISP DM: Cross-industry standard process for data mining. Image was taken from `https://en.wikipedia.org/wiki/Cross-industry_standard_process_for_data_mining`.

A predictive model is a function $f : \mathcal{X} \to \mathbb{R}$ which transforms a single point from $\mathcal{X}$ space into a real number and $\hat{y}^{(i)} = f(x^{(i)})$ stands for the model prediction for the i-th instance.

1.2. Feature engineering

Feature engineering process is a significant part of the predictive analysis and contains a variety of issues and tasks (Kuhn and Johnson, 2019)[31]. It involves broadly understood creation and manipulation of predictors that - conducted properly - might result in model performance improvement. Data is a fundamental component of every machine learning model but it can be really messy and overwhelming, especially nowadays when enormous databases are available due to rapid technological development in recent years. Thus, one should be able to deal with provided data and make them transparent and informative for the model so as to achieve higher accuracy.

Feature engineering refers to many different topics, including following:

- Feature filtering

One of the most important steps in feature engineering is identifying all the relevant predictors that should be included in the model. At the very beginning, based on his own experience, pertinent literature or experts knowledge, a data scientist should think through what data is required rather than focus on information already available.

- Feature transformation/extraction

  Once all necessary information is collected, you should consider various adjustments which could help the model work better. That includes creating indicator/dummy variables, combining sparse classes, logarithmic transformation, data scaling and many others. One may also think about possible interactions between two or more variables and try to add them somehow to the model. When performing all the modifications above, the type of machine learning model you are going to use ought to be taken into account. A simple example below presents consequences of modelling a target variable using raw, unmodified form of a predictor in a linear regression model.

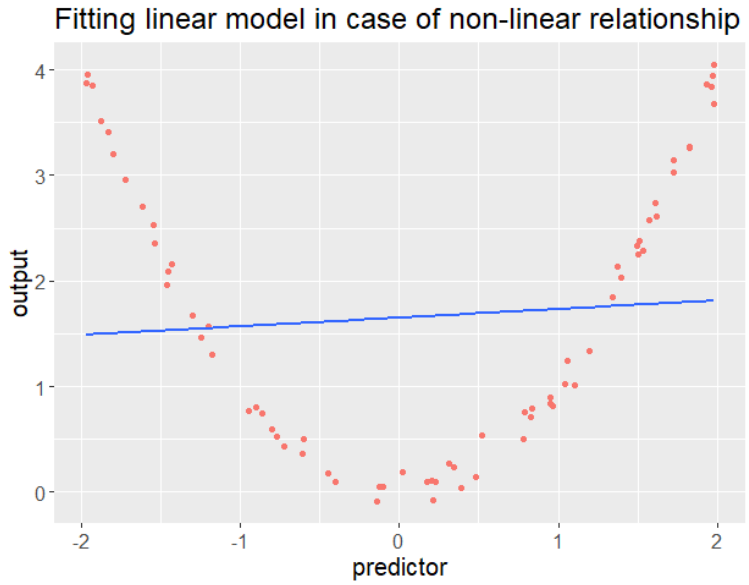


Figure 1.2: The consequences of fitting a linear model in case of non-linear relationship between a variable and an output. Not changing the form of the predictor will lead to poorly fitted linear model.

- Feature selection

  After extracting appropriate features the next step is deciding which of them to include in the model. Many of the algorithms will not be able to handle them all - some of the predictive models cannot cope with cases in which total number of variables exceeds the sample size. Other models encounter difficulties dealing with highly correlated predictors. Fur-

thermore, even when fitting the model is possible from a technical point of view, irrelevant or partially relevant features can negatively impact model performance. In general, this core concept of picking only some of the variables has also many other benefits, including overfitting reduction and training time decrease. Very often the procedure involves testing the effectiveness of different subsets of attributes and then choosing the best one.

1.3. Interpretability

A mathematical definition of interpretability does not exist, but in literature some informal ones occur. According to Miller's [34] approach interpretability is „the degree to which an observer can understand the cause of a decision". So higher interpretability means better human perception of reasons why a certain prediction has been made. In this paper the terms „interpretability" and „explainability" will stand for the same thing, whereas „explanation" will denote an explanation for a particular prediction.

There are many reasons why people look for model explanations. Natural curiosity might be mentioned as a top one but it is definitely more than that. Thorough understanding of the model behaviour can help you learn more about the data you own, the problem you are supposed to solve and might suggest in which situations the model is likely to make a mistake. Explainability is compulsory in some environments that have significant legal or regulatory consequences, such as financial institutions. It is also extremely desirable in fields concerning human health and safety.

However, the ability to interpret is valuable not only in high-risk environments. In the paper (Tulio Ribeiro et al., 2016)[42] there is analysis of the commonly known image classifier trained on the „Husky vs Wolf" task. The model was said to have remarkably high accuracy but interpretation methods have shown that it cannot be trustworthy - it turns out that it learned to differentiate the two animals by looking at snow in the background, actually not considering their color, position, etc. (figure 1.3).

There are two main ways in which explainability of machine learning model can be achieved. Above all, you might consider using only a fraction of existing algorithms which are simple and interpretable in themselves. A linear regression model may serve as a straightforward example. It predicts the output value as a weighted sum of the variables and in mathematical formula it can be written as follows:

$$y^{(i)} = \beta_0 + \beta_1 x_1^{(i)} + \ldots + \beta_p x_p^{(i)} + e^{(i)},$$

(a) Husky classified as wolf (b) Explanation

Figure 1.3: Raw image (on the left) and explanation of a bad model's prediction (on the right) in the „Husky vs Wolf" task. Images were taken from (Tulio Ribeiro et al., 2016)[42].

where $\beta_1, \ldots, \beta_p$ are coefficients learned by the model and $e^{(i)}$ is a residual for i-th observation. Such construction of the model makes it easy for humans to comprehend influence of a particular covariate as it is determined by the matching coefficient - increasing $x_j^{(i)}$ by one unit entails enhancement of a prediction by β_j.

Other examples of interpretable models could be logistic regression or decision/regression trees. Due to their simple structure they are often referred to as „glass-box" models, as opposed to the other part of the machine learning algorithms which are called „black-box" models. They are much more complex in their structure and usually involve a huge number of parameters, sometimes reaching millions or even billions (e.g. in deep neural networks). That prevents us from getting to know its operating principles and thus causes significant drop in our trustworthiness towards them.

Unfortunately, in many applications it is these complex models that have the greatest performance and giving up on them only because they are not transparent enough would not be desirable. So the second approach to preserving final explainability is to use a set of tools specially dedicated to that issue. In the last few years, countless methods have been proposed as a solution to interpretability problem and a SAFE algorithm is one of them.

2. Datasets

Throughout this paper, certain datasets are being used to present different techniques and models. In this chapter we provide detailed information on them in order to facilitate better understanding of the data we are working on.

2.1. Bike rentals

The *Bike rentals* dataset contains counts of rented bicycles from the bicycle rental company Capital-Bikeshare (`https://www.capitalbikeshare.com`) in Washington D.C, USA. It is based on original data from company system which is publicly available in `http://capitalbikeshare.com/system-data` and covers the period of two-year history (years 2011-2012). Fanaee-T and Gama (2013)[12] aggregated the data on two hourly and daily basis and added the corresponding weather and seasonal information (extracted from `http://www.freemeteo.com`). The dataset can be downloaded from the UCI Machine Learning Repository (`http://archive.ics.uci.edu/ml/datasets/Bike+Sharing+Dataset`s).

The dataset contains features such as:

- count of bicycles including both casual and registered users (used as the target in the modelling task),
- the season, one of spring, summer, fall, winter,
- indicator whether the day was a holiday or not,
- the year, one of 2011, 2012,
- number of days since the 01.01.2011 (the first day in the dataset) - this feature was introduced to take account of the trend over time,
- indicator whether the day was a working day or weekend,
- the weather situation, one of:

  - clear, few clouds, partly cloudy, cloudy,
  - mist + clouds, mist + broken clouds, mist + few clouds, mist,
  - light snow, light rain + thunderstorm + scattered clouds, light rain + scattered clouds,
  - heavy rain + ice pallets + thunderstorm + mist, snow + mist,

- temperature in degrees Celsius,
- relative humidity in percent (0 to 100),
- wind speed in km per hour.

Bike rental behaviors are highly correlated to season, day of the week, precipitation and other factors available in the dataset. The aim is to predict the number of bicycles being rented based on the seasonal and environmental settings. Since the count of bikes can be regarded as a continuous feature, it is a regression task.

2.2. Apartments

The *apartments* and *apartmentsTest* datasets contain information on prices of apartments in Warsaw. The structure of the data is based on original dataset from **PBImisc** package (Biecek, 2016)[3]. In both datasets the following features can be found:

- *m2.price* - price per square meter (used as a target in the modelling task),
- *surface* - apartment area in square meters,
- *no.rooms* - number of rooms (correlated with surface),
- *district* - district in which apartment is located, factor with 10 levels (Bemowo, Bielany, Mokotow, Ochota, Praga, Srodmiescie, Ursus, Ursynow, Wola, Zoliborz),
- *floor* - floor,
- *construction.year* - construction year.

The district feature is the only categorical one, all the other are on continuous scale. First few rows of the *apartments* dataset are displayed below.

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

Figure 2.1: First rows of the *apartments* dataset. The data contains one dependent variable *m2.price* and five explanatory features.

Both sets were artificially generated in a way to have the same measures of goodness of fit for linear model and random forest. The output feature *m2.price* was simulated as a function of all other variables:

$$5000 + 600 \cdot f(\text{construction.year}) - 10 \cdot \text{surface} - 100 \cdot \text{floor} - 50 \cdot \text{no.rooms} + 1000 \cdot g(\text{district}),$$

where:

$$f(\text{construction.year}) = \begin{cases} 0, & \text{if} \quad \text{construction.year} \in [1935, 1995] \\ 1, & \text{otherwise} \end{cases},$$

$$g(\text{district}) = \begin{cases} 2, & \text{if} \quad \text{district} = \text{Srodmiescie} \\ 1, & \text{if} \quad \text{district} \in \{\text{Mokotow, Ochota, Zoliborz}\} \\ 0, & \text{otherwise} \end{cases}.$$

Thus, the response was generated such that variables *surface*, *floor* and *no.rooms* have a linear impact on the output whereas variables *construction.year* and *district* do not.

The datasets can be used in the regression task of predicting prices of apartments based on other available features. Both datasets are derived from the **DALEX** package (Biecek, 2019)[6] but can also be found in the **rSAFE** package.

2.3. HR

The *HR_data* dataset contains information from a Human Resources department about employees. It consists of following features:

- *satisfaction_level* - level of satisfaction (0-1),

- *last_evaluation* - time since last performance evaluation (in years),
- *number_project* - number of projects completed while at work,
- *average_montly_hours* - average monthly hours at workplace,
- *time_spend_company* - number of years spent in the company,
- *Work_accident* - whether the employee had a workplace accident,
- *left* - whether the employee left the workplace or not, 0 or 1 (used as a target in the modelling task),
- *promotion_last_5years* - whether the employee was promoted in the last five years,
- *sales* - department in which the employee works for,
- *salary* - relative level of salary (low, medium, high).

First few rows of the *HR_data* set are displayed below.

satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	Work_accident	left	promotion_last_5years	sales	salary
0.38	0.53	2	157	3	0	1	0	sales	low
0.80	0.86	5	262	6	0	1	0	sales	medium
0.11	0.88	7	272	4	0	1	0	sales	medium
0.72	0.87	5	223	5	0	1	0	sales	low
0.37	0.52	2	159	3	0	1	0	sales	low
0.41	0.50	2	153	3	0	1	0	sales	low

Figure 2.2: First rows of the *HR_data* dataset. The data contains one dependent variable *left* and eight explanatory features.

This dataset may be used in the classification task of predicting whether an employee is likely to leave the company based on other available features. The data comes from the Kaggle competition „Human Resources Analytics" (`https://www.kaggle.com`) and is available in **breakDown** (Biecek, 2018)[4] and **rSAFE** packages.

2.4. Home Equity Line of Credit (HELOC)

The dataset contains Home Equity Line of Credit (HELOC) applications made by real homeowners. A HELOC is a line of credit typically offered by a bank as a percentage of home equity (the difference between the current market value of a home and its purchase price). The customers in this dataset have requested a credit line in the range of $5,000 - $150,000.

The fundamental task is to use the information about the applicant in their credit report to predict whether they will repay their HELOC account within 2 years. This prediction is then used to decide whether the homeowner qualifies for a line of credit and, if so, how much credit should be extended.

The predictor variables are all quantitative or categorical, and come from anonymized credit bureau data. The target variable to predict is a binary variable called *RiskPerformance*. The value „Bad" indicates that a consumer was 90 days past due or worse at least once over a period of 24 months from when the credit account was opened. The value „Good" indicates that they have made their payments without ever being more than 90 days overdue.

This is a real-world financial dataset provided by an analytics company FICO (`https://www.fico.com/`) for Explainable Machine Learning Challenge (`https://community.fico.com/s/explainable-machine-learning-challenge`). The challenge was focused on creating machine learning models with both high accuracy and explainability.

3. SAFE - Surrogate Assisted Feature Extraction

SAFE algorithm is a technique of distilling the knowledge from one machine learning model and then applying it to another one in order to boost the predictive power of the latter. The methodology focuses on feature transformations of specific type so it can constitute aid in early stages of the modelling workflow. It qualifies as a model-agnostic approach, meaning it may tackle any type of model, and its flexibility is a huge advantage over model-specific methods. Although there are no assumptions about the model structure (as it sometimes can be really complicated), this tool is designed to be used only for models operating on vectors from p-dimensional space and returning a single real value as an output for each case.

3.1. Distilling the knowledge from a model

As mentioned above, the SAFE algorithm works by extracting some information from a fitted model. Useful functionalities for this task can be found in two following R packages:

- **DALEX** (Descriptive mAchine Learning EXplanations) (Biecek, 2018)[5],
- **ingredients** (Biecek, 2019)[7].

These are the libraries that help us understand how particular variables impact final predictions, which may be beneficial especially in case of black-box models. They are both model-agnostic and include tools that refer to overall understanding of a model as well as explaining predictions made for specific data points. They also allow to compare a collection of models which may be useful when we are considering strengths and weaknesses of different methods.

In R modelling environment packages are very inconsistent and user interfaces vary widely making it harder to train, validate and use models. First thing **DALEX** does is wrapping up the model with specific metadata that facilitates its later investigation. Objects storing these additional information are called *explainers* since they are designed to clarify some aspects of a model. They can be used later for different purposes, such as explanations for model performance or variable importance. However, in this paper we will focus only on understanding

the effect of a single feature which will be discussed in the next section.

3.2. Effect of a single variable

Assessing the relationship between a predictor and the response is very often an important issue in machine learning tasks. This can be accomplished in many ways. In this section two methods are presented:

- partial dependence function,
- accumulated local effects function.

They will be used later for transformations of the features from a dataset.

3.2.1. Partial dependence function

One of the methods for obtaining the relationship between a variable and the target is constructing *partial dependence plots (PDPs)* introduced for the first time by Friedman (2001)[14]. They help us visualize how a certain predictor affects the output while accounting for the average effect of other features in our model.

Definition 3.1. The partial dependence function for j-th feature is defined as follows:

$$g_j^{PD}(z; f) = \mathbb{E}\left[f(x_j = z, X_{-j})\right] = \int f(x_j = z, x_{-j}) p_{-j}(x_{-j}) dx_{-j}, \tag{3.1}$$

where $p_{-j}(\cdot)$ denotes the marginal distribution of X_{-j}.

Thus, it is an expected output of our model f after setting the variable x_j to z. This function depends only on j-th feature we are interested in and by marginalizing over all other variables it shows the relationship between the j-th feature and the predicted output.

Example 3.1. Let X_1, X_2 be features in a model such that $X_1 \sim N(4, 1)$ and $X_2 \sim U([0, 1])$. If the model has the form of $f(x_1, x_2) = x_1 + x_2$, then:

$$g_1^{PD}(z; f) = z + 0.5, \qquad g_2^{PD}(z; f) = 4 + z.$$

Unfortunately, for real data we cannot compute the values of the partial dependence function directly as we usually do not know exactly neither the prediction function f nor the distribution of X_{-j}. However, the equation 3.1 may be estimated from a training set as follows:

$$\hat{g}_j^{PD}(z; f) = \frac{1}{n} \sum_{i=1}^{n} f(x_j^{(i)} = z, x_{-j}^{(i)}). \tag{3.2}$$

The partial dependence plot (PDP) for j-th feature is a plot of a partial dependence function estimator $\hat{g}_j^{PD}$ as a function of j-th feature.

Constructing a PDP for j-th feature in practice is pretty straightforward:

- Choose the number of points K and the grid $z_j^1, z_j^2, \ldots, z_j^K$ for which you want to compute values (points may be equidistant but not necessarily). If the variable is categorical, you should take all categories instead.
- For $k = 1, \ldots, K$:
  - Copy the training set and replace the original values of x_j with the constant z_j^k.
  - For each modified observation in the training set compute the prediction $f(x_j^{(i)} = z_j^k, x_{-j}^{(i)})$.
  - Average predictions over the whole dataset to obtain $\hat{g}_j^{PD}(z_j^k; f)$.
- Plot the pairs $\{z_j^k, \hat{g}_j^{PD}(z_j^k; f)\}_{k=1,\ldots,K}$.

Example 3.2. Let us get better understanding of the idea of PDPs on the example of bike rentals dataset (section 2.1). Having a regression problem of predicting a count of bicycles rented, we first fit a regression tree model which will serve us to construct relevant plots. We take only three of the variables into consideration: temperature (in degrees Celsius), relative humidity (in percent) and wind speed (km/h). The figure 3.1 illustrates the way these features influence the target variable. As we can see, higher temperatures encourage bike riding - at least to some point, since very hot weather has the opposite effect. Additionally, when humidity exceeds the level of 60%, number of rented bikes decreases. And the stronger the wind, the more unlikely people are to cycle which is quite intuitive. Although the plateau can be observed in the final stage of the plot, it should not be taken for granted since there is little training data in this region.

3.2.2. Accumulated local effects function

A partial dependence plot is a quite intuitive tool for analysing the impact a certain feature has on the final prediction. However, it has one major drawback - we assume here that there is no correlation between the feature for which the partial dependence function is computed and all the other variables. If such correlation occurs, then substituting feature values with one fixed constant from the grid may cause creation of points of very low probability. The alternative to the PDPs are *accumulated local effects plots (ALE plots)* introduced by Apley (2016)[1].

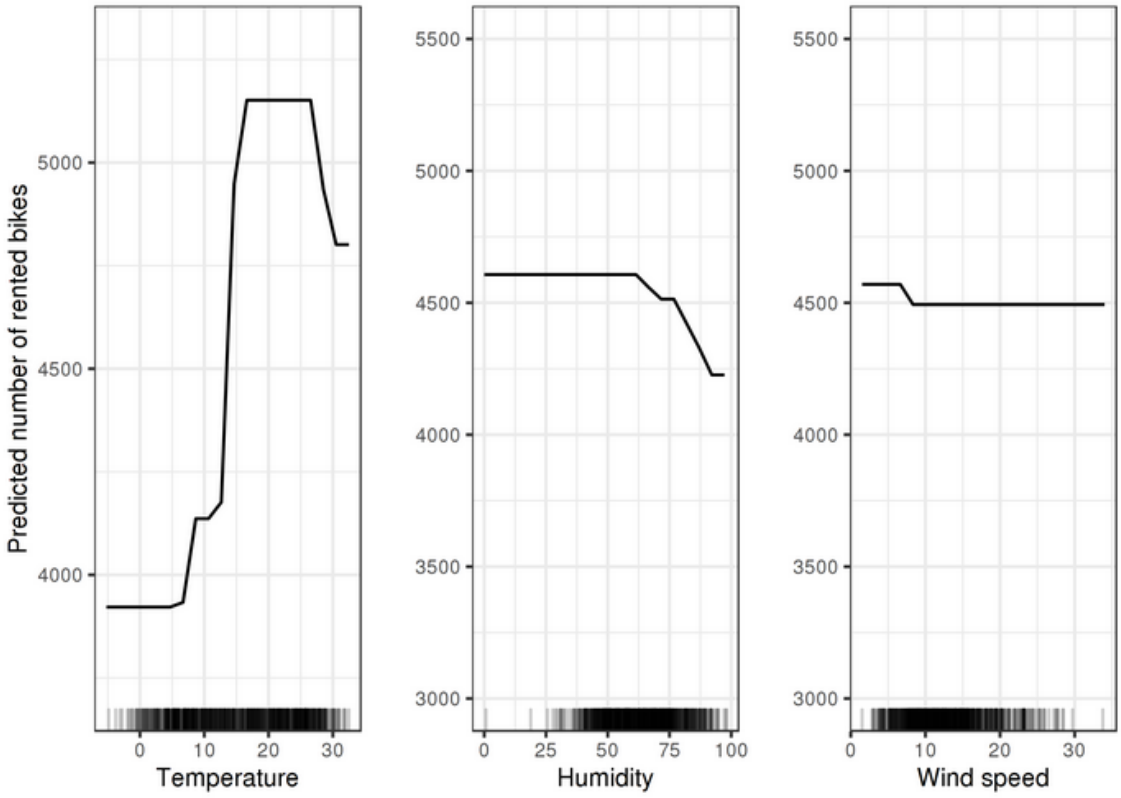


Figure 3.1: Partial dependence plots for the fitted regression tree and three features: temperature, humidity, wind speed. Bars on the lower sides of plots indicate variables distribution from the training set. Images were taken from (Molnar, 2019)[35].

Their operating principle is averaging predictions while adjusting all features to the current value of the j-th one and extracting effects of the other variables in the same time.

Definition 3.2. The accumulated local effects function for j-th feature is defined as follows:

$$\begin{aligned} g_j^{ALE}(z;f) &= \int_{\tau_0}^{z} \mathbb{E}\left[\left.\frac{\partial f(x)}{\partial x_j}\right|_{x_j=X_j, x_{-j}=X_{-j}} \middle| X_j = t\right] dt + c \\ &= \int_{\tau_0}^{z} \int \left.\frac{\partial f(x)}{\partial x_j}\right|_{x_j=t} p_{-j|j}(x_{-j}|t) dx_{-j} dt + c, \end{aligned} \tag{3.3}$$

where τ_0 is the lower bound of X_j, $p_{-j|j}(\cdot|\cdot)$ denotes the conditional distribution of X_{-j} given X_j and c is a constant.

In the equation above, the $\left.\frac{\partial f(x)}{\partial x_j}\right|_{x_j=t}$ component represents the local effect of the j-th variable on function f at point $(x_j = t, x_{-j})$. This local effect is averaged across all values of x_{-j} with weights $p_{-j|j}(x_{-j}|t)$ and then finally accumulated (integrated) over all values of t up to z. And that is where the name *accumulated local effects* comes from. The constant c may be selected arbitrarily but usually it is used to center the ALE function so that $g_j^{ALE}(X_j; f)$ has

a mean of zero with respect to the marginal distribution of X_j.

In practice, as in the case of partial dependence functions, we have to use the approximation of the equation 3.3. Let us divide the range of $\{x_j^{(i)}, i = 1, \ldots, n\}$ from the training set into K intervals: $N_j(1) = (z_j^0, z_j^1]$, $N_j(2) = (z_j^1, z_j^2]$, ..., $N_j(K) = (z_j^{K-1}, z_j^K]$ where z_j^k may be, for example, respective empirical quantiles of $\{x_j^{(i)}, i = 1, \ldots, n\}$, with z_j^0 selected just below the minimum and z_j^K selected as the greatest observation. The number of training instances which fall into the $N_j(k)$ interval will be denoted as $n_j(k)$ and $k_j(z)$ will stand for the index of the interval z value belongs to. With introduced notation the ALE function from 3.3 may be estimated in the following way:

$$\hat{g}_J^{ALE}(z; f) = \sum_{k=1}^{k_j(z)} \frac{1}{n_j(k)} \sum_{i: x_j^{(i)} \in N_j(k)} \left[f(x_j^{(i)} = z_j^k, x_{-j}^{(i)}) - f(x_j^{(i)} = z_j^{k-1}, x_{-j}^{(i)}) \right] + c. \tag{3.4}$$

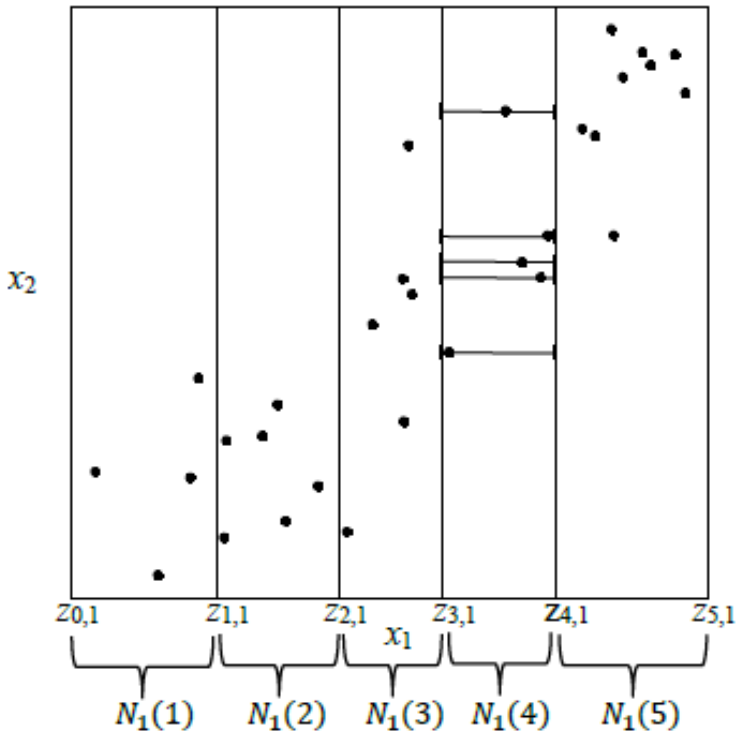


Figure 3.2: Intuition for calculating the estimator of ALE function for $p = 2$. The values of the x_1 from the training set are partitioned into $K = 5$ intervals. The horizontal lines shown in the fourth region indicate which points will be used to calculate the finite differences in prediction $f(x_1^{(i)} = z_1^4, x_2^{(i)}) - f(x_1^{(i)} = z_1^3, x_2^{(i)})$ that will correspond to the effect of x_1 feature in $N_1(4)$ neighbourhood. Those differences are then averaged across the whole region and added to the average local effects from the previous intervals to produce the final estimator. Image was taken from (Apley, 2016)[1].

The accumulated local effects plot (ALE plot) for j-th feature is a plot of an accumulated local effects function estimator $\hat{g}_j^{ALE}$ as a function of j-th feature.

Example 3.3. Let us illustrate accumulated local effects plots through the bike rentals dataset (section 2.1), similarly as it was done for PDPs in the example 3.2.

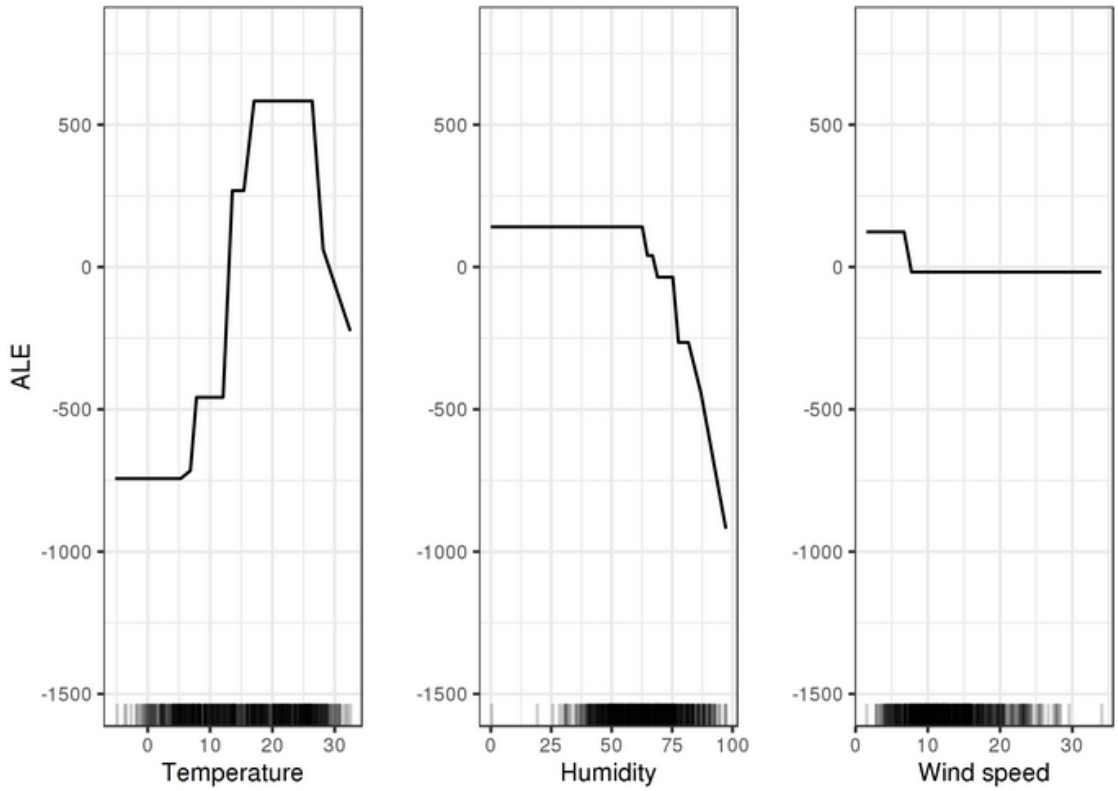


Figure 3.3: Accumulated local effects plots for the fitted regression tree and three features: temperature, humidity, wind speed. Bars on the lower sides of plots indicate variables distribution from the training set. As opposed to the PDPs from 3.2, the plots here are centered on zero. Images were taken from (Molnar, 2019)[35].

If we now compare obtained ALE plots with the previous PDPs, we can notice that they differ, e.g. ALE plot for humidity shows a greater decrease of rented bikes for higher values of the predictor than it was visible on the corresponding PD plot. This suggests that the features in the dataset are correlated to some extent and for this reason one should use accumulated local effects plots rather than partial dependence plots.

The methodology of ALE plots is applied only to continuous features, since for a categorical variable very often there is no established order within factors what precludes from dividing them into intervals and calculating the differences on the ends of segments.

The concepts of PD and ALE plots can be extended to larger numbers of features considered, resulting in high-dimensional plots. But in this paper only those for a single variable will be used. Furthermore, for classification tasks the theory of PD and ALE functions may be applied to models that return probabilities rather than predicted classes. The function f corresponds then to the probability of a chosen class.

3.3. Continuous features - detecting changepoints

3.3.1. Overview of the changepoint detection problem

In section 3.2 we discussed some methods for exploring the impact particular features have on the final prediction. And now we will use them to construct new variables, based on the knowledge a black-box model provides us. In this section we will focus only on continuous predictors. Having the set of points $\{z_j^k, \hat{g}_j(z_j^k; f)\}_{k=1,\ldots,K}$ from a PD or ALE plot, we will treat the sequence of the y-axis values $(\hat{g}_j(z_j^k; f))_{k=1,\ldots,K}$ as a time series and try to detect *changepoints (breakpoints)*, i.e. find points where the series *changes somehow*. Once done, these changepoints will be used to partition the range of the feature values and discretize it, creating new categorical variable. Thanks to this procedure, the values of the predictor which have the similar contribution to the prediction will be grouped into the same category. And this kind of data representation, learned by a complex model but maybe impossible to capture by an interpretable simpler model, can help the latter to perform better.

Changepoint analysis is a problem of identification of points within an ordered sequence of data where its statistical properties change somehow. The nature of these changes could be of different kinds, e.g. we can observe shifts only in mean or in variance as well as simultaneous changes of many statistics. Also, the number of changes is usually unknown, so the task involves calculating shifts positions together with their number.

Let us consider feature X_j and its corresponding PD/ALE plot. For simplicity, the sequence $(\hat{g}_j(z_j^k; f))_{k=1,\ldots,K}$ will be denoted as $v = (v_1, v_2, \ldots, v_K)$. We assume the existance of theoretical $M \in \mathbb{N}$ breakpoints and their positions $t = (t_1, t_2, \ldots, t_M)$ which indicate the moments of changes within the vector v. Each changepoint position t_m is an integer between 1 and $K-1$ inclusive. For a convenience purpose we also define $t_0 = 0$ and $t_{M+1} = K$ and assume that $0 = t_0 < t_1 < t_2 < \ldots < t_M < t_{M+1} = K$, i.e. the locations of breakpoints within t are ordered. Moreover, for $a \leq b$ we introduce the following notation for a subvector: $v_{a:b} = (v_a, v_{a+1}, \ldots, v_b)$.

Thus, the vector of indexes t divides our series into $(M+1)$ segments:

$$v_{1:t_1}, \quad v_{(t_1+1):t_2}, \quad v_{(t_2+1):t_3}, \quad \dots, \quad v_{(t_M+1):K}.$$

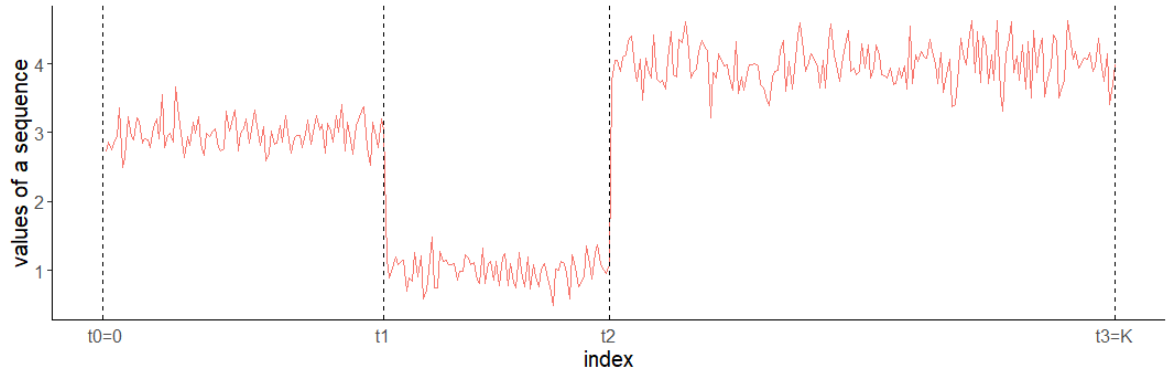


Figure 3.4: Changepoint detection problem. The presented time series consists of three homogenous segments. Thus, an efficient changepoint detection algorithm should return $M = 2$ as the number of breakpoints and $t = (t_1, t_2)$ as their positions.

As PD and ALE plots, to which the changepoint detection method will be applied, may have arbitrary shapes, we do not make any assumptions about the distribution of the vector v. And since the theoretical cumulative distribution function F of our data remains unknown, we will use its empirical form:

$$\forall w \in \mathbb{R} \qquad \hat{F}(w) = \frac{1}{K} \sum_{k=1}^{K} \left(\mathbb{1}(v_k < w) + 0.5 \cdot \mathbb{1}(v_k = w) \right), \tag{3.5}$$

where the component $0.5 \cdot \mathbb{1}(v_k = w)$ corresponds to the common „continuity correction" to the CDF in its discontinuity points. Furthermore, the equivalent of 3.5 for a subvector $v_{a:b}$ will be denoted as $\hat{F}_a^b(w)$.

In the proposed breakpoints detection method we assume that our sequence v is composed of independent random variables such that:

$$V_k \sim \sum_{m=1}^{M+1} F_m \mathbb{1}(t_{m-1} < k \leq t_m), \tag{3.6}$$

where F_m are CDFs for consecutive segments. So more formally, changepoints t_m are treated as bounds which partition the sequence into disjoint sets within which random variables are identically distributed.

3.3.2. Non-parametric likelihood and segmentation cost

The common approach to multiple changepoints identification is to minimise the following expression:

$$\sum_{m=1}^{M+1} cost_{t_{m-1}+1}^{t_m} + \gamma f(M), \tag{3.7}$$

where $cost$ is an appropriate cost function chosen so that it measures the „homogeneity" of the data and $cost_a^b$ stands for the cost of the segment $v_{a:b}$. The $cost$ function is expected to be large for a „heterogeneous" vector (containing one or more breakpoints) and low otherwise. The $\gamma f(M)$ component is a penalty for the complexity of the segmentation. Since the distribution of the sample is unknown, we cannot use the explicit log-likelihood function to construct the $cost$ function. Instead, we will introduce nonparametric log-likelihood, proposed by Zou et al. (2014)[48], in order to solve our problem.

If we look at our data as a binary sample with the probability of success $\hat{F}(w)$ for fixed w (where success means being less or equal to w), i.e. $V_1, V_2, \ldots, V_K \sim B(F(w))$, then we can write the likelihood function as follows:

$$L_1^K(w) = \prod_{k=1}^{K} F(w)^{\mathbb{1}(v_k \leq w)} \left(1 - F(w)\right)^{\mathbb{1}(v_k > w)} = F(w)^{K\hat{F}(w)} (1 - F(w))^{K(1-\hat{F}(w))}. \tag{3.8}$$

As usual, the logarithmic form of the likelihood function is more convenient:

$$l_1^K(w) = \log L_1^K(w) = K \left[\hat{F}(w) \log F(w) + (1 - \hat{F}(w)) \log(1 - F(w))\right]. \tag{3.9}$$

By calculating $\frac{dl}{dF}$, we can obtain that the expression above is maximised by the value of the empirical distribution function, which gives us the following form of the maximum of nonparametric log-likelihood function:

$$l_1^K(w) = K \left[\hat{F}(w) \log \hat{F}(w) + (1 - \hat{F}(w)) \log(1 - \hat{F}(w))\right]. \tag{3.10}$$

We can obtain similar log-likelihood maxima for single segments and use them in our $cost$ function, with the opposite sign since for the consistent data the cost values are supposed to be lower. Hence, for a particular w we can write:

$$\begin{aligned} cost_{t_{m-1}+1}^{t_m}(w) = -l_{t_{m-1}+1}^{t_m}(w) = -\left(t_m - t_{m-1}\right) \Big[\hat{F}_{t_{m-1}+1}^{t_m}(w) \log \hat{F}_{t_{m-1}+1}^{t_m}(w) \\ + (1 - \hat{F}_{t_{m-1}+1}^{t_m}(w)) \log(1 - \hat{F}_{t_{m-1}+1}^{t_m}(w))\Big]. \end{aligned} \tag{3.11}$$

The cost in equation 3.11 uses the CDF evaluated only at one value w and its choice may significantly impact the final segmentation. To cope with this issue, we can integrate this cost across all possible values:

$$cost_{t_{m-1}+1}^{t_m} = \int_{-\infty}^{\infty} cost_{t_{m-1}+1}^{t_m}(w) dW(w) \tag{3.12}$$

with weights $dW(w) = [F(w)(1-F(w))]^{-1} dF(w)$ as proposed in (Zou et al., 2014)[48] and (Zhang, 2002)[46]. In practical applications, the cost from 3.12 is approximated by summing across $S << K$ (specifically chosen) terms, as suggested in (Haynes et al., 2016)[24]. Let us first note that it can be estimated as:

$$cost^{t_m}_{t_{m-1}+1} = \int_{w_0}^{w_1} cost^{t_m}_{t_{m-1}+1}(w) \left[F(w)(1-F(w))\right]^{-1} dF(w), \tag{3.13}$$

where w_0, w_1 satisfy $F(w_0) = \frac{1}{2K} \approx 0$ and $F(w_1) = \frac{2K-1}{2K} \approx 1$. Below we present the lemma that will help us to dispose of the weights.

Lemma 3.1. Let $C = -\log(2K-1)$ and $p(x) = \frac{1}{1+\exp\{Cx\}}$ for $x \in \mathbb{R}$. Then:

$$\int_{w_0}^{w_1} cost^{t_m}_{t_{m-1}+1}(w) \left[F(w)(1-F(w))\right]^{-1} dF(w) = -C \int_{-1}^{1} cost^{t_m}_{t_{m-1}+1}(F^{-1}(p(x)))dx. \tag{3.14}$$

Proof. Showing the equality in the lemma boils down to making a change of variables in the integral.

$$\begin{aligned}
&F(w) = p(x),\\
&w = F^{-1}(p(x)),\\
&dF(w) = dp(x) = \frac{-C\exp\{Cx\}}{(1+\exp\{Cx\})^2}dx = -Cp(x)(1-p(x))dx = -CF(w)(1-F(w))dx,\\
&\left[F(w)(1-F(w))\right]^{-1} dF(w) = -Cdx.
\end{aligned}$$

Hence:

$$cost^{t_m}_{t_{m-1}+1}(w) \left[F(w)(1-F(w))\right]^{-1} dF(w) = -C \cdot cost^{t_m}_{t_{m-1}+1}(F^{-1}(p(x)))dx.$$

The limits of the integral are obtained from the fact that:

$$F(w_0) = \frac{1}{2K} = p(-1), \qquad F(w_1) = \frac{2K-1}{2K} = p(1).$$

□

Now, let $x_s = -1 + \frac{2s-1}{S}$ for $s = 1, \ldots, S$. These are equally spaced values from the $[-1, 1]$ interval and based on lemma 3.1 we can use the Monto Carlo estimator to approximate 3.13 as follows:

$$\begin{aligned}
cost^{t_m}_{t_{m-1}+1} &= -C \int_{-1}^{1} cost^{t_m}_{t_{m-1}+1}(F^{-1}(p(x)))dx = -2C \int_{-1}^{1} \frac{1}{2} cost^{t_m}_{t_{m-1}+1}(F^{-1}(p(x)))dx\\
&= -2C \cdot \mathbb{E}\left[cost^{t_m}_{t_{m-1}+1}(F^{-1}(p(X)))\right] = -\frac{2C}{S}\sum_{s=1}^{S} cost^{t_m}_{t_{m-1}+1}(F^{-1}(p(x_s))).
\end{aligned} \tag{3.15}$$

Since:

$$p(x_s) = \frac{1}{1+\exp\{Cx_s\}} = \frac{1}{1+\exp\{-\log(2K-1)\cdot\left(-1+\frac{2s-1}{S}\right)\}} = \frac{1}{1+(2K-1)\exp\{C\frac{2s-1}{S}\}}, \tag{3.16}$$

if we define $q_1, q_2, \ldots, q_S$ as $\frac{1}{1+(2K-1)\exp\{C\frac{2s-1}{S}\}}$ empirical quantiles of our data, then the cost 3.15 may be obtained finally in the following way:

$$\begin{aligned} cost^{t_m}_{t_{m-1}+1} = -\frac{2C}{S}\sum_{s=1}^{S} cost^{t_m}_{t_{m-1}+1}(q_s) = &\frac{2C}{S}\left(t_m - t_{m-1}\right)\sum_{s=1}^{S}\Big[\hat{F}^{t_m}_{t_{m-1}+1}(q_s)\log\hat{F}^{t_m}_{t_{m-1}+1}(q_s) \\ &+(1-\hat{F}^{t_m}_{t_{m-1}+1}(q_s))\log(1-\hat{F}^{t_m}_{t_{m-1}+1}(q_s))\Big]. \end{aligned} \tag{3.17}$$

3.3.3. PELT algorithm

In this subsection we present the algorithm PELT (Pruned Exact Linear Time) introduced by Killick et al. (2012)[29] and based on the Optimal Partitioning approach of Jackson et al. (2004)[25] that can be used to obtain the exact solution of the changepoint detection problem.

In our problem, both number of changes and its locations remain unknown and the objective function to minimise is the penalised sum of costs for single segments. Obviously, one could think about considering all possible segmentations and then choosing the best one, but this would result in huge complexity and make this solution impractical for longer sequences. In the proposed method we assume the linear form of the penalty for a number of breakpoints, meaning that our problem boils down to:

$$Q_1^K = \min_{M,(t_1,\ldots,t_M)}\left(\sum_{m=1}^{M+1}\left[cost^{t_m}_{t_{m-1}+1} + \gamma\right]\right), \tag{3.18}$$

where $cost$ is the estimator of the cost function specified in subsection 3.3.2, $\gamma > 0$ is a penalty value for adding another changepoint and Q_1^K stands for the overall cost for the sequence $(v_1, \ldots, v_K)$.

The Optimal Partitioning method (Jackson et al., 2004)[25] uses the particular type of the recursion. It begins first by conditioning on the last breakpoint and then considers the optimal value of the cost for entire sequence as the sum of the cost for the optimal segmentation prior to the last breakpoint and the cost for the segment from the last breakpoint to the end

of the data. In mathematical formulas it can be expressed as follows:

$$\begin{aligned} Q_1^K &= \min_{M,(t_1,\dots,t_M)} \left(\sum_{m=1}^{M+1} \left[cost_{t_{m-1}+1}^{t_m} + \gamma \right] \right) \\ &= \min_{t_M} \left[\min_{M,(t_1,\dots,t_{M-1})} \left(\sum_{m=1}^{M} \left[cost_{t_{m-1}+1}^{t_m} + \gamma \right] \right) + cost_{t_M+1}^{K} + \gamma \right] \\ &= \min_{t_M} \left[Q_1^{t_M} + cost_{t_M+1}^{K} + \gamma \right]. \end{aligned} \tag{3.19}$$

Solving the recursion above results in an $\mathcal{O}(K^2)$ algorithm (Jackson et al., 2004)[25] and provides the exact solution to the minimalization problem 3.18.

However, the computational efficiency of the Optimal Partitioning algorithm can be improved by a method called PELT which is the abbreviation for „Pruned Exact Linear Time" (Killick et al., 2012)[29]. Its major idea is substantially speeding up the calculations required to solve 3.19 by *pruning*, i.e. reducing the set of values t_M we have to minimise over.

Lemma 3.2. Let us consider index a. If for any $a < b$ holds:

$$Q_1^a + cost_{a+1}^b \geq Q_1^b, \tag{3.20}$$

then for any future time $T > b$ point a can never be the optimal last changepoint prior to T.

Proof. First we will show that introducing another changepoint into a sequence results in decrease of the overall cost for the sequence, i.e.:

$$\forall a < b < c \qquad cost_{a+1}^c \geq cost_{a+1}^b + cost_{b+1}^c. \tag{3.21}$$

Using the estimator of the cost from 3.17:

$$cost_{a+1}^c = -\frac{2C}{S} \sum_{s=1}^{S} cost_{a+1}^c(q_s) = \frac{2\log(2K-1)}{S} \sum_{s=1}^{S} cost_{a+1}^c(q_s), \tag{3.22}$$

we need to show that:

$$\forall a < b < c \qquad \sum_{s=1}^{S} cost_{a+1}^c(q_s) \geq \sum_{s=1}^{S} cost_{a+1}^b(q_s) + \sum_{s=1}^{S} cost_{b+1}^c(q_s) \tag{3.23}$$

which will be done by proving the following:

$$\forall w \quad \forall a < b < c \qquad cost_{a+1}^c(w) \geq cost_{a+1}^b(w) + cost_{b+1}^c(w). \tag{3.24}$$

We have:

$$L = cost_{a+1}^c(w) \overset{(3.11)}{=} -(c-a) \left[\hat{F}_{a+1}^c(w) \log \hat{F}_{a+1}^c(w) + (1 - \hat{F}_{a+1}^c(w)) \log(1 - \hat{F}_{a+1}^c(w)) \right]. \tag{3.25}$$

The expression $(c-a)\left[\hat{F}^c_{a+1}(w)\right]$ in the sum above can be written as follows:

$$\begin{aligned}
&(c-a)\left[\frac{1}{c-a}\sum_{k\in[a+1,c]}[\mathbb{1}(v_k<w)+0.5\cdot\mathbb{1}(v_k=w)]\right]\\
&=\sum_{k\in[a+1,b]}[\mathbb{1}(v_k<w)+0.5\cdot\mathbb{1}(v_k=w)]+\sum_{k\in[b+1,c]}[\mathbb{1}(v_k<w)+0.5\cdot\mathbb{1}(v_k=w)]\\
&=(b-a)\left[\hat{F}^b_{a+1}(w)\right]+(c-b)\left[\hat{F}^c_{b+1}(w)\right]
\end{aligned} \tag{3.26}$$

and with the analogous transformation for the $(c-a)\left[1-\hat{F}^c_{a+1}(w)\right]$ we obtain:

$$\begin{aligned}
L=&-(b-a)\left[\hat{F}^b_{a+1}(w)\log\hat{F}^c_{a+1}(w)+(1-\hat{F}^b_{a+1}(w))\log(1-\hat{F}^c_{a+1}(w))\right]\\
&-(c-b)\left[\hat{F}^c_{b+1}(w)\log\hat{F}^c_{a+1}(w)+(1-\hat{F}^c_{b+1}(w))\log(1-\hat{F}^c_{a+1}(w))\right]\\
&\geq-(b-a)\left[\hat{F}^b_{a+1}(w)\log\hat{F}^b_{a+1}(w)+(1-\hat{F}^b_{a+1}(w))\log(1-\hat{F}^b_{a+1}(w))\right]\\
&-(c-b)\left[\hat{F}^c_{b+1}(w)\log\hat{F}^c_{b+1}(w)+(1-\hat{F}^c_{b+1}(w))\log(1-\hat{F}^c_{b+1}(w))\right]=P,
\end{aligned} \tag{3.27}$$

where the inequality follows from the fact that for any fixed $x\in(0,1)$ the function $h(y)=x\log y+(1-x)\log(1-y)$ takes the maximum value at $y=x$. And using first the fact 3.21 we have just proved and then the lemma assumption we obtain:

$$Q^a_1+cost^T_{a+1}\geq Q^a_1+cost^b_{a+1}+cost^T_{b+1}\geq Q^b_1+cost^T_{b+1}, \tag{3.28}$$

so if we consider sequence $v_{1:T}$, then point a can never be the last optimal changepoint. □

Hence, at each iteration of the algorithm it can be checked whether 3.20 holds and some of the points may be discarded reducing the number of computations required to obtain the final set of breakpoints.

3.4. Categorical features – merging factors together

3.4.1. Overview of the cluster analysis problem

In the previous subsection we proposed the method for dealing with continuous numeric variables but they are not the only type of features that occur in analyzed datasets. Categorical data is typically found as well. In general, a categorical (factor) variable takes discrete values belonging to a specific finite set of categories (also called classes, labels or levels). Since there might be no ordering amongst the variable levels, we cannot repeat the same procedure as before. Instead, we will use the PD function (3.1) to calculate average model predictions for each category separately and based on results we will try to combine these categories into

groups via a cluster analysis, merging multiple levels together. This will cause the formation of the new categorical variable with smaller number of levels which deviate more one from another and both mentioned characteristics may positively impact the performance of the simpler model.

The main goal of the cluster analysis is grouping a set of observations into disjoint subsets called „clusters" such that objects from one cluster are, in a sense, similar (close) to each other and those coming from two clusters - different (distant) from each other.

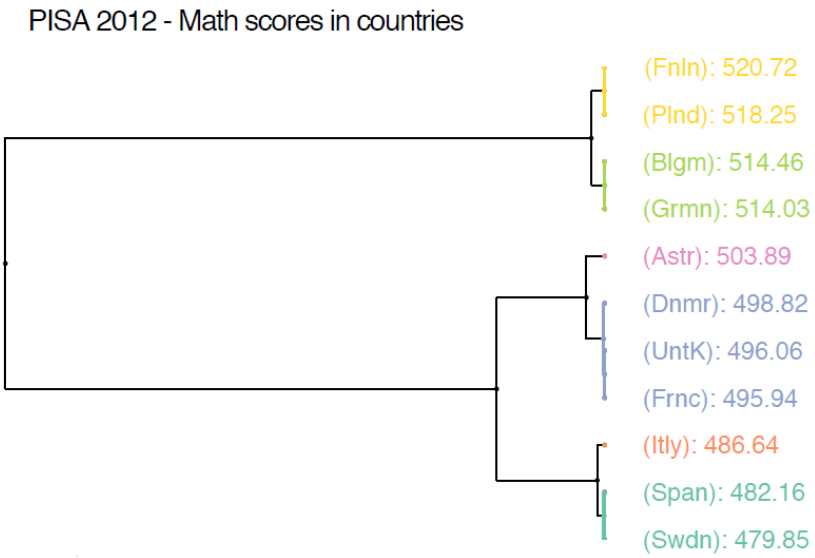


Figure 3.5: Clustering for a country feature in PISA 2012 dataset (Sitko et al., 2019)[39]. Colours annotate groups of countries with similar averages. Numbers displayed on the right side of country names stand for country averages. The image was taken from (Sitko and Biecek, 2017)[38].

Let us consider X_j and its corresponding PD plot. As with changepoint analysis for continuous variables, we denote the average responses for relevant categories $(\hat{g}_j(z_r; f))_{r=1,\ldots,R}$ as $v = (v_1, \ldots, v_R)$. Since we are going to partition the values of the vector v on the basis of how close they are to each other, we need to first specify the measure of dissimilarity between any two objects.

Definition 3.3. The dissimilarity (distance) measure between any two objects v_a, v_b from the given set S is a function d such that:

- it is symmetric with regard to its arguments:

$$d(v_a, v_b) = d(v_b, v_a),$$

- it is non-negative:

$$d(v_a, v_b) \geq 0,$$

- it is equal to zero if and only if its arguments are equal:

$$d(v_a, v_b) = 0 \quad \iff \quad v_a = v_b,$$

- it fulfils the triangle inequality:

$$\forall v_c \in S \qquad d(v_a, v_b) \leq d(v_a, v_c) + d(v_c, v_b).$$

There are many functions that fulfil the conditions required in 3.3 and thus can be used as dissimilarity measures. But since the elements of the vector v are 1-dimensional numeric scalars, we will use the squared Euclidean distance (identical to the squared difference in our case) which for objects v_a and v_b will be denoted as d_{ab}:

$$d_{ab} = (v_a - v_b)^2 . \tag{3.29}$$

Let T be a total sum of dissimilarities between all pairs of observations:

$$T = \frac{1}{2} \sum_{a=1}^{R} \sum_{b=1}^{R} d_{ab}. \tag{3.30}$$

For a fixed number of clusters K we also define sum of dissimilarities within groups and sum of dissimilarities between points belonging to two different classes, respectively:

$$\begin{aligned} W(C) &= \frac{1}{2} \sum_{k=1}^{K} \sum_{C(a)=k} \sum_{C(b)=k} d_{ab}, \\ B(C) &= \frac{1}{2} \sum_{k=1}^{K} \sum_{C(a)=k} \sum_{C(b)\neq k} d_{ab}. \end{aligned} \tag{3.31}$$

where function C maps an object to the index of cluster it belongs to. So if the number of clusters K is known beforehand, then the cluster analysis boils down to the well-defined optimization task of minimizing $W(C)$ or, equivalently, maximizing $B(C)$ (since $T = W(C) + B(C)$ holds and T is invariant with regard to the clustering). But in practical applications we do not know how many groups our data organizes into, so in the next subsections we are going to deal with this problem.

3.4.2. Hierarchical clustering

Hierarchical clustering methods do not depend on the choice of the number of clusters. However, they demand for the measure of dissimilarity between whole sets of objects (not between

single observations as it was defined in 3.3). They may be divided into two basic strategies: agglomerative (bottom-up) and divisive (top-down). In this paper we will focus only on the agglomerative approach which starts with each of the R objects representing a single cluster. Then at each of the $R-1$ consecutive steps the least dissimilar groups (dissimilar with regard to the chosen measure) are combined together, reducing the number of clusters by one. In the end of the procedure there is one big cluster containing all observations from the set.

The merging path in hierarchical clustering is largely determined by the choice of the dissimilarity measure used to describe how close two groups of objects A and B are to each other. This measure depends on the measure for comparing single observations $d(v_a, v_b)$ (which we introduced in definition 3.3) and will be denoted as $D(A, B)$. The following dissimilarities functions for groups are the ones that are widely used:

- single linkage - focuses on objects from two groups that are the closest pair:

$$D_{single}(A, B) = \min_{v_a \in A, v_b \in B} d_{ab},$$

- complete linkage - focuses on objects from two groups that are the furthest pair:

$$D_{complete}(A, B) = \max_{v_a \in A, v_b \in B} d_{ab},$$

- average linkage - takes into account all intergroup pairs:

$$D_{average}(A, B) = \frac{1}{N_A N_B} \sum_{a \in A} \sum_{b \in B} d_{ab},$$

  where N_A and N_B correspond to the numbers of objects in each of the two clusters.

The ordered sequence of groupings produced by an agglomerative method may be illustrated by a rooted binary tree called dendrogram. The R terminal nodes represent singleton clusters (containing single observations) which we have at the beginning of the agglomerative method, then each internal node corresponds to the cluster that was created at some point from two smaller groups represented by the children of the considered node, and eventually the root of the tree reflects the final situation with one big cluster consisting of all objects from the set. The plot below presents an example of dendrograms for three different dissimilarity measures applied to the same dataset.

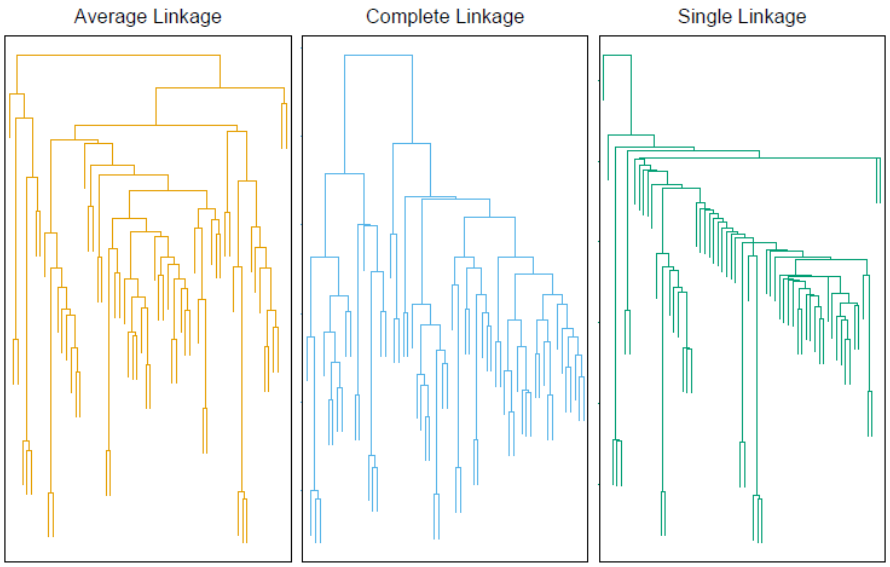


Figure 3.6: Dendrograms for agglomerative hierarchical clustering. Grouping paths are highly influenced by the choice of the dissimilarity measures. The image was taken from (Hastie et al., 2009)[23].

In general, there is no one single dissimilarity measure that produces the optimal merging path for an arbitrary dataset. In this paper, we consider the complete linkage as default but allow other as well.

3.4.3. Gap statistic

Hierarchical approach to the clustering problem is quite convenient since it provides us with the entire grouping path and allows the user to choose the most „natural" partition by picking up one of the R number of clusters. However, in our problem we want this number to be chosen automatically so there is no need to specify it in advance every time we deal with a categorical feature. So far, the universal algorithm for this task does not exist, but in this paper we will stick to the widely used method proposed by Tibshirani et al. (2001)[40] that is based on the **gap statistic**.

Let us assume that the hierarchical clustering for our vector v has been performed and we posses its entire grouping path. Let W_K denote the within-group dissimilarity $W(C)$ (3.31) calculated for the grouping C corresponding to the number of clusters equal to K. We compute all W_K through the merging path, obtaining $(W_1, W_2, \ldots, W_R)$ which is the sequence of decreasing values. Let K^* stand for the actual number of the distinct groupings. The idea of the gap statistic method is based on two following observations:

- for $K < K^*$ at least one cluster is composed of multiple true groups - so for $K < K^*$ each increase in the number of groups results in successive separation of the natural clusters and thus is likely to make W_K decrease substantially, i.e. $W_{K+1} \ll W_K$,

- for $K > K^*$ each additional partition splits a natural grouping consisting of similar objects into two subgroups and for this reason is likely to cause a smaller decrease of W_K.

Hence, it can be expected that:

$$(W_K - W_{K+1}|K < K^*) \gg (W_K - W_{K+1}|K \geq K^*)\,. \tag{3.32}$$

So estimating K^* boils down to finding the point from which differences $W_K - W_{K+1}$ decrease much slowly. Tibshirani et al. (2001)[40] suggest comparing $\log W_K$ curve to the curve of its expected value calculated for observations from the uniform distribution over an interval containing values $v_1, \ldots, v_R$. So if we define:

$$Gap(K) = \mathbb{E}\left[\log W_K\right] - \log W_K, \tag{3.33}$$

where the expectation $\mathbb{E}\left[\log W_K\right]$ is computed for the reference uniform distribution, then the estimator of K^* is the argument maximizing the Gap function.

In practical applications, $\log W_K$ for the reference distribution is calculated from a hierarchical clustering performed for a sample $V_1^*, \ldots, V_R^* \sim U([\min(v), \max(v)])$. And then the expected value $\mathbb{E}\left[\log W_K\right]$ is approximated by averaging over B copies of such a $\log W_K$ values:

$$\mathbb{E}\left[\log W_K\right] = \frac{1}{B}\sum_{b=1}^{B} \log W_{Kb}^*. \tag{3.34}$$

Let sd_K denote the standard deviation of $\log W_K^*$ over the all B repetitions. Accounting for the simulation error, we introduce the following correction: $s_K = \sqrt{\left(1 + \frac{1}{B}\right)} \cdot sd_K$ and then the final estimator of K^* is:

$$K^* = \underset{K \in \{2,\ldots,R-1\}}{\arg\min} \left(Gap(K) \geq Gap(K+1) - s_{K+1}\right). \tag{3.35}$$

We disregard cases $K = 1$ and $K = R$, since replacing all values with one category or leaving all original levels unchanged would be of no benefit in our problem. If there is no K satisfying the condition $Gap(K) \geq Gap(K+1) - s_{K+1}$, then we take $K^* = R - 1$ (merging two closest groups). For features with less than three categories clustering is not performed.

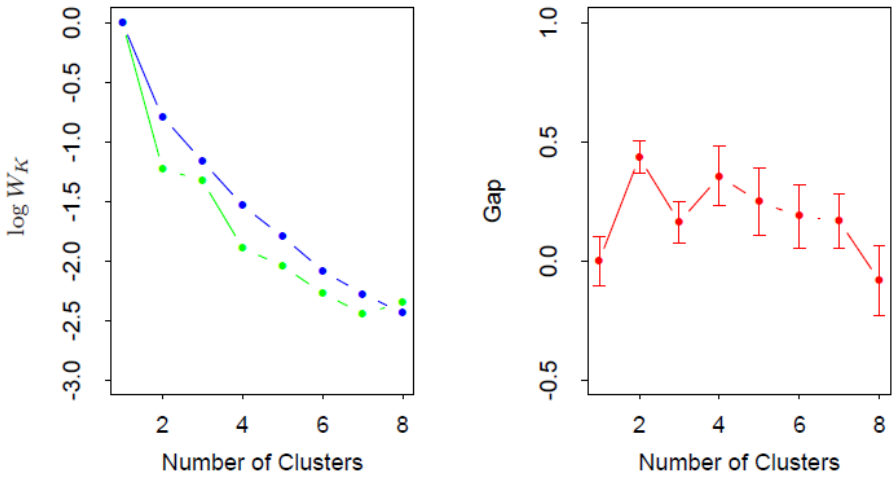


Figure 3.7: Choosing optimal number of clusters based on the gap statistic. On the left: $\log W_K$ values, green for the observed ones and blue for the ones expected with regard to the uniform distribution. On the right: values of the gap function for different clusters numbers. In this example $K^* = 2$. The images were taken from (Hastie et al., 2009)[23].

3.5. Feature interactions

In the previous sections we discussed the effect of one single feature. However, very often this effect depends on the values of other variables and thus aggregating all the single feature effects will not allow us to obtain the final prediction. In that situation we are dealing with *feature interactions*.

We demonstrate the problem on the example of two features taken from (Molnar, 2019)[35] in which the value of a house is predicted based only on its size and location.

location	size	prediction
good	big	**300**
good	small	200
bad	big	250
bad	small	150

location	size	prediction
good	big	**400**
good	small	200
bad	big	250
bad	small	150

The prediction for the table on the left can be expressed as follows:

$$\hat{y}_L = 150 + 50 \cdot \mathbb{1}\left(location = good\right) + 100 \cdot \mathbb{1}\left(size = big\right).$$

Thus, it may be decomposed into three parts: a constant term (150) and two components,

each for one feature. There is no interaction here and the model is additive, since the output is the sum of all single feature effects. Moving a house into a good location increases its value by 50, no matter how big it is, and we observe the analogous behaviour for the size feature.

The situation related to the right table is different, however. We have to introduce here an extra term accounting for the fact that for houses characterized by both the good location and the big size the prediction is larger by 100 in comparison to the previous additive model:

$$\hat{y}_R = 150 + 50 \cdot \mathbb{1}\left(location = good\right) + 100 \cdot \mathbb{1}\left(size = big\right) + 100 \cdot \mathbb{1}\left(location = good, size = big\right).$$

And now, moving a house into a good location increases its value by either 50 or 150, depending on its size.

The problem of detecting interactions has been studied broadly, yet there is no universal technique allowing to extract them from a model. In this paper we propose a simple method that is inspired by the work of Fisher et al. (2018)[13] and Gosiewska and Biecek (2019)[18].

The main idea is based upon the fact that the interaction can be regarded as the change in prediction that is left after taking into account all the individual feature effects. Let as pick up a single observation from the dataset $x^{(i)}$ and consider its j_1-th and j_2-th features. Let us assume that we want to check how the prediction for $x^{(i)}$ will change if we set $x^{(i)}_{j_1} = c_1$ and $x^{(i)}_{j_2} = c_2$ for some chosen constants c_1 and c_2. If $\Delta^{(i)}_j(c)$ denotes the change of prediction after setting $x^{(i)}_j = c$, then:

$$\begin{aligned}
\Delta^{(i)}_{j_1}(c_1) &= f\left(x^{(i)}_{j_1} = c_1, x^{(i)}_{-j_1}\right) - \hat{y}^{(i)}, \\
\Delta^{(i)}_{j_2}(c_2) &= f\left(x^{(i)}_{j_2} = c_2, x^{(i)}_{-j_2}\right) - \hat{y}^{(i)}, \\
\Delta^{(i)}_{j_1,j_2}(c_1, c_2) &= f\left(x^{(i)}_{j_1} = c_1, x^{(i)}_{j_2} = c_2, x^{(i)}_{-\{j_1,j_2\}}\right) - \hat{y}^{(i)}.
\end{aligned} \tag{3.36}$$

Then, if the model is additive and there is no interaction between X_{j_1} and X_{j_2}, the change in prediction triggered by the pair of variables is equal to the sum of changes caused by these features separately:

$$\Delta^{(i)}_{j_1,j_2}(c_1, c_2) = \Delta^{(i)}_{j_1}(c_1) + \Delta^{(i)}_{j_2}(c_2). \tag{3.37}$$

In practice, checking all possible replacement values c_1 and c_2 is unrealistic. Therefore, the permutations within j_1-th and j_2-th columns in the dataset are performed. As a result, each observation gets a new value of X_{j_1} and a new value of X_{j_2}. Then, for all instances in the dataset differences $diff^{(i)}_{j_1,j_2} = \Delta^{(i)}_{j_1,j_2} - \left(\Delta^{(i)}_{j_1} + \Delta^{(i)}_{j_2}\right)$ are computed. In case of an additive model they should be close to zero. Hence, if there are many big differences, that indicates the presence of the interaction.

In practical applications, one should also determine which interactions are significant and which may be negligible. We propose the following approach (with some fixed parameters $\alpha_1 > 0$ and $\alpha_2 \in [0, 1]$):

- First, at an instance level, the difference $diff^{(i)}_{j_1,j_2}$ between change caused by the pair and sum of changes caused by single features is considered as substantial if:

$$|diff^{(i)}_{j_1,j_2}| \geq \alpha_1 \cdot |\Delta^{(i)}_{j_1}(c_1) + \Delta^{(i)}_{j_2}(c_2)|.$$

- Second, at a general level, the interaction between X_{j_1} and X_{j_2} is regarded as meaningful if the percentage of significant differences from the previous point is not less than the threshold α_2.

Having established for which pairs of variables there may be an interaction, we construct additional categorical features. For this purpose, we use transformed versions of features (obtained either by changepoint discretization or hierarchical clustering). If transformed variables X_{j_1} and X_{j_2} have, accordingly, l_1 and l_2 levels, then their interaction categorical feature has $l_1 \cdot l_2$ levels which correspond to the Cartesian product of X_{j_1} levels and X_{j_2} levels.

The presented method can be extended to work with interactions of higher orders (between more than just two features), but in this paper we focus only on pairs of variables.

4. The rSAFE package

The **rSAFE** package (**Surrogate-Assisted Feature Extraction in R**) is a model-agnostic tool implemented in R that covers all methods described in chapter 3. Its goal is to make an interpretable glass-box model more accurate using alternative black-box model called a surrogate. Based on the complicated model, such as a neural network or a random forest, new features are being extracted and then used in the process of fitting a simpler interpretable model, improving its overall performance.

The package consists of multiple functionalities. Main functions (available for the user and essential in terms of working with the package) all starts with the „safe", as opposed to the other auxiliary functions. In this chapter we will describe them in detail and illustrate the way of dealing with **rSAFE** on the example of an `apartments` dataset (section 2.2).

The **rSAFE** package can be installed from a GitHub repository using the **devtools** package (Wickham et al., 2019)[45]:

```
install.packages("devtools")
devtools::install_github("ModelOriented/rSAFE")
```

Since the `apartments` dataset is available in **rSAFE**, we can get access to it by loading the package:

```
library(rSAFE)
```

A few first rows of the dataset are presented in figure 4.1. The dataset contains the response variable `m2.price` and five explanatory variables. More details on the features can be found in section 2.2.

Our aim is to predict the price per square meter of an apartment based on the features included in the table. It is the regression task because prices are measured on a continuous scale. Thus, first thing we do is choosing an appropriate regression model - in this example a random forest (Breiman et al., 2018)[10] with default parameters will be used.

```
library(randomForest)
set.seed(111)
```

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

Figure 4.1: First rows of the *apartments* dataset. The data contains one dependent variable *m2.price* and five explanatory features.

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

Random forest can be a really powerful tool but as a complex, ensemble algorithm it makes its predictions hard to explain and understand by a human. Hence, we are going to use it only as a surrogate which provides us with valuable information that can be passed on to a simpler model.

In order to extract the information from the random forest that has been just fitted we create an `explainer` object introduced in section 3.1.

```
library(DALEX)
explainer_rf <- explain(model_rf)
```

And now we are ready to start using the **rSAFE** package.

4.1. Function safely_detect_changepoints()

The `safely_detect_changepoints()` function provides the solution to the changepoint detection problem presented in section 3.3. It calculates the optimal positioning and number of breakpoints for given data using the PELT algorithm (subsection 3.3.3) with the non-parametric cost function (subsection 3.3.2). Although this function need not be used explicitly while working with **rSAFE**, it has been decided to give it the „safely" prefix and make it accessible to users. The implementation of the functionality was inspired to a large extent by the code written by Killick (2019)[28]. Unfortunately, her **changepoint.np** package cannot be used directly because it has been removed from the CRAN repository.

The function takes a numeric vector as an argument and returns the sequence of its (sorted) changepoint positions. As in subsection 3.3.1, the resulting vector $t = (t_1, \ldots, t_M)$ consists of indexes corresponding to segment ends (excluding the end of the last interval which is always determined), for example:

```
data <- c(2,2,2,2,2,7,7,7,7,4,4,4,4,4,4,4,4)
safely_detect_changepoints(data)
```

gives us the following output: `(5,9)`.

The method offers two optional arguments. Firstly, one can specify a penalty $\gamma > 0$ for adding another changepoint (3.18). There are four standard penalties available (K denotes the sequence length):

- AIC: penalty$= 2$,
- BIC/SIC: penalty$= \log K$,
- MBIC: penalty$= 3 \cdot \log K$ (Zhang and Siegmund, 2007)[47],
- Hannan-Quinn: penalty$= 2 \cdot \log(\log K)$.

or you can pass any non-negative numeric value (default is „MBIC"). Moreover, we can change the number of quantiles S used in the approximation 3.17 via the `nquantiles` parameter.

4.2. Function safely_transform_continuous()

The `safely_transform_continuous()` function is designed for dealing with continuous features in a dataset. It uses the partial dependence function $\hat{g}_j^{PD}$ (subsection 3.2.1) or the accumulated local effects function $\hat{g}_j^{ALE}$ (subsection 3.2.2) and applies them to the black-box model in order to assess the relationship between a variable and the response. Once done, the y-axis values $(\hat{g}_j(z_j^k; f))_{k=1,\ldots,K}$ from the obtained PD/ALE plot are treated as a time series and the changepoint detection algorithm (section 3.3) is performed resulting in a set of indexes $t = (t_1, \ldots, t_M)$ where these values are estimated to change somehow.

The main arguments of the function are an `explainer` object and a name of the feature we want to find the transformation for. By default, the ALE plot is constructed but we can switch to the PD version via the `response_type` argument. Also, the grid size for the plot can be modified - either increased to boost the accuracy or decreased to accelerate the calculations. Since the `safely_transform_continuous()` function makes use

of the `safely_detect_changepoints()` function (section 4.1), all arguments of the latter are also available to adjust. If no breakpoints have been found by the PELT algorithm, then the range of the feature is split into `no_segments` segments on the basis of sample quantiles.

Let us go back to the `apartments` example. So far, we have created the `explainer_rf` object that stores the information about the fitted `model_rf`. Now we can use it to obtain the transformation for a continuous variable from our dataset, e.g. for `construction.year` run:

```
safely_transform_continuous(explainer_rf, "construction.year")
```

This command returns a list of three elements:

- `sv` - a data frame with information on how the single variable influences the output, it can be used later in constructing the PD/ALE plots,
- `break_points` - a vector of interval ends suggested by the PELT algorithm; these are not the indexes $(t_1, \ldots, t_M)$ but their corresponding values from the variable range;
  in our case: `(1933,1949,1983)`,
- `new_levels` - names for $(M + 1)$ levels of the new categorical feature, each level is called after the relevant interval;
  in our case: `("(-Inf,1933]","(1933,1949]","(1949,1983]","(1983,Inf)")`.

As described in section 2.2, `construction.year` is a binary variable indicating whether the flat was built before 1935 or after 1995. That refers to the three intervals: $(-Inf, 1935), [1935, 1995], (1995, Inf)$. So we can see that the changepoint method produces one segment more. That can be changed via the `penalty` argument – calling e.g.:

```
safely_transform_continuous(explainer_rf, "construction.year",
        penalty = 25)
```

gives the following partition: `"(-Inf,1936.49]","(1936.49,1993]","(1993,Inf)"`. Hence, carefully chosen penalty argument can lead to maybe not the exact points of change but their quite good estimates.

4.3. Function safely_transform_categorical()

The `safely_transform_categorical()` function is the equivalent of the `safely_transform_continuous()` (4.2) for categorical features. Firstly, average predictions for all levels are assessed through the partial dependence function $\hat{g}_j^{PD}$

(subsection 3.2.1) constructed for the black-box model. And then, the categories are grouped using the hierarchical clustering algorithm (subsection 3.4.2) and the optimal number of clusters is chosen via the gap statistic method (subsection 3.4.3).

Similarly to `safely_transform_continuous()`, an `explainer` and a feature name are the mandatory arguments. Furthermore, one can set out an agglomeration method related to the dissimilarity measure (subsection 3.4.2) which is used while merging groups together. Since `safely_transform_categorical()` uses the `stats::hclust` function, any method available there can be used: `"ward.D"`, `"ward.D2"`, `"single"`, `"complete"`, `"average"`, `"mcquitty"`, `"median"`, `"centroid"` (details of the methods can be found in (Murtagh and Contreras, 2011)[36] and (Murtagh and Legendre, 2014)[37]). We can also specify how many samples are used in the estimation 3.34 needed in the gap statistic method.

The only categorical feature in the `apartments` dataset is `district` - we will use it to demonstrate the functionality:

```
safely_transform_categorical(explainer_rf, "district")
```

The code above produces a list of two elements: `clustering`, being the output of the `stats::hclust` function and used later to draw a dendrogram, and `new_levels` which is a data frame showing how original levels have been mapped to new ones. In our example it has the following form:

```
      district                              district_new
1       Bemowo Bemowo_Bielany_Praga_Ursus_Ursynow_Wola
2      Bielany Bemowo_Bielany_Praga_Ursus_Ursynow_Wola
3        Ursus Bemowo_Bielany_Praga_Ursus_Ursynow_Wola
4      Ursynow Bemowo_Bielany_Praga_Ursus_Ursynow_Wola
5        Praga Bemowo_Bielany_Praga_Ursus_Ursynow_Wola
6         Wola Bemowo_Bielany_Praga_Ursus_Ursynow_Wola
7     Zoliborz                 Mokotow_Ochota_Zoliborz
8      Mokotow                 Mokotow_Ochota_Zoliborz
9       Ochota                 Mokotow_Ochota_Zoliborz
10 Srodmiescie                              Srodmiescie
```

The left column consists of all original categories whereas the right one contains their new equivalents indicated by the clustering algorithm. We can see that $K^* = 3$ groups have been selected: the city centre ("Srodmiescie"), districts in close vicinity to the centre and well communicated with it ("Mokotow", "Ochota", "Zoliborz") and others. The result is in line with expectations since it is consistent with true relationship between `district` and `m2.price` described

in section 2.2. It is worth mentioning that if there are less than three categories, the clustering procedure is redundant and thus not performed.

4.4. Function safely_detect_interactions()

The `safely_detect_interactions()` function helps us to identify interactions between pairs of variables via the approach described in section 3.5. It iterates over all possible pairs, for each performing permutations within relevant columns and calculating corresponding changes in prediction (3.36). As a result, we get those pairs for which the interaction is strong enough to be considered significant.

Let us use this function for the `apartments` dataset:

```
safely_detect_interactions(explainer_rf)
```

In this particular example the function returns `NULL` as it has not identified any second order interaction. That is desired behaviour since the data was generated such that a response value is a sum of individual feature effects.

In the implemented algorithm parameters α_1 and α_2 (section 3.5) play an important role and have an impact on the obtained set of feature pairs. They can be adjusted through arguments `inter_param` and `inter_threshold`, respectively (by default, both are equal to 0.25). If we wish to get all the possible pairs, setting `inter_threshold = 0` will do the work:

```
safely_detect_interactions(explainer_rf, inter_threshold = 0)
```

Now we obtain the following data frame:

```
           variable1 variable2 strength
1              floor  no.rooms    0.130
2            surface     floor    0.106
3            surface  no.rooms    0.106
4  construction.year   surface    0.103
5  construction.year     floor    0.095
6  construction.year  no.rooms    0.089
7           no.rooms  district    0.060
8  construction.year  district    0.059
9              floor  district    0.050
10           surface  district    0.041
```

Since there are five explanatory features in our dataset, we have $\frac{5 \cdot 4}{2} = 10$ pairs. They are sorted by the last column named `strength` which specifies the percentage of significant instance-level differences $diff_{j_1,j_2}^{(i)}$ and in a way measures the interaction strength.

4.5. Function safe_extraction()

The methods for dealing with a single feature (continuous or categorical) have been presented in section 4.2 and section 4.3. But datasets may contain tens or even hundreds of variables and analyzing each of them separately would be tedious and impractical. For this reason, the **rSAFE** package provides the `safe_extraction()` function that iterates over all the features and based on their type calls either `safely_transform_continuous()` or `safely_transform_categorical()`. Transformations for all variables are systematically saved and at the end returned in a large object of class `safe_extractor`. This is the core function of the **rSAFE** package and should be used directly.

The function involves assessing single feature effects as well as detecting pairs that interact with each other. Therefore, all arguments of the `safely_transform_continuous()`, `safely_transform_categorical()` and `safely_detect_interactions()` are available here.

Let us create an `safe_extractor` object for the `apartments` data, setting `penalty = 25` as in section 4.2:

```
safe_extractor_rf <- safe_extraction(explainer_rf, penalty = 25)
```

We can print its summary by running `print(safe_extractor_rf)`:

```
Variable 'construction.year' - selected intervals:
(-Inf, 1936.49]
(1936.49, 1993]
(1993, Inf)
Variable 'surface' - selected intervals:
(-Inf, 47.87755]
(47.87755, 105]
(105, Inf)
Variable 'floor' - selected intervals:
(-Inf, 6]
(6, Inf)
```

```
Variable 'no.rooms' - selected intervals:
(-Inf, 3]
(3, Inf)
Variable 'district' - created levels:
Bemowo, Bielany, Ursus, Ursynow, Praga, Wola ->
        Bemowo_Bielany_Praga_Ursus_Ursynow_Wola
Zoliborz, Mokotow, Ochota ->  Mokotow_Ochota_Zoliborz
Srodmiescie ->  Srodmiescie
```

We can see propositions of transformation for all five features. Each of the continuous variables has been split into two or three intervals and the only categorical variable has been grouped into three new levels. The `safe_extractor_rf` object is composed of three elements:

- `explainer` - this is our `explainer_rf` passed as an argument to the function,
- `interaction_effects` - this is a data frame with pairs of variables returned by `safely_detect_interactions()`, in our case it is `NULL`,
- `variables_info` - this is a list containing information returned by all calls of `safely_transform_continuous()` and `safely_transform_categorical()` functions. e.g. executing:

  ```
  safe_extractor_rf$variables_info$floor
  ```

  results in:

  ```
  $`type`
  [1] "continuous"


  $sv
  Top profiles    :
  _vname_      _label_ _x_     _yhat_ _ids_
  floor.randomForest.1   floor randomForest   1    0.00000     0
  floor.randomForest.2   floor randomForest   2  -71.64424     0
  floor.randomForest.3   floor randomForest   3 -143.88893     0
  floor.randomForest.4   floor randomForest   4 -182.49439     0
  floor.randomForest.5   floor randomForest   5 -299.84194     0
  floor.randomForest.6   floor randomForest   6 -356.01271     0
  ```

```
$break_points
[1] 6

$new_levels
[1] "(-Inf, 6]" "(6, Inf)"
```

For continuous features we can visualize their ALE plots (or PD plots, if changed so via the `response_type` argument):

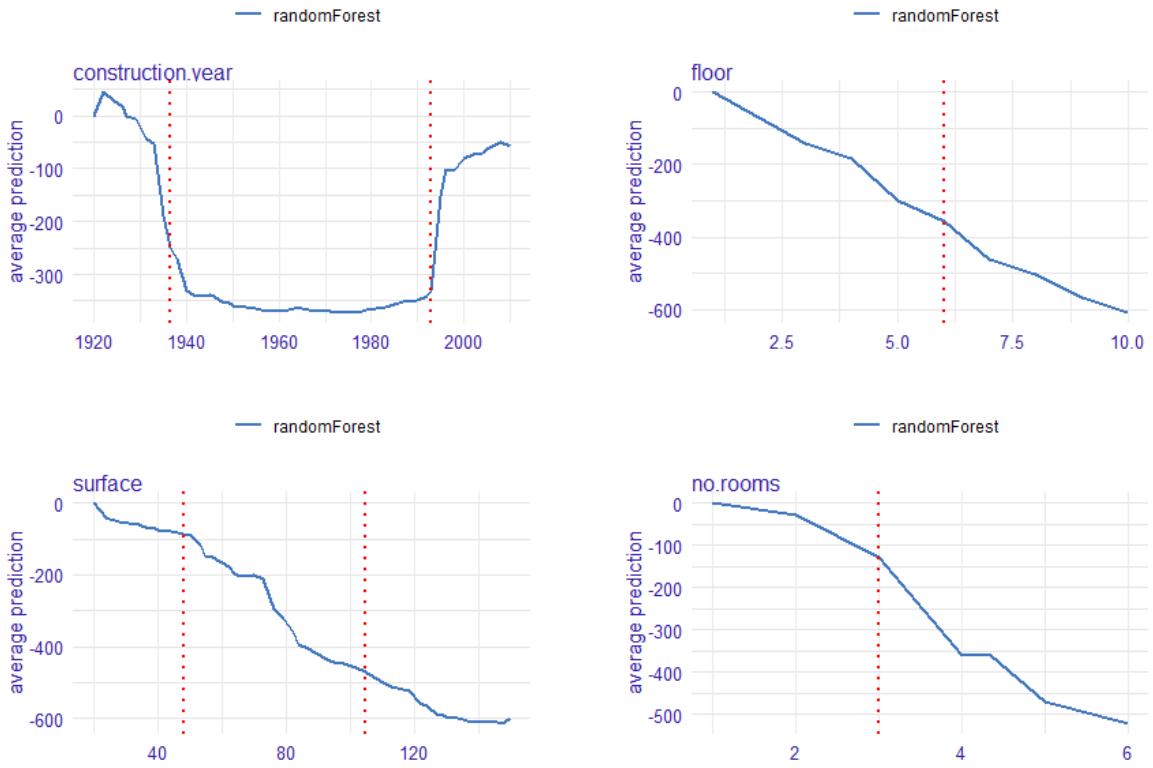


Figure 4.2: ALE plots for all four continuous features in `apartments` dataset. Each plot presents the effect a particular feature has on the response (blue solid line) together with the breakpoints chosen by the PELT algorithm (red dashed line).

by running:

```
plot(safe_extractor_rf, variable = "construction.year")
plot(safe_extractor_rf, variable = "surface")
plot(safe_extractor_rf, variable = "floor")
plot(safe_extractor_rf, variable = "no.rooms")
```

If we are interested in categorical features, **rSAFE** provides us with plots of merging paths

produced by the hierarchical clustering:

```
plot(safe_extractor_rf, variable = "district")
```

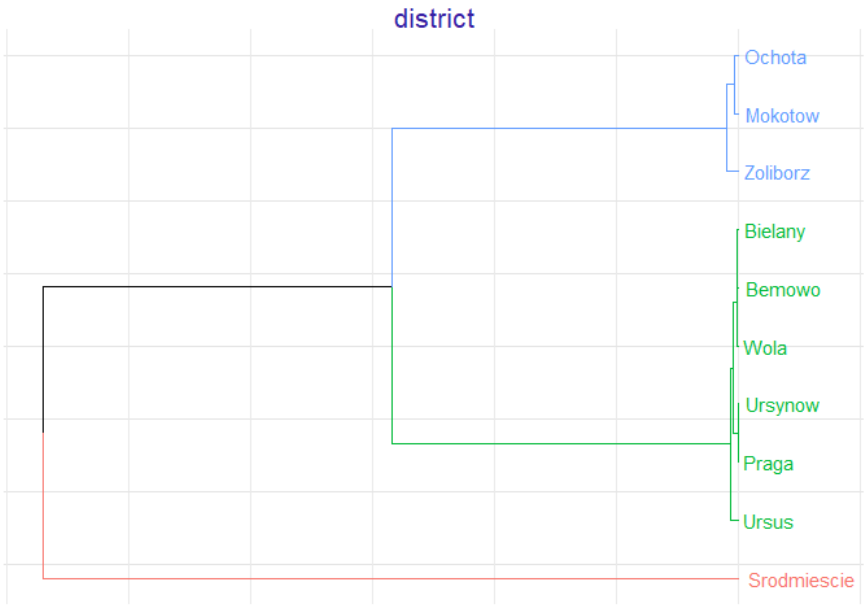


Figure 4.3: Merging path plot for `district` feature from `apartments` dataset. We can observe in which order levels have been merged. Colours indicate the optimal clustering.

4.6. Function safely_transform_data()

So far, we have presented the functionalities that are able to use a model to distil the valuable knowledge about the features, save it in a structured way and visualize it via quite intuitive graphics. But what we are really striving for is applying all those transformations from previous sections to a dataset so that new variables will be created and we will might use them in the process of fitting another model. And this can be done thanks to the `safely_transform_data()` function. It extracts the information from the passed `safe_extractor` object and for a specified dataset it adds new columns which are transformations of original features. For a continuous variable breakpoints are used to partition the range of its values into segments and each of the original values on a continuous scale is mapped to its corresponding interval. For a categorical feature original levels are replaced by relevant new categories coming from the clustering algorithm.

Transformation of the `apartments` data may be performed as follows:

```
apartments_new <- safely_transform_data(safe_extractor_rf,
      data = apartments)
```

The `apartments_new` object is a data frame containing eleven columns: the response feature `m2.price`, five original explanatory features and their five transformations. For example, the first row consists of following values:

```
m2.price              "5897"
construction.year     "1953"
surface               "25"
floor                 "3"
no.rooms              "1"
district              "Srodmiescie"
construction.year_new "(1936.49, 1993]"
surface_new           "(-Inf, 47.87755]"
floor_new             "(-Inf, 6]"
no.rooms_new          "(-Inf, 3]"
district_new          "Srodmiescie"
```

It can be observed that each created variable has a name that corresponds to the name of the original feature with sufix „_new" added.

4.7. Function safely_select_variables()

In general, the SAFE algorithm ends with the step of creating new features described in section 4.6. But for some variables the transformed form has the advantage over the original one whereas for others applying a transformation may only worsen the model performance and genuine values are preferred. Since transformed and original forms are highly correlated, most of the models are likely to have troubles handling them all. For this reason, the **rSAFE** package contains one more function - `safely_select_variables()` - that helps to assess which form of the feature to choose.

The `safely_select_variables()` function takes as an argument a dataset with already transformed features and returns a vector of p variable names, for each single feature proposing either its new or original form. Starting from the set containing only genuine features, it iterates across them all and at every turn checks whether applying the transformation to the currently considered variable improves results. The comparison is carried out by fitting two models -

to the actual best set of predictors and to the same set with this particular feature replaced by its transformed form. Since the **rSAFE** methodology is intended to be used with simple explainable models, two such models are considered - for regression problem it is a linear model and for classification problem it is a logistic regression (in multiclass cases the class of interest has to be specified and then all other are combined together). So depending on the task, adequate models are fitted to different subsets of features and then their log-likelihood values are compared.

Running the following line of code:

```
safely_select_variables(safe_extractor_rf, data = apartments,
        which_y = "m2.price")
```

gives us the set of five feature names: `"surface"`, `"floor"`, `"no.rooms"`, `"construction.year_new"`, `"district_new"`. Hence, some of the features one should use in their original form and some after applying an appropriate transformation. Since the linear model has been used in feature selection, it is not surprising that for predictors having approximately linear effect on the response (`"surface"`, `"floor"`, `"no.rooms"`) the method suggests leaving their basic forms.

5. rSAFE - examples of usage

5.1. Regression task - apartments data

In chapter 4 the `apartments` dataset was used to illustrate the consecutive functionalities. Now we will use it to demonstrate the standard workflow and show that **rSAFE** can really help us improve model performance.

We will use both of the `apartments` and `apartmentsTest` datasets (section 2.2). In order to ensure the final errors are computed on the data which has not been seen by an appropriate model, we divide our data as follows:

- `data1` (`apartments`) – the data which initial black-box and glass-box models are fitted to,
- `data2` (rows $1 - 3000$ of `apartmentsTest`) – the data used to create `explainers` for both models from previous point (serves as a test data) and a `safe_extractor` for the black-box model only,
- `data3` (rows $3001 - 6000$ of `apartmentsTest`) – the data for which transformations and feature selection are performed, used also as a training set for new models,
- `data4` (rows $6001 - 9000$ of `apartmentsTest`) – the data used as a test set for the new models, allowing to compare the results.

```
library(rSAFE)
data1 <- apartments
data2 <- apartmentsTest[1:3000,]
data3 <- apartmentsTest[3001:6000,]
data4 <- apartmentsTest[6001:9000,]
```

Our goal is to predict the price per square meter of an apartment based on available explanatory variables. As in chapter 4, we use a random forest which will serve us as a surrogate model:

```
library(randomForest)
set.seed(111)
model_rf1 <- randomForest(m2.price ~ construction.year + surface +
      floor + no.rooms + district, data = data1)
```

Then we create an `explainer` for the fitted model. Since we are interested in results for the test data not used during training process, we pass this data via `data` and `y` arguments.

```
library(DALEX)
explainer_rf1 <- explain(model_rf1, data = data2[,2:6],
      y = data2[,1], label = "rf1")
```

The `explainer_rf1` object may now be used for feature extraction. We increase the penalty for a single changepoint as in chapter 4.

```
safe_extractor <- safe_extraction(explainer_rf1, penalty = 25)
```

```
Variable 'construction.year' - selected intervals:
(-Inf, 1937]
(1937, 1994]
(1994, Inf)
Variable 'surface' - selected intervals:
(-Inf, 47]
(47, 101]
(101, Inf)
Variable 'floor' - selected intervals:
(-Inf, 5]
(5, Inf)
Variable 'no.rooms' - selected intervals:
(-Inf, 3]
(3, Inf)
Variable 'district' - created levels:
Bemowo, Bielany, Ursus, Ursynow, Praga, Wola ->
        Bemowo_Bielany_Praga_Ursus_Ursynow_Wola
Zoliborz, Mokotow, Ochota ->  Mokotow_Ochota_Zoliborz
Srodmiescie ->  Srodmiescie
```

The `safe_extractor` object provides us with the list of transformations which we can now apply to another dataset.

```
data3_trans <- safely_transform_data(safe_extractor, data3)
```

The `data3_trans` contains the dual set of predictors. We can reduce it doing feature selection.

```
selected_variables <- safely_select_variables(safe_extractor,
      data3_trans, which_y = "m2.price")
data3_trans_sel <- data3_trans[,c("m2.price", selected_variables)]
```

The final dataset after feature transformation and then selection looks like this:

m2.price	surface	floor	no.rooms	construction.year_new	district_new
3542	21	6	1	(1937, 1994]	Bemowo_Bielany_Praga_Ursus_Ursynow_Wola
5631	107	2	4	(1994, Inf)	Srodmiescie
2989	41	9	2	(1937, 1994]	Bemowo_Bielany_Praga_Ursus_Ursynow_Wola
3822	28	2	2	(1937, 1994]	Bemowo_Bielany_Praga_Ursus_Ursynow_Wola
2337	146	3	6	(1937, 1994]	Bemowo_Bielany_Praga_Ursus_Ursynow_Wola
3381	97	8	3	(1937, 1994]	Mokotow_Ochota_Zoliborz

Figure 5.1: First rows of the `apartments` dataset after applying feature transformations.

Having the dataset containing new predictors, we can use it to train a simple interpretable model. For this example we choose a linear model (categorical features are automatically one-hot encoded).

```
model_lm2 <- lm(m2.price ~ ., data = data3_trans_sel)
```

Since we want to find out how the feature transformations influence model predictive abilities, we also fit a linear model to the initial data and a random forest to the final one.

```
model_lm1 <- lm(m2.price ~ ., data = data1)
set.seed(111)
model_rf2 <- randomForest(m2.price ~ ., data = data3_trans_sel)
```

To compare the performance of four created models, we use the **DALEX** package. First, we have to create `explainers` for all of them (the one for initial random forest model has been already made). For models `model_rf2` and `model_lm2` we need to perform transformations on the independent dataset.

```
data4_trans <- safely_transform_data(safe_extractor, data4)
data4_trans_sel <- data4_trans[,c("m2.price", selected_variables)]
explainer_lm1 <- explain(model_lm1, data = data2[,2:6],
```

```
      y = data2[,1], label = "lm1")
explainer_rf2 <- explain(model_rf2, data = data4_trans_sel[,2:6],
      y = data4_trans_sel[,1], label = "rf2")
explainer_lm2 <- explain(model_lm2, data = data4_trans_sel[,2:6],
      y = data4_trans_sel[,1], label = "lm2")
```

We will look at residuals for each of the models using the `DALEX::model_performance()` function.

```
mp_lm1 <- model_performance(explainer_lm1)
mp_rf1 <- model_performance(explainer_rf1)
mp_lm2 <- model_performance(explainer_lm2)
mp_rf2 <- model_performance(explainer_rf2)
```

We can visualize residuals by plotting reversed empirical CDFs for their absolute values:

```
plot(mp_lm1, mp_rf1, mp_lm2, mp_rf2)
```

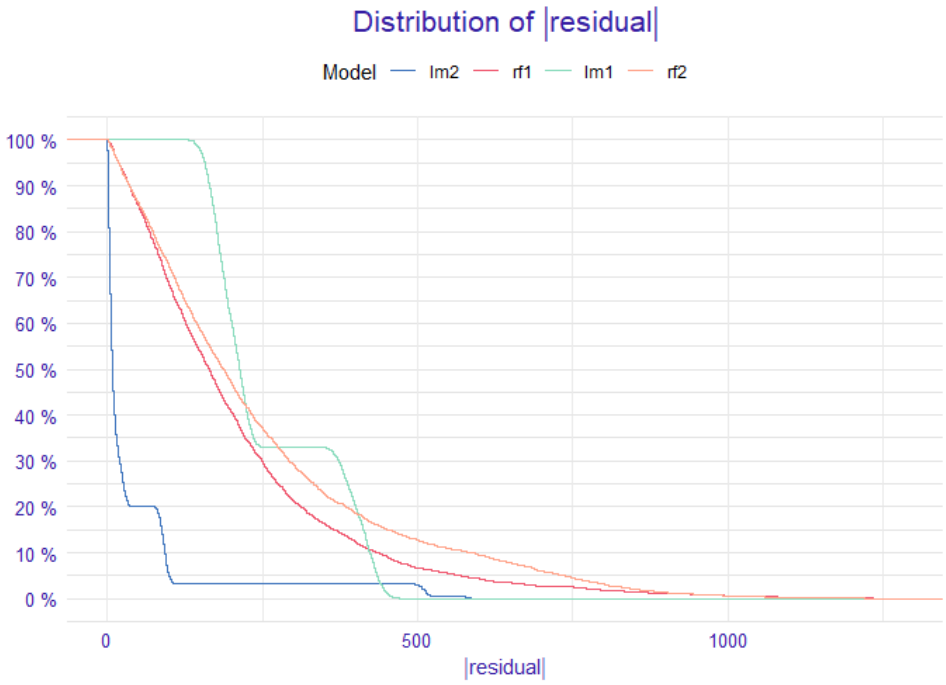


Figure 5.2: Reversed empirical CDFs for residuals in four fitted models: `model_lm1`, `model_rf1`, `model_lm2`, `model_rf2`.

Or we may create relevant boxplots for absolute values of residuals:

```
plot(mp_lm1, mp_rf1, mp_lm2, mp_rf2, geom = "boxplot")
```

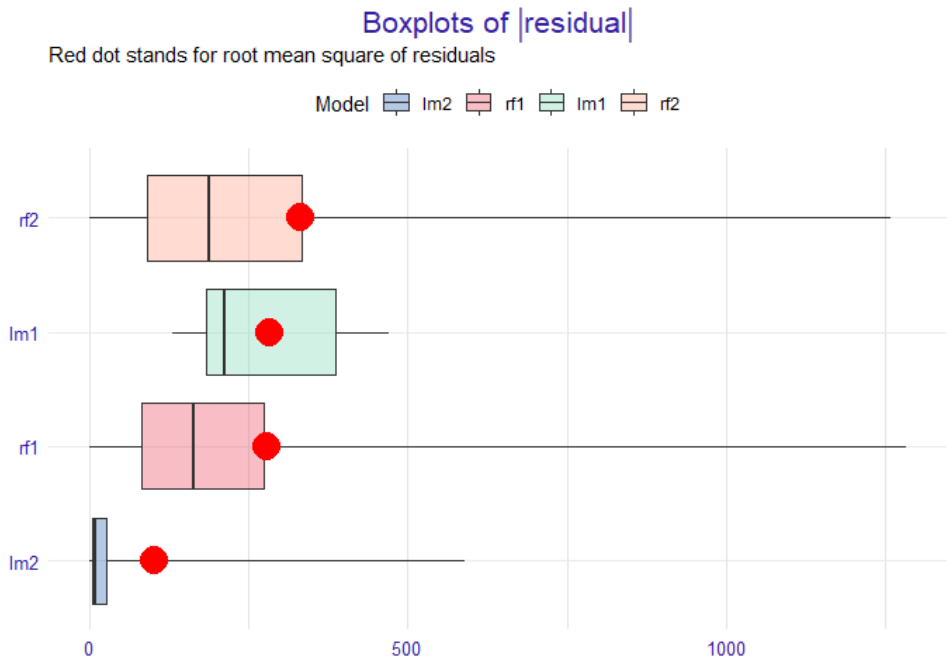


Figure 5.3: Boxplots for absolute values of residuals in four fitted models: `model_lm1`, `model_rf1`, `model_lm2`, `model_rf2`.

The random forest models have a similar errors distribution - and it is not surprising because the transformations that were applied to the data had been derived from `model_rf1` and they do not provide any additional information `model_rf2` cannot learn by itself. However, a significant difference can be observed within the linear models. The second model that was enriched with feature transformations has generally more accurate predictions than the first one fitted to the original data.

5.2. Classification task - HR data

In section 5.1 we presented the example of an application of the **rSAFE** package for the regression problem. Now we are going to show that the package may be helpful also in classification tasks.

For this example we use the `HR_data` set described in section 2.3. Let us remind how few first rows of this dataset look like.

```
library(rSAFE)
head(HR_data)
```

satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	Work_accident	left	promotion_last_5years	sales	salary
0.38	0.53	2	157	3	0	1	0	sales	low
0.80	0.86	5	262	6	0	1	0	sales	medium
0.11	0.88	7	272	4	0	1	0	sales	medium
0.72	0.87	5	223	5	0	1	0	sales	low
0.37	0.52	2	159	3	0	1	0	sales	low
0.41	0.50	2	153	3	0	1	0	sales	low

Figure 5.4: First rows of the *HR_data* dataset. The data contains one dependent variable *left* and eight explanatory features.

As opposed to the `apartments` data, now we do not know the true relations between particular features and the output so transformations we should obtain via the SAFE algorithm also remain unknown.

Our aim is to predict whether an employee will leave his/her workplace based on the information from the dataset. The response variable `left` takes two values, 0 and 1, and determines what happened to the particular employee. As explanatory variables we use all available in the dataset except for `sales` which specifies department in which the employee works for.

```
data <- HR_data[, colnames(HR_data) != "sales"]
```

The data consists of 14999 instances and, as in section 5.1, we split it into four subsets (shuffling rows first to ensure they are evenly distributed).

```
set.seed(111)
data <- data[sample(1:nrow(data)),]
data1 <- data[1:4000,]
data2 <- data[4001:8000,]
data3 <- data[8001:12000,]
data4 <- data[12001:14999,]
```

We start with the logistic regression model as a the model which is interpretable and simple enough.

```
model_lr1 <- glm(left ~ ., data = data1, family = binomial())
```

In this example we decide to use the xgboost model (Greenwell et al., 2019)[21] as a black-box.

```
n.trees <- 10000
library(gbm)
set.seed(111)
model_xgb1 <- gbm(left ~ ., data = data1,
        distribution = "bernoulli", n.trees = n.trees)
```

For creating the relevant explainer, we use another data. Furthermore, for classification problems we need to specify `predict_function` - a function that may be used for model predictions and returns a single numerical value for each observation.

```
library(DALEX)
predict_function <- function(model, x) predict(model, x,
      type = "response", n.trees = n.trees)
explainer_xgb1 <- explain(model_xgb1, data2[,-7], y = data2$left,
      predict_function = predict_function, verbose = FALSE)
```

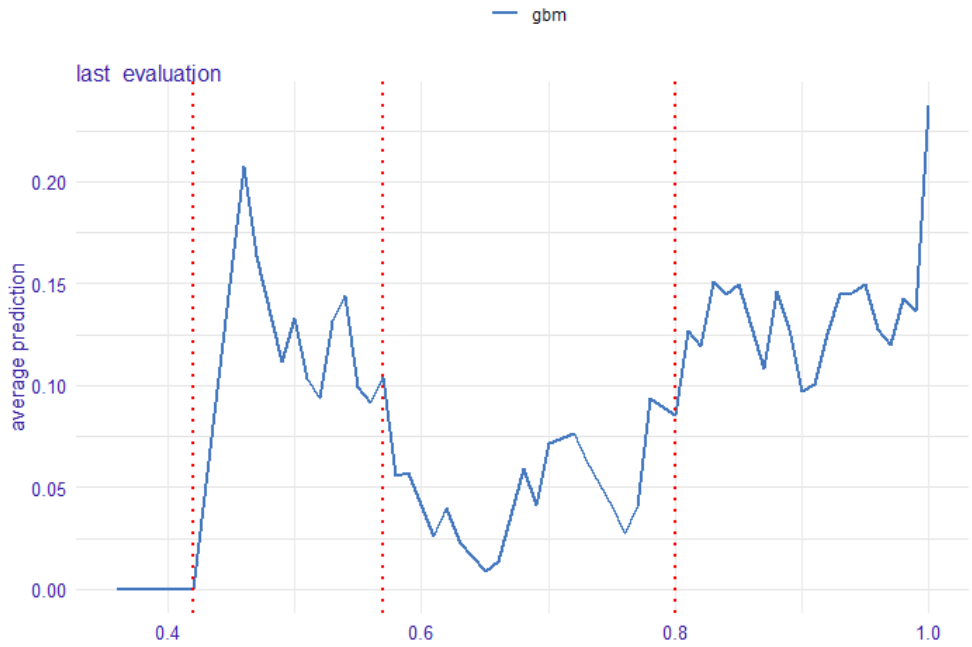
And now we are ready to start using the **rSAFE** package. We create a `safe_extractor` object for the `explainer_xgb1` with an adjustment for `penalty` argument:

```
safe_extractor <- safe_extraction(explainer_xgb1, penalty = 20)
```

```
Variable 'satisfaction_level' - selected intervals:
(-Inf, 0.45]
(0.45, 0.91]
(0.91, Inf)
Variable 'last_evaluation' - selected intervals:
(-Inf, 0.42]
(0.42, 0.57]
(0.57, 0.8]
(0.8, Inf)
Variable 'number_project' - selected intervals:
(-Inf, 4]
(4, Inf)
Variable 'average_montly_hours' - selected intervals:
(-Inf, 211]
(211, Inf)
Variable 'time_spend_company' - selected intervals:
(-Inf, 3]
(3, Inf)
Variable 'Work_accident' - no transformation suggested.
Variable 'promotion_last_5years' - no transformation suggested.
Variable 'salary' - created levels:
high ->  high
low, medium ->  low_medium
```

We can view what transformation has been chosen for a particular feature.

```
plot(safe_extractor, variable = "last_evaluation")
```



Using the `safe_extractor` object we may now perform transformations and feature selection on new dataset.

```
data3_trans <- safely_transform_data(safe_extractor, data3)
selected_variables <- safely_select_variables(safe_extractor,
      data3_trans, which_y = "left")
data3_trans_sel <- data3_trans[,c("left", selected_variables)]
```

Having the `data3_trans_sel` set, we now use it to fit another glass-box and black-box models:

```
model_lr2 <- glm(left ~ ., data = data3_trans_sel,
      family = binomial())
set.seed(111)
model_xgb2 <- gbm(left ~ ., data = data3_trans_sel,
      distribution = "bernoulli", n.trees = n.trees)
```

And finally, we prepare the test set on which models accuracy will be compared:

```
data4_trans <- safely_transform_data(safe_extractor, data4)
data4_trans_sel <- data4_trans[,c("left", selected_variables)]
```

Four models have been fitted in this example: `model_lr1`, `model_xgb1`, `model_lr2` and `model_xgb2`. For each of them we make predictions on the relevant test set, i.e. we use `model_lr1` and `model_xgb1` to predict the output for `data2` and `model_lr2` and `model_xgb2` to predict the output for `data4`. The performance of the models is then evaluated based on confusion matrices with relative percentages. Xgboost models give the following results:

	predicted 0	**predicted 1**
actual 0	0.97	0.03
actual 1	0.09	0.91

Before transformations

	predicted 0	**predicted 1**
actual 0	0.93	0.07
actual 1	0.27	0.73

After transformations

The model trained on original data has higher predictive power - feature transformations have caused the loss of valuable information which was apparently used by the first model. However, within logistic regression models the significant improvement can be observed:

	predicted 0	**predicted 1**
actual 0	0.92	0.08
actual 1	0.63	0.37

Before transformations

	predicted 0	**predicted 1**
actual 0	0.93	0.07
actual 1	0.27	0.73

After transformations

The initial logistic regression model has difficulties with „true negatives” - for employees that actually quit it very often predicts the opposite. On the other hand, `model_lr2` which was trained on the set of features modified by the SAFE algorithm has much better accuracy.

5.3. rSAFE applied to real data

In previous sections we illustrated the abilities of **rSAFE** on the example of artificial datasets. We decided to test whether it is helpful also in real data. In this section we apply the SAFE methodology to HELOC dataset used in FICO Challenge (section 2.4). Our aim is to predict whether a consumer will be at least 90 days overdue in a period of 2 years from when the credit account was opened (`RiskPerformance` feature taking two values, `"Good"` and `"Bad"`).

The original data was at first slightly preprocessed and then split into two subsets: `train`, used for training models and feature engineering, and `test`, used for validation.

As with HR example, we use logistic regression model as a glass-box and xgboost model as a black-box. We fit the xgboost model with `n.trees = 10000` and `interaction.depth = 3` parameters. And then `safe_extractor` object is created, with a change only in one of the default parameters: `penalty = 20`. All variable transformations from the `safe_extractor` are listed in table 5.5. All categorical features are binary so no clustering is proposed for them. For continuous variables, though, the algorithm suggests splitting each feature into two intervals. It can be noticed that for many of the continuous features the point separating those two intervals is equal to the median of the values in the dataset. This means that for these features no changepoints have been detected by the PELT algorithm. That could be easily changed by decreasing the value of the `penalty` argument. However, doing so would result later in some problems with fitting a logistic regression model so we stick to the chosen value of `20`.

After creating `safe_extractor`, we apply its transformations to `train` set and perform feature selection on it. Column „chosen form" in table 5.5 indicates which form of a feature has been selected. For some variables the original form is preferred and for others the transformed one. It can be observed that the algorithm more often decides to leave a feature unchanged - for majority of variables discretizing it into two intervals would be harmful to a model rather than beneficial.

Having both transformations and a set of final features, we transform `test` set and fit another logistic regression model. Then we compare results for three fitted models using AUC metric. Table below presents AUC measured on both `train` and `test` sets.

	AUC - train	AUC - test
xgboost (original data)	0.9968	0.7494
logistic regression (original data)	0.8040	0.7875
logistic regression (**rSAFE**)	0.8055	**0.7906**

The last row corresponds to the model fitted to the data transformed with SAFE methodology. We can see that the xgboost model combined with the **rSAFE** package improve slightly the performance of simple logistic regression model. To check which modifications contribute most towards the AUC increase, for each feature we compare its importance in initial model with the importance in model fitted to transformed dataset. Using `feature_importance` function from **ingredients** package (Biecek, 2019)[7], we compute permutation based feature importance (Biecek and Burzykowski, 2019)[9] for each of the features (in both logistic regression models). Columns „dropout1" and „dropout2" in table 5.5 correspond to the decrease in AUC

no	feature name	transformation	created levels	chosen form	dropout1	dropout2
1	PercentTradesNeverDelq	discretization by detected changepoints	$(-\infty, 85]; (85, \infty)$	transformed	0.003	0.002
2	NumTotalTrades		$(-\infty, 24]; (24, \infty)$		<0.001	<0.001
3	PercentInstallTrades		$(-\infty, 46]; (46, \infty)$		0.003	0.005
4	ExternalRiskEstimate		$(-\infty, 67]; (67, \infty)$	original	0.031	0.028
5	MSinceOldestTradeOpen		$(-\infty, 180]; (180, \infty)$		<0.001	<0.001
6	AverageMInFile		$(-\infty, 74]; (74, \infty)$		0.009	0.009
7	NumSatisfactoryTrades		$(-\infty, 17]; (17, \infty)$		0.019	0.020
8	NetFractionRevolvingBurden		$(-\infty, 37]; (37, \infty)$		0.016	0.014
9	NumTrades60Ever2DerogPubRec	no changepoints detected – discretization by sample median	$(-\infty, 0]; (0, \infty)$	transformed	<0.001	<0.001
10	MSinceMostRecentDelq		$(-\infty, 21.8]; (21.8, \infty)$		0.002	0.005
11	MSinceMostRecentInqexcl7days		$(-\infty, 2]; (2, \infty)$		0.009	0.017
12	NetFractionInstallBurden		$(-\infty, 68.6]; (68.6, \infty)$		<0.001	0.001
13	NumInstallTradesWBalance		$(-\infty, 2]; (2, \infty)$		<0.001	<0.001
14	MSinceMostRecentTradeOpen		$(-\infty, 6]; (6, \infty)$	original	0.001	0.002
15	NumTrades90Ever2DerogPubRec		$(-\infty, 0]; (0, \infty)$		<0.001	<0.001
16	MaxDelq2PublicRecLast12M		$(-\infty, 6]; (6, \infty)$		<0.001	0.002
17	MaxDelqEver		$(-\infty, 6.4]; (6.4, \infty)$		0.001	<0.001
18	NumTradesOpeninLast12M		$(-\infty, 1.9]; (1.9, \infty)$		<0.001	<0.001
19	NumInqLast6M		$(-\infty, 1]; (1, \infty)$		0.067	0.064

20	NumInqLast6Mexcl7days		$(-\infty, 1]; (1, \infty)$		0.029	0.034
21	NumRevolvingTradesWBalance		$(-\infty, 4]; (4, \infty)$		0.011	0.009
22	NumBank2NatlTradesWHighUtilization		$(-\infty, 1]; (1, \infty)$		0.004	0.004
23	PercentTradesWBalance		$(-\infty, 67]; (67, \infty)$		<0.001	<0.001
24	NoBureau	binary features – no transformation suggested	–	original	<0.001	0.002
25	NoValid_MSinceOldestTradeOpen				<0.001	<0.001
26	NoValid_MSinceMostRecentDelq				<0.001	<0.001
27	No_MSinceMostRecentDelq				0.006	0.017
28	NoValid_MSinceMostRecentInqexcl7days				0.005	0.001
29	No_MSinceMostRecentInqexcl7days				0.001	0.014
30	NoValid_NetFractionRevolvingBurden				<0.001	<0.001
31	NoValid_NetFractionInstallBurden				<0.001	0.002
32	NoValid_NumRevolvingTradesWBalance				<0.001	<0.001
33	NoValid_NumInstallTradesWBalance				<0.001	<0.001
34	NoValid_NumBank2NatlTradesWHighUtilization				<0.001	<0.001

Figure 5.5: Transformations of the features from FICO dataset proposed by the SAFE algorithm. Since all categorical variables are binary, there are no transformations for them. Column „chosen form" contains information on which form of a feature has been chosen in the feature selection. In two last columns there are measures of permutation based feature importance calculated for logistic regression model - fitted to, respectively, original data and data transformed with the **rSAFe** package. Each feature importance corresponds to the decrease in AUC on `test` set after permuting the feature values.

after permuting the feature values. All values are measured on `test` set and „dropout1" refers to the initial logistic regression model and „dropout2" to the one fitted to transformed data. The plot below illustrates how importance of particular features changes after applying SAFE transformations.

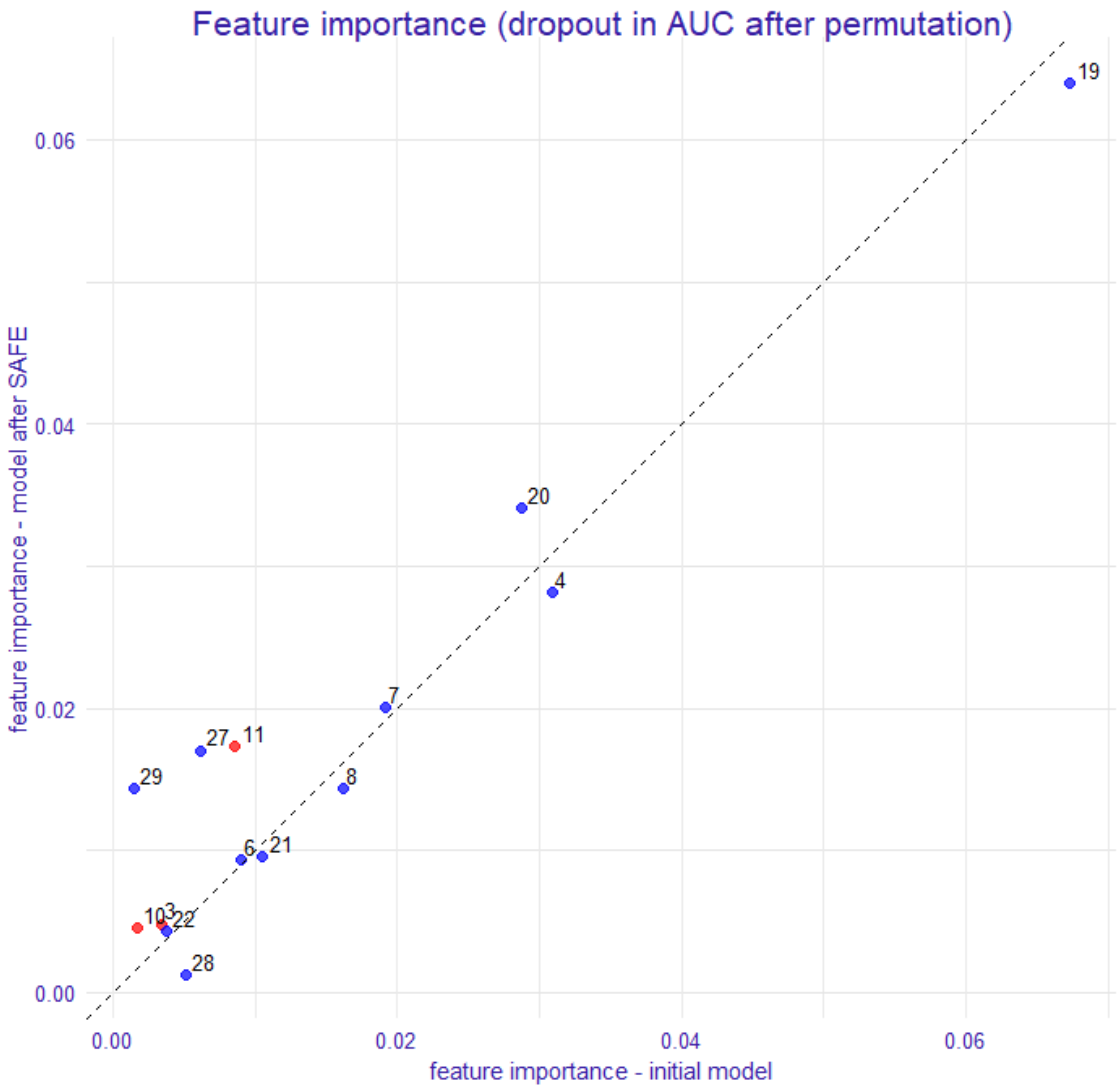


Figure 5.6: Comparison between importance of features in two logistic regression models - fitted to original data (x axis) and fitted to data transformed with `safe_extractor` object (y axis). The plot includes only 14 most important variables. Red points correspond to features which have been transformed, blue points - to the ones for which original form has been preserved. Features are labelled according to table 5.5.

It can be noticed that after introducing transformations for some of the features the importance of other variables (remaining untransformed) increases. For example,

`No_MSinceMostRecentInqexcl7days` (29.) and `No_MSinceMostRecentDelq` (27.) are binary features that have been used in both models in the same form. Yet, in the second model they affect predictions to a much greater extent than in the first model. Amongst 14 most influential predictors there are three continuous variables which have been used in the second model in discretized form: `PercentInstallTrades` (3.), `MSinceMostRecentDelq` (10.) and `MSinceMostRecentInqexcl7days` (11.). The biggest difference may be observed for the last one - replacing its original values with one of the two intervals $(-\infty, 2]; (2, \infty)$ increases its permutation based importance from 0.009 to 0.017. Interestingly, only one of the mentioned three features has been split through the PELT algorithm - for the other two a sample median serves as a changepoint. One feature clearly stands out against others - `NumInqLast6M` (19.) is the most important predictor in both logistic regression models. The ALE plots for all mentioned continuous features are presented in 5.7.

Despite the improvement of model performance after applying SAFE method, the differences in AUC are not huge. The reason for that may be the choice of the black-box model - test AUC for the fitted xgboost model is smaller than test AUC measured for logistic regression models. The shape of ALE plots presented in 5.7 also may indicate some problems with the model. So maybe training a more accurate elastic model would provide us with more valuable transformations which could help us obtain better simple model. Nevertheless, in real applications, such as those in credit risk world, even little difference in model performance can be beneficial for a company and translate into large profits.

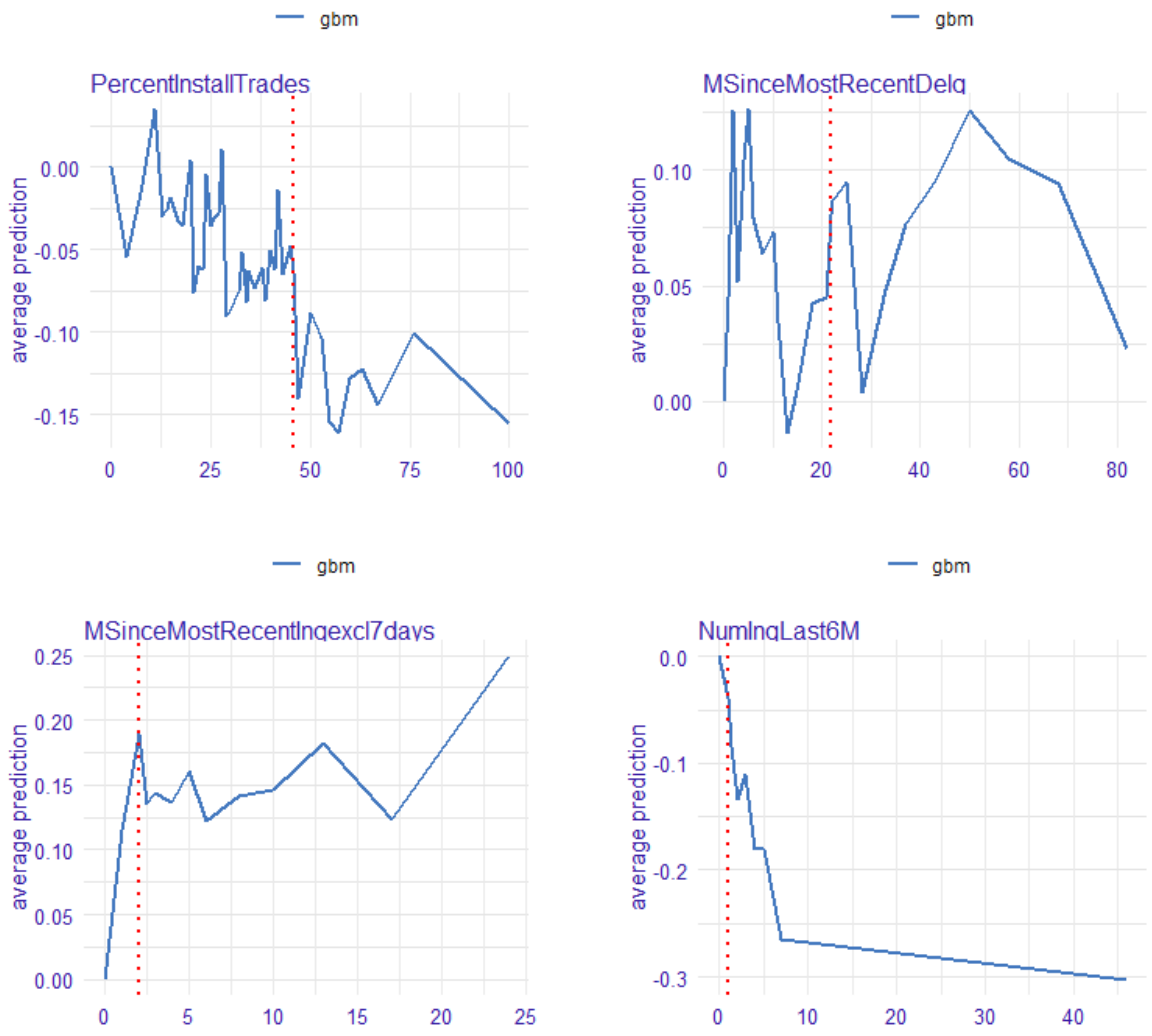


Figure 5.7: ALE plots for chosen features from FICO dataset. Variable `PercentInstallTrades` (3.) is split by one changepoint which is in line with our intuition. Features `MSinceMostRecentDelq` (10.) and `MSinceMostRecentInqexcl7days` (11.) are split by a sample median since no obvious breakpoints are found. Variable `NumInqLast6M` (19.) is the strongest predictor for which discretization by a median is not preferred.

6. Conclusion

In this paper, we presented the method that may be used to tackle the interpretability issue which occupies an important place in the modern machine learning world. The SAFE algorithm is about distilling the knowledge from one predictive model and then applying it to another one. It uses a surrogate model to obtain valuable feature transformations that may be used later to train a simpler, explainable model. In chapter 5 we showed that the described methodology can boost the performance of machine learning models, both regression and classification ones.

The introduced methods have been implemented in the **rSAFE** package for R. It contains multiple functionalities which can be used in applications of the SAFE algorithm to real data. This tool is designed for an automatic process of feature transformations, yet this process can be controlled by the user via a wide set of parameters, making it more flexible. The **rSAFE** package is available on GitHub (development version, `https://github.com/ModelOriented/rSAFE`) and its stable version will be soon available on CRAN. Package documentation and examples can be found at `https://modeloriented.github.io/rSAFE/`.

We believe that presented approach and implemented solution can serve as valuable assistance during the process of building machine learning models. However, we also notice some potential areas in which future research can be done. Firstly, one might think about better ways to obtain the relationship between a predictor and a response variable. Moreover, we could consider more flexible forms of feature transformations, not limiting ourselves only to discretization. And last but not least, more advanced techniques for dealing with interactions problem could be investigated.

Bibliography

[1] D. Apley. Visualizing the effects of predictor variables in black box supervised learning models. 2016. URL https://arxiv.org/abs/1612.08468.

[2] D. Apley. *ALEPlot: Accumulated Local Effects (ALE) Plots and Partial Dependence (PD) Plots*, 2018. URL https://cran.r-project.org/package=ALEPlot.

[3] P. Biecek. *PBImisc: A Set of Datasets Used in My Classes or in the Book 'Modele Liniowe i Mieszane w R, Wraz z Przykladami w Analizie Danych'*, 2016. URL https://cran.r-project.org/package=PBImisc.

[4] P. Biecek. *breakDown: Model Agnostic Explainers for Individual Predictions*, 2018. URL https://cran.r-project.org/package=breakDown.

[5] P. Biecek. DALEX: Explainers for Complex Predictive Models in R. *Journal of Machine Learning Research*, 19(84):1–5, 2018. URL http://jmlr.org/papers/v19/18-416.html.

[6] P. Biecek. *DALEX: Descriptive mAchine Learning EXplanations*, 2019. URL https://cran.r-project.org/package=DALEX.

[7] P. Biecek. *ingredients: Effects and Importances of Model Ingredients*, 2019. URL https://cran.r-project.org/package=ingredients.

[8] P. Biecek. Model Development Process. 2019. URL https://arxiv.org/abs/1907.04461.

[9] P. Biecek and T. Burzykowski. *Predictive Models: Explore, Explain, and Debug*. 2019. https://pbiecek.github.io/PM_VEE/.

[10] L. Breiman, A. Cutler, A. Liaw, and M. Wiener. *randomForest: Breiman and Cutler's Random Forests for Classification and Regression*, 2018. URL https://cran.r-project.org/package=randomForest.

[11] T. Chen and C. Guestrin. XGBoost: A Scalable Tree Boosting System. 2016. URL https://arxiv.org/abs/1603.02754.

[12] H. Fanaee-T and J. Gama. Event labeling combining ensemble detectors and background knowledge. *Progress in Artificial Intelligence*, pages 1–15, 2013. ISSN 2192-6352. doi: 10.1007/s13748-013-0040-3. URL http://dx.doi.org/10.1007/s13748-013-0040-3.

[13] A. Fisher, C. Rudin, and F. Dominici. All Models are Wrong but Many are Useful: Variable Importance for Black-Box, Proprietary, or Misspecified Prediction Models, using Model Class Reliance. 2018. URL https://arxiv.org/abs/1801.01489.

[14] J. H. Friedman. Greedy Function Approximation: A Gradient Boosting Machine. *Annals of Statistics*, 29(5):1189–1232, 2001. URL https://doi.org/10.1214/aos/1013203451.

[15] T. Galili, Y. Benjamini, G. Simpson, and G. Jefferis. *dendextend: Extending 'dendrogram' Functionality in R*, 2019. URL https://cran.r-project.org/package=dendextend.

[16] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[17] B. Goodman and S. Flaxman. European Union regulations on algorithmic decision-making and a „right to explanation". 2016. URL https://arxiv.org/abs/1606.08813.

[18] A. Gosiewska and P. Biecek. iBreakDown: Uncertainty of Model Explanations for Non-additive Predictive Models. 2019. URL https://arxiv.org/abs/1903.11420v1.

[19] A. Gosiewska, A. Gacek, P. Lubon, and P. Biecek. SAFE ML: Surrogate Assisted Feature Extraction for Model Learning. 2019. URL https://arxiv.org/abs/1902.11035.

[20] B. Greenwell. *pdp: Partial Dependence Plots*, 2018. URL https://cran.r-project.org/package=pdp.

[21] B. Greenwell, B. Boehmke, and J. Cunningham. *gbm: Generalized Boosted Regression Models*, 2019. URL https://cran.r-project.org/package=gbm.

[22] B. M. Greenwell. pdp: An R Package for Constructing Partial Dependence Plots. *The R Journal*, 9(1):421–436, 2017. URL https://journal.r-project.org/archive/2017/RJ-2017-016/index.html.

[23] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer-Verlag New York, 2009. Second Edition.

[24] K. Haynes, P. Fearnhead, and I. A. Eckley. A computationally efficient nonparametric approach for changepoint detection. 2016. URL https://arxiv.org/abs/1602.01254.

[25] B. Jackson, J. D. Scargle, D. Barnes, S. Arabhi, A. Alt, P. Gioumousis, E. Gwin, P. Sangtrakulcharoen, L. Tan, and T. T. Tsai. An Algorithm for Optimal Partitioning of Data on an Interval. 2004. URL https://arxiv.org/abs/math/0309285.

[26] A. Kassambara. *ggpubr: 'ggplot2' Based Publication Ready Plots*, 2019. URL https://cran.r-project.org/package=ggpubr.

[27] A. Kassambara and F. Mundt. *factoextra: Extract and Visualize the Results of Multivariate Data Analyses*, 2017. URL https://cran.r-project.org/package=factoextra.

[28] R. Killick. *A place for the development version of the changepoint package on CRAN.*, 2019. URL https://github.com/rkillick/changepoint.

[29] R. Killick, P. Fearnhead, and I. A. Eckley. Optimal detection of changepoints with a linear computational cost. 2012. URL https://arxiv.org/abs/1101.1438.

[30] J. Koronacki and J. Ćwik. *Statystyczne systemy uczące się*. Akademicka Oficyna Wydawnicza EXIT, 2008. Wydanie drugie.

[31] M. Kuhn and K. Johnson. *Feature Engineering and Selection: A Practical Approach for Predictive Models*. 2019. www.feat.engineering.

[32] S. Lundberg and S.-I. Lee. A Unified Approach to Interpreting Model Predictions. 2017. URL https://arxiv.org/abs/1705.07874.

[33] D. Meyer, K. Hornik, and C. Buchta. *sets: Sets, Generalized Sets, Customizable Sets and Intervals*, 2017. URL https://cran.r-project.org/package=sets.

[34] T. Miller. Explanation in artificial intelligence: Insights from the social sciences. 2017. URL https://arxiv.org/abs/1706.07269.

[35] C. Molnar. *Interpretable Machine Learning*. 2019. https://christophm.github.io/interpretable-ml-book/.

[36] F. Murtagh and P. Contreras. Methods of Hierarchical Clustering. 2011. URL https://arxiv.org/abs/1105.0121v1.

[37] F. Murtagh and P. Legendre. Ward's Hierarchical Agglomerative Clustering Method: Which Algorithms Implement Ward's Criterion? 2014.

[38] A. Sitko and P. Biecek. The Merging Path Plot: adaptive fusing of k-groups with likelihood-based model selection. 2017. URL https://arxiv.org/abs/1907.04461.

[39] A. Sitko, A. Grudziąż, and P. Biecek. *factorMerger: The Merging Path Plot*, 2019. URL https://cran.r-project.org/package=factorMerger.

[40] R. Tibshirani, G. Walther, and T. Hastie. Estimating the number of clusters in a data set via the gap statistic. 2001. URL https://statweb.stanford.edu/~gwalther/gap.

[41] C. Truong, L. Oudre, and N. Vayatis. Selective review of offline change point detection methods. 2019. URL http://arxiv.org/abs/1801.00718.

[42] M. Tulio Ribeiro, S. Singh, and C. Guestrin. „Why Should I Trust You?": Explaining the Predictions of Any Classifier. pages 97–101, 02 2016. URL https://arxiv.org/abs/1602.04938.

[43] H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016. ISBN 978-3-319-24277-4. URL https://ggplot2.tidyverse.org.

[44] H. Wickham, W. Chang, L. Henry, T. L. Pedersen, K. Takahashi, C. Wilke, K. Woo, and H. Yutani. *ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics*, 2019. URL https://cran.r-project.org/package=ggplot2.

[45] H. Wickham, J. Hester, and W. Chang. *devtools: Tools to Make Developing R Packages Easier*, 2019. URL https://cran.r-project.org/package=devtools.

[46] J. Zhang. Powerful Goodness-of-Fit Tests Based on the Likelihood Ratio. *Journal of the Royal Statistical Society. Series B*, 64(2):281–294, 2002.

[47] N. R. Zhang and D. O. Siegmund. A Modified Bayes Information Criterion with Applications to the Analysis of Comparative Genomic Hybridization Data. 2007.

[48] C. Zou, G. Yin, L. Feng, and Z. Wang. Nonparametric maximum likelihood approach to multiple change-point problems. 2014. URL https://arxiv.org/abs/1405.7173.