Explainable Machine Learning

Mango Solutions

2019-11-01

1 Introduction to the Training

1.1 Course Aims

This course is designed to provide an overview of commonly used frameworks in Explainable Machine Learning (EML) and how to apply them using the R language. It explains why obtaining interpretable model explanations is key in any Machine Learning project and explores tools that can be used to better understand model predictions on global and local level.

1.2 Course Materials

Items appearing in this material are sometimes given a special appearance to set them apart from regular text. Here's how they look:

```
This is a section of code                  # This is a comment
```

A warning, typically describing non-intuitive aspects of the R language

A tip: additional features of R or "shortcuts" based on user experience

Exercises to be performed during (or after) the training course

1.3 Course Script and Exercise Answers

A great deal of code will be executed within R during the delivery of this training. This includes the answers to each exercise, as well as other code written to answer questions that arise. Following the course, you will be sent a script containing all the code that was executed.

2 Motivation

Machine Learning has seen an unprecedented development over the last decade. This was triggered by the rapid development of complex learning algorithms and computing power that together now allow us to crunch data of bigger volume, variety and velocity than ever before.

Machine Learning is widely used in almost all industries these days. For example, it can be used for interpreting health scans, recommending movies and music, recruiting new employees and retaining the existing ones, predicting the next purchase or equipment failure. It is understandable why many businesses and governments choose a machine learning solution over a human decision, as it scales better and it tends to produce more accurate predictions. But how do we know whether to trust even most accurate predictions?

2.1 Machine Learning and Trust

High accuracy is an undeniable benefit of automated decision making, however not all algorithms were made equal in this respect and some tend to be more accurate than others. More accurate algorithms also tend to be more complex, e.g., they contain more parameters and use features in a way that is less intuitive to humans. This complexity makes them less interpretable, i.e., it is difficult to understand **why** they made certain predictions. Algorithms that don't offer human-interpretable decision rules are called **black-box**. Neural networks are a typical example of (potentially) highly accurate black-box algorithms.

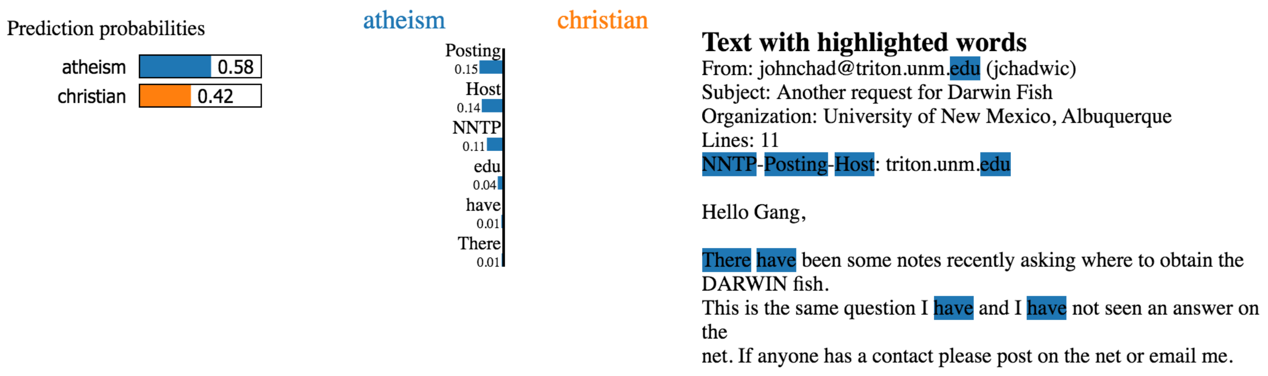
Still, one could argue that if high accuracy is the only metric that we care about, interpretability is then irrelevant. This could not be farther from the truth.

When we trust a highly accurate model without understanding what drives its predictions, we make an assumption that the model is accurate because it correctly captured intricate relationships between the features and the target variable. However, in reality there are many other factors that may account for high model accuracy, for example:

- **over-fitting** - train and validation sets do not cover a wide enough range of possible outcomes and/or a high-variance algorithm was used for training and identifies noise as signal.
- **correlation** - data contains undesirable artefacts that are correlated to the target variable but bear little significance in reality, e.g., identifying a wolf in the picture every time there is snow in the background.
- **data leakage** - information between train and validation data sets is accidentally shared, e.g., using a feature whose value would not be available in practice at the time you'd want to use the model to make a prediction.
- **truth** - model correctly captures signal in the data.

Let's take an example of where data correlation confuses model predictions: in the original LIME repo [1], Marco Tulio Ribeiro gives an example of the project where the aim was to distinguish between documents covering atheism or Christianity using the 20 newsgroups dataset [2]. A random forest with 500 trees achieved a surprisingly high accuracy of over 92%. Without any access to drivers behind model decisions we would assume that this is a good model that we can trust. However, if we look at the model explanations (**Figure 2.1**) it will become clear that the words associated with atheism label (highlighted blue, e.g. "Posting", "Host", "edu", etc.) have little to do with the actual topic. As Mario put it:

> This is a case where the classifier predicts the instance correctly, but for the wrong reasons. A little further exploration shows us that the word "Posting" (part of the email header) appears in 21.6% of the examples in the training set, only two times in the class 'Christianity'. This is repeated on the test set, where it appears in almost 20% of the examples, only twice in 'Christianity'. This kind of quirk in the dataset makes the problem much easier than it is in the real world, where this classifier would not be able to distinguish between Christianity and atheism documents. This is hard to see just by looking at accuracy or raw data, but easy once explanations are provided. Such insights become common once you understand what models are actually doing, leading to models that generalize much better.



Text with highlighted words
From: johnchad@triton.unm.edu (jchadwic)
Subject: Another request for Darwin Fish
Organization: University of New Mexico, Albuquerque
Lines: 11
NNTP-Posting-Host: triton.unm.edu

Hello Gang,

There have been some notes recently asking where to obtain the
DARWIN fish.
This is the same question I have and I have not seen an answer on
the
net. If anyone has a contact please post on the net or email me.

Figure 2.1 Prediction explanations of the highly accurate classification model distinguishing between emails written on Atheism or Catholicism. Text highlighted blue indicates words supporting atheism label most. Still, these words have little to do with the actual topic.

2.1.1 Trust in high-stake decisions

Having trust in model predictions is even more important where wrong decisions can affect people's lives, potentially at scale. This would include, but would not be limited to, situations such as recruiting/promoting employees, predicting loan default or who's most likely to commit a crime in defined future.

For example, in 2014 Amazon developed a recruiting engine[3] to review job applicants' resumes with the aim of optimizing the search for top talent. But a year later, the company realized its new system was not rating candidates in a gender-neutral way, e.g., for technical posts, such as software developer, it would prefer male candidates over the female ones. The way it was biased was not obvious - gender was not included as a feature, but instead the model penalized resumes that included the word "women's," as in "women's chess club captain" or "women's college". At the end, Amazon decided to stop using the tool, but understanding these biases would be impossible without having access to interpretable explanations.

Another example is the Correctional Offender Management Profiling for Alternative Sanctions (COMPAS) [4], tool that predicts a defendant's risk of committing another crime. It works through a proprietary algorithm, developed by a private company, that is based on the answers to a 137-item questionnaire. The tool is routinely used in a number of US States, such as Florida, New York, Wisconsin or California. COMPAS has been criticised by some for being biased against African Americans, although the debate continues. More importantly, COMPAS accuracy can be matched by very simple models [5], e.g., a linear regression with only 2 predictors, age and number of committed crimes. Finally, because COMPAS is a commercial product, we can't get access to its interpretable explanations - if such exist - and thus we can't know for sure whether we can trust it or not. Are there good reasons to think that it is biased? Or that it performs consistently better than simplistic models?

Given the widespread use of Machine Learning algorithms and its potential to affect even the most vulnerable, it is absolutely critical that we understand why models make certain decisions. It is clear that relying on high accuracy metrics alone may be very misleading and counter-productive long-term.

2.2 Explainable Machine Learning

Explainable Machine Learning (EML) provides methods and techniques that enable human experts to understand the results of the ML/AI solutions, thus making even black-box models interpretable. In recent years there has been a surge of work in this area and new techniques and frameworks are developed on a regular basis.

This is important as it will alleviate some of the social and ethical issues in applying black-box algorithms at scale. Still, there are good reasons to apply EML to any Machine Learning project, even if it is not intended to negatively affect others.

TRUST

Interpretable explanations build trust by offering insights into the following:

- Sanity check - model explanation may provide reassurance that the rules behind the predictions are sound and follow common sense and/or expert knowledge.
- Generalizability - if the model explanations follow common sense and/or expert knowledge, they are more likely to generalise well.
- Fairness - model explanations may uncover any learned biases that are unfair, unethical or undesirable in any other way.

PREDICT

Explainable Machine Learning can give us foresight of model behaviour when applied to unseen data. This could reduce the uncertainty around post-deployment model performance.

IMPROVE

Finally, understanding how a model makes predictions can help modify underlying features and ultimately improve model performance.

3 DALEX

DALEX stands for **D**escriptive M**a**chine **L**earning **Ex**planations and is an R package that provides a set of tools that help understand how complex models are working. It is part of the [DrWhy.AI (https://github.com/ModelOriented/DrWhy/)](https://github.com/ModelOriented/DrWhy/) universe - a collection of tools for Visual Exploration, Explanation and Debugging of Predictive Models developed by Przemyslaw Biecek.

In many applications we need to understand how our model is working: how the input variables are used, what is their impact on the target variable and when our models performs well (or badly). We need tools to extract this information no matter what kind of algorithm we're working with.

Each of these tasks can be tackled using existing tools for model understanding and validation, but we would need a number of packages - with their own grammar and naming conventions - to achieve it comprehensively. DALEX creates a consistent grammar and visualizations for these packages. More importantly, these tools are model-agnostic, which means they can be applied to any algorithm, including black-box models. Finally, DALEX enables quick and efficient model comparison which is an integral part of any modelling process.

3.1 Starting with DALEX

Understanding our model can be quite a complex process and depending on the stage of the model development we may be interested in different aspects of it.

Firstly, you may be interested in how your model works globally. For example, we may want to know how good is the model: how well it performs under what circumstances.

Secondly, we may be interested in the variables that affect model predictions most (feature importance) and in the relationship between specific variables, be it continuous or categorical, and the target variable.

Finally, we may want to understand how each of the features contributes to a specific prediction (prediction attribution). See the below scheme for a better visualization of the process:

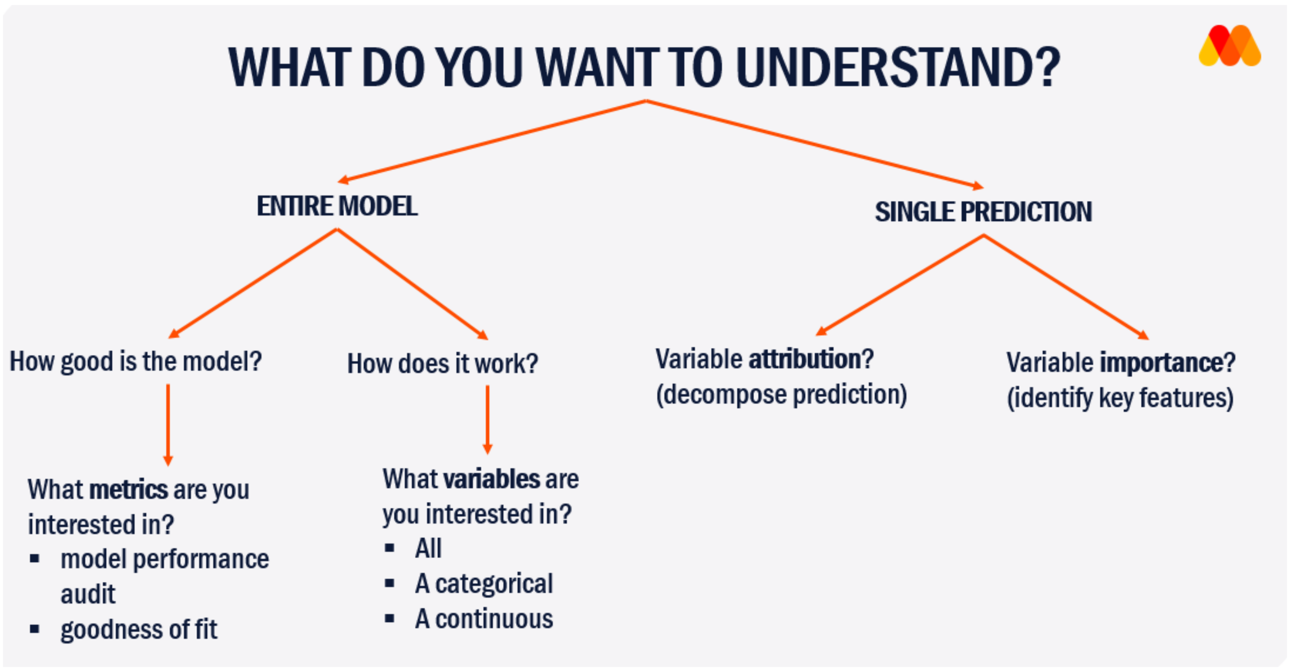

WHAT DO YOU WANT TO UNDERSTAND?
ENTIRE MODEL
SINGLE PREDICTION
How good is the model?
How does it work?
Variable attribution? (decompose prediction)
Variable importance? (identify key features)
What metrics are you interested in?
model performance audit
goodness of fit
What variables are you interested in?
All
A categorical
A continuous

Figure 3.1 Flow of understanding your Machine Learning model

DALEX can help us explore all of these questions, but in this chapter we will focus only on understanding the global model.

3.2 DALEX explainers

DALEX is designed to work with different types of models, such as linear models, neural networks, tree ensembles, etc. In practice, these models can be created using different packages that use different interfaces to train and validate models, and capture different meta-data about the models themselves. For example, you can train a Random Forest model using at least three different packages: `randomForest`, `caret` or `mlr`. So, in order to make the model understanding process consistent, DALEX unifies the model interfacing and meta-data and saves it in a DALEX explainer object. Conveniently, all of the DALEX object can be used with the generic `print()` and `plot()` functions. To see how it works in practice, let's first build our models. To do this, we are going to use `apartments` and `apartments_test` datasets that come with the DALEX package.

3.2.1 Train the models

```r
# load packages
library(DALEX)
library(randomForest)
library(ggplot2)

# load data
data("apartments")
data("apartments_test")
```

They contain information about apartment prices per square meter in the Polish capital Warsaw (`m2.price`), and their features: when they were built (`construction.year`), how big they are (`surface`), which floor they are on (`floor`), how many rooms they have (`rooms`) and which neighbourhood they are located in (`district`):

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

Our target variable is `m2.price` and as it is a continuous variable, we are dealing with a regression problem. Let's build two regression models using all the available features: a linear model and a random forest:

```
# train a linear regression
apartments_lm_model <-  lm(m2.price ~ .,
                           data = apartments)


# train as random forest
set.seed(519)
apartments_rf_model <- randomForest(m2.price ~ .,
                                    data = apartments)
```

Now, let's use these models on unseen data, `apartments_test`:

```
predicted_lm <- predict(apartments_lm_model, apartments_test)

predicted_rf <- predict(apartments_rf_model, apartments_test)
```

How do these models compare in terms of accuracy? We will calculate a root mean squared error (RMSE), or a mean difference between the actual value and prediction, for each of them:

```
# calculate root mean squared error for both models
sqrt(mean((predicted_rf - apartments_test$m2.price)^2))
```

```
## [1] 282.2885
```

```
sqrt(mean((predicted_lm - apartments_test$m2.price)^2))
```

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

As we can see, the model accuracy is almost identical, does it mean these models are equally good? Can we trust both of them to the same extend? We'll explore this question in the coming chapters, but first we need to create the DALEX explainer objects.

3.2.2 Create DALEX explainer objects

In order to unify the data on the model and its interface, DALEX provides the `explain()` function that takes at least three arguments: i) the `model` we trained, ii) test features as `data` and iii) the test target variable as `y`:

```
explainer_rf <- DALEX::explain(
  model = apartments_rf_model,
  data = apartments_test[,2:6],
  y = apartments_test$m2.price)
```

```
## Preparation of a new explainer is initiated
##   -> model label       :  randomForest  (  [33m default  [39m )
##   -> data              :  9000  rows  5  cols
##   -> target variable   :  9000  values
##   -> predict function  :  yhat.randomForest  will be used (  [33m default  [39m
##   -> predicted values  :  numerical, min =  1970.683 , mean =  3506.927 , max =
##   -> residual function :  difference between y and yhat (  [33m default  [39m )
##   -> residuals         :  numerical, min =  -749.1102 , mean =  4.596643 , max
##   -> model_info        :  package randomForest , ver. 4.6.14 , task regression
##   [32m A new explainer has been created!  [39m
```

```
explainer_lm <- DALEX::explain(
  model = apartments_lm_model,
  data = apartments_test[,2:6],
  y = apartments_test$m2.price)
```

```
## Preparation of a new explainer is initiated
##   -> model label       :  lm  (  [33m default  [39m )
##   -> data              :  9000  rows  5  cols
##   -> target variable   :  9000  values
##   -> predict function  :  yhat.lm  will be used (  [33m default  [39m )
##   -> predicted values  :  numerical, min =  1792.597 , mean =  3506.836 , max =
##   -> residual function :  difference between y and yhat (  [33m default  [39m )
##   -> residuals         :  numerical, min =  -257.2555 , mean =  4.687686 , max
##   -> model_info        :  package stats , ver. 3.6.0 , task regression (  [33m
##   [32m A new explainer has been created!  [39m
```

When we print the explainer object to the console, we see only basic information about the model class and a snippet of underlying data, but this object contains much more information than that (which can be uncovered by running `str()` on the explainer object).

```
explainer_rf
```

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

DALEX explainer objects will be almost universally used as the first argument in any DALEX-related tasks.

3.3 How good is the model?

Now, that we have the explainer objects ready, we can dive deeper into the model performance. The function `model_performance()` calculates predictions and residuals (the difference between the observed and predicted values) for the validation dataset.

```
mp_lm <- model_performance(explainer_lm)

mp_rf <- model_performance(explainer_rf)
```

3.3.1 Distribution of residuals

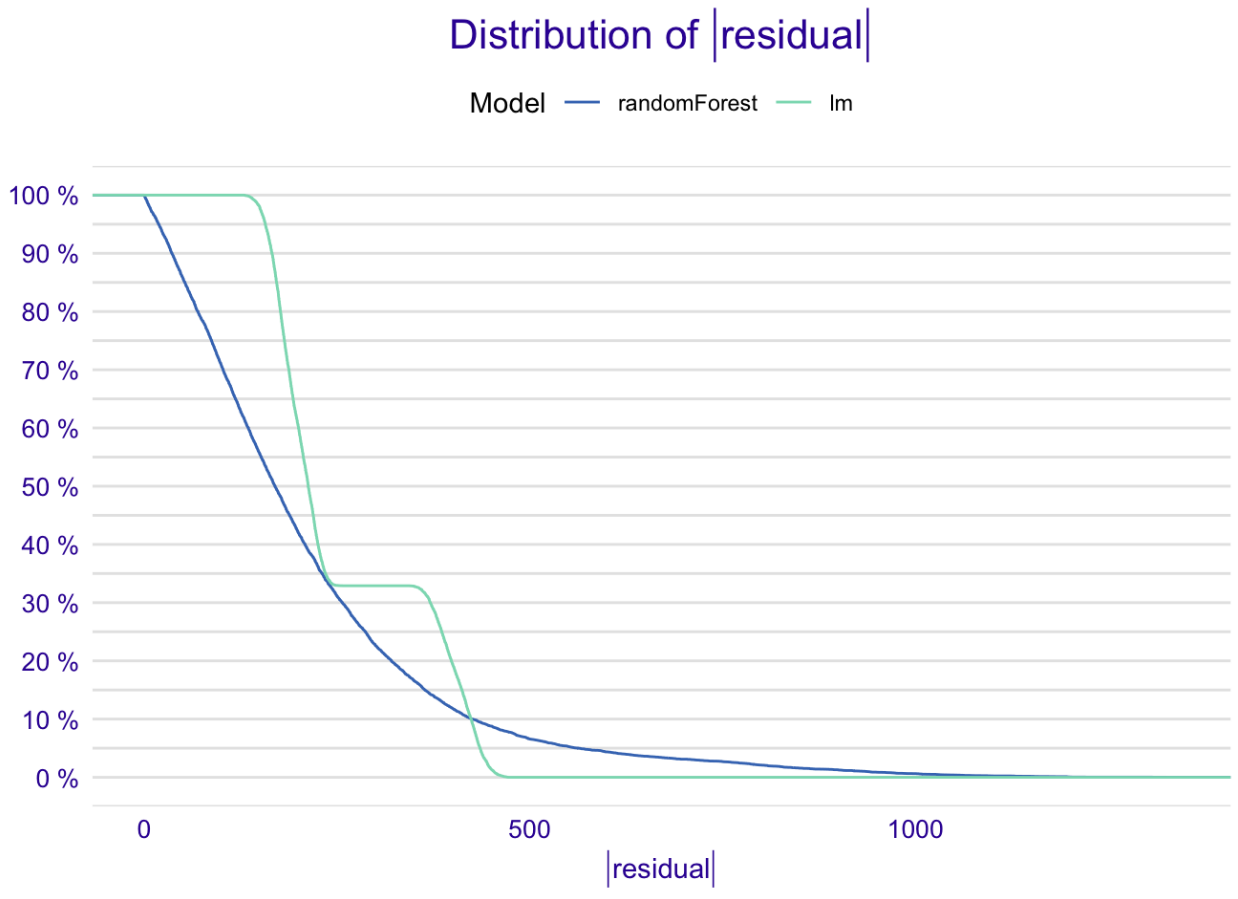
If we print the model performance objects to the console, we'll get the quantiles for residuals. In the example below, 30% of linear model's residuals are less or equal to -370 but 40% are less or equal to 161.

```
mp_lm
```

```
##         0%       10%       20%       30%       40%       50%       60% 
## -472.3560 -423.9131 -398.2811 -370.8841  161.2473  174.0677  184.1412 
##       70%       80%       90%      100% 
##  195.8834  209.2460  221.4659  257.2555
```

We can use the generic `plot()` function to visualise the results. Conveniently, we can plot together as many model performance objects as we like, making the comparison between different models very easy:

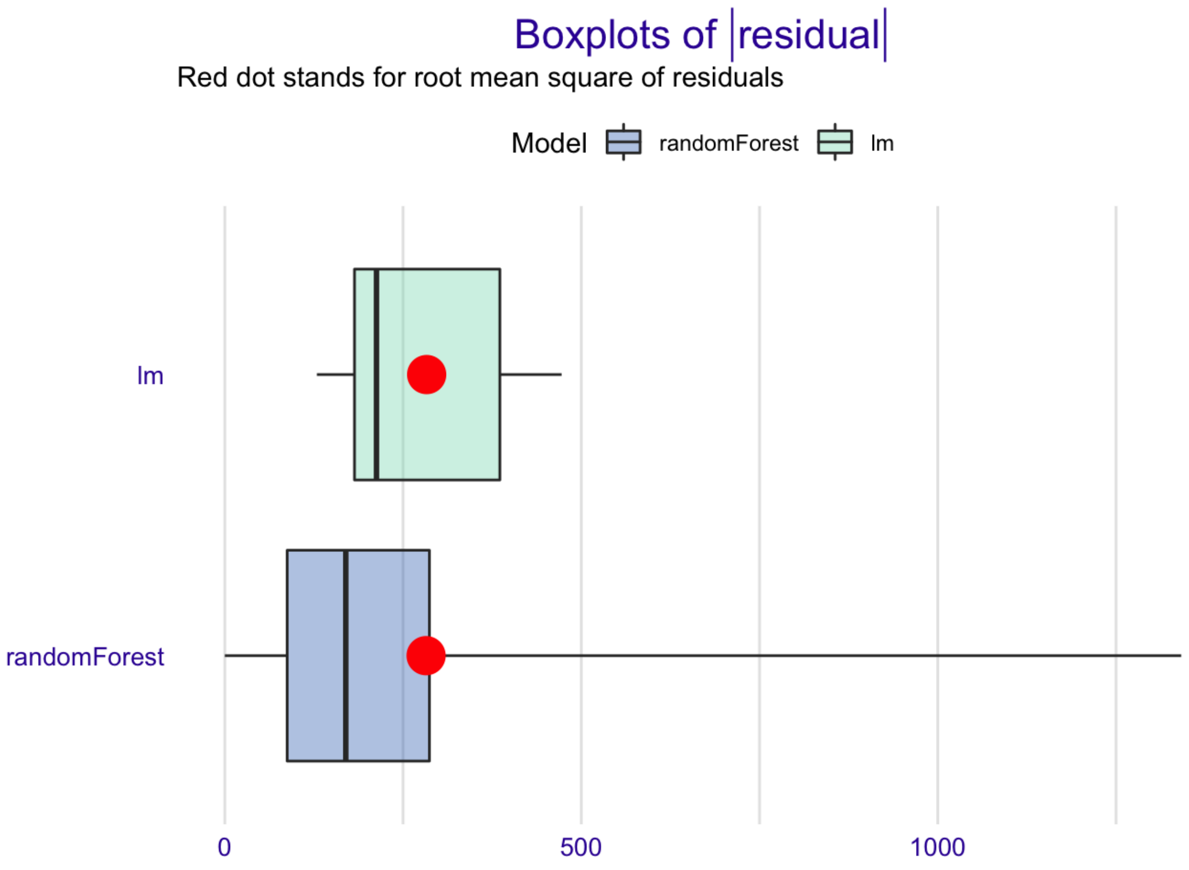
```r
plot(mp_lm, mp_rf)
```



The above plot shows the reversed cumulative distribution function (CDF) for absolute values of the residuals, i.e., moving from left to right the line shows the fraction of residuals larger than the x-axis value. In other words, the closer the CDF is to x = 0 line, the more accurate the model is. From this we can gather that the random forest model is overall more accurate than the linear model, except for a fraction of large residuals around 500 value that affect the root mean squared error.

We can visualize the distribution of the residuals via a boxplot by setting the `geom` argument to `boxplot`. Here, the red dot indicates root mean squared error.

```
plot(mp_lm, mp_rf, geom = 'box-plot')
```



Again, from the plot above it's clear that the random forest has overall lower residuals but greater outliers compared to the linear model.

3.3.2 Residuals over the actual values

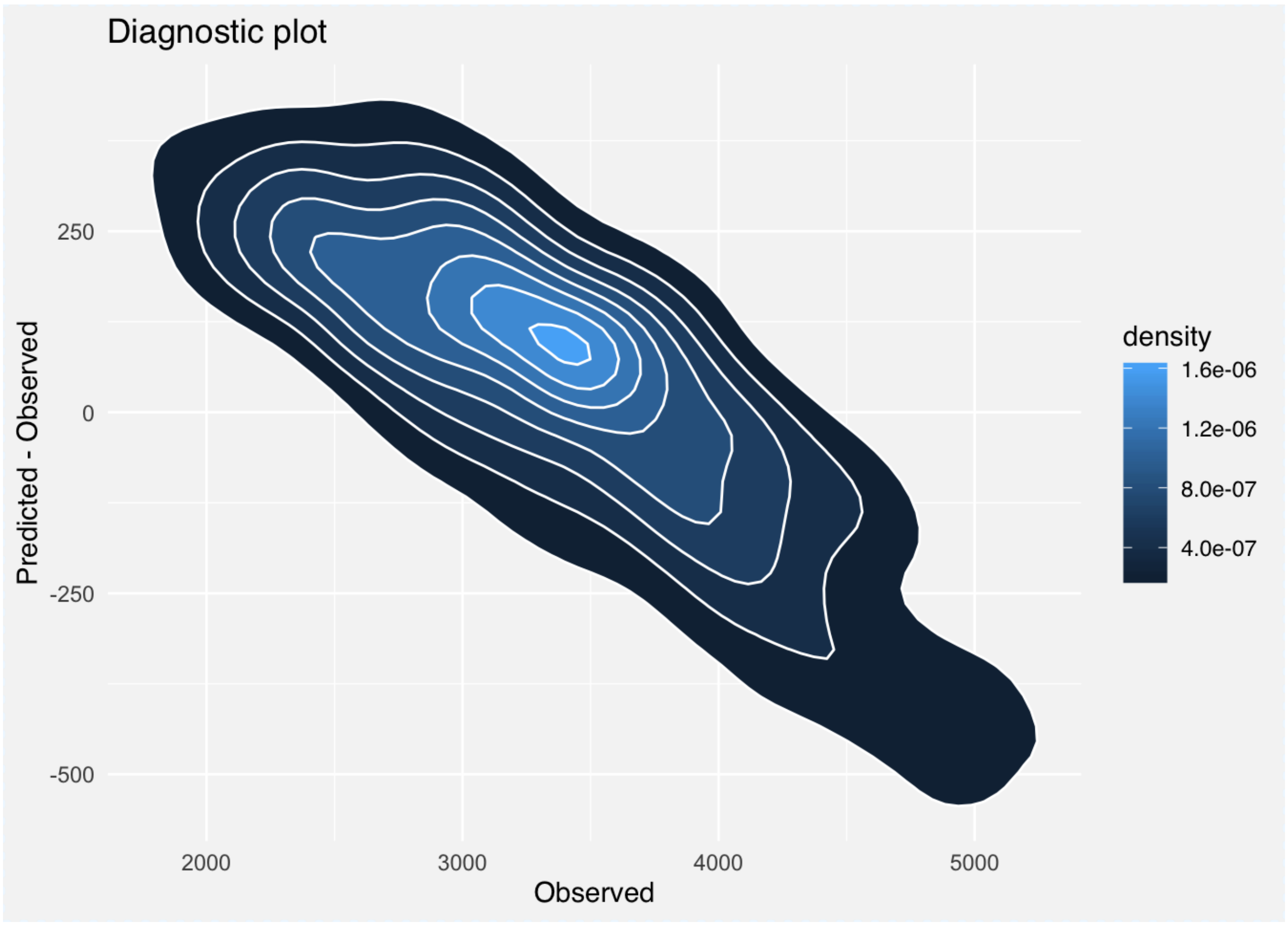
The distribution of absolute residuals gives us a useful view of model performance but that is not the only way we can use the model performance object. If we inspect this object you'll see that it contains predicted and actual values, as well as residuals (`diff` variable).

```
str(mp_rf)
```

```
## Classes 'model_performance_explainer' and 'data.frame':  9000 obs. of  4 variab
##  $ predicted: num  4228 3199 2692 2730 2973 ...
##  $ observed : num  4644 3082 2498 2735 2781 ...
##  $ diff     : num  -416.4 117.4 193.6 -5.3 192.1 ...
##  $ label    : chr  "randomForest" "randomForest" "randomForest" "randomForest"
```

Now, we can use these variables to understand for which observation the model is making biggest mistakes by plotting actual values over the residuals:

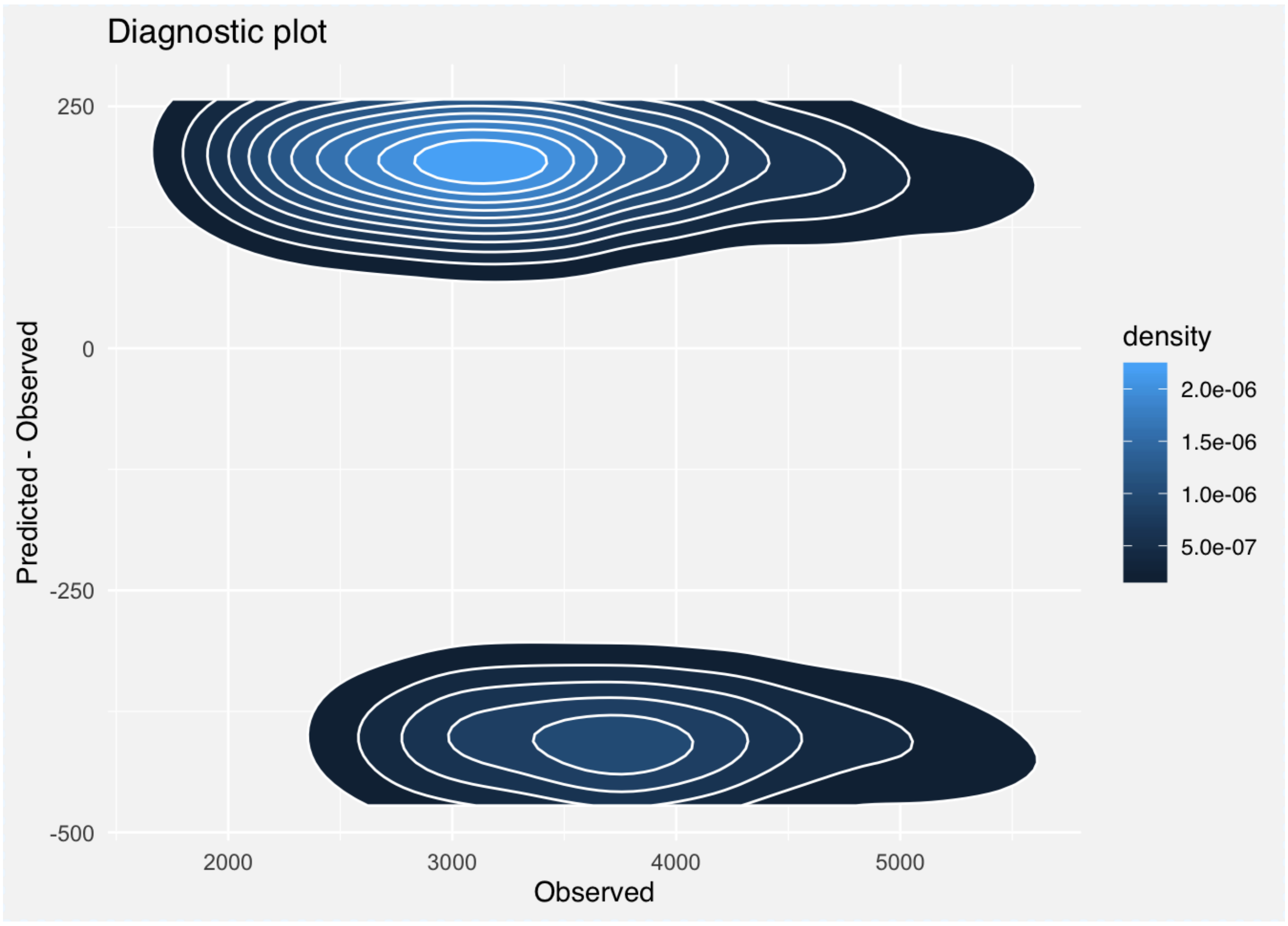
```
ggplot(mp_rf,
       aes(observed, diff) ) +
  stat_density_2d(aes(fill = ..level..), geom = "polygon", colour =
"white") +
  scale_fill_gradient(name = "density") +
  xlab("Observed") +
  ylab("Predicted - Observed") +
  ggtitle("Diagnostic plot") +
  theme_mi2()
```



In case of the random forest we can see that the higher the value of the apartment (x axis) the more the model underestimates the actual apartment price.

A similar plot for the linear model reveals two distinct groups of residuals (corresponding to the bumps in the inverse CDF).

```
ggplot(mp_lm,
       aes(observed, diff) ) +
  stat_density_2d(aes(fill = ..level..), geom = "polygon", colour =
"white") +
  scale_fill_gradient(name = "density") +
  xlab("Observed") +
  ylab("Predicted - Observed") +
  ggtitle("Diagnostic plot") +
  theme_mi2()
```



1. Load DALEX package and explore its in-built `dragons` and `dragons_test` datasets. The feature to predict is `life_length`.
2. Create two explainers – one for a linear model and one for a random forest.
3. Compare the models' performance in terms of the distribution of residuals. Which one performs better?

3.4 How does the global model work?

We understand how our models perform overall but can we trust them? Do they use features that are known to explain the target variable? If so, do they do it in a way that agrees with domain knowledge? We can answer these questions by first looking at the features that contribute the most to the predictions, i.e., feature importance. Then, we can explore how our model uses individual variables - continuous or categorical - to make a prediction.

3.4.1 Feature importance

Many algorithms use model-specific methods to calculate feature importance. For example, in random forest models it can be calculated based on mean decrease in accuracy or node impurity. In contrary, DALEX uses a model-agnostic measure of feature importance, i.e., permutation importance. It relies on randomly shuffling each feature (and therefore breaking the relationship between the feature and the target) and observing the effect on model accuracy (loss function). This method can be applied to any type of model because it does not rely on internal model parameters, such as node purity or model coefficients. We can calculate it using `feature_importance()` from the ingredients package:

```
vi_rf <- ingredients::feature_importance(
  explainer_rf,
  loss_function = loss_root_mean_square)

vi_lm <- ingredients::feature_importance(
  explainer_lm,
  loss_function = loss_root_mean_square)
```

It takes two arguments: the explainer object and the loss function that should be used. As we're dealing with the regression problem, we use root mean squared error. The lower the loss function value, the better the model fit.

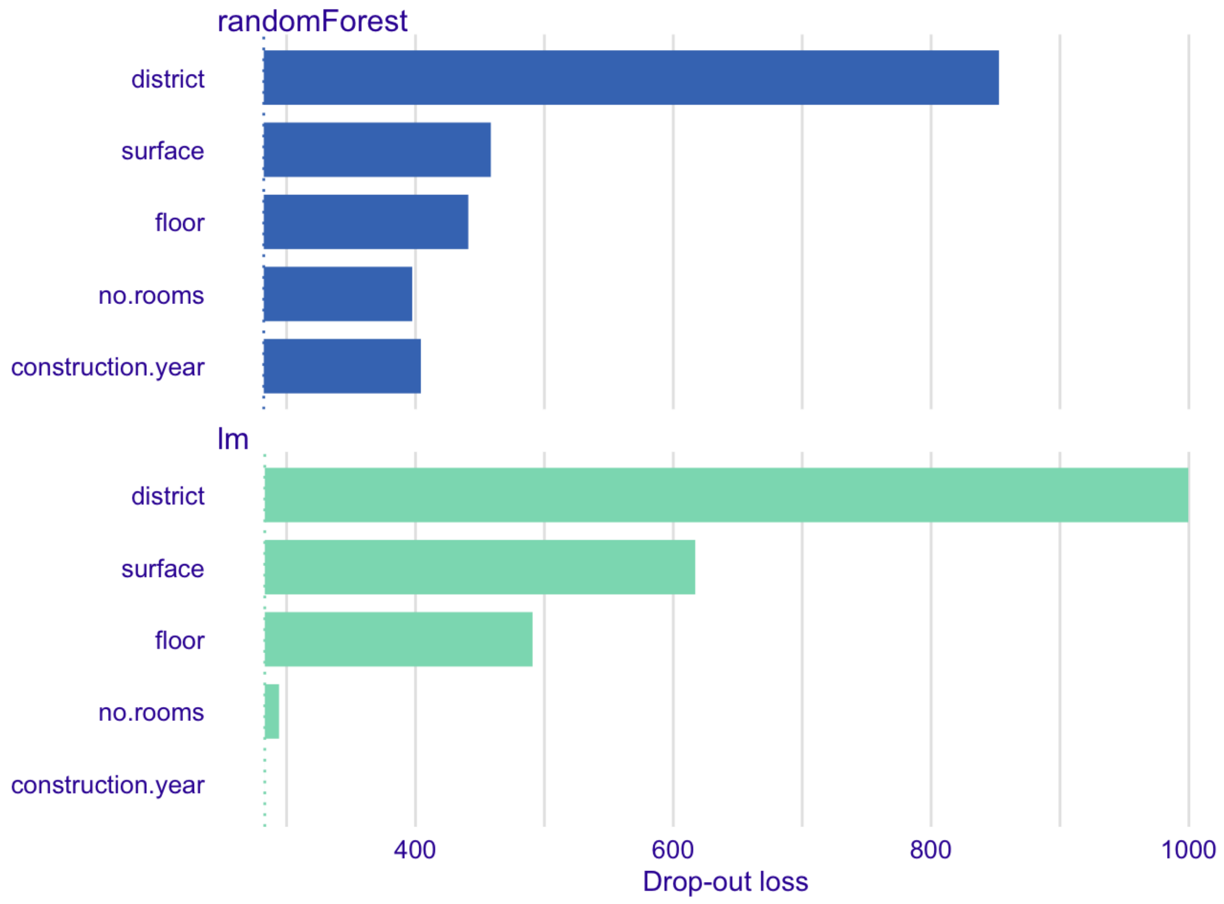
When we print this object to the console we can see the loss function value for the full model (the model we trained) and then for each case when a relevant feature was shuffled. Finally, the baseline model value describes the loss function for when all the features where shuffled and thus presents the 'worst case' scenario. The higher `dropout_loss`, the worse the model performance and thus the more important the feature.

```
vi_rf
```

```
##            variable dropout_loss        label
## 1       _full_model_     282.2885 randomForest
## 2          no.rooms     397.4441 randomForest
## 3 construction.year     404.1588 randomForest
## 4             floor     440.9568 randomForest
## 5           surface     458.4539 randomForest
## 6          district     852.6393 randomForest
## 7        _baseline_    1123.3749 randomForest
```

As in previous examples, we can plot feature importance objects using the generic `plot()` function:

```
plot(vi_rf, vi_lm)
```



In both models the order of feature importance is almost identical, however, in the random forest model `construction.year` seems to carry more weight than in the linear model. Let's explore why.

3.4.2 Explore a single continuous variable

Once we understand the importance of each of the features, we may want to explore the relationship between one or two of them and the target variable in more detail. In case of continuous variables, we can use Partial Dependence Plots (PDP) that show how each feature affects the model's prediction. This is done by running the fitted model over a grid of predictor values and then taking a mean output value per predictor value. It is a very useful way to gain insight on predictor-target relationship for any model, including black-box models.

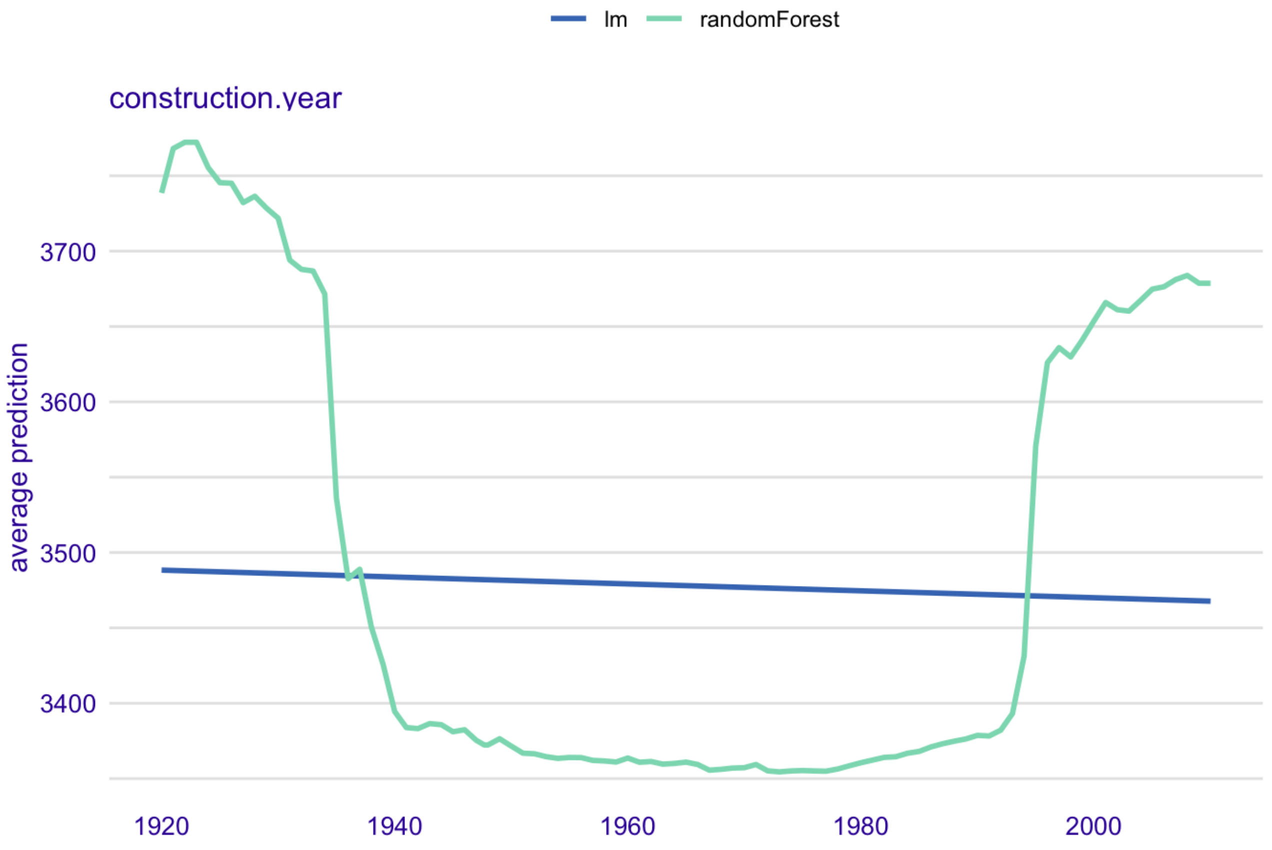
Let's build a single variable explainer for a continuous variable using the `partial_dependency()` function from the ingredients package, where we define a predictor of interest in the `variables` argument and the type of the response to be calculated in the `variable_type` argument:

```
pdp_rf <- ingredients::partial_dependency(
  explainer_rf, variables = "construction.year",
  variable_type = "numerical")

pdp_lm <- ingredients::partial_dependency(
  explainer_lm, variables = "construction.year",
  variable_type = "numerical")
```

Now, let's plot the two explainer objects and compare the output:

```
plot(pdp_rf, pdp_lm)
```



What we can see is that the linear model described dependence of the apartment price and the year when the apartment was built as a slowly decreasing linear relationship. On the other hand, the random forest partial dependence plot shows a non-linear relationship with higher prices for new and pre-war flats and the lower price for all the properties in between (which can be seen in the training data). This explains why `construction.year`’s feature importance was so low in the linear model but not in the random forest.

3.4.3 Explore a single categorical variable

Partial dependence plots are excellent for visualising the relationship between a continuous feature and the target variable but are inappropriate for categorical predictors. In such cases, in order to understand the relationship between the feature and the target, we can use Merging Paths Plots (MPP).

MPPs offer a way to present the average target value associated with a given group level and assess how different groups are similar to each other. In a nutshell, merging paths is a method of clustering categorical variables into non-overlapping groups by fitting a series of models, where each consecutive model is created based on the fusion of two closest groups with respect to their goodness of fit measure (the Likelihood Ratio Test-based distance). This hierarchy is summarized in a consistent graphical way in merging paths plots from the `factorMerger` package.

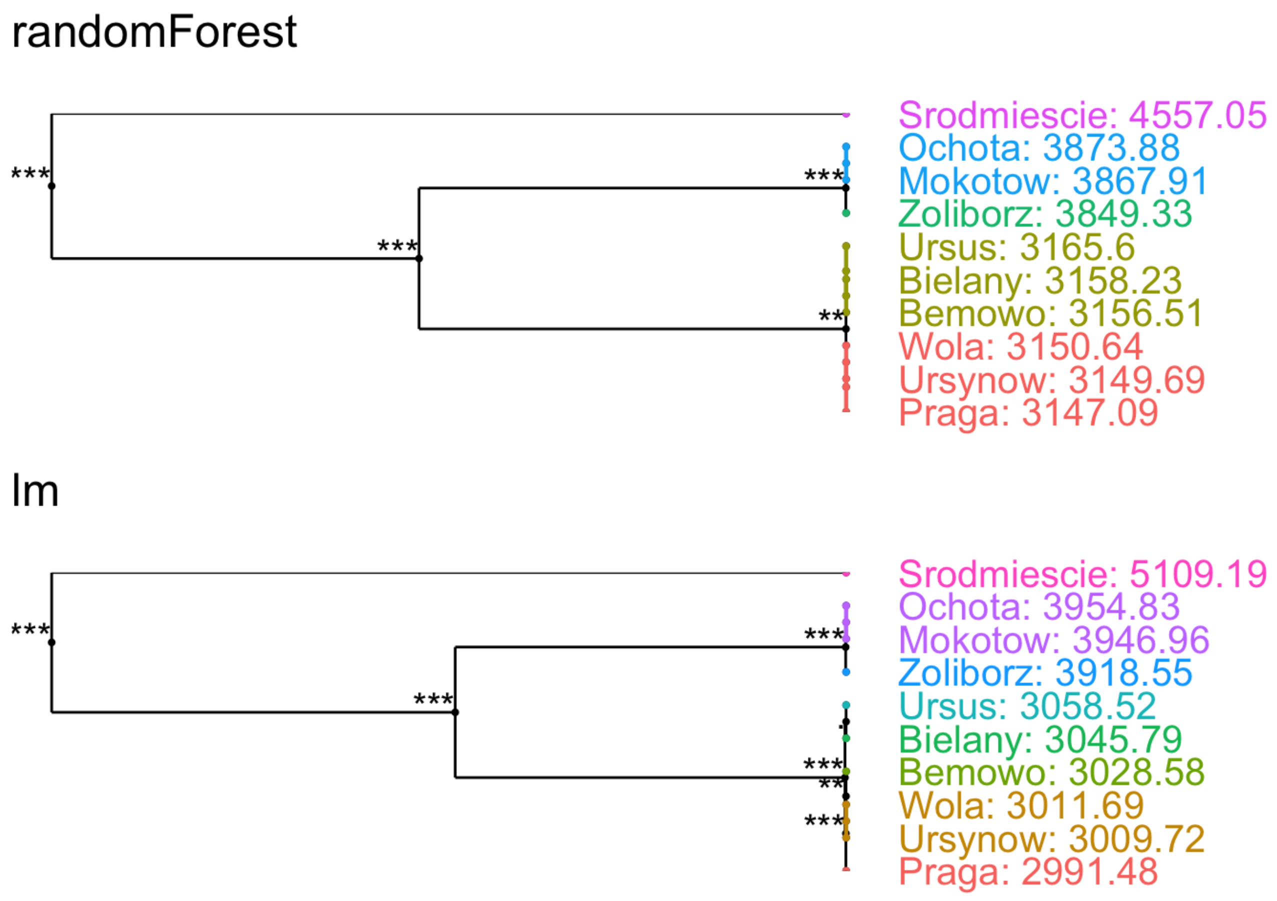
In order to produce MPPs, we will use the `single_variable()` function, and explore the `district` variable with the `type` argument set to `factor`:

```
svd_rf  <- single_variable(
  explainer_rf,
  variable =  "district",
  type = "factor")

svd_lm  <- single_variable(
  explainer_lm,
  variable =  "district",
  type = " factor")
```

We can obtain merging paths plots by calling `plot()` on these objects:

```
plot(svd_rf, svd_lm)
```



The above plot summarizes the structure of group similarities. Here, colours signify groups of districts with similar averages. Numbers displayed on the right-hand side show district averages. The position on which branches are merged corresponds to the likelihood of a model with combined groups. Stars denote the results of pairwise tests for selected groups of variables (p-values from likelihood ratio tests between consecutive models)

If we compare the two models we can see that the group structure is very similar: Srodmiescie, the city centre, has the highest mean price and creates a separate cluster. More central and well connected districts, such as Ochota, Mokotow and Zoliborz cluster together and are set aside from the remaining, less affluent districts. It's worth noting that even though the general group structure is very similar between the models, the random forest's group means for more central districts are on average lower than those from the linear model.

1. Use previous linear and random forest models trained on `dragons` dataset, and their explainers.
2. Extract and compare the models' variable importance. How are they similar? How are they different?
3. Explore `height` in both models using Partial Dependence Plots. What are the differences and why?
4. Explore `colour` in both models using Merging Paths Plots. How do they compare?

3.5 Summary

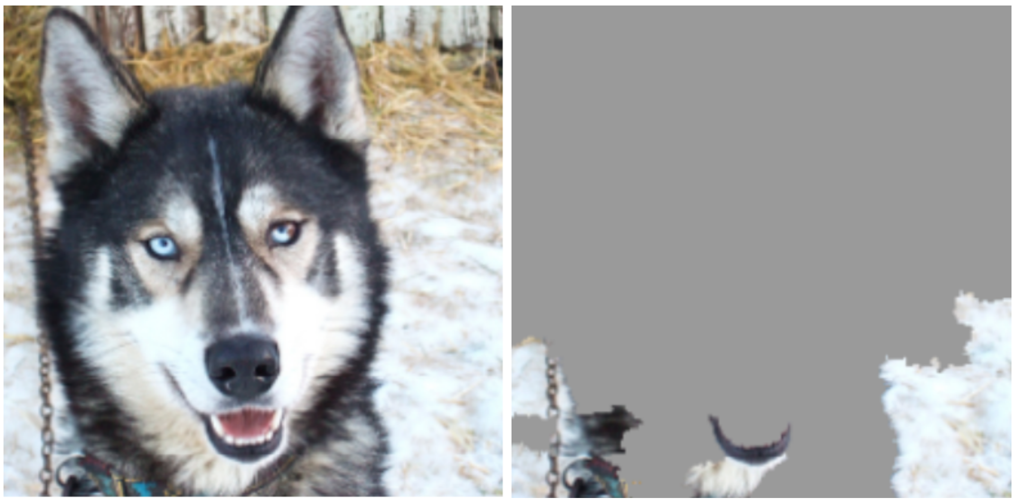
What have we learned about our global models? That despite very similar mean accuracy, random forest is overall more accurate except for most expensive apartments, price of which it consistently underestimates. We also understand why this is: linear model fails to capture a non-linear relationship between the flat price and the year it was built, while random forest tends to underestimate the price of more expensive flats located in the most central district. We achieved this level of understanding by running only a handful of functions, all of which had a consistent syntax and required a single package to be loaded in the workspace (DALEX).

4 LIME - Local Interpretable Model Explanations

4.1 Motivational Examples

When we check model's accuracy we are looking at how often the model makes the right prediction. We are not, however, looking at *why* it makes a certain prediction. Both are important in assessing a model's usefulness.

Let's revisit two examples from the second chapter of classifiers which may well be very accurate but for the "wrong" reasons. A classifier is supposed to distinguish between huskies and wolves. In the training set, all pictures with wolves also show snow whereas none of the pictures of huskies do. This can lead to the classifier learning to detect snow, rather than the difference between huskies and wolves. This is hard to pick up on in the raw data but easy to spot when the regions contributing to a prediction are highlighted.



Similarly, if the explanations for a model classifying emails as covering atheism vs Christianity include whether or not "words" like "Re:" or "Nntp-Posting-Host" are present in the email we may not trust this model to have captured the difference between "atheism" and "Christianity" in a meaningful way. In contrast, if we were to review why a model predicted that a person has the flu and the explanations were that sneezing and having a headache were indicators for the flu while "no fatigue" was an indicator for "no flu", we probably would trust the model to have captured something sensible.

There are some important aspects to note about these examples:

- We look at the explanations of multiple instances to assess if we trust the model as a whole but the explanations are always local, i.e., specific to a single image or email. So while we may conclude that the model is actually detecting snow, the detected region of snow (i.e., the local explanation) is different for each image.

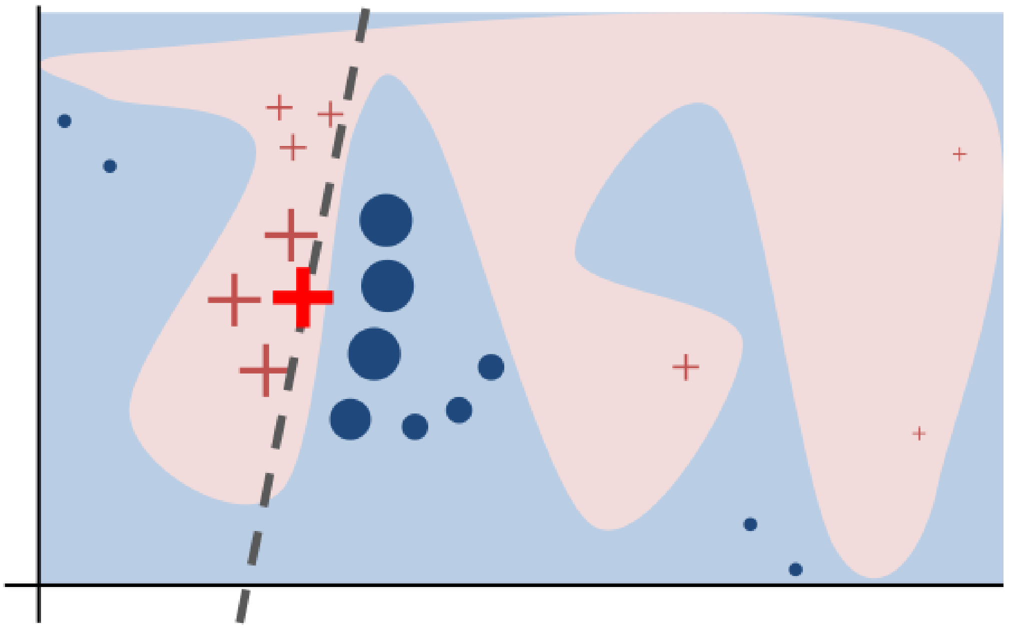
- The explanations involve a simplified version ("interpretable representation") of the original features: the explanations for the image are based on a few regions ("super-pixels") instead of individual pixels, and the explanations for the text are based on the presence/absence of individual words rather than, e.g., word embeddings.

In the next section we are introducing the theoretic framework which formalises and generalises these examples before we apply it to the iris dataset in the following section.

4.2 Theory

At its core, the idea of LIME is to approximate the behaviour of the complex model for a specific observation by fitting a simple, interpretable model to sampled data close to the specific observation.

To break this down, we are introducing some clarifying notation but are referring the reader to the work by Ribeiro et al (2016)[6] for the full details.


The basic idea of LIME (Ribeiro et al, 2016)

For an observation $x \in R^d$ we are explaining the prediction $f(x)$ of the complex model f by fitting a simple model g locally around x. A single observation x is not enough data to fit a model so we generate a perturbed sample z of n observations. We calculate the distance between each point in z to the original observation x and convert that to a similarity measure $\pi_x(z)$. The simple model tries to explain the predictions from the complex model, $f(z)$, and uses the similarities $\pi_x(z)$ as weights in the estimation. This means that data points closer to the original observation have a stronger influence, resulting in a local model.

The simple model g should be interpretable with regard to two aspects:

1. The model type itself should be interpretable, e.g., a linear model.

2. The covariates used in the model should be easily interpretable.

You may recall that in the example of the email text classifier, the explanations, i.e., the covariates in the simple model, were binary indicators whether a word was present, whereas the complex model may take word embeddings as features. This is referred to an interpretable representation $x' \in \{0,1\}^{d'}$ of x. The simple model g takes these interpretable representations as covariates and explains $f(z)$ through z' rather than z.

The choice of g is based on two considerations: it should a) be faithful to the complex model and b) be as simple as possible. These two aspects are captured in the loss $\mathcal{L}$ between f and g, weighted by $\pi_x(z)$ and the complexity Ω. The optimal model ξ of all possible models $g \in \mathcal{G}$ is thus

$$\xi = \operatorname{argmin}_g \mathcal{L}(f, g, \pi_x(z)) + \Omega(g)$$

For the implementation, $\mathcal{L}$ was chosen as the locally weighted quadratic loss and the complexity measure Ω such that only models with up to K covariates are considered (by setting the complexity of a model with more than K covariates to infinity). These choices of $\mathcal{L}$ and Ω make the optimisation problem intractable so the authors suggest to approximate this by a two-step approach:

- first, the best K covariates are selected based on a procedure like forward selection or shrinkage via a LASSO model,
- then a linear model is fitted with these covariates.

The coefficients of this linear model are then reported back for interpretation and are referred to as the local explanations.

4.3 Application to Iris Dataset

The main choices users of the LIME implementation get to make are:

- How to simplify the features to the interpretable representations.
- How and how many of those interpretable representations are selected as covariates for the simple model.
- How to calculate the similarity between perturbed data points and original observation.

One of the most commonly used example datasets in statistics and machine learning is the iris dataset, containing four measurements (width and length of sepal and petal, respectively) for three different species of iris flowers. When trying to predict the plant species, the two measurements of the petal are a lot more useful than the measurements of the sepal.

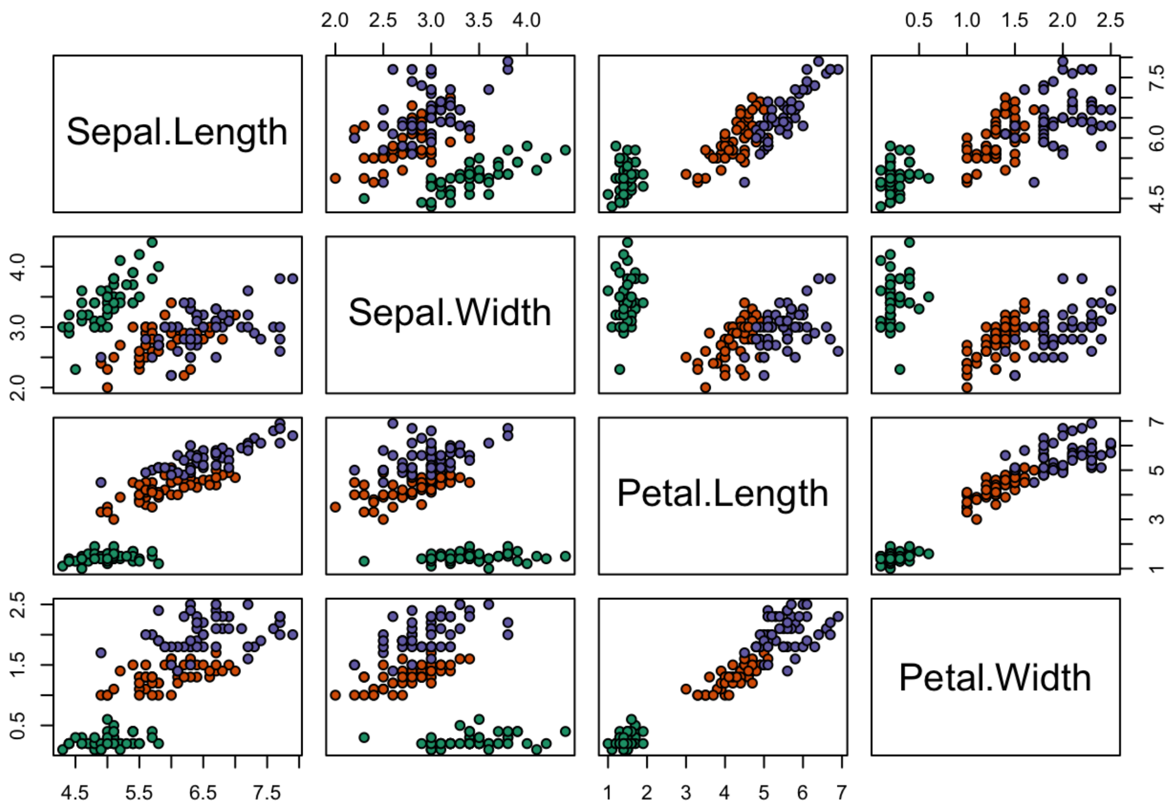


Figure 4.1: Iris data

We sample 5 observations for which we want to use LIME to explain our model's predictions and use the remaining 145 observations to train the model (here a random forest).

```
set.seed(304)
ind <- sample(1:nrow(iris), 5)
iris_explain <- iris[ind, 1:4]
iris_train <- iris[-ind, 1:4]
iris_lab <- iris[[5]][-ind]

library(caret)
model <- train(iris_train, iris_lab, method = "rf")
```

First, we create an `explainer` object, using the `lime()` function. This object specifies how the features of the complex model are transformed to their interpretable representations. For tabular data this includes whether or not continuous variables should be binned (`bin_continuous`) and if so how, i.e., how many bins (`n_bins`) and if the bins should be based on quantiles of the training data (`quantile_bins`).

```r
library(lime)
explainer <- lime(iris_train, model,
                  bin_continuous = TRUE, n_bins = 4,
                  quantile_bins = TRUE)
```

We can then use this explainer in the `explain()` function together with the observation(s) whose predictions by the complex model we'd like to explain. For the iris dataset we are doing this for the 5 observations in `iris_explain`. This means that LIME will fit 5 local simple models, one for each prediction.

We can request the local explanations for the `n_labels` most likely classes. While for the model here we could set this parameter up to 3 for the 3 species, we will only request explanations for the most likely class.

The number of covariates K in the local simple model can be set by the `n_features` argument. This number should be small enough for us to still be able to interpret them all. In this example, we are making things very simple by only using 2 covariates.

The sample size of z can be set by `n_permutations`.

The argument `feature_select` can be used to specify how the K best covariates are selected: options include `"none"` to use all available covariates, `"forward_selection"` in a Ridge regression, `"heighest_weights"` in a Ridge regression, `"lasso_path"` for selection based on a LASSO model, `"tree"` for feature selection based on XGBoost, and the default `"auto"` which uses `"forward_selection"` if `n_features` is less or equal to 6 and `"heighest_weights"` otherwise.

Further parameters allow specification of how the distance and similarity between the perturbed sample points z and the observation x is calculated.

```r
explanation <- explain(iris_explain, explainer,
                       n_labels = 1, n_features = 2,
                       n_permutations = 5000,
                       feature_select = "auto")
head(explanation)
```

```
## # A tibble: 6 x 13
##   model_type case  label label_prob model_r2 model_intercept
##   <chr>      <chr> <chr>      <dbl>    <dbl>           <dbl>
## 1 classific… 135   virg…      0.638    0.353          0.183
## 2 classific… 135   virg…      0.638    0.353          0.183
## 3 classific… 51    vers…      0.994    0.289          0.220
## 4 classific… 51    vers…      0.994    0.289          0.220
## 5 classific… 105   virg…      1        0.701          0.0935
## 6 classific… 105   virg…      1        0.701          0.0935
## # … with 7 more variables: model_prediction <dbl>, feature <chr>,
## #   feature_value <dbl>, feature_weight <dbl>, feature_desc <chr>,
## #   data <list>, prediction <list>
```

The resulting explanations are returned in a data frame along with model statistics. With the `plot_features()` function, we can get an easy to interpret visualisation of the explanations.

```r
plot_features(explanation)
```

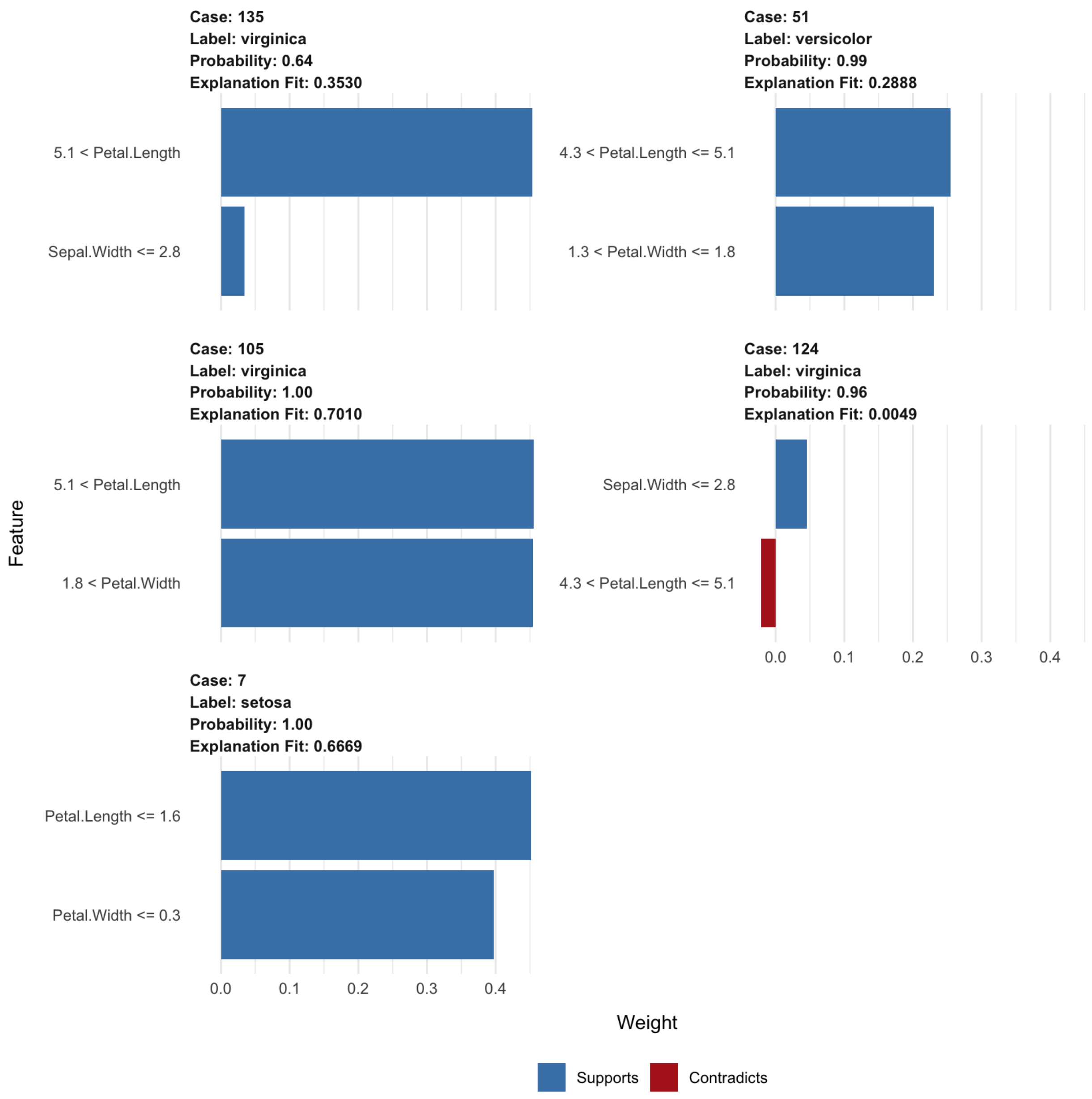


Figure 4.2: Explanations for prediction on iris data

Large coefficients translate to important covariates. Covariates supporting the prediction are coloured in green, those contradicting the prediction are coloured in red. In 4 out of our 5 cases, petal length is the important covariate, which we would expect given the plot in Figure 4.1. Taking a closer look at the explanations for the first observation (case 135), we see that the important feature for the prediction "virginica" is petal length larger than 5.1. When we compare this to the plot in Figure 4.1, we see that flowers with such measurements are nearly all coloured in blue, i.e., are indeed of species "virginica". Therefore we argue that we have indeed created a trustworthy model.

1. Prepare the `AdultUCI` dataset from the **arules** package by omitting all rows with missing values, recoding `education` and `income` as unordered factors, and sampling 1000 rows.
2. Set aside 5 rows as the observations we want to explain the predictions for.
3. Fit a random forest on the remaining data with `income` as the response.
4. Check the explanations for the predictions of the 5 observations set aside earlier. Do you trust this model? Explain your reasoning to your neighbour.

5 Further Concepts

LIME is not the only approach to understanding why a model makes a certain prediction. Lundberg and Lee (2017)[7] provide a framework called Additive Feature Attribution Methods which includes as special cases amongst others

- LIME
- DeepLIFT
- Layer-Wise Relevance Propagation
- Shapley values

However, Shapley values are the only one with theoretically proven properties so we'll give them a further look.

5.1 Shapley Values

Shapley Values were developed in game theory to distribute the pay-off of a game fairly amongst the players, i.e., how much has each player contributed to the pay-off? Translated to a model explanation context this becomes: how can we attribute a prediction to the features in a model, i.e., how much has each feature contributed to a prediction?

The proven theoretical properties of Shapley values are local accuracy, consistency, and missingness (i.e., if a feature is not in the model, its feature attribution value should be 0).

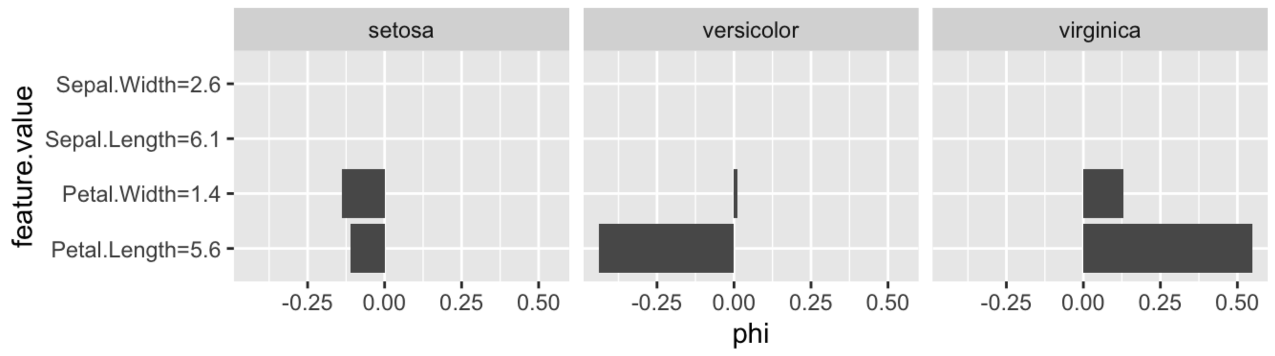
The basic idea is that how much a player increases the final pay-off when added to a team depends not only on their own skills but also on who else is in the team and their respective skills. In a modelling context, this dependence between players corresponds to, for example, correlation between features. To find the fair pay-off, we try all the combinations of players and average across all the marginal contributions of a player. Applied to a modelling context, we refit the model with different combinations of features and average across the marginal prediction.

The **iml** package contains an implementation of Shapley values in R. Using the **caret** `model` for the iris data from the previous chapter, we can construct a predictor object. Note that iml uses R6 classes and thus methods for an object are called with the `$` sign.

```r
library(iml)
predictor <- Predictor$new(model, data = iris_train, y = iris_lab)
```

To calculate the Shapley values for a prediction of interest, we pass the predictor object to the Shapley constructor function together with the features of the observation in whose prediction we are interested. To visualise the Shapley values, we can use the `plot()` method.

```r
shapley <- Shapley$new(predictor, x.interest = iris_explain[1,])
shapley$plot()
```



This approach requires (re-)fitting a potentially large number of models for the different combinations of features present/absent in the model. Shapley values and LIME are both part the framework of Additive Feature Attribution Methods and LIME works on binary features which can be translated to feature present/absent.

We can choose the loss function, complexity measure, and similarity measure such that we can leverage the LIME procedure to approximate the Shapley values without refitting all the models. This is implemented in the Python library SHAP (SHapley Additive exPlanation). The **shapper** package is a wrapper for this but requires **reticulate**.

5.2 Other Approaches

Further approaches include

- *Anchors* is an extension of LIME with an implementation in Python which currently supports classifiers for tabular and text data.
- **breakDown** is an R package which contains another approximation of Shapley values.
- *DeepLift* for deep learning models is available in Python.
- **xgboostExplainer** is an R package for xgboost models.

Footnotes:

1. Marco Tulio Ribeiro, GitHub, viewed 4 Sept 2019, https://github.com/marcotcr/lime/blob/master/doc/blog_post.md↩
2. Unknown, Quone, viewed 4 Sept 2019, http://qwone.com/~jason/20Newsgroups/↩
3. Jeffrey Dastin, Reuters, viewed 4 Sept 2019, https://www.reuters.com/article/us-amazon-com-jobs-automation-insight/amazon-scraps-secret-ai-recruiting-tool-that-showed-bias-against-women-idUSKCN1MK08G↩
4. Unknown, Wikipedia, viewed 4 Sept 2019, https://en.wikipedia.org/wiki/COMPAS_(software)↩
5. Julia Dressel and Hany Farid, Science Advances, viewed 4 Sept 2019, https://advances.sciencemag.org/content/4/1/eaao5580↩
6. Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "Why should i trust you?: Explaining the predictions of any classifier". In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM. 2016, pp. 1135–1144.↩
7. Lundberg, S.M. and Lee, S.I., 2017. A unified approach to interpreting model predictions. In Advances in Neural Information Processing Systems (pp. 4765-4774).↩