CIS 8398
Advanced AI Topics in Business

#Model-Agnostic Methods for XAI

Yu-Kai Lin

Agenda

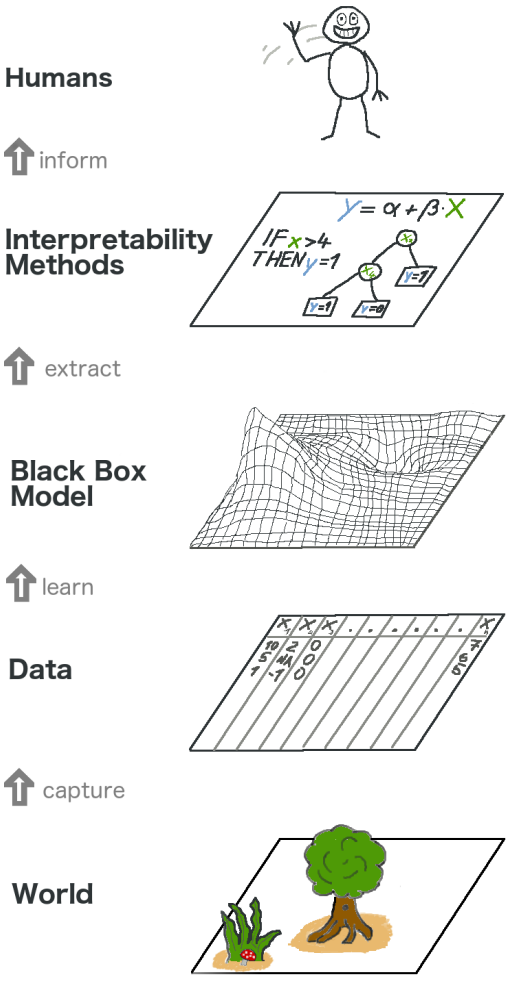
- Instance-level (local) explanations
- Dataset-level (global) explanations

[**Acknowledgements**] The materials in the following slides are based on the source(s) below:

- Interpretable Machine Learning by Christoph Molnar
- Explanatory Model Analysis by Przemyslaw Biecek and Tomasz Burzykowski
- AAAI 2022 Tutorial on Explainable AI

Achieving Explainable AI

- The prevailing solution to explanation problems is to use so called **interpretable models**, such as decision trees, k-nearest neightbors, or linear regressions
- As we discussed, interpretable models tend to be simple and hence have a lower prediction accuracy
- As such, the future of XAI is really about developing **model-agnostic methods** that can be applied to any ML models



Prerequisites

DALEX: moDel Agnostic Language for Exploration and eXplanation

```
install.packages("DALEX")
install.packages("DALEXtra")
install.packages("lime")
install.packages(c("rms", "randomForest", "e1071"))

library("tidyverse")
library("DALEX")
library("DALEXtra")
```

DALEXverse

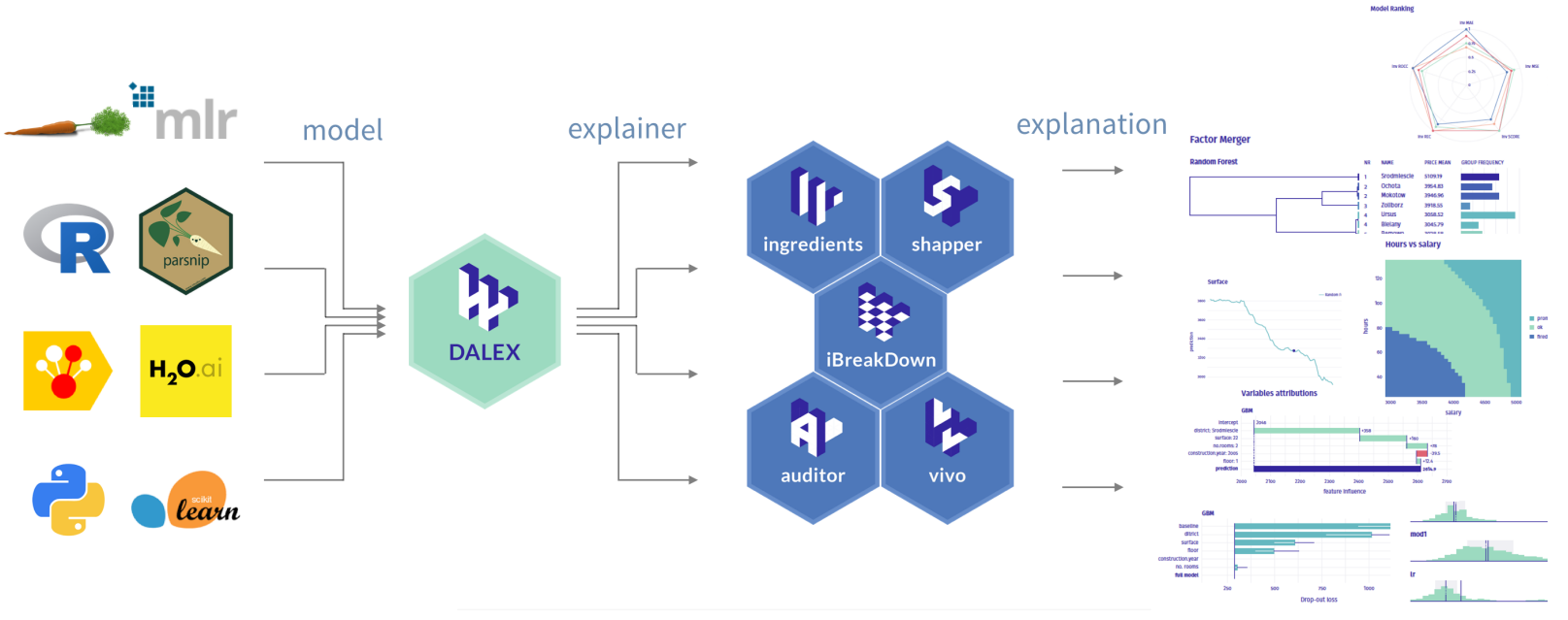


Image source: DrWhy

Summary of this lecture

Instance level explanations		Dataset level explanations	
Question *method*	*Function* *arguments*	*Question* *method*	*Function* *arguments*
What is the model prediction for the selected instance?	predict()	How good is the model? *ROC curve* *LIFT, Gain charts*	model_performance() geom = ecdf, boxplot, gain, lift, histogram
Which variables contribute to the selected prediction? Break Down SHAP, LIME	predict_parts() type = break_down, lime, shap, oscillations	Which variables are important to the model? Permutational Variable Importance	model_parts() type = variable_importance
How a variable affects the prediction? Ceteris paribus	predict_profile() type = ceteris_paribus	How a variable affects the average prediction? Partial Dependence Profile Accumulated Local Effects	model_profile() type = partial, accumulated, conditional geom = aggregates, profiles, points
Is the model well fitted around the prediction?	predict_diagnostics()	Is the model well fitted in general?	model_diagnostics()

Image source: mlr3 book

Datasets and Models

We will illustrate model-agnostic XAI methods by using two datasets from the `DALEX` package related to:

- **Classification**: predicting probability of survival for passengers of Titanic
  - This dataset is called `titanic` in the `DALEX` package
- **Regression**: predicting apartment prices in Warsaw, Poland
  - This dataset is called `apartments` in the `DALEX` package

We will consider the following ML models:

- Logistic/linear regression
- Random forest
- Support vector machine

Explore the Titanic data

The `titanic` data frame contains 2207 observations (for 1317 passengers and 890 crew members) and nine variables:

- *gender*, person's (passenger's or crew member's) gender, a factor (categorical variable) with two levels (categories): "male" and "female";
- *age*, person's age in years, a numerical variable; the age is given in (integer) years;
- *class*, the class in which the passenger travelled, or the duty class of a crew member; a factor with seven levels: "1st" (14.7%), "2nd" (12.9%), "3rd" (32.1%), "deck crew" (3%), "engineering crew" (14.7%), "restaurant staff" (3.1%), and "victualling crew" (19.5%);
- *embarked*, the harbor in which the person embarked on the ship, a factor with four levels: "Belfast" (8.9%), "Cherbourg" (12.3%), "Queenstown" (5.6%), and "Southampton" (73.2%);
- *country*, person's home country, a factor with 48 levels; the most common levels are "England" (51%), "United States" (12%), "Ireland" (6.2%), and "Sweden" (4.8%);
- *fare*, the price of the ticket (only available for passengers; 0 for crew members);
- *sibsp*, the number of siblings/spouses aboard the ship;
- *parch*, the number of parents/children aboard the ship;
- *survived*, a factor with two levels: "yes" (67.8%) and "no" (32.2%) indicating whether the person survived or not.

The first six rows of this dataset are presented in the table below.

gender	age	class	embarked	fare	sibsp	parch	survived
male	42	3rd	Southampton	7.11	0	0	no
male	13	3rd	Southampton	20.05	0	2	no
male	16	3rd	Southampton	20.05	1	1	no
female	39	3rd	Southampton	20.05	1	1	yes
female	16	3rd	Southampton	7.13	0	0	yes
male	25	3rd	Southampton	7.13	0	0	yes

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

```
glimpse(titanic)
```

```
## Rows: 2,207
## Columns: 9
## $ gender   <fct> male, male, male, female, female, male, male, female, male, m…
## $ age      <dbl> 42.0000000, 13.0000000, 16.0000000, 39.0000000, 16.0000000, 2…
## $ class    <fct> 3rd, 3rd, 3rd, 3rd, 3rd, 3rd, 2nd, 2nd, 3rd, 3rd, 3rd, 3rd, 3…
## $ embarked <fct> Southampton, Southampton, Southampton, Southampton, Southampt…
## $ country  <fct> United States, United States, United States, England, Norway,…
## $ fare     <dbl> 7.1100, 20.0500, 20.0500, 20.0500, 7.1300, 7.1300, 24.0000, 2…
## $ sibsp    <dbl> 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0…
## $ parch    <dbl> 0, 2, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0…
## $ survived <fct> no, no, no, yes, yes, yes, no, yes, yes, yes, no, no, no, yes…
```

The `titanic` data has some missing values.

- We will replace the missing *age* values by the mean of the observed ones, i.e., 30.
- Missing *country* is encoded by `"X"`.
- For *sibsp* and *parch*, we replace the missing values by the most frequently observed value, i.e., 0.
- For *fare*, we use the mean fare for a given *class*, i.e., 0 pounds for crew, 89 pounds for the first, 22 pounds for the second, and 13 pounds for the third class.

```
titanic = titanic |>
  mutate(country =  as.character(country)) |>
  replace_na(list(age = 30, country = "X", sibsp = 0, parch = 0)) |>
  mutate(country =  factor(country))

# impute missing fare based on class
titanic$fare[is.na(titanic$fare) & titanic$class == "1st"] = 89
titanic$fare[is.na(titanic$fare) & titanic$class == "2nd"] = 22
titanic$fare[is.na(titanic$fare) & titanic$class == "3rd"] = 13
```

Explore the apartment data

The `apartments` dataset contains 1000 observations (apartments) and six variables:

- *m2.price*, apartment's price per square meter (in EUR), a numerical variable in the range of 1607--6595;
- *construction.year*, the year of construction of the block of flats in which the apartment is located, a numerical variable in the range of 1920--2010;
- *surface*, apartment's total surface in square meters, a numerical variable in the range of 20--150;
- *floor*, the floor at which the apartment is located (ground floor taken to be the first floor), a numerical integer variable with values ranging from 1 to 10;
- *no.rooms*, the total number of rooms, a numerical integer variable with values ranging from 1 to 6;
- *district*, a factor with 10 levels indicating the district of Warsaw where the apartment is located.

The first six rows of this dataset are presented in the table 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

Models considered for this dataset will use *m2.price* as the (continuous) dependent variable. Models' predictions will be validated on a set of 9000 apartments included in data frame `apartments_test`.

```
glimpse(apartments)
```

```
## Rows: 1,000
## Columns: 6
## $ m2.price          <dbl> 5897, 1818, 3643, 3517, 3013, 5795, 2983, 2346, 4745…
## $ construction.year <dbl> 1953, 1992, 1937, 1995, 1992, 1926, 1970, 1985, 1928…
## $ surface           <dbl> 25, 143, 56, 93, 144, 61, 127, 105, 145, 112, 74, 75…
## $ floor             <int> 3, 9, 1, 7, 6, 6, 8, 8, 6, 9, 5, 7, 1, 6, 9, 2, 6, 6…
## $ no.rooms          <dbl> 1, 5, 2, 3, 5, 2, 5, 4, 6, 4, 2, 3, 3, 5, 3, 4, 2, 1…
## $ district          <fct> Srodmiescie, Bielany, Praga, Ochota, Mokotow, Srodmi…
```

Models for Titanic

```
library("rms")
# the rcs() function allows us to model potentially non-linear effect of age
titanic_lmr <- lrm(survived == "yes" ~ gender + rcs(age) + class +
                   sibsp + parch + fare + embarked, titanic)

library("randomForest")
set.seed(123)
titanic_rf <- randomForest(survived ~ class + gender + age +
                           sibsp + parch + fare + embarked, data = titanic)

library("e1071")
titanic_svm <- svm(survived == "yes" ~ class + gender + age +
                   sibsp + parch + fare + embarked, data = titanic,
                   type = "C-classification", probability = TRUE)
```

Models' predictions

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

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

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

```
(pred_lmr <- predict(titanic_lmr, johnny, type = "fitted"))
```

```
##         1
## 0.7677036
```

```
(pred_rf <- predict(titanic_rf, johnny, type = "prob"))
```

```
##      no   yes
## 1 0.622 0.378
## attr(,"class")
## [1] "matrix" "array"  "votes"
```

```
(pred_svm <- predict(titanic_svm, johnny, probability = TRUE))
```

```
##     1
## FALSE
## attr(,"probabilities")
##       FALSE      TRUE
## 1 0.7786512 0.2213488
## Levels: FALSE TRUE
```

The predicted probabilities of survival are 0.77, 0.38, and 0.22 for the logistic regression, random forest, and SVM models, respectively.

Which one should you trust? This is why XAI can be useful.

Let's try another passenger: Henry

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

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

```
##         1
## 0.4318245
```

```
predict(titanic_rf, henry, type = "prob")[,2]
```

```
## [1] 0.236
```

```
attr(predict(titanic_svm, henry, probability = TRUE),"probabilities")[,2]
```

```
## [1] 0.1777008
```

For Henry, the predicted probability of survival is lower than for Johnny.

Models for apartment price

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

library("randomForest")
set.seed(123)
apartments_rf <- randomForest(m2.price ~ ., data = apartments)

library("e1071")
apartments_svm <- svm(m2.price ~ construction.year + surface + floor +
                                 no.rooms + district, data = apartments)
```

Models' predictions

The actual and predicted prices for the first five observations from `apartments_test` are provided below:

```
apartments_test$m2.price[1:5]
```

```
## [1] 4644 3082 2498 2735 2781
```

```
predict(apartments_lm, apartments_test[1:5,])
```

```
##     1001     1002     1003     1004     1005
## 4820.009 3292.678 2717.910 2922.751 2974.086
```

```
predict(apartments_rf, apartments_test[1:5,])
```

```
##     1001     1002     1003     1004     1005
## 4216.896 3193.303 2697.534 2742.046 2965.499
```

```
predict(apartments_svm, apartments_test[1:5,])
```

```
##     1001     1002     1003     1004     1005
## 4590.076 3012.044 2369.748 2712.456 2681.777
```

Let's summarize the predictive performance of the linear regression and random forest models by computing the square root of the mean-squared-error (RMSE):

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

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

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

```
## [1] 287.7215
```

```
predicted_apartments_svm <- predict(apartments_svm, apartments_test)
sqrt(mean((predicted_apartments_svm - apartments_test$m2.price)^2))
```

```
## [1] 160.901
```

It appears that SVM made more accurate predictions, but we still cannot explain how SVM reached these predictions.

Your turn

Follow the previous slides to:

1. Install and load all the necessary packages
2. Inspect and explore the `titanic` and `apartments` datasets
3. Build models for each dataset
4. Obtain predictions from each model

Models' explainers

Black-box models may have very different structures. The `explain` function from the `DALEX` package creates a unified representation of a model.

Each explainer-object contains all elements needed to create a model explanation. With explainers, we don't need to worry about different structures in each model. This can make our code look a lot cleaner.

The code in the following slides creates explainers for all our models.

Explainer for `titanic_lmr`:

```
titanic_lmr_exp <- explain(
  model = titanic_lmr, data = titanic[, -9],
  y = titanic$survived == "yes", label = "Logistic Regression",
  type = "classification")
```

```
## Preparation of a new explainer is initiated
##   -> model label       :  Logistic Regression
##   -> data              :  2207  rows  8  cols
##   -> target variable   :  2207  values
##   -> predict function  :  yhat.lrm  will be used (  default  )
##   -> predicted values  :  No value for predict function target column. (  default
##   -> model_info        :  package rms , ver. 6.7.1 , task classification (  defau
##   -> model_info        :  type set to  classification
##   -> model_info        :  Model info detected classification task but 'y' is a lo
##   -> predicted values  :  numerical, min =  0.002671631 , mean =  0.3221568 , max
##   -> residual function :  difference between y and yhat (  default  )
##   -> residuals         :  numerical, min =  -0.9845724 , mean =  -2.491758e-09 ,
##   A new explainer has been created!
```

Explainer for `titanic_rf`:

```
titanic_rf_exp <- explain(
  model = titanic_rf, data = titanic[, -9],
  y = titanic$survived == "yes", label = "Random Forest",
  type = "classification")
```

```
## Preparation of a new explainer is initiated
##   -> model label       :  Random Forest
##   -> data              :  2207  rows  8  cols
##   -> target variable   :  2207  values
##   -> predict function  :  yhat.randomForest  will be used (  default  )
##   -> predicted values  :  No value for predict function target column. (  default
##   -> model_info        :  package randomForest , ver. 4.7.1.1 , task classificati
##   -> model_info        :  type set to  classification
##   -> model_info        :  Model info detected classification task but 'y' is a lo
##   -> predicted values  :  numerical, min =  0 , mean =  0.2371355 , max =  1
##   -> residual function :  difference between y and yhat (  default  )
##   -> residuals         :  numerical, min =  -0.882 , mean =  0.0850213 , max =  1
##   A new explainer has been created!
```

Explainer for `titanic_svm`:

```
titanic_svm_exp <- explain(
  model = titanic_svm, data = titanic[, -9],
  y = titanic$survived == "yes", label = "Support Vector Machine",
  type = "classification")
```

```
## Preparation of a new explainer is initiated
##   -> model label       :  Support Vector Machine
##   -> data              :  2207  rows  8  cols
##   -> target variable   :  2207  values
##   -> predict function  :  yhat.svm  will be used (  default  )
##   -> predicted values  :  No value for predict function target column. (  default
##   -> model_info        :  package e1071 , ver. 1.7.13 , task classification (  de
##   -> model_info        :  type set to  classification
##   -> model_info        :  Model info detected classification task but 'y' is a lo
##   -> predicted values  :  numerical, min =  0.08769169 , mean =  0.3241748 , max
##   -> residual function :  difference between y and yhat (  default  )
##   -> residuals         :  numerical, min =  -0.8669913 , mean =  -0.002018072 , m
##   A new explainer has been created!
```

Explainer for `apartments_lm`:

```
apartments_lm_exp <- explain(
  model = apartments_lm, data = apartments_test[, -1],
  y = apartments_test$m2.price, label = "Linear Regression",
  type = "regression")
```

```
## Preparation of a new explainer is initiated
##   -> model label       :  Linear Regression
##   -> data              :  9000  rows  5  cols
##   -> target variable   :  9000  values
##   -> predict function  :  yhat.lm  will be used (  default  )
##   -> predicted values  :  No value for predict function target column. (  default
##   -> model_info        :  package stats , ver. 4.3.1 , task regression (  default
##   -> model_info        :  type set to  regression
##   -> predicted values  :  numerical, min =  1792.597 , mean =  3506.836 , max =
##   -> residual function :  difference between y and yhat (  default  )
##   -> residuals         :  numerical, min =  -257.2555 , mean =  4.687686 , max =
##   A new explainer has been created!
```

Explainer for `apartments_rf`:

```
apartments_rf_exp <- explain(
  model = apartments_rf, data = apartments_test[, -1],
  y = apartments_test$m2.price, label = "Random Forest",
  type = "regression")
```

```
## Preparation of a new explainer is initiated
##   -> model label       :  Random Forest
##   -> data              :  9000  rows  5  cols
##   -> target variable   :  9000  values
##   -> predict function  :  yhat.randomForest  will be used (  default  )
##   -> predicted values  :  No value for predict function target column. (  default
##   -> model_info        :  package randomForest , ver. 4.7.1.1 , task regression (
##   -> model_info        :  type set to  regression
##   -> predicted values  :  numerical, min =  1970.401 , mean =  3506.833 , max =
##   -> residual function :  difference between y and yhat (  default  )
##   -> residuals         :  numerical, min =  -761.9362 , mean =  4.691025 , max =
##   A new explainer has been created!
```

Explainer for `apartments_svm`:

```
apartments_svm_exp <- explain(
  model = apartments_svm, data = apartments_test[, -1],
  y = apartments_test$m2.price, label = "Support Vector Machine",
  type = "regression")
```

```
## Preparation of a new explainer is initiated
##   -> model label       :  Support Vector Machine
##   -> data              :  9000  rows  5  cols
##   -> target variable   :  9000  values
##   -> predict function  :  yhat.svm  will be used (  default  )
##   -> predicted values  :  No value for predict function target column. (  default
##   -> model_info        :  package e1071 , ver. 1.7.13 , task regression (  defaul
##   -> model_info        :  type set to  regression
##   -> predicted values  :  numerical, min =  1721.466 , mean =  3502.935 , max =
##   -> residual function :  difference between y and yhat (  default  )
##   -> residuals         :  numerical, min =  -547.0168 , mean =  8.588531 , max =
##   A new explainer has been created!
```

Your turn

Create the 6 explainers shown on the previous slides.

The following slides will show how to use these explainers to create local and global interpretations, so you must have them in your R environment.

Instance-level Interpretation

Instance-level interpretation methods help us understand how a model yields a prediction for a particular single observation.

Popular approaches:

- Break-down Plots for Additive Attributions
- Break-down Plots for Interactions
- Shapley Additive Explanations (SHAP) for Average Attributions
- Local Interpretable Model-agnostic Explanations (LIME)
- Ceteris-paribus Profiles

For each approach, we will highlight the intuition, provide some examples, discuss its pros and cons, and give you an opportunity to practice it in R.

Break-down (BD) Plots

Intuition

Prediction $f(X)$ is an approximation of the expected value of the dependent variable Y given values of explanatory variables X.

Suppose we have in total J variables in X: $x_1, x_2, \ldots, x_J$, we assume that their effects are **additive**. In other words,

$$f(X) = f(x_1) + f(x_2) + \ldots + f(x_n)$$

The underlying idea of BD plots is to capture the contribution of an explanatory variable to the model's prediction by computing the **shift** in the expected value of Y, while fixing the values of other variables.

Break-down (BD) Plots

Examples

```
# to explain Random Forest's prediction for johnny
titanic_rf_exp_bd <- predict_parts(
  titanic_rf_exp, new_observation = johnny,
  type = "break_down")

# to explain Random Forest'ss prediction for the first test apartment
new_apt = apartments_test[1, -1]
apartments_rf_exp_bd <- predict_parts(
  apartments_rf_exp, new_observation = new_apt,
  type = "break_down")
```

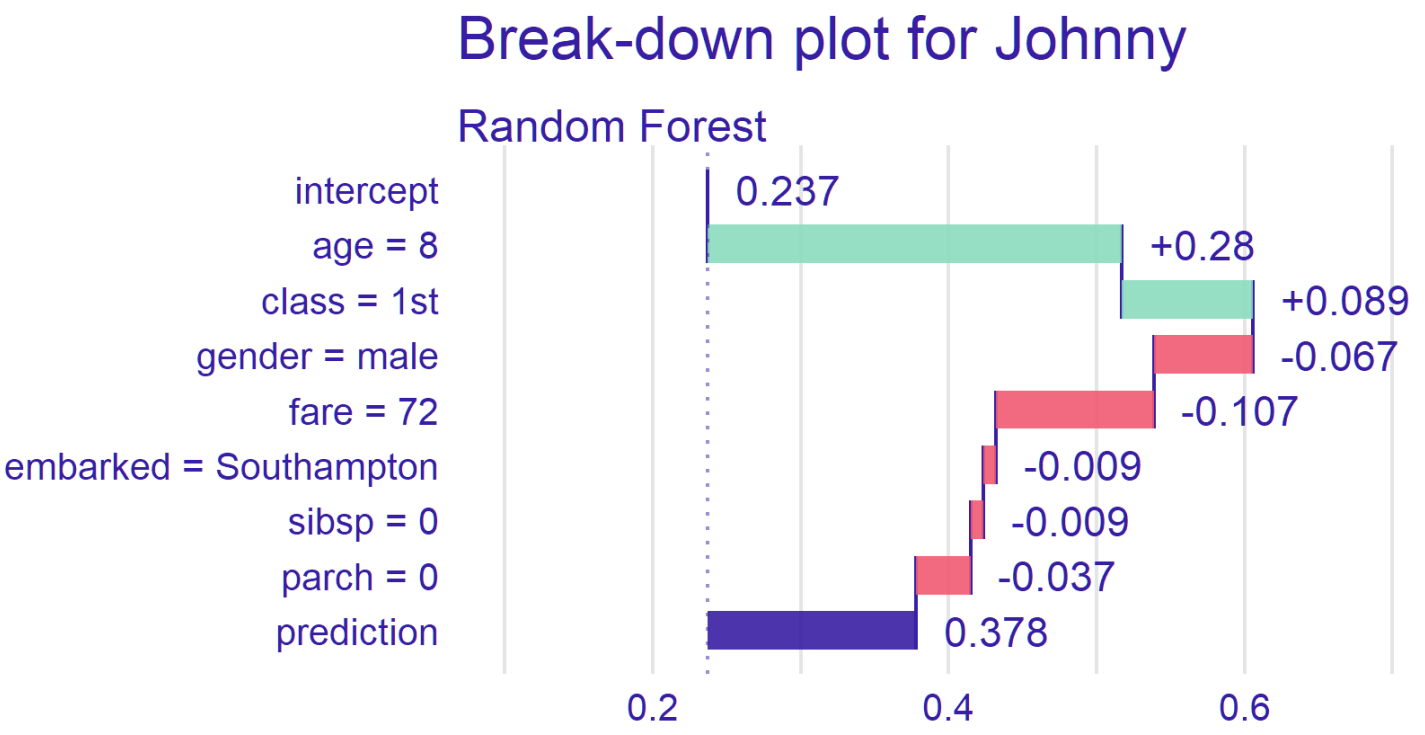
```
titanic_rf_exp_bd # contributions to johnny's survival probability
```

```
##                                          contribution
## Random Forest: intercept                        0.237
## Random Forest: age = 8                          0.280
## Random Forest: class = 1st                      0.089
## Random Forest: gender = male                   -0.067
## Random Forest: fare = 72                       -0.107
## Random Forest: embarked = Southampton          -0.009
## Random Forest: sibsp = 0                       -0.009
## Random Forest: parch = 0                       -0.037
## Random Forest: prediction                       0.378
```

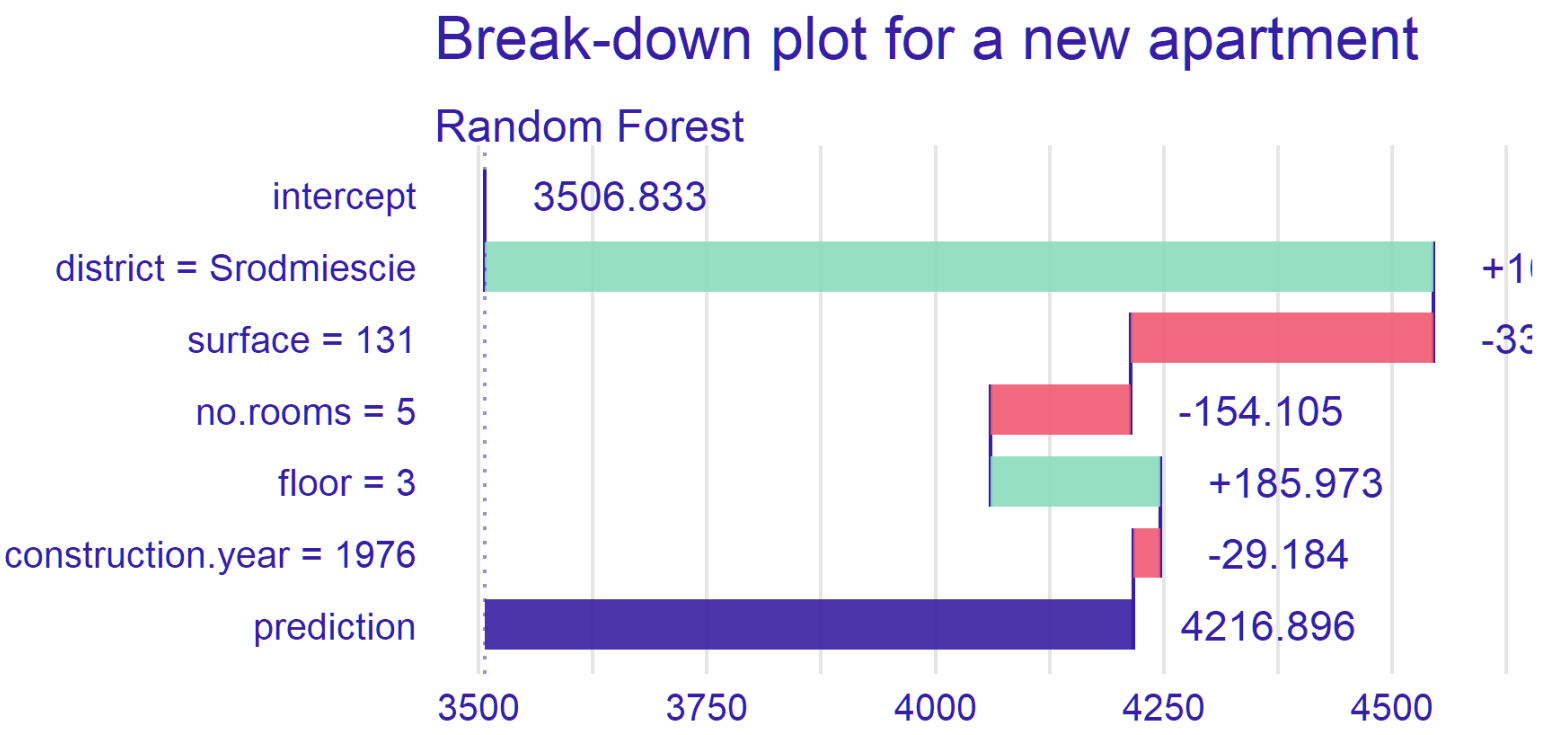
```
apartments_rf_exp_bd # contributions to new_apt's price
```

```
##                                            contribution
## Random Forest: intercept                       3506.833
## Random Forest: district = Srodmiescie          1038.432
## Random Forest: surface = 131                   -331.052
## Random Forest: no.rooms = 5                    -154.105
## Random Forest: floor = 3                        185.973
## Random Forest: construction.year = 1976         -29.184
## Random Forest: prediction                      4216.896
```

```
plot(titanic_rf_exp_bd) + ggtitle("Break-down plot for Johnny")
```



```
plot(apartments_rf_exp_bd) + ggtitle("Break-down plot for a new apartment")
```



Break-down (BD) Plots

Pros and Cons

Pros:

- The plots are, in general, compact and easy to understand.
- Numerical complexity of the BD algorithm is linear in relation to the number of explanatory variables.

Cons:

- BD plots may be misleading for models including interactions. This is because the plots show only the **additive** attributions.
- The choice of the ordering of the explanatory variables that is used in the calculation of the variable-importance measures is important.
- For models with a large number of variables, BD plots may be complex and include many explanatory variables with small contributions to the instance prediction.

Break-down (BD) Plots

Your turn

Use BD plots to interpret `titanic_svm` and `apartments_svm`

BD Plots for Interactions (iBD Plots)

Intuition

Interaction (deviation from additivity) means that the effect of an explanatory variable depends on the value(s) of other variable(s).

For the sake of simplicity, let's consider only two variables, *age* and *class*, from male passengers in Titanic. *Age* is a continuous variable, but we will use a dichotomized version of it, with two levels: boys (0-16 years old) and adults (17+ years old). Also, for *class*, we will consider just "2nd class" and "other".

Class	Boys (0-16)	Adults (>16)	Total
2nd	11/12 = 91.7%	13/166 = 7.8%	24/178 = 13.5%
other	22/69 = 31.9%	306/1469 = 20.8%	328/1538 = 21.3%
Total	33/81 = 40.7%	319/1635 = 19.5%	352/1716 = 20.5%

Question: What are the impacts of *age* and *class* to survival? (Two explanations on the next slide)

Explanation 1:

- Males, overall: 20.5%
  - Males, 2nd class: 13.5% (**-7 percentage points**)
  - Males, 2nd class, boys: 91.7% (**+78.2 percentage points**)

- By considering first the effect of *class*, and then the effect of *age*, we can conclude the effect of **−7 percentage points** for *class* and **+78.2 percentage points** for *age* (being a boy).

Explanation 2:

- Males, overall: 20.5%
  - Males, boys: 40.7% (**+20.2 percentage points**)
  - Males, boys, 2nd class: 91.7% (**+51 percentage points**)

- By considering first the effect of *age*, and then the effect of *class*, we can conclude the effect of **+20.2 percentage points** for *age* (being a boy) and **+51 percentage points** for *class*

BD Plots for Interactions (iBD Plots)

Examples

To calculate the attributions with the break-down method with interactions, we use the `predict_parts()` method with `type='break_down_interactions'` argument.

```
titanic_rf_exp_bdi <- predict_parts(
  titanic_rf_exp, new_observation = johnny,
  type = "break_down_interactions")

apartments_rf_exp_bdi <- predict_parts(
  apartments_rf_exp, new_observation = new_apt,
  type = "break_down_interactions")
```

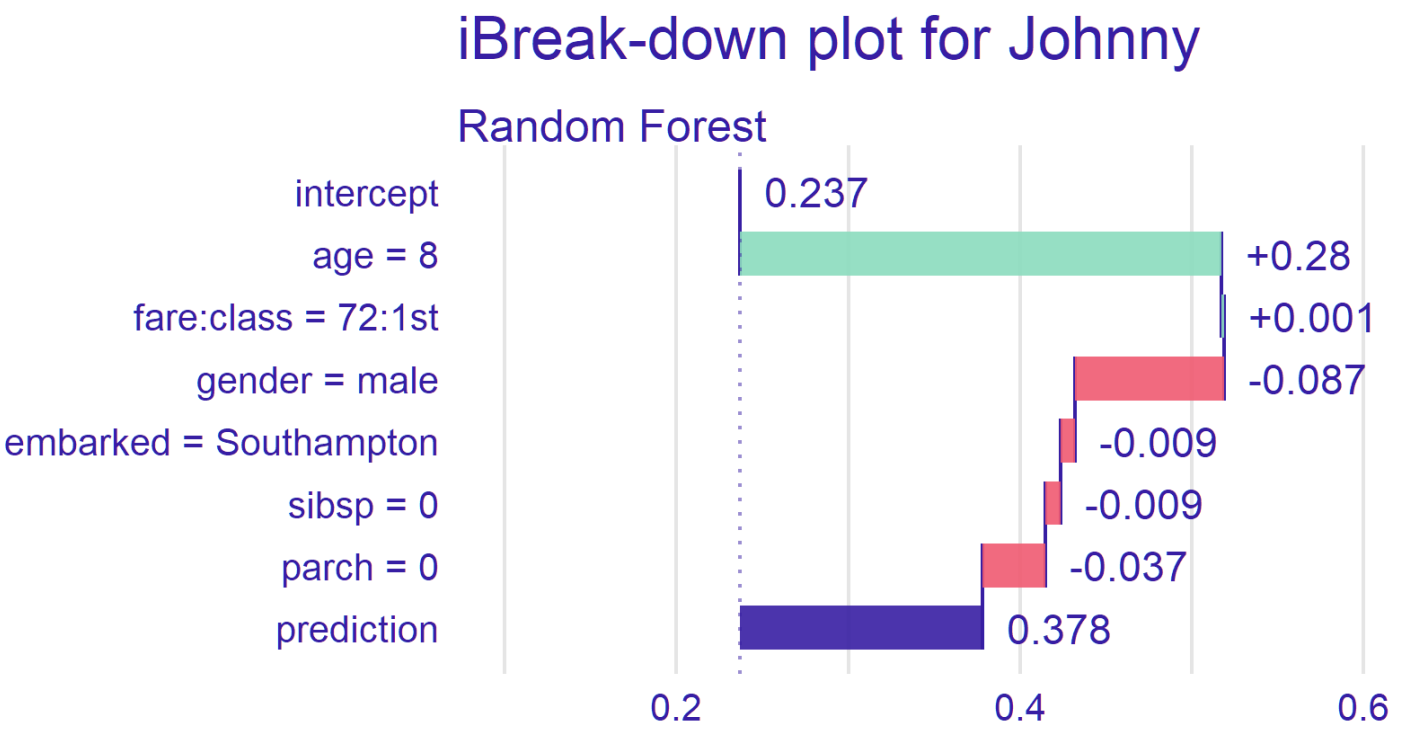
```
titanic_rf_exp_bdi # contributions to johnny's survival probability
```

```
##                                          contribution
## Random Forest: intercept                        0.237
## Random Forest: age = 8                          0.280
## Random Forest: fare:class = 72:1st              0.001
## Random Forest: gender = male                   -0.087
## Random Forest: embarked = Southampton          -0.009
## Random Forest: sibsp = 0                       -0.009
## Random Forest: parch = 0                       -0.037
## Random Forest: prediction                       0.378
```

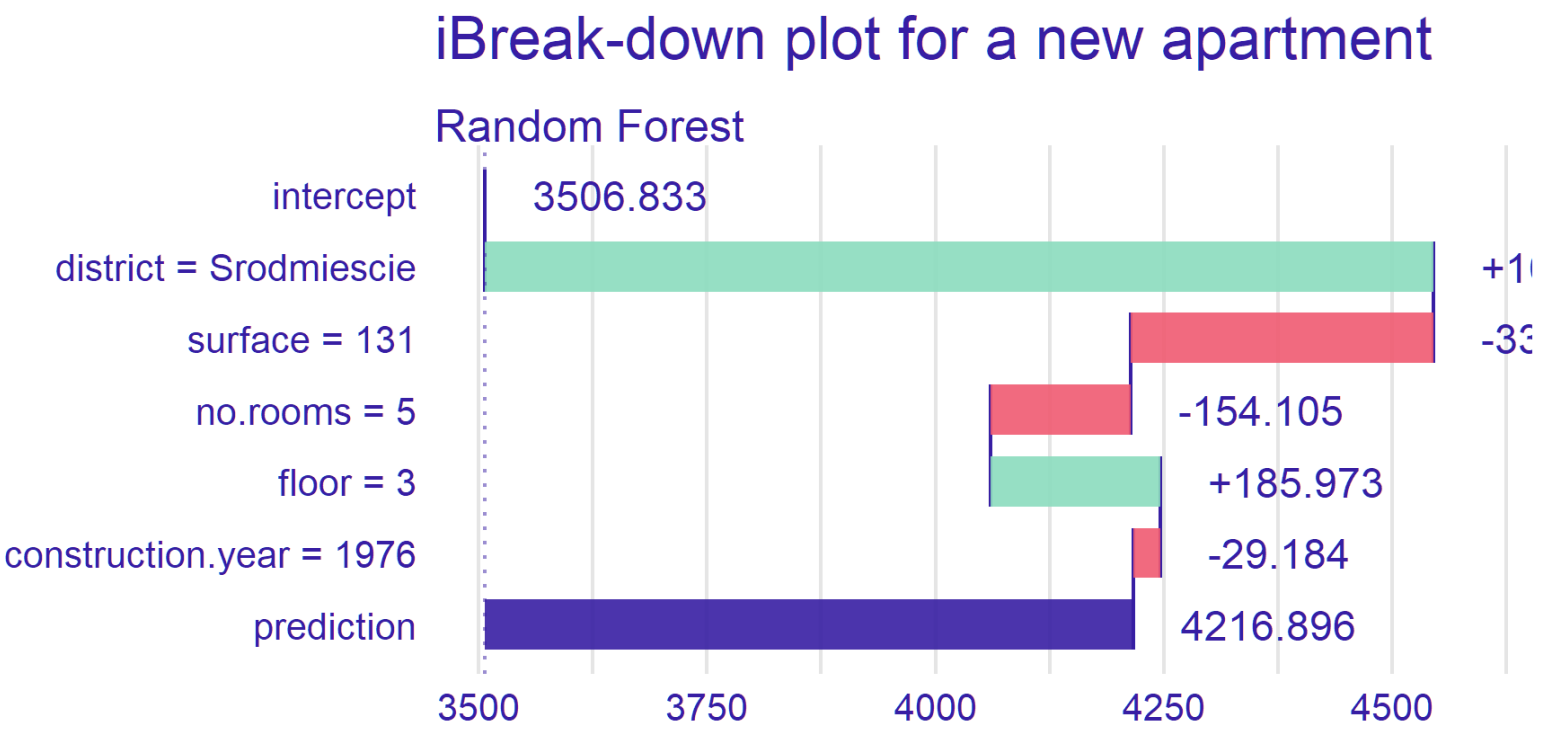
```
apartments_rf_exp_bdi # contributions to new_apt's price
```

```
##                                            contribution
## Random Forest: intercept                       3506.833
## Random Forest: district = Srodmiescie          1038.432
## Random Forest: surface = 131                   -331.052
## Random Forest: no.rooms = 5                    -154.105
## Random Forest: floor = 3                        185.973
## Random Forest: construction.year = 1976         -29.184
## Random Forest: prediction                      4216.896
```

```
plot(titanic_rf_exp_bdi) + ggtitle("iBreak-down plot for Johnny")
```



```
plot(apartments_rf_exp_bdi) + ggtitle("iBreak-down plot for a new apartment")
```



BD Plots for Interactions (iBD Plots)

Pros and Cons

iBD plots share many advantages and disadvantages of BD plots for models without interactions. However, in the case of models with interactions, iBD plots provide more correct explanations.

Cons:

- The numerical complexity of the iBD procedure is quadratic
- It may be time-consuming in case of models with a large number of explanatory variables. For a model with p explanatory variables, we have got to calculate $p * (p + 1)/2$ net contributions for single variables and pairs of variables.
- For datasets with a small number of observations, the calculations of the net contributions will be subject to a larger variability and, therefore, larger randomness in the ranking of the contributions.

BD Plots for Interactions (iBD Plots)

Your turn

Use iBD plots to interpret `titanic_svm` and `apartments_svm`

SHAP for Average Attributions

Intuition

SHAP (**SH**apley **A**dditive ex**P**lanations) is based on the idea of averaging the value of a variable's attribution over all (or a large number of) possible orderings.

The idea is closely linked to "Shapley values" developed originally for cooperative games by Lloyd Shapley, who won the Nobel Prize in Economics for it in 2012

Cooperative game theory in attributions

- Classic result in game theory on distributing gain in a coalition game
- Shapley values are a fair way to attribute the total gain to the players based on their contributions

The key insight here is that we can consider features as players in explaining a prediction; important features will have higher Shapley values

---

Reference: Scott Lundberg and Su-In Lee. "A unified approach to interpreting model predictions." NIPS 2017. https://arxiv.org/abs/1705.07874

SHAP for Average Attributions

Examples

```
titanic_rf_exp_shap <- predict_parts(
  explainer = titanic_rf_exp, new_observation = johnny,
  type = "shap", B = 25)

apartments_rf_exp_shap <- predict_parts(
  explainer = apartments_rf_exp, new_observation = new_apt,
  type = "shap", B = 25)
```

The `B = 25` argument indicates that we want to select 25 **random orderings** of explanatory variables for which the Shapley values are to be computed.

Summary statistics over the 25 random orderings of explanatory variables in the Titanic dataset:

```
titanic_rf_exp_shap
```

```
##                                                 min          q1       median
## Random Forest: age = 8                   0.09093883  0.24927219  0.270352968
## Random Forest: class = 1st              -0.14559583 -0.02871137  0.008598768
## Random Forest: embarked = Southampton -0.02435342 -0.01098686 -0.008320743
## Random Forest: fare = 72                -0.13416765 -0.09177889 -0.064896692
## Random Forest: gender = male            -0.16986226 -0.12697848 -0.107850023
## Random Forest: parch = 0                -0.03689986 -0.02232533 -0.013382420
## Random Forest: sibsp = 0                -0.02328863 -0.01229338  0.004863616
##                                                  mean           q3          max
## Random Forest: age = 8                   0.2537493792  0.279992750  0.381924785
## Random Forest: class = 1st               0.0209419846  0.063998414  0.189125510
## Random Forest: embarked = Southampton -0.0081104486 -0.004772542  0.003758043
## Random Forest: fare = 72                -0.0034429361  0.121010421  0.171706389
## Random Forest: gender = male            -0.1064120707 -0.067400091 -0.043777979
## Random Forest: parch = 0                -0.0155532759 -0.005705483 -0.001868600
## Random Forest: sibsp = 0                -0.0003081106  0.008254644  0.013338469
```

Summary statistics over the 25 random orderings of explanatory variables in the apartment dataset:

```
apartments_rf_exp_shap
```

```
##                                                 min        q1    median
## Random Forest: construction.year = 1976 -133.8106 -130.1802 -111.1709
## Random Forest: district = Srodmiescie   1019.5133 1038.4316 1056.5628
## Random Forest: floor = 3                 160.4427  175.7202  184.6053
## Random Forest: no.rooms = 5             -238.1715 -215.0412 -201.3793
## Random Forest: surface = 131            -337.8705 -317.9568 -282.5910
##                                                mean         q3        max
## Random Forest: construction.year = 1976  -97.26083  -67.60464  -29.18417
## Random Forest: district = Srodmiescie   1083.06209 1152.70756 1152.70756
## Random Forest: floor = 3                 183.23785  194.17732  197.41587
## Random Forest: no.rooms = 5             -180.19491 -124.24912 -103.85564
## Random Forest: surface = 131            -278.78115 -278.78369 -205.45335
```

SHAP for Average Attributions

Pros and Cons

Pros:

- Has a solid theoretical foundation in game theory
- Shapley values provide a uniform approach to decompose a model's predictions into contributions that can be attributed additively to different explanatory variables.

Cons:

- An important drawback of Shapley values is that they provide additive contributions (attributions) of explanatory variables. If the model is not additive, then the Shapley values may be misleading.
- For large models, the calculation of Shapley values is time-consuming. However, sub-sampling can be used to address the issue.

SHAP for Average Attributions

Your turn

Use SHAP to interpret `titanic_svm` and `apartments_svm`

LIME

Intuition

LIME = Local Interpretable Model-agnostic Explanations

BD/iBD plots and SHAP are most suitable for models with a small or moderate number of features because these methods need to consider **all** features

For datasets with a large number of features, sparse explanations with **a small number** of features offer a useful alternative

The most popular example of such sparse explainers is LIME

The key idea behind LIME is to locally approximate a black-box model by a simpler glass-box model, which is easier to interpret

- The typical choices of the glass-box models are regularized linear models like LASSO regression or decision trees

---

Reference: Marco Tulio Ribeiro, Sameer Singh, Carlos Guestrin. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. KDD 2016 (pp. 1135-1144). https://arxiv.org/abs/1602.04938

The idea behind the LIME approximation with a local glass-box model.

- The colored areas are decision regions for a complex binary classification model.
- The black cross indicates the instance (observation) of interest. *Why* should the instance be classified in the green class?

- Dots correspond to **artificial data** around the instance of interest.
  - Cannot use real data points because they are often too "far" away
  - The size of the dots corresponds to proximity to the instance of interest
  - Created by using perturbations of the instance of interest
  - For binary variables, the common choice is to switch ($0 \leftrightarrow 1$) the value of a randomly-selected number of variables.
  - For continuous variables, we may (1) add Gaussian noise or (2) discretize continuous variables and then perturb the discretized versions of the variables

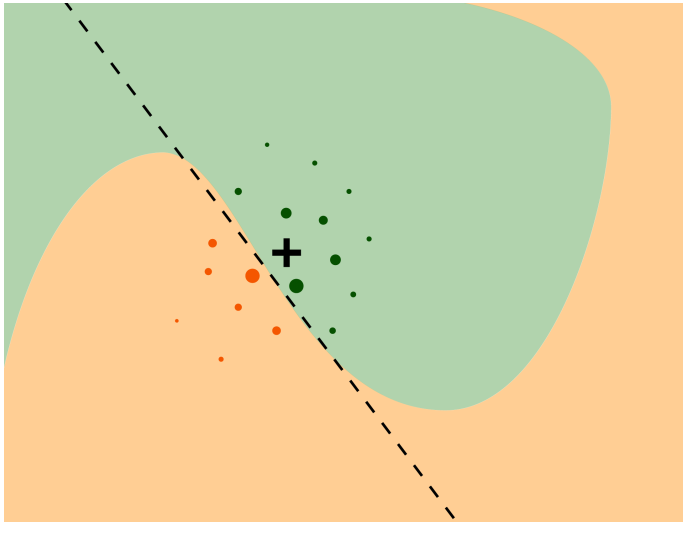


Image source: Explanatory Model Analysis

- The dashed line represents a simple linear model fitted to the artificial data. The simple model "explains" local behavior of the black-box model around the instance of interest.

LIME

Examples

```
library("DALEXtra") #Extension for 'DALEX' Package

model_type.dalex_explainer <- DALEXtra::model_type.dalex_explainer
predict_model.dalex_explainer <- DALEXtra::predict_model.dalex_explainer

# use predict_surrogate(), not predict_parts()
titanic_rf_exp_lime <- predict_surrogate(
  explainer = titanic_rf_exp, new_observation = johnny,
  n_features = 3, n_permutations = 1000, type = "lime")

apartments_rf_exp_lime <- predict_surrogate(
  explainer = apartments_rf_exp, new_observation = new_apt,
  n_features = 3, n_permutations = 1000, type = "lime")
```

```
(as.data.frame(titanic_rf_exp_lime))
```

```
##   model_type case  model_r2 model_intercept model_prediction feature
## 1 regression    1 0.6294761       0.5374413        0.4957616  gender
## 2 regression    1 0.6294761       0.5374413        0.4957616     age
## 3 regression    1 0.6294761       0.5374413        0.4957616   class
##   feature_value feature_weight  feature_desc                  data prediction
## 1             2     -0.3822677 gender = male 2, 8, 1, 4, 72, 0, 0      0.378
## 2             8      0.1859486     age <= 22 2, 8, 1, 4, 72, 0, 0      0.378
## 3             1      0.1546393   class = 1st 2, 8, 1, 4, 72, 0, 0      0.378
```

- `case`: provides indices of observations for which the explanations are calculated (that is, row number)
- `feature`: indicates which explanatory variables were given non-zero coefficients in the LASSO method.
- `feature_value`: provides information about the values of the original explanatory variables for the observations for which the explanations are calculated.
- `feature_desc`: indicates how the original explanatory variable was transformed. Note that the applied implementation of the LIME method dichotomizes continuous variables by using quartiles. Hence, `age <= 22`.
- `feature_weight`: provides the estimated coefficients for the variables selected by the LASSO method for the explanation.
- `model_intercept`: provides of the value of the intercept.

With the glass-box model shown on the previous slide, LIME approximates the random forest model around the instance of interest, Johnny, through the model intercept and feature weights:

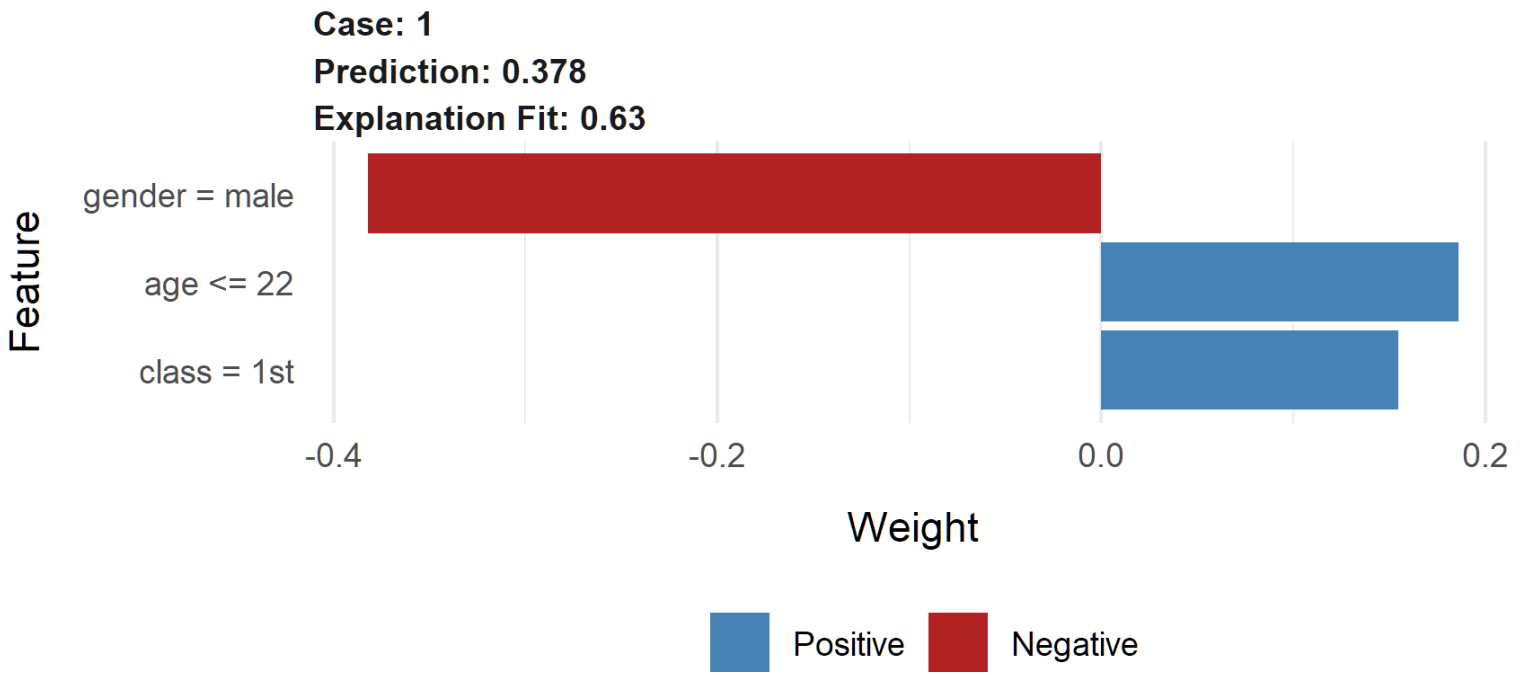
$$\hat{p}_{lime} = 0.54673 - 0.38885 + 0.17915 + 0.15510 = 0.49213$$

- 0.54673: model intercept
- -0.38885: gender = male
- 0.17915: age <= 22
- 0.15510: class = 1st
- 0.49213: overall model prediction

Note that the computed value corresponds to the number given in the column `model_prediction` in the printed output on the previous slide

By applying the `plot()` function to the object containing the explanation, we obtain a graphical presentation of the results.

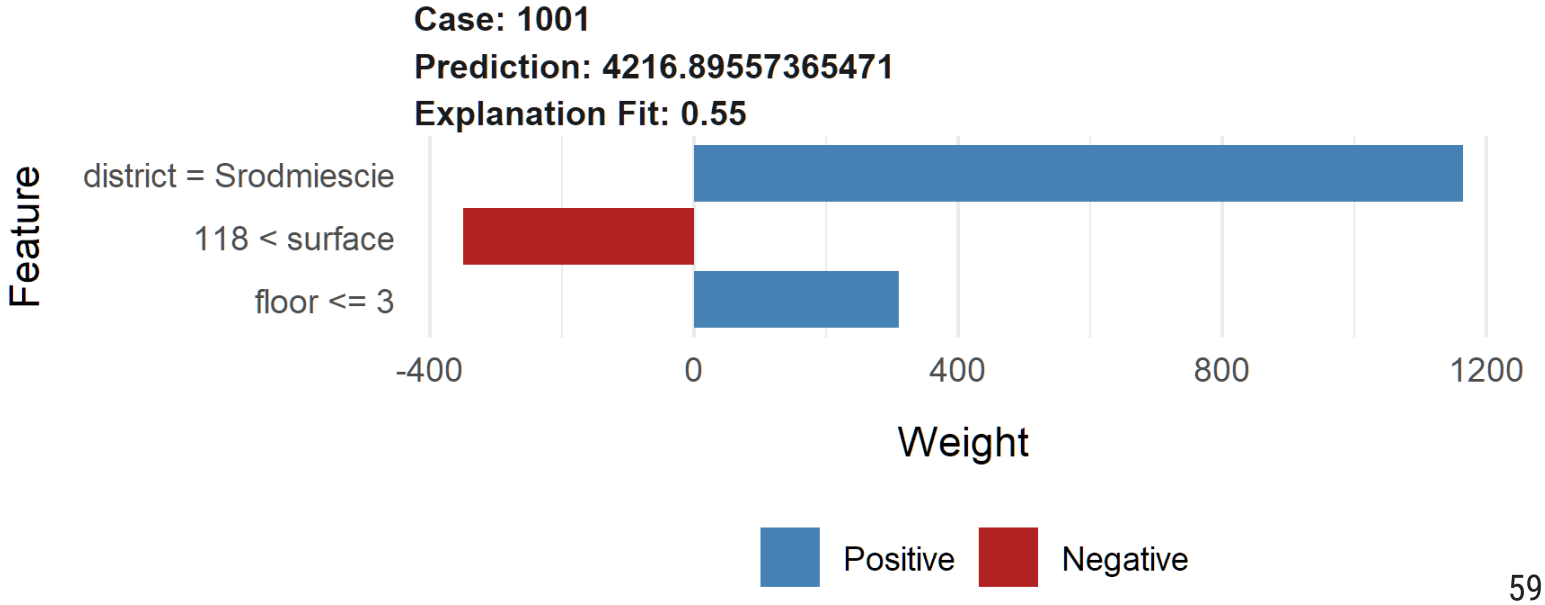
```
plot(titanic_rf_exp_lime)
```



```
(as.data.frame(apartments_rf_exp_lime))
```

```
##   model_type case  model_r2 model_intercept model_prediction  feature feature_value
## 1 regression 1001 0.5474026        3392.755         4517.614 district             6
## 2 regression 1001 0.5474026        3392.755         4517.614  surface           131
## 3 regression 1001 0.5474026        3392.755         4517.614    floor             3
##   feature_weight           feature_desc                 data prediction
## 1      1163.6115 district = Srodmiescie 1976, 131, 3, 5, 6   4216.896
## 2      -349.0794          118 < surface 1976, 131, 3, 5, 6   4216.896
## 3       310.3262             floor <= 3 1976, 131, 3, 5, 6   4216.896
```

```
plot(apartments_rf_exp_lime)
```



LIME

Pros and Cons

Pros:

- Can handle datasets with many features (and no need to consider all features)
- Works for text and image analysis
- Intuitive and easy to understand: a simpler model is used to approximate a more complex one

Cons:

- Because the glass-box model is selected to approximate the black-box model, and not the data themselves, the method does not control the quality of the local fit of the glass-box model to the data. Thus, the latter model may be misleading.
- In high-dimensional data, data points are sparse. Defining a "local neighborhood" of the instance of interest may not be straightforward. Sometimes even slight changes in the neighborhood strongly affect the obtained explanations.

LIME

Your turn

Use LIME to interpret `titanic_svm` and `apartments_svm`

Ceteris-paribus Profiles

Intuition

Ceteris-paribus (CP) profiles show how a model's prediction would change if the value of **a single exploratory variable** changed.

"Ceteris paribus" is Latin for "other things held constant" or "all else unchanged".

The method examines the influence of **an explanatory variable** by assuming that the values of all other variables do not change. The main goal is to understand how changes in the values of the variable affect the model's predictions.

Also known as "What-if" model analysis or Individual Conditional Expectations (ICE).

- What if Johnny was 20 years old? (other things held constant)
- What if Johnny traveled in the second class? (other things held constant)

Ceteris-paribus Profiles

Examples

```
# use predict_profile(), not predict_parts()
titanic_rf_exp_cp <- predict_profile(
    explainer = titanic_rf_exp, new_observation = johnny)

apartments_rf_exp_cp <- predict_profile(
    explainer = apartments_rf_exp, new_observation = new_apt)
```

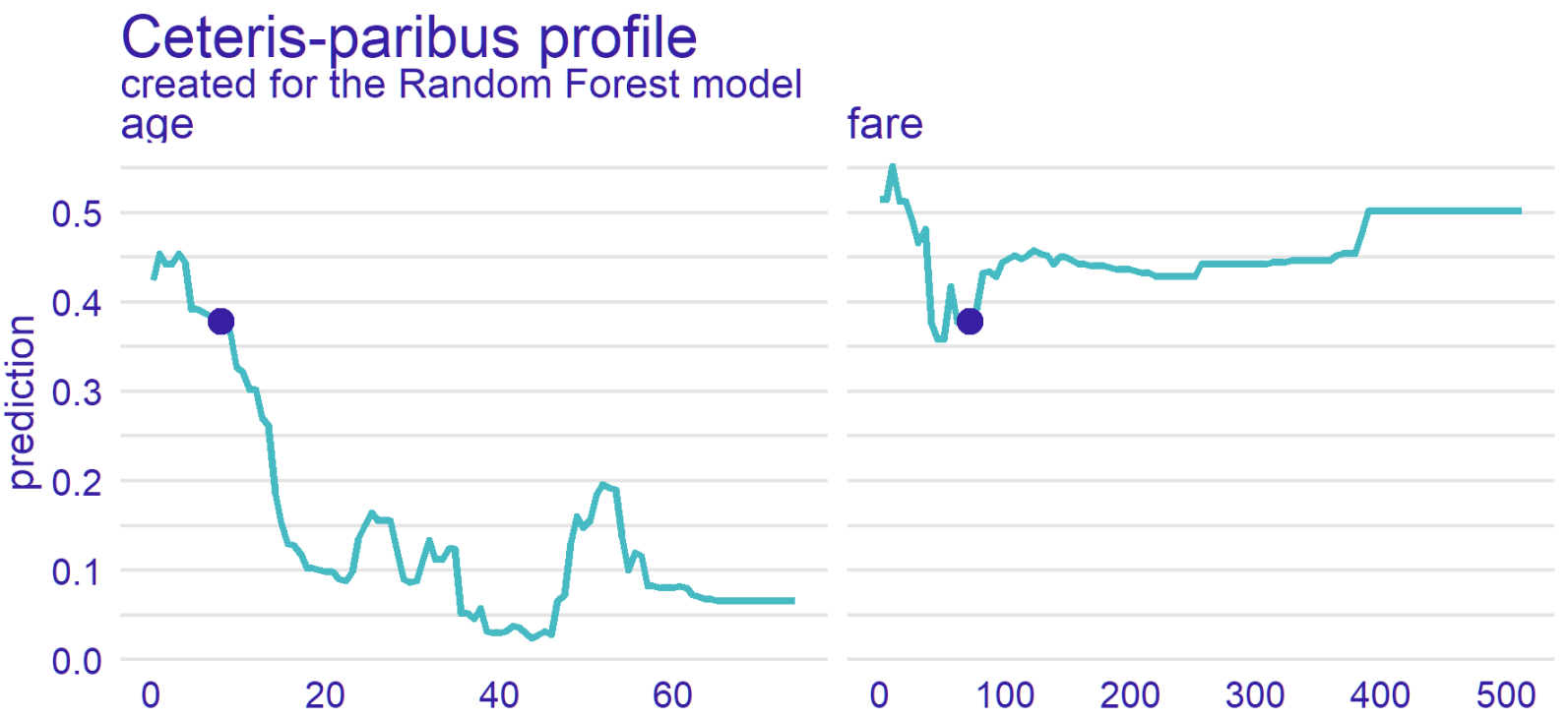
```
titanic_rf_exp_cp
```

```
## Top profiles    :
##                class gender age sibsp parch fare    embarked _yhat_ _vname_
## 1                1st   male   8     0     0   72 Southampton  0.378   class
## 1.1              2nd   male   8     0     0   72 Southampton  0.392   class
## 1.2              3rd   male   8     0     0   72 Southampton  0.430   class
## 1.3        deck crew   male   8     0     0   72 Southampton  0.468   class
## 1.4 engineering crew   male   8     0     0   72 Southampton  0.352   class
## 1.5 restaurant staff   male   8     0     0   72 Southampton  0.372   class
##     _ids_       _label_
## 1       1 Random Forest
## 1.1     1 Random Forest
## 1.2     1 Random Forest
## 1.3     1 Random Forest
## 1.4     1 Random Forest
## 1.5     1 Random Forest
##
##
## Top observations:
##   class gender age sibsp parch fare    embarked _yhat_       _label_ _ids_
## 1   1st   male   8     0     0   72 Southampton  0.378 Random Forest     1
```

```
apartments_rf_exp_cp
```

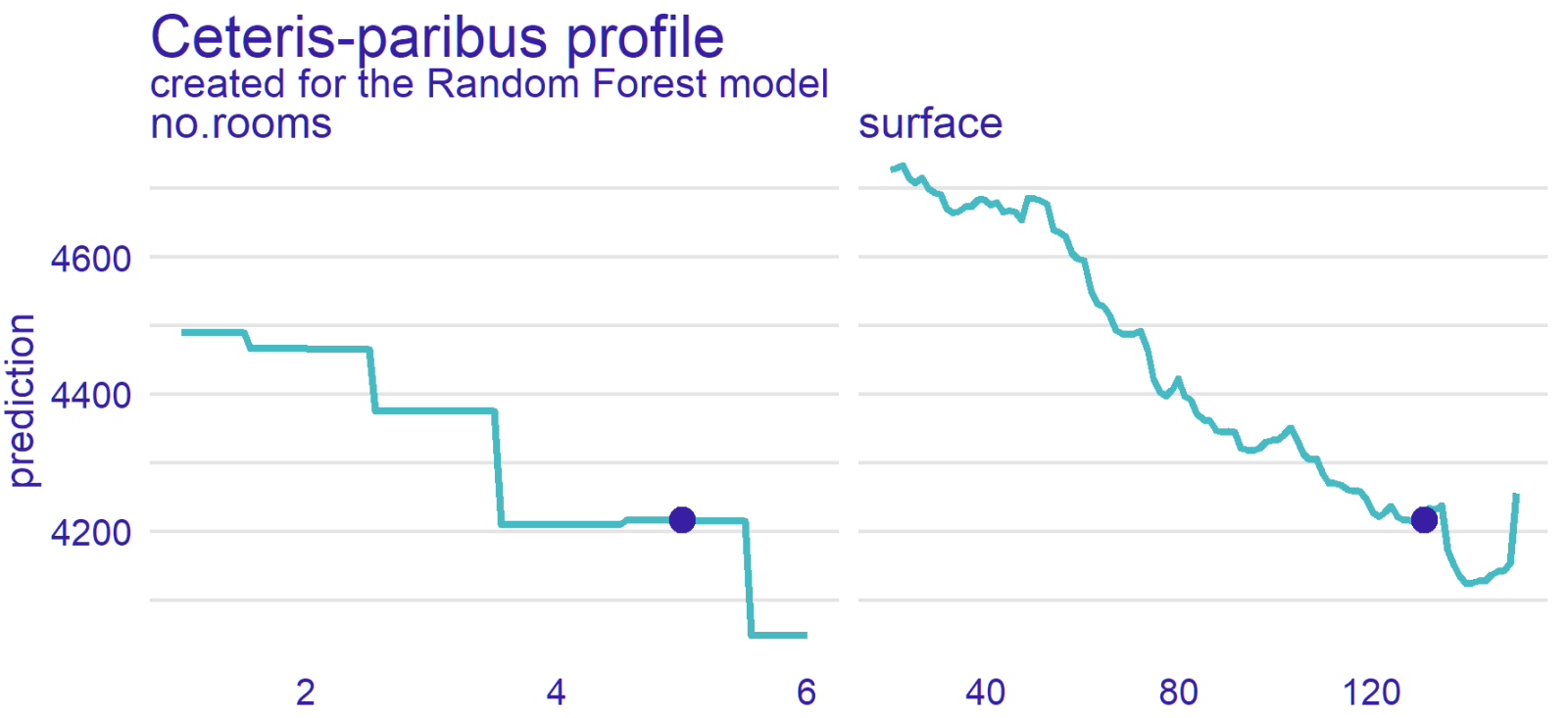
```
## Top profiles    :
##        construction.year surface floor no.rooms    district   _yhat_
## 1001              1920.0     131     3        5 Srodmiescie 4284.818
## 1001.1            1920.9     131     3        5 Srodmiescie 4324.882
## 1001.2            1921.8     131     3        5 Srodmiescie 4335.271
## 1001.3            1922.7     131     3        5 Srodmiescie 4348.070
## 1001.4            1923.6     131     3        5 Srodmiescie 4331.370
## 1001.5            1924.5     131     3        5 Srodmiescie 4327.520
##                  _vname_ _ids_       _label_
## 1001   construction.year  1001 Random Forest
## 1001.1 construction.year  1001 Random Forest
## 1001.2 construction.year  1001 Random Forest
## 1001.3 construction.year  1001 Random Forest
## 1001.4 construction.year  1001 Random Forest
## 1001.5 construction.year  1001 Random Forest
##
##
## Top observations:
##      construction.year surface floor no.rooms    district   _yhat_
## 1001              1976     131     3        5 Srodmiescie 4216.896
##            _label_ _ids_
## 1001 Random Forest     1
```

```
plot(titanic_rf_exp_cp, variables = c("age", "fare")) +
  ggtitle("Ceteris-paribus profile")
```



- The blue dots are the actual values from Johnny

```
plot(apartments_rf_exp_cp, variables = c("no.rooms", "surface")) +
  ggtitle("Ceteris-paribus profile")
```



- The blue dots are the actual values from the new apartment

Ceteris-paribus Profiles

Pros and cons

Pros:

- One-dimensional CP profiles offer a uniform, easy to communicate, and extendable approach to model exploration.
- Their graphical representation is easy to understand and explain
- CP profiles are easy to compare, as we can overlay profiles for two or more models to better understand differences between the models

Cons:

- In the presence of correlated explanatory variables, the application of the ceteris-paribus principle may lead to unrealistic settings and misleading results
- In case of a model with hundreds or thousands of variables, the number of plots to inspect may be daunting

Ceteris-paribus Profiles

Your turn

Use CP profiles to interpret `titanic_svm` and `apartments_svm`

Dataset-level Interpretation

Dataset-level explainers help us understand how do the model predictions perform overall, for an entire set of observations

Popular approaches:

- Variable-importance Measures
- Partial-dependence Profiles

Variable-importance Measures

Intuition

The main idea is to measure how much does a model's performance change if the effect of a selected explanatory variable, or of a group of variables, is removed?

To remove the effect, we use perturbations, like resampling from an empirical distribution or permutation of the values of the variable.

If a variable is important, then we expect that, after permuting the values of the variable, the model's performance will worsen. The larger the change in the performance, the more important is the variable.

Despite the simplicity of the idea, the permutation-based approach to measuring an explanatory-variable's importance is a very powerful model-agnostic tool for model exploration.

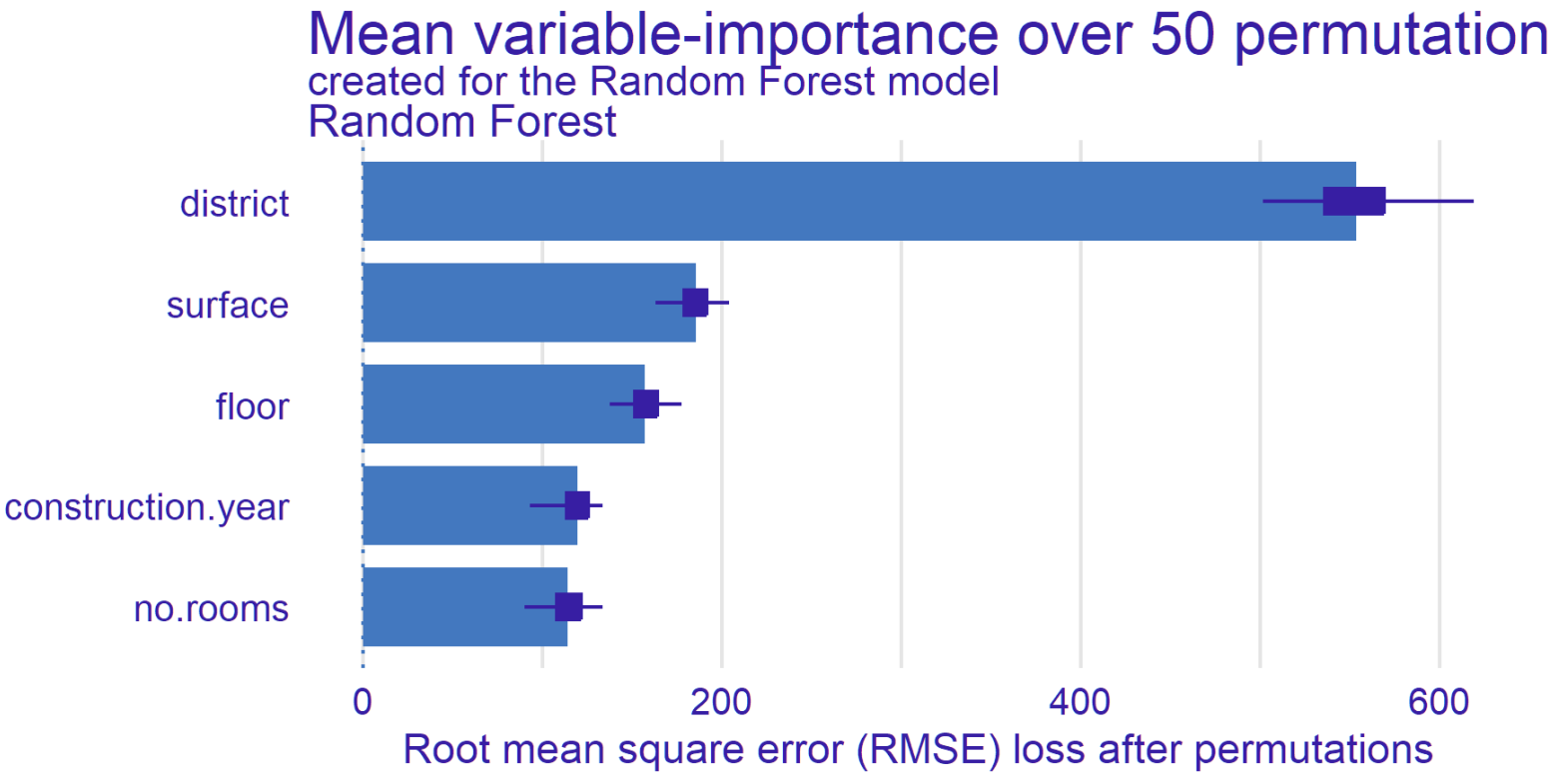
Variable-importance Measures

Examples

```
# use model_parts(), not predict_parts()
apartments_rf_exp_vip <- model_parts(
    explainer = apartments_rf_exp,
    loss_function = loss_root_mean_square,
    B = 50, type = "difference")
```

- `loss_function = loss_root_mean_square` specifies how to determine variable importance
- `B = 50` specifies the number of permutations to be used for the purpose of calculation of the (mean) variable-importance measures

```
plot(apartments_rf_exp_vip) +
  ggtitle("Mean variable-importance over 50 permutations")
```

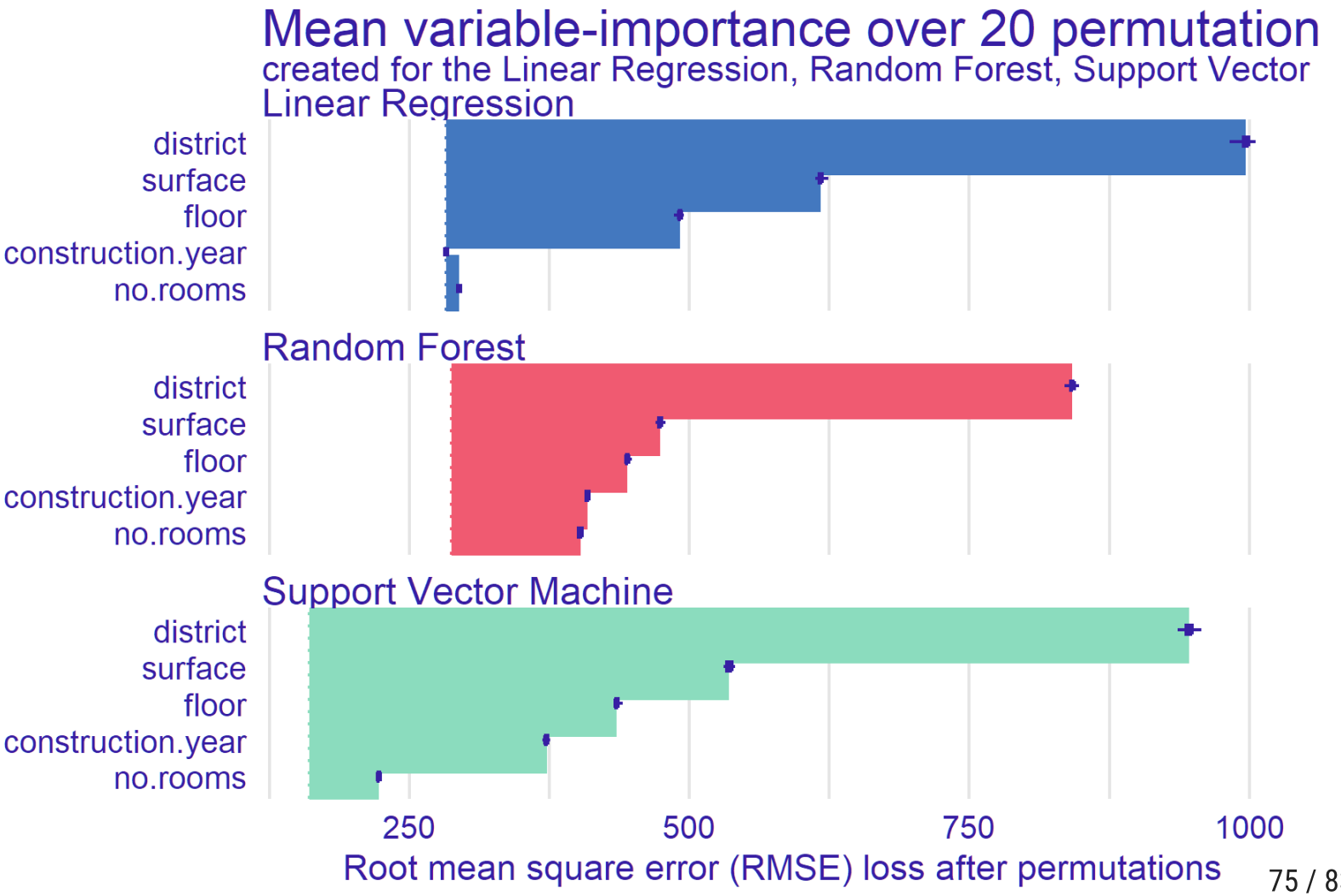


We can also compare variable-importance across models:

```
# this part will take a few minutes
vip_lm  <- model_parts(explainer = apartments_lm_exp,  B = 20, N = NULL)
vip_rf  <- model_parts(explainer = apartments_rf_exp,  B = 20, N = NULL)
vip_svm <- model_parts(explainer = apartments_svm_exp, B = 20, N = NULL)
```

- `B`, the number of permutations to be used for the purpose of calculation of the (mean) variable-importance measures
- `N`, the number of observations that are to be sampled from the data available in the explainer-object for calculating the variable-importance measure; by default, `N = 1000` is used; if `N = NULL`, the entire dataset is used

```
plot(vip_lm, vip_rf, vip_svm) +
  ggtitle("Mean variable-importance over 20 permutations")
```



Variable-importance Measures

Pros and cons

Pros:

- The plots of variable-importance measures are easy to understand
- The measures can be compared between models and may lead to interesting insights. For example, if variables are correlated, then models like random forest are expected to spread importance across many variables, while in regularized-regression models the effect of one variable may dominate the effect of other correlated variables.

Cons:

- The main disadvantage of the permutation-based variable-importance measure is its dependence on the random nature of the permutations. As a result, for different permutations, we will, in general, get different results.

Variable-importance Measures

Your turn

Use variable-importance measures to assess our models for the titanic data.

Partial-dependence Profiles

Intuition

The idea of PD profiles is to show how does the expected value of model prediction behave as a function of a selected explanatory variable

To this end, the average of a set of individual ceteris-paribus (CP) profiles can be used

Recall that a CP profile shows the dependence of an instance-level prediction on an explanatory variable.

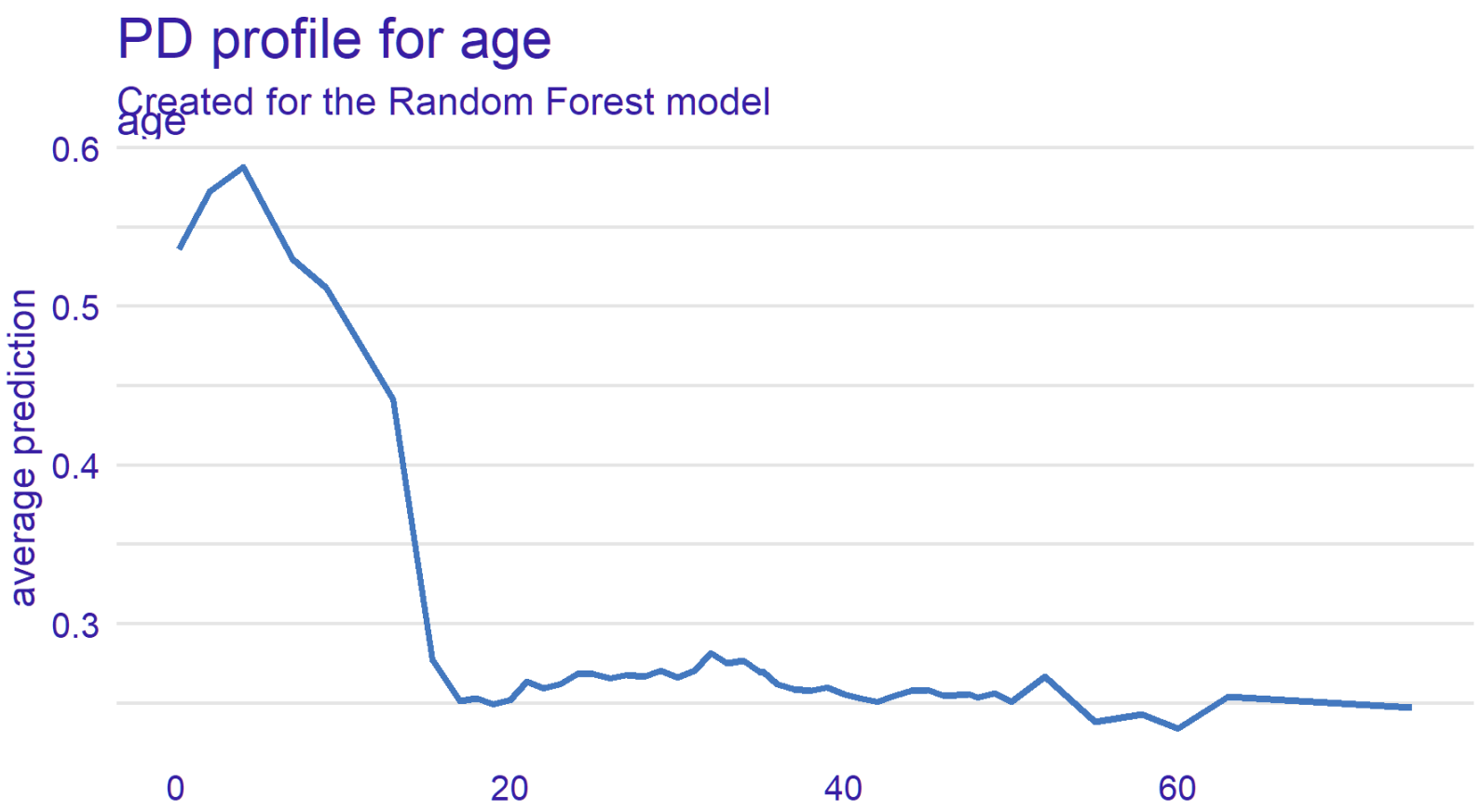
A PD profile is estimated by the mean of the CP profiles for all instances (observations) from a dataset.

Partial-dependence Profiles

Examples

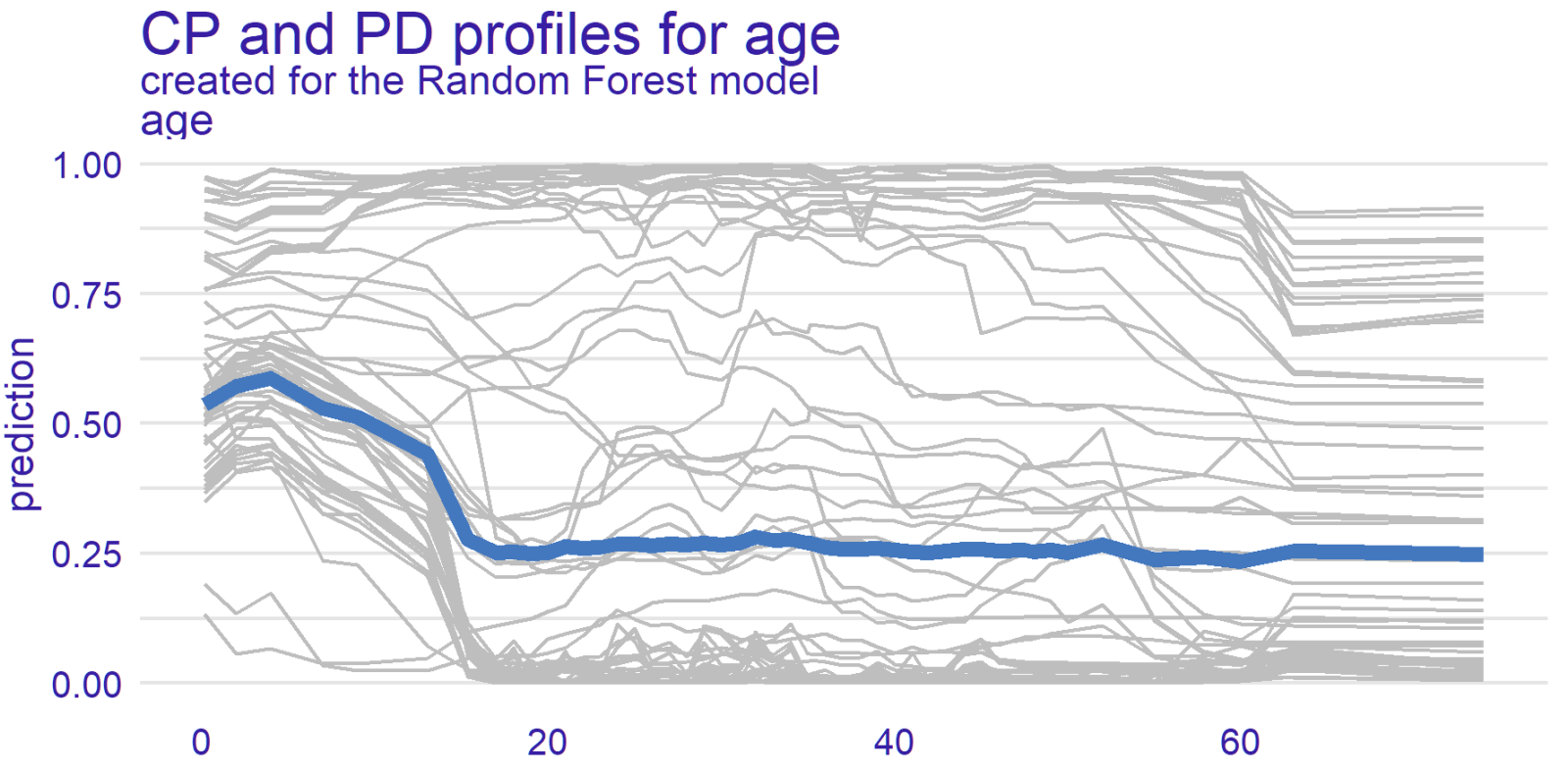
```
titanic_rf_exp_pdp <- model_profile(
  explainer = titanic_rf_exp,
  variables = "age")
```

```
plot(titanic_rf_exp_pdp) + ggtitle("PD profile for age")
```



By specifying the argument `geom = "profiles"` in the `plot()` function, we add the CP profiles to the plot of the PD profile. This is useful if we want to check how well the PD profile captures the CP profiles.

```
plot(titanic_rf_exp_pdp, geom="profiles")+ggtitle("CP & PD profiles for age")
```



Partial-dependence Profiles

Pros and cons

Pros:

- PD profiles offer a simple way to summarize the effect of a particular explanatory variable on the dependent variable
- easy to explain and intuitive
- can be obtained for sub-groups of observations and compared across different models

Cons:

- Given that the PD profiles are averages of CP profiles, they inherit the limitations of the latter. In particular, as CP profiles are problematic for correlated explanatory variables, PD profiles are also not suitable for that case

Partial-dependence Profiles

Your turn

Use PD plots to interpret `titanic_svm` and `apartments_svm`