DALEX: Descriptive mAchine Learning EXplanations. Tools for exploration, validation and explanation of complex machine learning models

Mateusz Staniak

University of Wrocław

Budapest, 12.05.2017

Outline

Why explain a single prediction? - bird's-eye view

- when important decision are made based on ML model, it needs to be **trustworthy**
- trust comes from **understanding**
- the demand for interpretable algorithms is growing (see: *Weapons of math destruction*, Facebook feed controversies etc.)

Why explain a single prediction? - worm-eye view

- this demand is transfered into legal regulations (see: GDPR) $\rightarrow$ more and more institutions are obliged to explain model predictions (debt collection, loans ...)
- understanding models helps improve them

Which predictions need explanation?

1. Every prediction the client (or the boss) wants to understand
2. Predictions that seem suspicious
    - How to spot them?

   $\rightarrow$ **model performance**

   $\rightarrow$ **model diagnostics**

    - How to explain them?
3. Representative predictions (see: LIME)

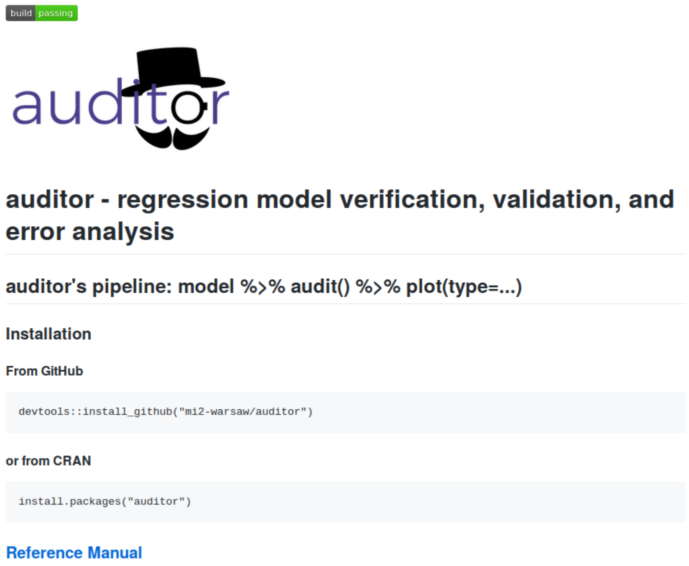
Model diagnostics: example

```
library(tidyverse)
library(randomForest)
load(url("https://github.com/pbiecek/DALEX_docs/
raw/master/workshops/eRum2018/houses.rda"))
hrf <- randomForest(sqm_price ~., data = houses)
head(houses)
```

rooms_num	area	sqm_price	year	floor	max_floor	district
3	89.00	5270	2007	2	2	Krzyki
4	163.00	6687	2002	2	2	Psie Pole
3	52.00	6731	1930	1	2	Srodmiescie
4	95.03	5525	2016	1	2	Krzyki
4	88.00	5216	1930	3	4	Srodmiescie
2	50.00	5600	1915	3	4	Krzyki

Thanks to Piotr Sobczyk for this dataset (http://szychtawdanych.pl)

Auditor



auditor - regression model verification, validation, and error analysis

auditor's pipeline: model %>% audit() %>% plot(type=...)

Installation

From GitHub

```
devtools::install_github("mi2-warsaw/auditor")
```

or from CRAN

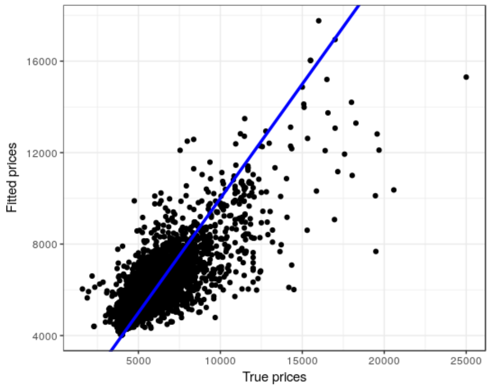
```
install.packages("auditor")
```

Reference Manual

Model diagnostics: example

```
library(DALEX)
library(auditor)
rf_explainer <- DALEX::explain(hrf, data = houses, y = houses$sqm_pri
rf_audit <- audit(rf_explainer)
plotPrediction(rf_audit)
```

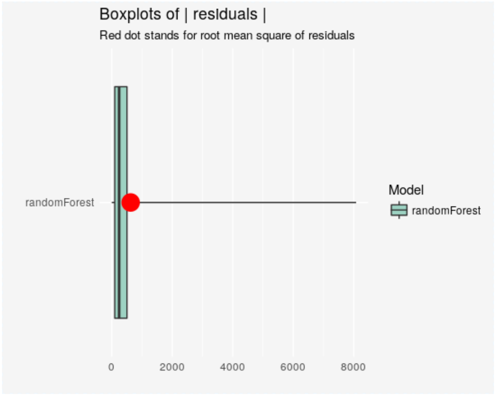
Model diagnostics: fitted vs observed



- Points on the plot should be close to the `y = x` line,
- Questions:
  - is there a pattern? (For example: does the devation from true value grow as true value grows?)
  - are there any points especially far from the line (meaning: points with large residuals)?
- More diagnostic tools: `auditor` package

Model performance

```
rf_perf <- model_performance(rf_explainer)
plot(rf_perf, geom = "boxplot")
```



- shortly summarizes the distribution of the absolute value of residuals
- red dot is the root mean square error
- we can put boxplots for several models on the same plot (simply by passing all of them as arguments to `plot`) to compare models
- boxplots help discover outliers

Single prediction explanations

Once we identified predictions we want to explain, we need tools that will help us!

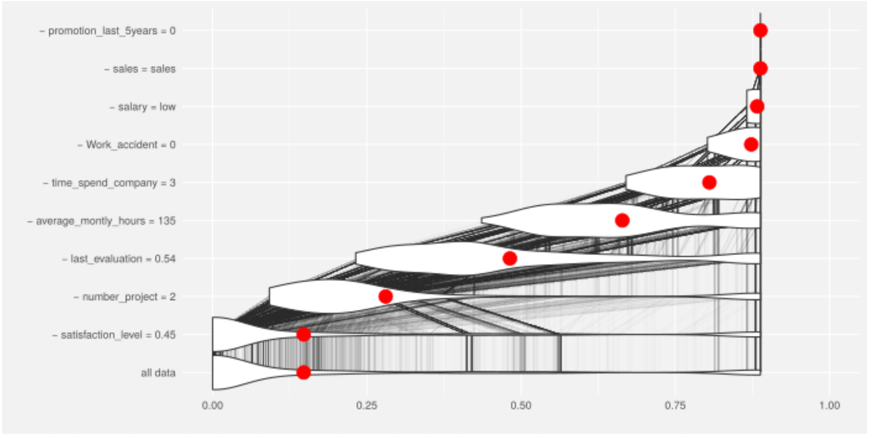
- These tools were a subject of study since at least 2008 (*Explaining Classifications For Individual Instances* by Robnik-Šikonja and Kononenko)
- LIME method introduced in 2016 by Tulio Ribeiro, Singh and Guestrin gained larger recognition (paper *"Why Should I Trust You?" Explaining the Predictions of Any Classifier*)
- Two main ideas:
    1. Attribute scores to explanatory variables according to their influence on the prediction (**contributions**)
    2. Fit a model locally around an observation and investigate it

Single predictions explanations

Methods:

1. Contributions
   - `breakDown` package
   - Shapley values
2. Local approximations
   - LIVE (Local Interpretable Visual Explanations)
   - LIME (Locallly Interpretable Model-agnostic Explanations)

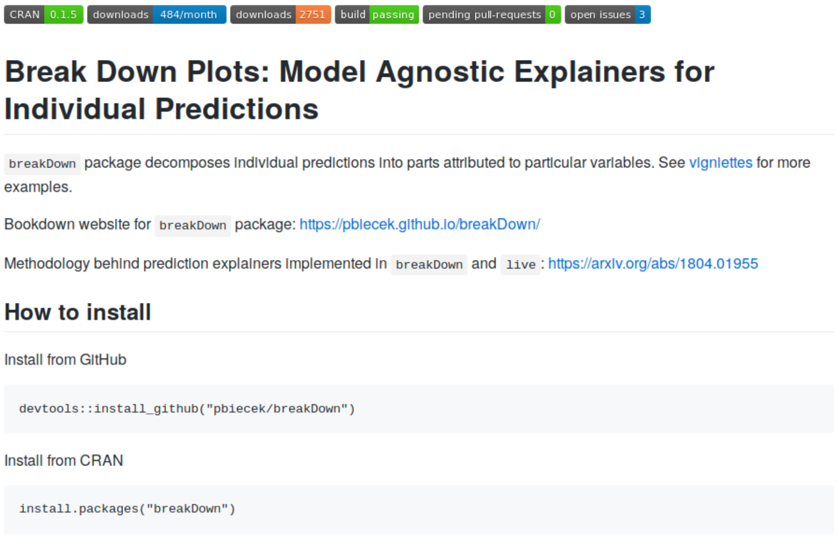
Break Down: general idea



- Approach: finding **additive** feature contributions
- Contributions are assigned in a greedy manner
- Waterfall plots as a visual tool

→ intuitive interpretation

breakDown



Break Down Plots: Model Agnostic Explainers for Individual Predictions

`breakDown` package decomposes individual predictions into parts attributed to particular variables. See vignettes for more examples.

Bookdown website for `breakDown` package: https://pbiecek.github.io/breakDown/

Methodology behind prediction explainers implemented in `breakDown` and `live` : https://arxiv.org/abs/1804.01955

How to install

Install from GitHub

```
devtools::install_github("pbiecek/breakDown")
```

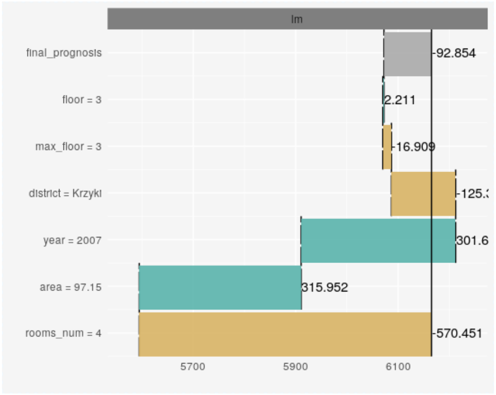
Install from CRAN

```
install.packages("breakDown")
```

Break Down for linear models

```r
linear_model <- lm(sqm_price ~., data = houses)
lm_explainer <- DALEX::explain(linear_model,
                               data = houses,
                               y = houses$sqm_price)
breakdown_linear <- single_prediction(lm_explainer,
                                      houses[4036, -3])
plot(breakdown_linear)
```

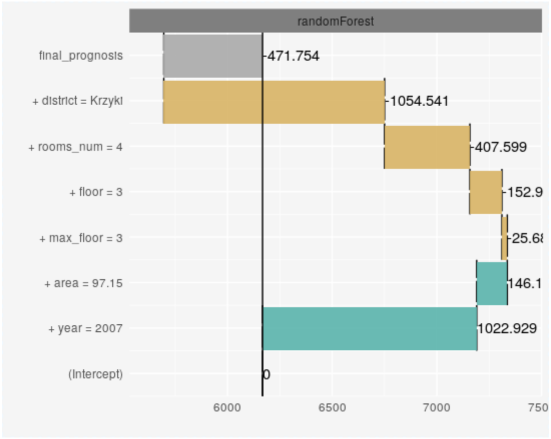
Break Down for linear models



- Contributions are scaled, so they do not depend on the units
- We can see actual contributions, not just the weights (as in LIME plots)

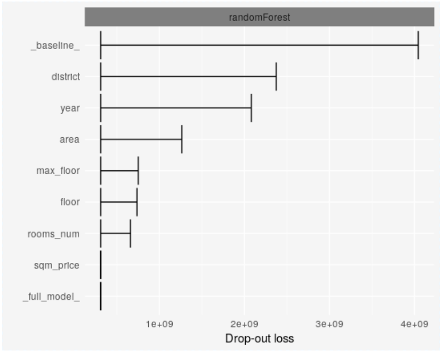
Model-agnostic Break Down

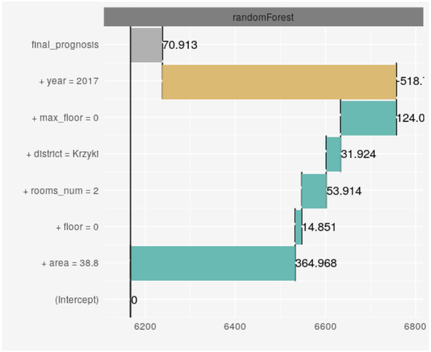
```
breakdown_explanation <- single_prediction(rf_explainer, houses[4036,
plot(breakdown_explanation)
```



- Notice the importance of *District* and Year

But isn't it enough to calculate feature importance?





```
global_feat_imp <- DALEX::variable_importance(rf_explainer)
plot(global_feat_imp)
```

No. Particular instances can be influenced the most by different features, not necessarily the ones that are most important globally.

Shapley values: general idea

- The goal is a decomposition of prediction into a **sum** of scores related to (simplified) features.
- The problem is solved using game theory: *Shapley values*.
- Variables are *players* who contribute to the outcome - the prediction - and we try to *pay* them accordingly to their contributions.
- This approach unifies several methods (including `LIME`).

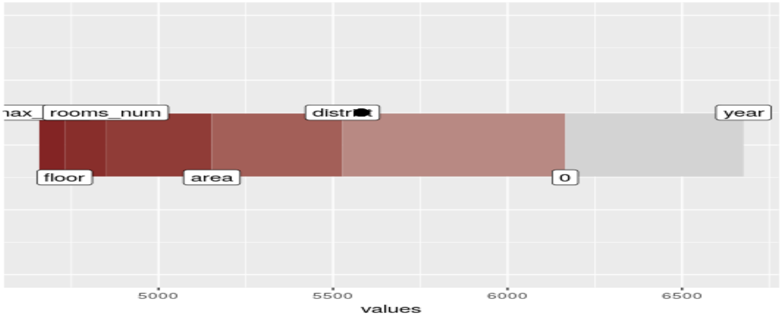
Shapley values: some details

- Exact methods exist for linear models and tree ensemble methods. In other cases, approximations are needed.
- The classic way: sample permutations of variables, then average contributions.
- The better way: approximation based on LIME and Shapley values for regression.
- This method has good theoretical properties, but will not produce sparse explanations

Shapley values: example

```r
library(shapleyr)
library(mlr)
house_task <- makeRegrTask(data = houses, target = "sqm_price")
house_rf_mlr <- train("regr.randomForest", house_task)

shapley_explanation <- shapley(4036,
                               task = house_task,
                               model = house_rf_mlr)
class(shapley_explanation) <- c("shapley.singleValue", "list")
plot(shapley_explanation)
```

Shapley values: example



- **0** is the mean of all predictions
- the **black dot** is the prediction we are explaining
- values and the plot describe how we move from the global mean of predictions to this particular predictions
- most important features are the ones that help move the most
- many different *paths* from the mean to the predictions are considered and averaged

Shapley values: example

```
gather(shapley_explanation$values, "feature", "shapley.score")
```

	feature	shapley.score
1	_Id	4036
2	_Class	
3	rooms_num	-32.25
4	area	-597.239
5	year	461.175
6	floor	-137.767
7	max_floor	-32.202
8	district	-122.961

Time to practice!

Remember, you can find some help in the `Workshop_eRum_2018_part2.R` file

Exercises

1. Run the following code to fit random forest, linear regression and SVM to the housing prices data.

```r
library(tidyverse)
library(live)
library(DALEX)
library(randomForest)
library(e1071)
library(auditor)
load("./rda_files/houses.rda")

set.seed(33)
house_rf <- randomForest(sqm_price ~., data = houses)
house_svm <- svm(sqm_price ~., data = houses)
house_lm <- lm(sqm_price ~., data = houses)
```

Create DALEX explainer object for each of the models. Create and compare boxplots of residuals for all the models (`model_performance`).

Exercises

- Which model is the best?
- Are there any outlying predictions?
- Find the observation with the largest absolute value of residual among houses cheaper than 7000 PLN.

TIP: object returned by `model_performance` function is a data frame with colnames *predicted*, *observed*, *diff* and *label*.

Exercises

2. Create single prediction explainers for the instance chosen in Exercise 1. Create Break Down plots for each of the observations. What are the key factors that drive the prediction? Are they the same for every model?
3. Run the following code to train model random forest model using *mlr* interface (this is necessary for ‘shapleyR‘ package).

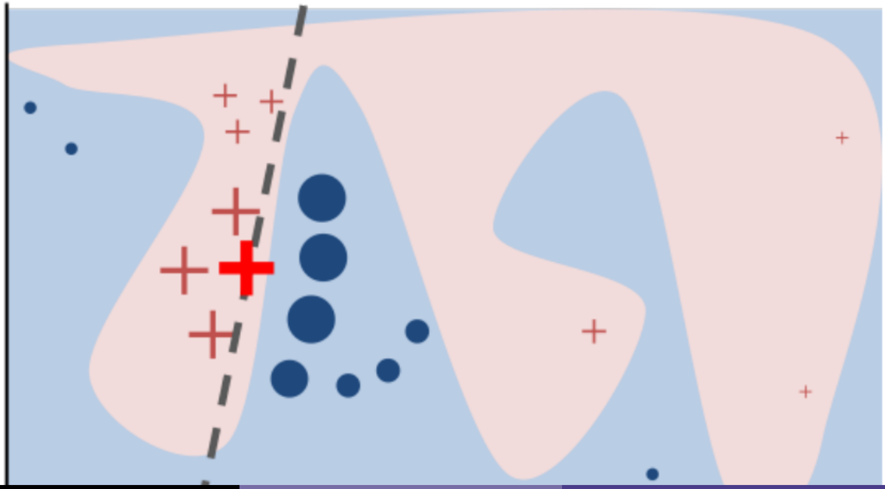
```r
library(mlr)
load("./rda_files/houses.rda")
n_obs <- 1189

house_task <- makeRegrTask(data = houses, target = "sqm_price")
house_rf_mlr <- train("regr.randomForest", house_task)
```

Use shapleyR package to calculate Shapley values for prediction chosen in Exercise 1 (its index is in `n_obs` object). Are the results consistent with Break Down results from Exercise 2? Draw a plot of Shapley values.
TIP: remember to set class of the object returned by `shapley` function to `shapley.singleValue` before using `plot`.

Bonus exercises

- Draw plots of fitted vs observed values for each of these models. Can you spot any problems with the predictions? Are the prices usually underestimated of overestimated?
- Create variable importance explainer. Compare global variable importance to scores obtained in Exercise 2 and Exercise 3.

LIME: General idea



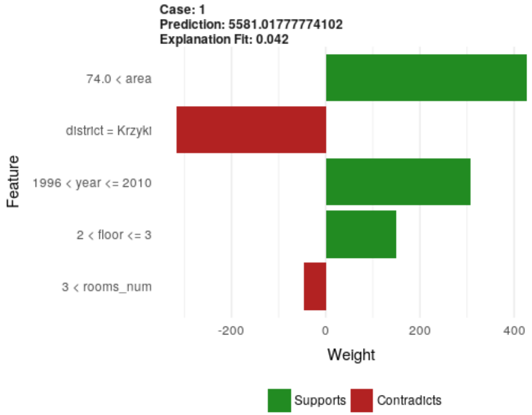
LIME: some details

- Observations are transformed to *interpretable input space*
- Gaussian sampling for tabular, uniform sampling from interpretable inputs for image/text.
- Scores for new observation are weighted by the distance from original observation.
- Variable selection is usually based on ridge/lasso regression.
- Weights are assigned to interpretable inputs to decide if they *vote* for or against a given label.
- Note: method depends on many hyperparameters

LIME: example

```
library(lime)

explained_prediction <- houses[4036, ]

lime_explainer <- lime(houses,
                       model = house_rf_mlr)
lime_explanation <- lime::explain(houses[4036, ],
                                  explainer = lime_explainer,
                                  n_features = 5)
plot_features(lime_explanation)
```

LIME: example



- weights from ridge regression are on the plot (NOT weights multiplied by actual feature values)
- positive weights are *votes for*, negative weights are *against*

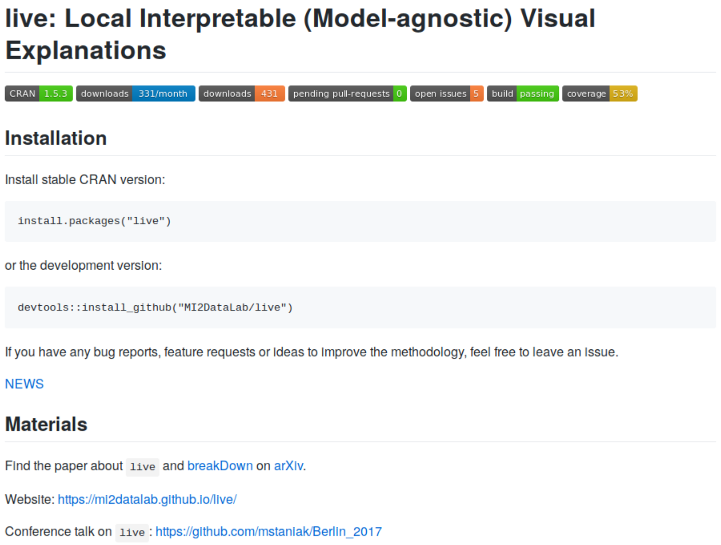
LIVE: general idea

- Modification of `LIME` for tabular data and regression problems with emphasis on model visualization.
- Similar observations for *fake* dataset are sampled from empirical distributions.
- Variable selection is possible (LASSO, then explanation model is fitted to selected features).

LIVE: some details

- Two methods of creating the new dataset are available: by permuting each variable and by changing one feature per observations (method does matter)
- We can control which variables are allowed to vary through `fixed_variables` variable argument to `sample_locally` (keeping date/factor/correlated variables unchanged)
- Features can be standardized before fitting explanation model
- Black box model can be pre-trained or it can be trained using `mlr` interface
- Hyperparameters of both black box and explanation models can be set

live



live: Local Interpretable (Model-agnostic) Visual Explanations

CRAN 1.5.3 | downloads 331/month | downloads 431 | pending pull-requests 0 | open issues 5 | build passing | coverage 53%

Installation

Install stable CRAN version:

```
install.packages("live")
```

or the development version:

```
devtools::install_github("MI2DataLab/live")
```

If you have any bug reports, feature requests or ideas to improve the methodology, feel free to leave an issue.

NEWS

Materials

Find the paper about `live` and breakDown on arXiv.

Website: https://mi2datalab.github.io/live/

Conference talk on `live`: https://github.com/mstaniak/Berlin_2017

LIVE: example

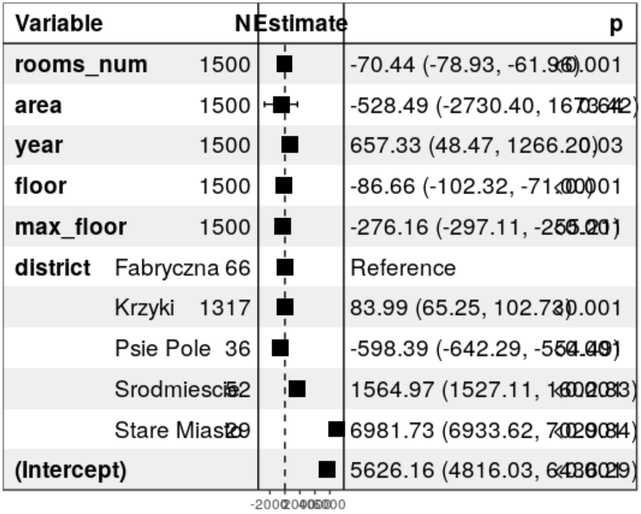
```r
library(live)
library(mlr)

new_dataset <- sample_locally2(data = houses,
                               explained_instance = houses[4036, ],
                               explained_var = "sqm_price",
                               size = 1000)
with_predictions <- add_predictions2(new_dataset, hrf)
live_explanation <- fit_explanation2(with_predictions,
                                     "regr.lm",
                                     standardize = TRUE)
```

LIVE: example

```
live_explanation

Dataset:
  Observations:  500
  Variables:  7
  Response variable:  sqm_price
Explanation model:
  Name: regr.lm
  Variable selection wasnt performed
  Weights present in the explanation model
  R-squared: 0.8923
```

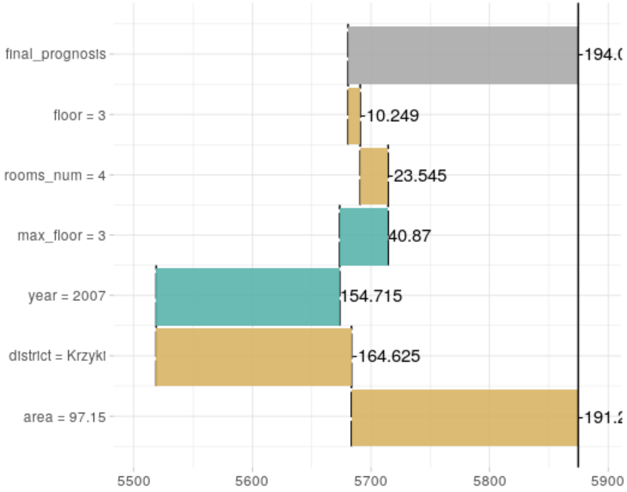
- Default method of sampling is `live`
- Default explanation model is linear regression and distance is measured (weights are assigned) by gaussian kernel

Plot local model structure: forest plot



Variable		N	Estimate	p
rooms_num		1500		-70.44 (-78.93, -61.96)0.001
area		1500		-528.49 (-2730.40, 1673.642)
year		1500		657.33 (48.47, 1266.20)03
floor		1500		-86.66 (-102.32, -71<0.0001
max_floor		1500		-276.16 (-297.11, -255.201)
district	Fabryczna	66		Reference
	Krzyki	1317		83.99 (65.25, 102.73)0.001
	Psie Pole	36		-598.39 (-642.29, -554.49)
	Srodmiescie	52		1564.97 (1527.11, 1602.83)
	Stare Miasto	29		6981.73 (6933.62, 7029.84)
(Intercept)				5626.16 (4816.03, 6436.29)

Plot local variable contributions: waterfall plot



Time to practice!

Remember, you can find some help in the `Workshop_eRum_2018_part2.R` file

Exercises

4. Simulate new data around the observation from Exercise 1 (row 1189) and the add random forest predictions (from `house_rf` object). Then fit a linear model locally.
TIP: remember to load `mlr` package.
TIP2: don't use too small `size` for the simulated dataset (try at least 1000).

5. Visualize approximation created in Exercise 4. Use `plot_explanation2` function to create a forest plot of the linear model and then the Break Down plot.

6. Use `lime` package to approximate random forest model around prediction chosen in Exercise 1. Follow the `lime` work flow: create an explainer, approximate the model around the explained instance and then use `plot_features` function to see, how features influence this prediction.
TIP: use `house_rf_mlr` object from Exercise 3, because `lime` works well with `mlr` objects.

Bonus exercises

- Use add_predictions function to add SVM and LM predictions to the simulated dataset. Compare plots for all three models.
- Run the following code to see largest residuals for *Psie Pole* district.

```
library(tidyverse)
houses %>%
mutate(id = 1:n()) %>%
mutate(rf_pred = predict(house_rf)) %>%
mutate(abs_res = abs(sqm_price - rf_pred)) %>%
arrange(desc(abs_res)) %>%
filter(sqm_price < 7000,
district == "Psie␣Pole") %>%
head(5)

n_obs2 <- 5830
```

Using live package, fit a linear model around the top observation. Compare waterfall plots for this prediction and the prediction from Exercise 5. How are they different?

Acknowledgement

Uniwersytet Wrocławski

Thank you for attending our workshop!

What did we learn?

1. Ability to provide an explanation for a single prediction is becoming more and more important
2. Main ideas are
   - decomposing prediction into a sum of scores corresponding to predictors
   - approximating the black box model locally by a simpler function
3. Different tools are available for this tasks:
   - breakDown
   - Shapley values
   - live
   - LIME