Explanations of model predictions with `live` and `breakDown` packages

by Mateusz Staniak and Przemysław Biecek

Abstract Complex models are commonly used in predictive modeling. In this paper we present R packages that can be used to explain predictions from complex black box models and attribute parts of these predictions to input features. We introduce two new approaches and corresponding packages for such attribution, namely **live** and **breakDown**. We also compare their results with existing implementations of state of the art solutions, namely **lime** that implements Locally Interpretable Model-agnostic Explanations and **ShapleyR** that implements Shapley values.

Introduction

Predictive modelling is a very exciting field with many different applications. Lots of algorithms have been developed in this area. According to many Kaggle competitions (Fogg, 2016), winning solutions are often obtained with elastic tools like random forest, gradient boosting or neural networks.

These algorithms have many strengths but also share a major weakness, which is the lack of interpretability of a model structure. A single random forest, an **xgboost** model or a neural network may be parametrized with thousands of parameters which makes these models hard to understand. Lack of interpretability results in the lack of trust in model predictions. Lack of trust is a major obstacle when one thinks about applications in regulated areas such as personalized medicine or similar. An interesting example of a situation in which trust issues are fully justified is given in Tulio Ribeiro et al. (2016b). Authors compare two classifier that were trained to recognize if a text describes Christianity or Atheism. After explanations were provided, it turned out that the model with superior performance on the test set often based its prediction on irrelevant words, for example prepositions. To overcome this problem, the interpretability of complex machine learning models has been a matter of interest of much research, also regarding model visualization. Find some examples in Štrumbelj and Kononenko (2011); Tzeng and Ma (2005); Zeiler and Fergus (2014).

The general approach to the interpretability is to identify important variables (features) in the model and then learn the expected model response given a single variable. Description of general framework of permutation based variable importance rankings may be found in Altmann et al. (2010). An interesting and widely adopted tool for estimation of marginal model response are Partial Dependency Plots (see Friedman (2001)), that presents the marginal relation between the variable of interest and a single variable from the model. An effective and very elastic implementation of this method is available in the **pdp** package for `R` (see Greenwell (2017)). This method has many extensions, like for example Individual Conditional Expectations (Goldstein et al. (2015)). The ICE method allows to trace predictions for individual variables and is very useful for identification of interactions. On the other hand, ALE plots (Apley (2016)) were proposed to better handle the case of strongly correlated predictors by describing conditional distribution of predicted values. It can be used to asses both main effects and interactions between predictors. All these methods are focused on the relation between the effect of a single variable or small set of variables within the black box model.

A different approach is presented in the article Tulio Ribeiro et al. (2016b). Here authors proposed LIME (Locally Interpretable Model agnostic Explanations) as a method of explaining black box predictions by fitting an interpretable model locally around a prediction of interest. This methodology was illustrated with examples from image and text classification areas. Later, it was extended by Puri et al. (2017) to MAGIX methodology (Model Agnostic Globally Interpretable Explanations) and modified by the authors of the original article to aLIME (anchor-LIME) in Tulio Ribeiro et al. (2016a).

So far, two implementations of the method have been created. Python library was developed by the authors of the original article is available on GitHub at `https://github.com/marcotcr/lime`. It works for any text or image classifier and also for tabular data. Regression models can be explained using simple linear regression. The R package **lime** which is a port to the original Python package is maintained by Thomas Lin Pedersen. This package works with tabular and text data and handles all models supported by either **caret** or **mlr** package and can be easily extended to work with other models. Neither of the packages implements *sp-LIME* algorithm that was proposed in the original article to choose representative observations that would explain the behavior of the model globally.

In this article we give a short overview of methods of explaining predictions made by complex models and introduce two new methods implemented in R packages **live** and **breakDown**.

Related work

In this section we present two most recognized methods for explanations of a single prediction from a complex black box model.

Locally Interpretable Model-agnostic Explanations (LIME)

In 2016 Tulio Ribeiro et al. (2016b) proposed **LIME** method for explaining prediction for a single observation which takes a significantly different approach compared to methods described above. The algorithm is designed to explain predictions of any classifier, primarily for image and text data. First, original observation of interest is transformed to simplified input space of binary vectors (for example presence or absence of words). Then a dataset of similar observations is created by sampling features that are present in the representation of explained instance. Closeness of these observations to the original observations is measured via specified similarity kernel. This distance is taken into account while the explanation model is fitted to the new dataset. The interpretable model can be penalized to assure that it doesn't become too complex itself.

Shapley values (SHAP)

In 2017 Lundberg and Lee (2017) introduced a general framework for explaining machine learning models that encompasses LIME among other methods. The method is associated with some specific visualization techniques that present how predictors contribute to the predicted values. In this framework, observations are transformed to the space of simplified inputs. Explanation models are restricted to so-called **additive feature attributions methods**, which means that values predicted by the explanation model are linear combinations of these binary input vectors. Formally, if $z = (z_1, \ldots, z_p)$ is a vector in simplified inputs space and g is the explanation model, then

$$g(z) = \phi_0 + \sum_{j=1}^{M} \phi_j z_j,$$

where $\phi_j, j = 0, \ldots, M$ are weights. These weights measure how each feature contributes to the prediction. Authors prove that in this class of explanation models **Shapley values** provide unique solutions to the problem of finding optimal weights ϕ_j that assure the model has desirable properties of local accuracy and consistency. For formal treatment and examples, please refer to the original article Lundberg and Lee (2017). Currently, SHAP values cannot be computed using any R package available on CRAN, but a development version of a package can be found on `https://github.com/redichh/ShapleyR`.

Local Interpretable Visual Explanations (LIVE)

Next two sections introduce two alternative approaches to explaining model predictions. Both are implemented in R packages, **live** and **breakDown**, respectively.

Motivation

live is an alternative implementation of LIME for regression problems, which emphasizes the role of model visualization in understanding of complex models. Compared to the original LIME, both the method of local exploration and handling of interpretable inputs are changed. Dataset for local exploration is simulated by perturbing the explained instance one feature at a time. The process is described in section 2.3.2. All generated observations are treated as similar to the observation of interest and thus the identity kernel is used. Original variables are used as interpretable inputs. Interpretability of the local explanation comes from a tractable relationship between inputs and the predicted response. Variable selection is optional for linear regression when sparsity is required.

One of the main purposes of live is to provide tools for model visualization, which is why emphasis is put on models that are easy to visualize. For linear models, waterfall plots can be drawn to present how predictors contribute to the overall model score for a given prediction and forest plots (Kennedy, 2017) can be drawn to summarize the structure of local linear approximation. Examples for both techniques are given in section 2.5. Other interpretable models that are equipped with generic `plot` function can be visualized, too. In particular, decision trees which can be plotted using **partykit** package are well suited for this task, as they can help discover interactions. The package uses the `mlr` interface to handle machine learning algorithms, so any classifier or regression method supported by

`mlr` can be used as a black box model and similarly any classifier or regression method can be used as an interpretable model.

Methodology

live package uses a two-step procedure to explain prediction of a selected black box model in the point x. First, an artificial dataset X' is created around point x. Second, the white box model is fitted to the model predictions in points X'.

The first step is described by the Algorithm 1. When the number of predictors is smaller or equal than the desired size of the simulated dataset for local exploration, it is created according to the following procedure. When the number of predictors is bigger than the size of *fake* dataset, a random

Algorithm 1 Simulating X' - surroundings around the selected x.

1: $p \leftarrow$ number of predictors
2: $n \leftarrow$ number of observations to generate
3: Duplicate the given observation $n$times
4: **if** $p > n$ **then**
5: Randomly pick n predictors
6: **for** i in $\{1, \ldots, n\}$ **do**
7: Draw number $k \in \{1, \ldots, n\}$ uniformly. Replace k-th variable in I with a random draw from the empirical distribution of this variable [in i-th duplicate of the original observation]
8: **end for**
9: **else**
10: **for** i in $\{1, \ldots, p\}$ **do**
11: Replace the value of i-th variable in i-th observation with a draw from empirical distribution of this variable
12: For the remaining n - p observations, proceed as in case $p > n$.
13: **end for**
14: **end if**

subset of n predictors is chosen, where n is the number of observations, and the procedure described above is performed on this subset. In other words, the procedure amounts to iterating over the set of n observation identical to given instance and changing a value of one variable at each step. Current implementation of this algorithm relies on **data.table** package for performance (Dowle and Srinivasan, 2017).

Model agnostic greedy explanations of model predictions (breakDown)

Motivation

live package approximates the local structure of the black box model around a single point in feature space. The idea behind the **breakDown** is different. Here the main goal is to decompose model predictions into parts that can be attributed to particular variables. It is straightforward for linear (and more general: additive) models. Here we present a model agnostic approach that works also for nonlinear models.

Let us use a following notation. $x = (x_1, x_2, ..., x_p) \in X \subset \mathcal{R}^p$ is a vector in feature space X. $f : X \to R$ is a scoring function for the model under consideration, that may be used for regression of classification problems. X^{train} is a training dataset with n observations.

For a single observation x^{new} the model prediction is equal to $f(x^{new})$. Our goal is to attribute parts of this score to variables (dimensions) in the X space.

The lm-break: version for additive models

For linear models (and also generalized linear models) the scoring function (e.g. link function) may be expressed as linear combination of feature vectors.

$$f(x^{new}) = (1, x^{new})(\mu, \beta)^T = \mu + x_1^{new}\beta_1 + \ldots + x_p^{new}\beta_p. \quad (1)$$

In this case it is easy to attribute the impact of feature x_i on prediction $f(x^{new})$. The most straightforward approach would be to use the $x_i^{new}\beta_i$ as the attribution. However, it is easier to interpret variable attributions if they are invariant to scale-location transformations of x_i, like change of the unit or origin. This is why for linear models the **lm-break** variable attributions are defined as $(x_i^{new} - \bar{x}_i)\beta_i$. The equation 1 may be rewritten as

$$f(x^{new}) = (1, x^{new})(\mu, \beta)^T = baseline + (x_1^{new} - \bar{x}_1)\beta_1 + ... + (x_p^{new} - \bar{x}_p)\beta_p \tag{2}$$

where

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

Components $(x_i^{new} - \bar{x}_i)\beta_i$ are all expressed in the same units. For `lm` and `glm` models these values are calculated and plotted by the generic `broken()` function from the **breakDown** package.

The ag-break: model agnostic approach

Interpretation of **lm-break** attributions is straightforward, but limited only to additive models. In this section we present an extension for non-additive models. This extension uses additive attributions to explain predictions from non-additive models thus some information about the model structure will be lost. Still, for many models such attribution may be useful. For additive models the **ag-break** approach gives the same results as **lm-break** approach.

The intuition behind **ag-break** approach is to identify components of x^{new} that cannot be changed without a significant change in the prediction $f(x^{new})$. In order to present this approach in a more formal way we need to first introduce some definitions.

Definition 2.4.1 (Relaxed model prediction) *Let $f^{IndSet}(x^{new})$ denote an expected model prediction for x^{new} relaxed on the set of indexes $IndSet \subset \{1, \ldots, p\}$.*

$$f^{IndSet}(x^{new}) = E[f(x)|x_{indSet} = x_{indSet}^{new}]. \tag{3}$$

Thus $f^{IndSet}(x^{new})$ is an expected value for model response conditioned on variables from set $indSet$ in a way, that $\forall_{i \in indSet} x_i = x_i^{new}$.

Intuition behind relaxed prediction is that we are interested in average model response for observations that are equal to x^{new} for features from set $indSet^C$ and follow the population distribution for features from set $indSet$. Clearly, two extreme cases are

$$f^{\{1,\ldots,p\}}(x^{new}) = f(x^{new}), \tag{4}$$

which is the case of no relaxation, and

$$f^{\emptyset}(x^{new}) = E[f(x)]. \tag{5}$$

which corresponds to full relaxation. We will say that a variable was relaxed, when we don't fix its value and we let it follow the population distribution. This will play a crucial part in algorithm presented in this section.

Since we do not know the joint distribution of x, we will use it's estimate instead.

$$\widehat{f^{IndSet}(x^{new})} = \frac{1}{n}\sum_{i=1}^{n} f(x_{i,-indSet}, x_{indSet}^{new}). \tag{6}$$

We will omit the dashes to simplify the notation.

Definition 2.4.2 (Distance to relaxed model prediction) *For a set of indexes $indSet$ let us define the distance between model prediction and relaxed model prediction.*

$$d(x^{new}, indSet) := |f^{IndSet}(x^{new}) - f(x^{new})|. \tag{7}$$

It is the difference between model prediction for observation x^{new} and observation relaxed on features $indSet$. The smaller the difference, the less important are variables in the set $indSet$.

Definition 2.4.3 (Added feature contribution) *For j-th feature we define its contribution relative to a set of indexes $IndSet$ (added contribution) as*

$$contribution(j) = f^{IndSet \cup \{j\}}(x^{new}) - f^{IndSet}(x^{new}). \tag{8}$$

It is the change in model prediction for x^{new} after relaxation on j.

The model agnostic feature contribution is based on distances to relaxed model predictions. In this approach we are looking for a series of variables that can be relaxed in a way to move model prediction from $f(x^{new})$ to fully relaxed prediction $E[f(x)]$ (expected value over all model predictions). The order of features in this series is important. But here we are using a greedy strategy in which we add features to the *indSet* iteratively (one feature per iteration) and minimize locally the distance to relaxed model prediction.

This approach can be seen as a approximation of Shapley values where feature contribution is linked with the average effect of a feature across all possible relaxations. For additive models these approaches are identical. For non-additive models the additive attribution is just an approximation in both cases, yet the greedy strategy produces explanations that are easier to interpret. It is worth noting that similar decomposition of predictions and measures of contribution for classifiers have been studied in Robnik-Šikonja and Kononenko (2008).

The greedy search can start from null set of indexes (then in each step a single feature is being relaxed) or it can start from full set of relaxed features (then in each step a single feature is removed from this set). These approaches are named *step-up* and *step-down*, respectively. They are presented in algorithms 2 and 3.

The algorithm 2 presents the procedure that generates a sequence of variables with increasing contributions. This sequence corresponds to variables that can be relaxed in such a way that minimizes the distance to the original prediction. Resulting sequence of *Contributions* and *Variables* may be plotted with Break Down Plots, see an example in Figure 2. Figure 1 summarizes the idea behind algorithm 2. By relaxing consecutive variables one finds path between single prediction and average model prediction.

One can also consider an opposite strategy. Instead of starting from $IndSet = \{1, \ldots, p\}$ one can start with $IndSet = \emptyset$, this strategy is called *step-up* approach and is presented in Algorithm 3.

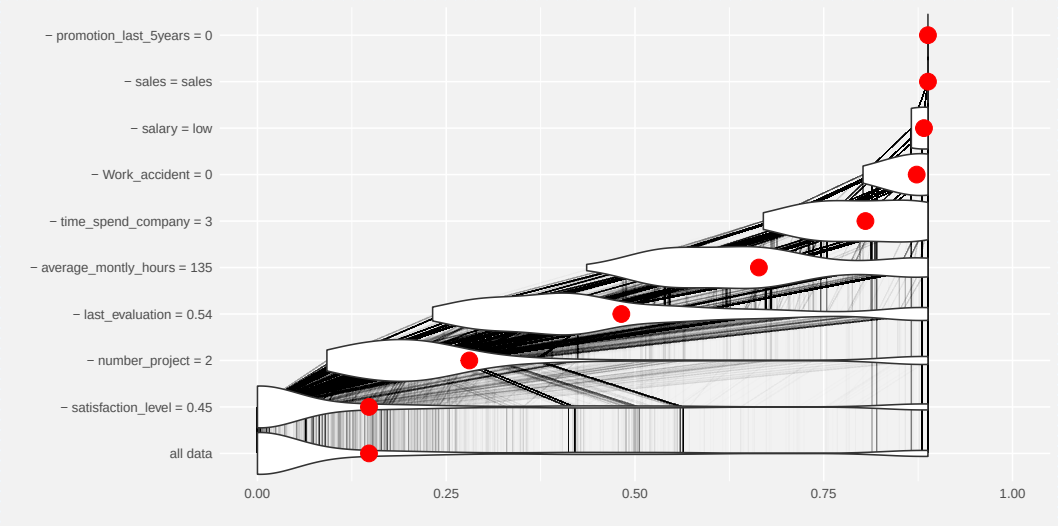

Figure 1: An illustration of algorithm 2. Each row in this plot corresponds to a distribution of model scores $f(x)|x_{indSet} = x^{new}_{indSet}$ for different sets of indexes $indSet$. Initially $indSet = \{1, \ldots, p\}$ and in each step single variable is removed from this set. Labels on the right side of the plots shows which variable is removed in a given step. Red dots stand for conditional average - an estimate of relaxed predictions $f^{IndSet}(x^{new})$. Violin plots summarize conditional distributions of scores while gray lines show how model predictions change for particular observations between consecutive relaxations.

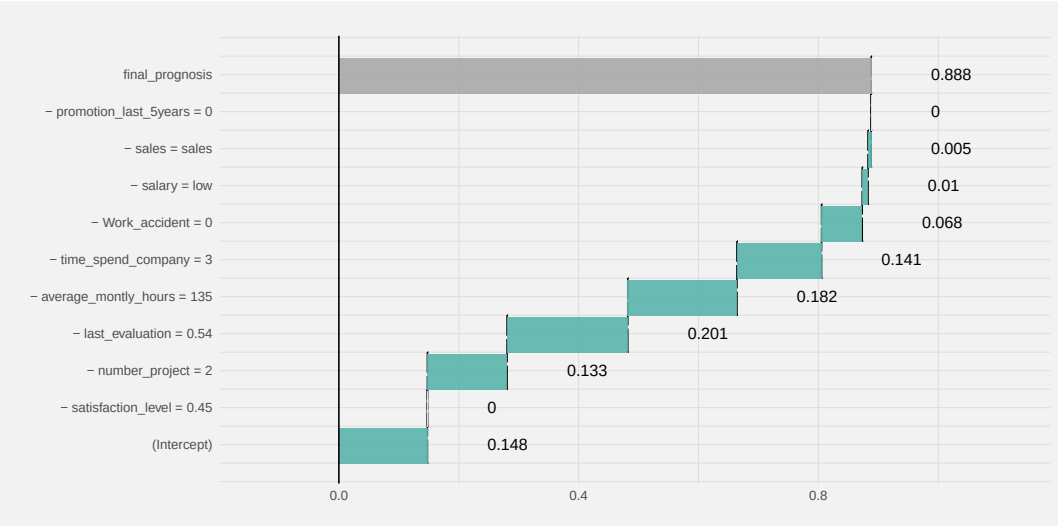

Figure 2: Break Down Plot for decomposition identified in Figure 1. Beginning and end of each rectangle correspond to relaxed prediction (red dot in figure 1) with and without particular feature.

Algorithm 2 Model agnostic break down of model predictions. The *step-down* approach.

$p \leftarrow$ number of variables
$IndSet \leftarrow \{1, \ldots, p\}$ set of indexes of all variables
for i in $\{1, \ldots, p\}$ **do**
    Find new variable that can be relaxed with small loss in relaxed distance to $f(x^{new})$
    for j in $IndSet$ **do**
        Calculate relaxed distance with j removed
        $dist(j) \leftarrow d(x^{new}, IndSet \setminus \{j\})$
    end for
    Find and remove j that minimizes loss
    $j_{min} \leftarrow \arg\min_j dist(j)$
    $Contributions(i) \leftarrow f^{IndSet}(x^{new}) - f^{IndSet \setminus \{j_{min}\}}(x^{new})$
    $Variables(i) \leftarrow j_{min}$
    $IndSet \leftarrow IndSet \setminus \{j_{min}\}$
end for

Algorithm 3 Model agnostic break down of model predictions. The *step-up* approach.

$p \leftarrow$ number of variables
$IndSet \leftarrow \emptyset$ empty set
for i in $\{1, \ldots, p\}$ **do**
    Find new variable that can be relaxed with large distance to $f^{\emptyset}(x^{new})$
    for j in $\{1, \ldots, p\} \setminus IndSet$ **do**
        Calculate relaxed distance with j added
        $dist(j) \leftarrow d(x^{new}, IndSet \cup \{j\})$
    end for
    Find and add j that maximize distance
    $j_{max} \leftarrow \arg\max_j dist(j)$
    $Contributions(i) \leftarrow f^{IndSet \cup \{j_{max}\}}(x^{new}) - f^{IndSet}(x^{new})$
    $Variables(i) \leftarrow j_{max}$
    $IndSet \leftarrow IndSet \cup \{j_{max}\}$
end for

Case study: How good is this red wine?

The wine quality data (Cortez et al., 2009) is a well-known dataset which is commonly used as an example in predictive modeling. The main objective associated with this dataset is to predict the quality of some variants of Portuguese „Vinho Verde" based on 11 chemical properties. A single observation from the dataset can be found in Table 1. According to the results from the original article, the Support Vector Machine (SVM) model performs better than other models including linear regression, neural networks and others.

In this section we will show how **live** package can be used to fit linear regression model locally and generate a visual explanation for the black box model and also how **breakDown** package can be used to attribute parts of the final prediction to particular features.

fixed acidity	volatile acidity	citric acid	res. sugar	Cl^-	free SO_2	total SO_2	D	pH	SO_4^{2-}	alcohol
7.40	0.70	0.00	1.90	0.08	11.00	34.00	1.00	3.51	0.56	9.40

Table 1: The fifth observation in wine quality dataset. D denotes density, Cl^- stands for chlorides, "res." for residual and SO_4^{2-} for sulphates.

The SVM model used in this example is trained with the use of **e1071** package.

```
library("e1071")
wine_svm_model <- svm(quality~., data = wine)
```

Different approaches for explanations of a single prediction are illustrated based on prediction for fifth wine from this dataset, the one presented in table 1. The actual quality of this wine is 5, while quality predicted by the SVM model is 5.03.

```
nobs <- wine[5, ]
predict(wine_svm_model, nobs)
##        1
## 5.032032
```

The live package

The **live** package approximates black box model (here SVM model) with a simpler white box model (here linear regression model) to explain the local structure of a black box model and in consequence to assess how features contribute into a single prediction.

To do this, first we need to generate artificial observations around the selected observation x^{new} for local exploration. We use `sample_locally` function from **live** package.

```
library("live")
similar <- sample_locally(data = wine,
                          explained_instance = wine[5, ],
                          explained_var = "quality",
                          size = 500)
similar <- add_predictions(data = wine,
                           to_explain = similar,
                           black_box_model = wine_svm_model)
```

If multiple models are to be explained, there is no need to generate multiple *artificial* datasets. Predictions of each model on a single simulated dataset can be added with the use of `add_predictions` function. A different object should be created for each model, but the same result of a call to `sample_locally` function should be used a `to_explain` argument. Black box model can be passed as a model object or as a name of **mlr** learner. While the object created by `sample_locally` function stores the dataset and the name of the response variable, object returned by `add_predictions` function also stores the fitted black box model. The result of applying `sample_locally` functions doesn't contain the response and the result of `add_predictions` contains a column with model predictions, which has the same name as response in original dataset.

Once the artificial data points around x^{new} are generated, we may fit the white box model to them. Here we fit linear regression model using `fit_explanation` function.

```
wine_expl <- fit_explanation(live_object = similar,
                             white_box = "regr.lm")
```

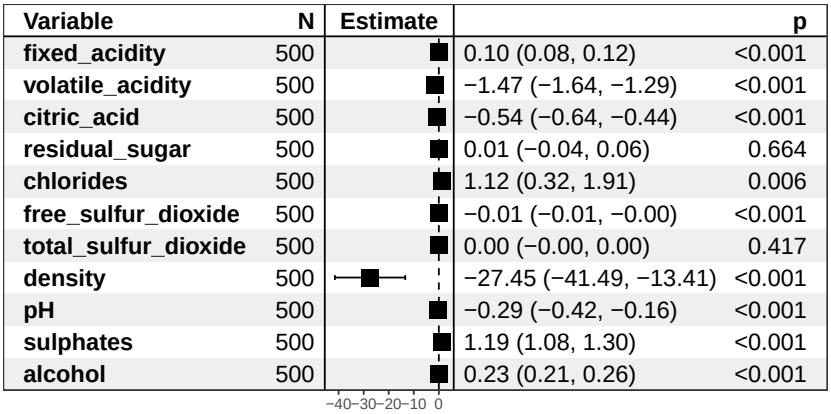

Figure 3: *Forest plot* for a linear white box model that approximates the black box model.

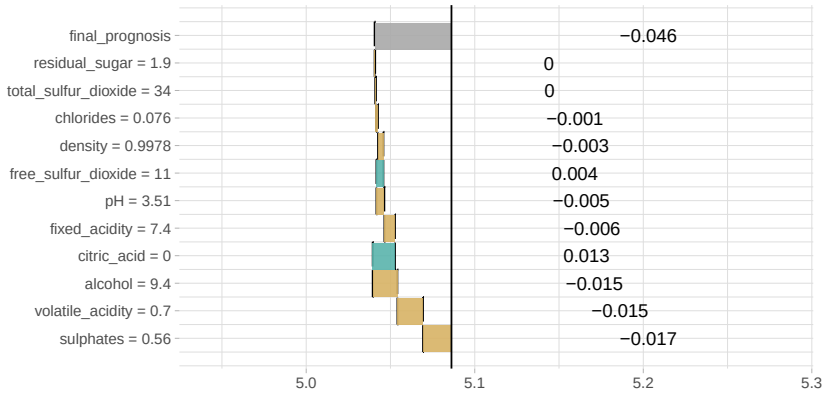

Figure 4: *Waterfall plot* for additive components of linear model that approximates the black box model around x^{new}.

This function returns a native `mlr` object. Model object (for example lm object) can be extracted with the use of `getLearnerModel` function.

The white box model `wine_expl` approximates the black box model `wine_svm_model` around x^{new}. Coefficients of this model can be graphically presented with the `plot_explanation` function. See the corresponding Forest Plot in Figure 3 and corresponding Waterfall Plot in Figure 4.

```
plot_explanation(model = wine_expl, regr_plot_type = "forestplot",
                 explained_instance = wine[5, ])
plot_explanation(model = wine_expl, regr_plot_type = "waterfallplot",
                 explained_instance = wine[5, ])
```

For datasets with larger number of variables, we could obtain sparse results by setting `selection = TRUE` in the `fit_explanation` function. With this option variable selection based on LASSO implemented in **glmnet** package is performed ((Friedman et al., 2010), (Simon et al., 2011)). When using Generalized Linear Model as a white box model it is possible to set `family` argument to one of the distribution families available in `glm` and `glmnet` functions via `response_family` argument to `fit_explanation`.

The lime package

The LIME method is implemented in the R package **lime** (Pedersen and Benesty, 2017). By default it produces sparse explanations.

In the first step a `lime` object is created for a specified dataset and fitted black box model.

```
library("lime")
wine_expl <- lime(nobs, wine_svm_model)
```

Then we use the `explain` function, which in case of regression takes the observation of interest, `lime` object and the number of top features to be used for explanation. In this case data are low dimensional, so we can use all predictors. Alternatively, we could set `feature_select` to `none` to skip the selection part.

```
model_type.svm <- function(x, ...) "regression"
svm_explained <- explain(nobs, wine_expl, n_features = 11)
plot_explanation(svm_explained)
```

Results produced by the `plot_explanation` function are presented in Figure 5.

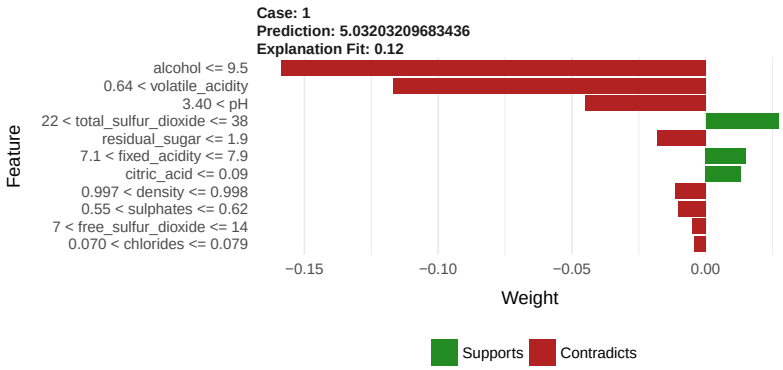

Figure 5: Contributions of particular features to the prediction calculated with SVM model assessed with **lime** package.

The breakDown package

The **breakDown** package directly calculates variable attributions for a selected observation. It does not use any surrogate model.

The `broken()` function is used to calculate feature attributions. Generic functions `print()` and `plot()` show feature attributions as a text or with the use of waterfall plots. The `baseline` argument specifies the origin of a waterfall plot. By default it's 0. Use `baseline = "intercept"` to set the origin to average model prediction.

```
library("breakDown")
explain_5 <- broken(wine_svm_model, new_observation = nobs,
                    data = wine,
                    baseline = "intercept",
                    direction = "up")
explain_5
##                                contribution
## baseline                          5.613
## + alcohol = 9.4                  −0.318
## + volatile_acidity = 0.7         −0.193
## + sulphates = 0.56               −0.068
## + pH = 3.51                      −0.083
## + residual_sugar = 1.9           −0.035
## + density = 0.9978               −0.031
## + chlorides = 0.076              −0.021
## + total_sulfur_dioxide = 34      −0.003
## + quality = 5                     0.000
## + free_sulfur_dioxide = 11        0.004
## + fixed_acidity = 7.4             0.024
## + citric_acid = 0                 0.144
## final_prognosis                   5.032

plot(explain_5)
```

Figure 6 shows variable contributions for step-up and step-down strategy. Variable ordering is different but the contributions are consistent across both strategies.

Find more examples for classification and regression models created with **caret**, **mlr**, **randomForest** and other frameworks in package vignettes at `https://pbiecek.github.io/breakDown/`.

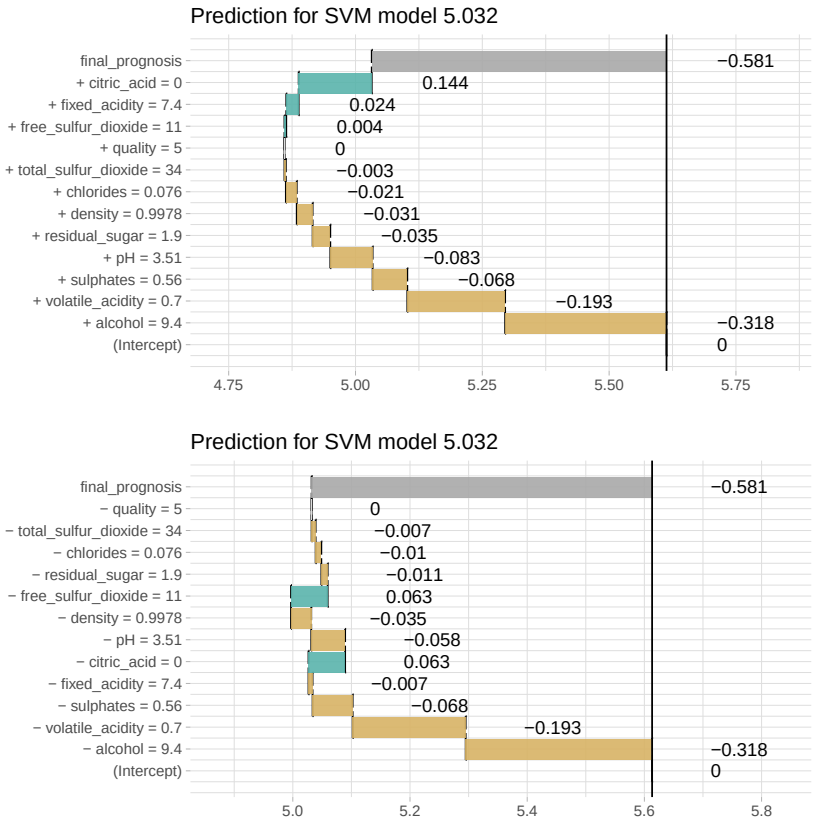

Figure 6: ag-break feature attributions for SVM model calculated for 5th wine. Upper plot presents feature attributions for the step-up strategy, while the bottom plot presents results for the step-down strategy. Attributions are very similar even if the ordering is different. Vertical black line shows the average prediction for the SVM model. The 5th wine gets final prediction 5.032 which is 0.581 point below the average for this model.

Shapley values (SHAP)

Authors of the original article about Shapley values maintain a Python package which implements several methods of computing the Shapley values and provides visual diagnostic tools that help understand black box models such a plotting Shapley values for all instances in the dataset, exploring the influence of a single variable on predictions by plotting it's value vs Shapley values and interaction Shapley values.

The only R package for Shapley values is in development stage. It can be found at at address https://github.com/redichh/ShapleyR. So far, it allows user to compute Shapley values. It uses the `mlr` interface to train models, so first, we need to create `mlr` task, and then pass it to `shapley` function along with the row number of observation for which we explain the prediction and an `mlr` object with fitted model.

```
library(shapleyr)
tsk <- makeRegrTask("wine", wine, "quality")
shp <- shapley(5, model = train("regr.svm", tsk), task = tsk)
```

In table 2 we present Shapley values and compare them to contribution calculated with **ag-break** algorithm.

	Variable	Shapley values	**ag-break** contributions
	Baseline prediction	5.61	5.61
1	alcohol	-0.22	-0.32
2	sulphates	-0.19	-0.07
3	volatile_acidity	-0.16	-0.19
4	citric_acid	0.13	0.14
5	pH	-0.08	-0.08
6	fixed_acidity	-0.04	0.02
7	free_sulfur_dioxide	0.03	0.00
8	residual_sugar	-0.02	-0.03
9	total_sulfur_dioxide	-0.01	-0.00
10	density	-0.01	-0.03
11	chlorides	0.01	-0.02
	Final prediction	5.032	5.032

Table 2: Comparison of feature attributions calculated with **ShapleyR** and **breakDown** packages.

Discussion

In this paper we presented four approaches and four R packages that can be used for explanations of predictions from complex black box models. Two of them were introduced in literature while **live** and **brakDown** are introduced in this article for the first time.

All four approaches are model agnostic in a way, that the method does not depend on any particular structure of black box model. Yet there are also some differences between these approaches.

- Surrogate models vs. conditional expected responses. **live** and **lime** packages use surrogate models (so called white box models) that approximate local structure of the complex black box model. Coefficients of these surrogate models are used for explanations. On the contrary, **breakDown** and **shapleyr** construct feature attributions based on conditional responses of a black box model.
- **live** and **lime** packages differ in a way how the surrounding of x^{new} is defined. This task is highly non-trivial especially for mixed data with continuous and categorical features. **live** does not use *interpretable input space* (and so does not fall under the *additive feature attribution methods* category), but approximates the black box model directly in the data space, which can be considered a more effective use of data. It comes with no theoretical guarantees that are provided for *Shapley values*, but is very intuitive and offers several tools for visual inspection of the model.
- **shapleyr** and **breakDown** take conditional expectation of the predictor function with respect to explanatory features. They differ in how conditioning is applied to calculate feature attributions. **shapleyr** is based on results from game theory, here contribution of a single feature is averaged across all possible conditionings. **breakDown** uses a greedy approach in which only single series of nested conditionings is considered. The greedy approach is easier to interpret and faster to compute. Moreover, exact methods of computing *Shapley values* exist only for linear regression and tree ensemble models. Approximate computations are also problematic, as they require the choice of number of samples of subsets of predictors which will be used. These two methods produce nearly identical results for linear models (see table Table 2), but for more complex models the estimated contributions can be very different, even pointing in opposite directions. An advantage of *Shapley values* are proven theoretical properties, though they are restricted to explanation models that belong to the *additive feature attribution methods* class.
- When parameters (kernel and regularization term) are chosen as in Lundberg and Lee (2017), **lime** produces estimates of *Shapley values*. Thus other choices of kernel and penalty term lead to inconsistent results. The fact that the suggested penalty term is equal to 0 can be considered a huge limitation of LIME and SHAP, because in this setting they won't produce sparse explanations.
- All presented methods decompose final prediction into additive components attributed to particular features. This approach will not work well for models with heavy components related to interactions between features.

Comparison of these methods presented in the previous section is far from being comprehensive. More studies are needed to better understand differences between these approach and new approaches are needed to overcome constrains listed above. Yet the availability of mentioned packages creates an opportunity for further studies on model exploration.

Acknowledgements

This work was financially supported by the *NCN Opus grant 2016/21/B/ST6/02176*.

Bibliography

A. Altmann, L. Toloşi, O. Sander, and T. Lengauer. Permutation importance: a corrected feature importance measure. *Bioinformatics*, 26(10):1340–1347, May 2010. ISSN 1460-2059, 1367-4803. doi: 10.1093/bioinformatics/btq134. [p]

D. W. Apley. Visualizing the Effects of Predictor Variables in Black Box Supervised Learning Models. *ArXiv e-prints*, Dec. 2016. [p]

B. Bischl, M. Lang, L. Kotthoff, J. Schiffner, J. Richter, E. Studerus, G. Casalicchio, and Z. M. Jones. mlr: Machine learning in r. *Journal of Machine Learning Research*, 17(170):1–5, 2016. URL http://jmlr.org/papers/v17/15-066.html. [p]

P. Cortez, A. Cerdeira, F. Almeida, T. Matos, and J. Reis. Modeling wine preferences by data mining from physicochemical properties. *Decis. Support Syst.*, 47(4):547–553, Nov. 2009. ISSN 0167-9236. doi: 10.1016/j.dss.2009.05.016. URL http://dx.doi.org/10.1016/j.dss.2009.05.016. [p]

M. Dowle and A. Srinivasan. *data.table: Extension of 'data.frame'*, 2017. URL https://CRAN.R-project.org/package=data.table. R package version 1.10.4-3. [p]

A. Fogg. *Anthony Goldbloom gives you the secret to winning Kaggle competitions*, 2016. URL https://www.import.io/post/how-to-win-a-kaggle-competition/. [p]

J. Friedman, T. Hastie, and R. Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1):1–22, 2010. URL http://www.jstatsoft.org/v33/i01/. [p]

J. H. Friedman. Greedy function approximation: A gradient boosting machine. *Ann. Statist.*, 29(5):1189–1232, 10 2001. doi: 10.1214/aos/1013203451. URL https://doi.org/10.1214/aos/1013203451. [p]

A. Goldstein, A. Kapelner, J. Bleich, and E. Pitkin. Peeking Inside the Black Box: Visualizing Statistical Learning with Plots of Individual Conditional Expectation. *ArXiv e-prints*, Sept. 2013. [p]

A. Goldstein, A. Kapelner, J. Bleich, and E. Pitkin. Peeking Inside the Black Box: Visualizing Statistical Learning With Plots of Individual Conditional Expectation. *Journal of Computational and Graphical Statistics*, 24(1):44–65, Jan. 2015. ISSN 1061-8600, 1537-2715. doi: 10.1080/10618600.2014.907095. URL http://www.tandfonline.com/doi/full/10.1080/10618600.2014.907095. [p]

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

N. Kennedy. *forestmodel: Forest Plots from Regression Models*, 2017. URL https://CRAN.R-project.org/package=forestmodel. R package version 0.4.3. [p]

S. Lundberg and S.-I. Lee. A unified approach to interpreting model predictions. *ArXiv e-prints*, May 2017. [p]

S. Marco Tulio Ribeiro and Carlos Guestrin. "why should i trust you?" explaining the predictions of any classifier. *KDD '16 Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1135–1144, 2016. [p]

T. L. Pedersen and M. Benesty. *lime: Local Interpretable Model-Agnostic Explanations*, 2017. URL https://CRAN.R-project.org/package=lime. R package version 0.3.1. [p]

N. Puri, P. Gupta, P. Agarwal, S. Verma, and B. Krishnamurthy. MAGIX: Model Agnostic Globally Interpretable Explanations. *ArXiv e-prints*, June 2017. [p]

M. Robnik-Šikonja and I. Kononenko. Explaining classifications for individual instances. *IEEE Transactions on Knowledge and Data Engineering*, 20(5):589–600, May 2008. ISSN 1041-4347. doi: 10.1109/TKDE.2007.190734. [p]

N. Simon, J. Friedman, T. Hastie, and R. Tibshirani. Regularization paths for cox's proportional hazards model via coordinate descent. *Journal of Statistical Software*, 39(5):1–13, 2011. URL http://www.jstatsoft.org/v39/i05/. [p]

C. Strobl, A.-L. Boulesteix, T. Kneib, T. Augustin, and A. Zeileis. Conditional variable importance for random forests. *BMC Bioinformatics*, 9(307), 2008. URL http://www.biomedcentral.com/1471-2105/9/307. [p]

M. Tulio Ribeiro, S. Singh, and C. Guestrin. Nothing Else Matters: Model-Agnostic Explanations By Identifying Prediction Invariance. *ArXiv e-prints*, Nov. 2016a. [p]

M. Tulio Ribeiro, S. Singh, and C. Guestrin. Model-Agnostic Interpretability of Machine Learning. *ArXiv e-prints*, June 2016b. [p]

F. Y. Tzeng and K. L. Ma. Opening the black box - data driven visualization of neural networks. In *VIS 05. IEEE Visualization, 2005.*, pages 383–390, Oct 2005. doi: 10.1109/VISUAL.2005.1532820. [p]

E. Štrumbelj and I. Kononenko. A general method for visualizing and explaining black-box regression models. In *Proceedings of the 10th International Conference on Adaptive and Natural Computing Algorithms - Volume Part II*, ICANNGA'11, pages 21–30, Berlin, Heidelberg, 2011. Springer-Verlag. ISBN 978-3-642-20266-7. URL http://dl.acm.org/citation.cfm?id=1997005.1997009. [p]

M. D. Zeiler and R. Fergus. *Visualizing and Understanding Convolutional Networks*, page 818–833. Springer International Publishing, 2014. ISBN 978-3-319-10590-1. [p]

Mateusz Staniak
Mathematical Institute
University of Wroclaw
Poland
mateusz.staniak@math.uni.wroc.pl

Przemysław Biecek
Faculty of Mathematics and Information Science
Warsaw University of Technology
Faculty of Mathematics, Informatics and Mechanics
University of Warsaw
Poland
przemyslaw.biecek@gmail.com