This is code that will encompany an article that will appear in a special edition of a German IT magazine. The article is about explaining black-box machine learning models. In that article I’m showcasing three practical examples:

1. Explaining supervised classification models built on tabular data using caret and the iml package
2. Explaining image classification models with keras and lime
3. Explaining text classification models with xgboost and lime

Below, you will find the code for the first part:

# data wrangling

library(tidyverse)

library(readr)

# ml

library(caret)

# plotting

library(gridExtra)

library(grid)

library(ggridges)

library(ggthemes)

theme\_set(theme\_minimal())

# explaining models

library(iml)

First we train a Random Forest to predict the Boston median housing value. How does lstat influence the prediction individually and on average? (Accumulated local effects)

library("iml")

library("randomForest")

data("Boston", package = "MASS")

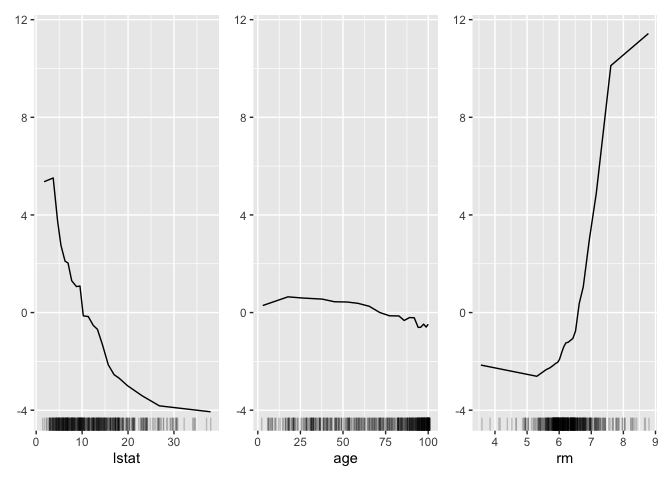
rf = randomForest(medv ~ ., data = Boston, ntree = 50)

X = Boston[which(names(Boston) != "medv")]

model = Predictor$new(rf, data = X, y = Boston$medv)

effect = FeatureEffects$new(model)

effect$plot(features = c("lstat", "age", "rm"))

[](https://github.com/christophM/iml/blob/main/man/figures/README-unnamed-chunk-2-1.png)

library(breakDown)

The R code

library(breakDown)

url <- 'https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-white.csv'

wine <- read.table(url, header = T, sep=";")

head(wine, 3)

## fixed.acidity volatile.acidity citric.acid residual.sugar chlorides free.sulfur.dioxide total.sulfur.dioxide density pH

## 1 7.0 0.27 0.36 20.7 0.045 45 170 1.0010 3.00

## 2 6.3 0.30 0.34 1.6 0.049 14 132 0.9940 3.30

## 3 8.1 0.28 0.40 6.9 0.050 30 97 0.9951 3.26

## sulphates alcohol quality

## 1 0.45 8.8 6

## 2 0.49 9.5 6

## 3 0.44 10.1 6

model <- lm(quality ~ fixed.acidity + volatile.acidity + citric.acid + residual.sugar + chlorides + free.sulfur.dioxide + total.sulfur.dioxide + density + pH + sulphates + alcohol,

data = wine)

new\_observation <- wine[1,]

br <- broken(model, new\_observation)

br

## contribution

## (Intercept) 5.90000

## residual.sugar = 20.7 1.20000

## density = 1.001 -1.00000

## alcohol = 8.8 -0.33000

## pH = 3 -0.13000

## free.sulfur.dioxide = 45 0.03600

## sulphates = 0.45 -0.02500

## volatile.acidity = 0.27 0.01500

## fixed.acidity = 7 0.00950

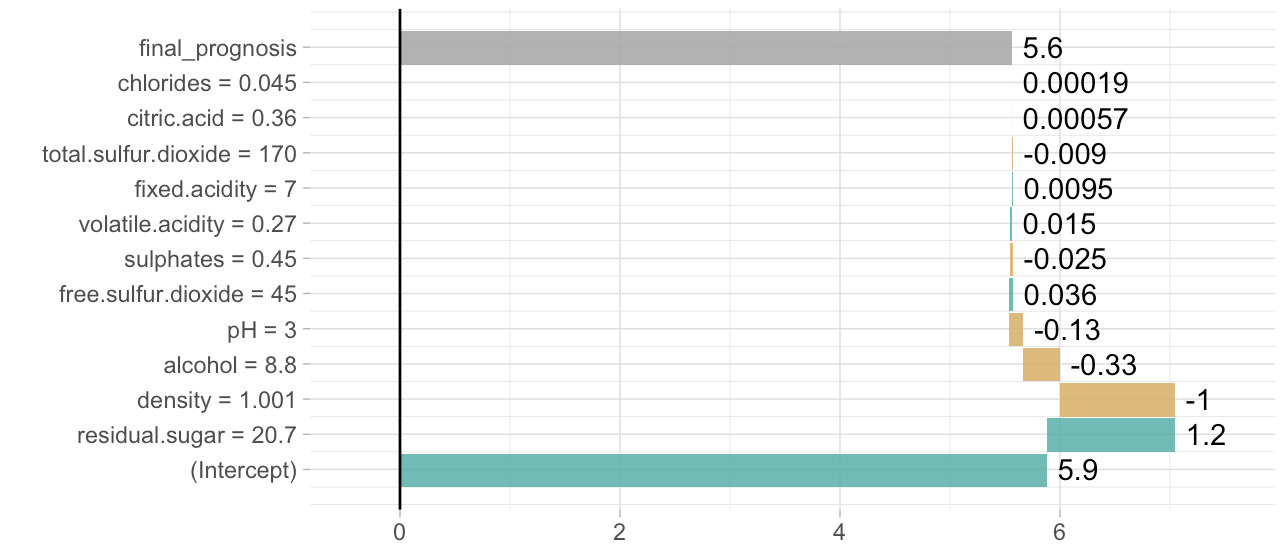
## total.sulfur.dioxide = 170 -0.00900

## citric.acid = 0.36 0.00057

## chlorides = 0.045 0.00019

## final\_prognosis 5.60000

plot(br)



library(DALEX)

**Supervised classfication models with tabular data**

**The data**

The example dataset I am using in this part is the [wine quality data](https://www.kaggle.com/uciml/red-wine-quality-cortez-et-al-2009). Let’s read it in and do some minor housekeeping, like

* converting the response variable quality into two categories with roughly equal sizes and
* replacing the spaces in the column names with “\_" to make it easier to handle in the tidyverse

wine\_data <- read\_csv("/Users/shiringlander/Documents/Github/ix\_lime\_etc/winequality-red.csv") %>%

mutate(quality = as.factor(ifelse(quality < 6, "qual\_low", "qual\_high")))

## Parsed with column specification:

## cols(

## `fixed acidity` = col\_double(),

## `volatile acidity` = col\_double(),

## `citric acid` = col\_double(),

## `residual sugar` = col\_double(),

## chlorides = col\_double(),

## `free sulfur dioxide` = col\_double(),

## `total sulfur dioxide` = col\_double(),

## density = col\_double(),

## pH = col\_double(),

## sulphates = col\_double(),

## alcohol = col\_double(),

## quality = col\_integer()

## )

colnames(wine\_data) <- gsub(" ", "\_", colnames(wine\_data))

glimpse(wine\_data)

## Observations: 1,599

## Variables: 12

## $ fixed\_acidity <dbl> 7.4, 7.8, 7.8, 11.2, 7.4, 7.4, 7.9, 7.3, ...

## $ volatile\_acidity <dbl> 0.700, 0.880, 0.760, 0.280, 0.700, 0.660,...

## $ citric\_acid <dbl> 0.00, 0.00, 0.04, 0.56, 0.00, 0.00, 0.06,...

## $ residual\_sugar <dbl> 1.9, 2.6, 2.3, 1.9, 1.9, 1.8, 1.6, 1.2, 2...

## $ chlorides <dbl> 0.076, 0.098, 0.092, 0.075, 0.076, 0.075,...

## $ free\_sulfur\_dioxide <dbl> 11, 25, 15, 17, 11, 13, 15, 15, 9, 17, 15...

## $ total\_sulfur\_dioxide <dbl> 34, 67, 54, 60, 34, 40, 59, 21, 18, 102, ...

## $ density <dbl> 0.9978, 0.9968, 0.9970, 0.9980, 0.9978, 0...

## $ pH <dbl> 3.51, 3.20, 3.26, 3.16, 3.51, 3.51, 3.30,...

## $ sulphates <dbl> 0.56, 0.68, 0.65, 0.58, 0.56, 0.56, 0.46,...

## $ alcohol <dbl> 9.4, 9.8, 9.8, 9.8, 9.4, 9.4, 9.4, 10.0, ...

## $ quality <fct> qual\_low, qual\_low, qual\_low, qual\_high, ...

**EDA**

The first step in my machine learning workflows in exploratory data analysis (EDA). This can get pretty extensive, but here I am only looking at the distributions of my features and the class counts of my response variable.

p1 <- wine\_data %>%

ggplot(aes(x = quality, fill = quality)) +

geom\_bar(alpha = 0.8) +

scale\_fill\_tableau() +

guides(fill = FALSE)

p2 <- wine\_data %>%

gather(x, y, fixed\_acidity:alcohol) %>%

ggplot(aes(x = y, y = quality, color = quality, fill = quality)) +

facet\_wrap( ~ x, scale = "free", ncol = 3) +

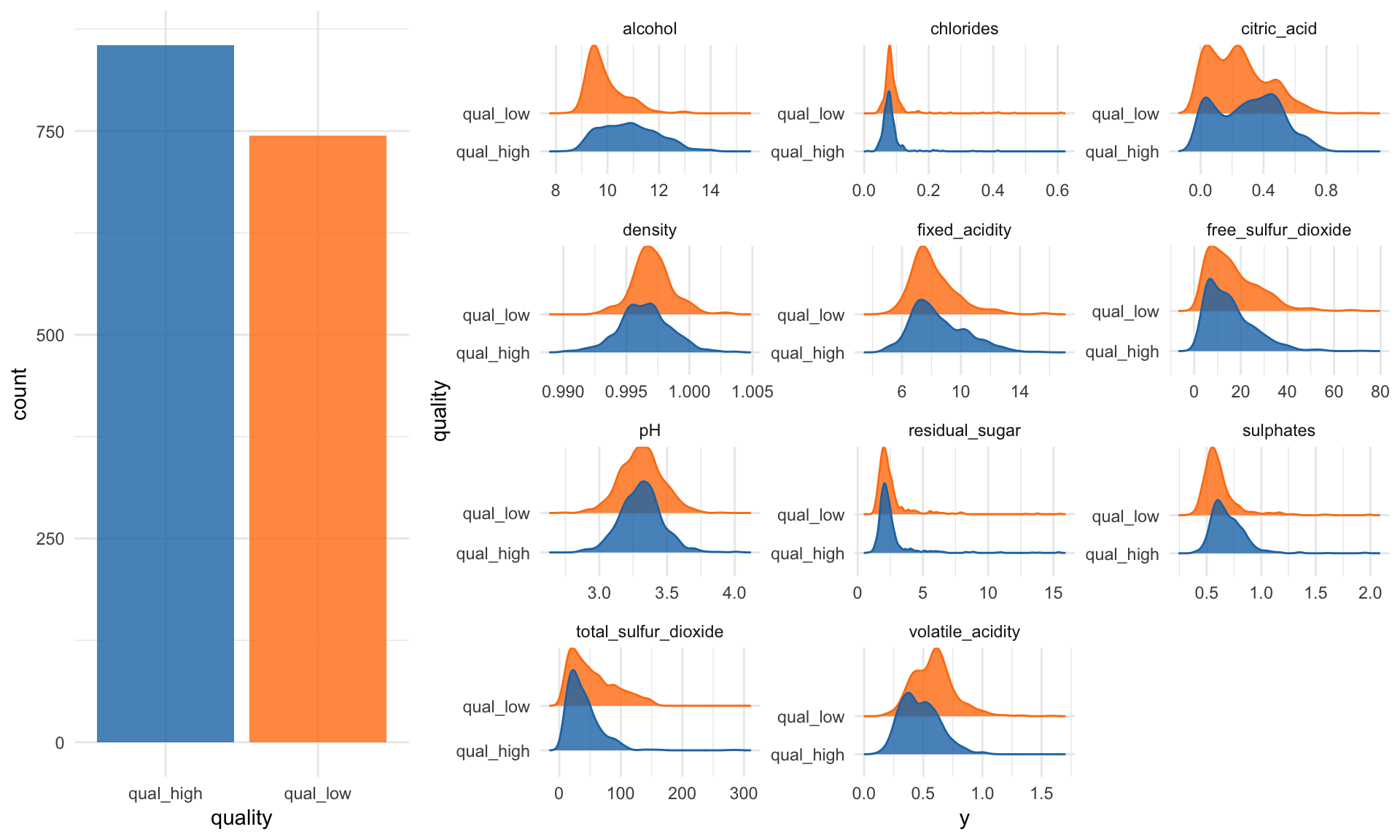
scale\_fill\_tableau() +

scale\_color\_tableau() +

geom\_density\_ridges(alpha = 0.8) +

guides(fill = FALSE, color = FALSE)

grid.arrange(p1, p2, ncol = 2, widths = c(0.3, 0.7))



**Model**

For modeling, I am splitting the data into 80% for training and 20% for testing.

set.seed(42)

idx <- createDataPartition(wine\_data$quality,

p = 0.8,

list = FALSE,

times = 1)

wine\_train <- wine\_data[ idx,]

wine\_test <- wine\_data[-idx,]

I am using 5-fold cross-validation, repeated 3x and scale and center the data. The example model I am using here is a Random Forest model.

fit\_control <- trainControl(method = "repeatedcv",

number = 5,

repeats = 3)

set.seed(42)

rf\_model <- train(quality ~ .,

data = wine\_train,

method = "rf",

preProcess = c("scale", "center"),

trControl = fit\_control,

verbose = FALSE)

rf\_model

## Random Forest

##

## 1280 samples

## 11 predictor

## 2 classes: 'qual\_high', 'qual\_low'

##

## Pre-processing: scaled (11), centered (11)

## Resampling: Cross-Validated (5 fold, repeated 3 times)

## Summary of sample sizes: 1023, 1024, 1025, 1024, 1024, 1024, ...

## Resampling results across tuning parameters:

##

## mtry Accuracy Kappa

## 2 0.7958240 0.5898607

## 6 0.7893104 0.5766700

## 11 0.7882738 0.5745067

##

## Accuracy was used to select the optimal model using the largest value.

## The final value used for the model was mtry = 2.

Now let’s see how good our model is:

test\_predict <- predict(rf\_model, wine\_test)

confusionMatrix(test\_predict, as.factor(wine\_test$quality))

## Confusion Matrix and Statistics

##

## Reference

## Prediction qual\_high qual\_low

## qual\_high 140 26

## qual\_low 31 122

##

## Accuracy : 0.8213

## 95% CI : (0.7748, 0.8618)

## No Information Rate : 0.5361

## P-Value [Acc > NIR] : <2e-16

##

## Kappa : 0.6416

## Mcnemar's Test P-Value : 0.5962

##

## Sensitivity : 0.8187

## Specificity : 0.8243

## Pos Pred Value : 0.8434

## Neg Pred Value : 0.7974

## Prevalence : 0.5361

## Detection Rate : 0.4389

## Detection Prevalence : 0.5204

## Balanced Accuracy : 0.8215

##

## 'Positive' Class : qual\_high

##

Okay, this model isn’t too accurate but since my focus here is supposed to be on explaining the model, that’s good enough for me at this point.

**Explaining/interpreting the model**

There are several methods and tools that can be used to explain or interpret machine learning models.

**Feature importance**

The first metric to look at for Random Forest models (and many other algorithms) is feature importance:

“Variable importance evaluation functions can be separated into two groups: those that use the model information and those that do not. The advantage of using a model-based approach is that is more closely tied to the model performance and that it may be able to incorporate the correlation structure between the predictors into the importance calculation.”

The varImp() function from the caret package can be used to calculate feature importance measures for most methods. For Random Forest classification models such as ours, the prediction error rate is calculated for

1. permuted out-of-bag data of each tree and
2. permutations of every feature

These two measures are averaged and normalized as described here:

“Here are the definitions of the variable importance measures. The first measure is computed from permuting OOB data: For each tree, the prediction error on the out-of-bag portion of the data is recorded (error rate for classification, MSE for regression). Then the same is done after permuting each predictor variable. The difference between the two are then averaged over all trees, and normalized by the standard deviation of the differences. If the standard deviation of the differences is equal to 0 for a variable, the division is not done (but the average is almost always equal to 0 in that case). The second measure is the total decrease in node impurities from splitting on the variable, averaged over all trees. For classification, the node impurity is measured by the Gini index. For regression, it is measured by residual sum of squares.” randomForest help function for importance()

rf\_model\_imp <- varImp(rf\_model, scale = TRUE)

p1 <- rf\_model\_imp$importance %>%

as.data.frame() %>%

rownames\_to\_column() %>%

ggplot(aes(x = reorder(rowname, Overall), y = Overall)) +

geom\_bar(stat = "identity", fill = "#1F77B4", alpha = 0.8) +

coord\_flip()

We can also use a ROC curve for evaluating feature importance. For this, we have the caret::filterVarImp() function:

“The importance of each predictor is evaluated individually using a “filter” approach. For classification, ROC curve analysis is conducted on each predictor. For two class problems, a series of cutoffs is applied to the predictor data to predict the class. The sensitivity and specificity are computed for each cutoff and the ROC curve is computed. The trapezoidal rule is used to compute the area under the ROC curve. This area is used as the measure of variable importance. For multi–class outcomes, the problem is decomposed into all pair-wise problems and the area under the curve is calculated for each class pair (i.e class 1 vs. class 2, class 2 vs. class 3 etc.). For a specific class, the maximum area under the curve across the relevant pair–wise AUC’s is used as the variable importance measure. For regression, the relationship between each predictor and the outcome is evaluated. An argument, nonpara, is used to pick the model fitting technique. When nonpara = FALSE, a linear model is fit and the absolute value of the \(t\)–value for the slope of the predictor is used. Otherwise, a loess smoother is fit between the outcome and the predictor. The \(R^2\) statistic is calculated for this model against the intercept only null model.” caret help for filterVarImp()

roc\_imp <- filterVarImp(x = wine\_train[, -ncol(wine\_train)], y = wine\_train$quality)

p2 <- roc\_imp %>%

as.data.frame() %>%

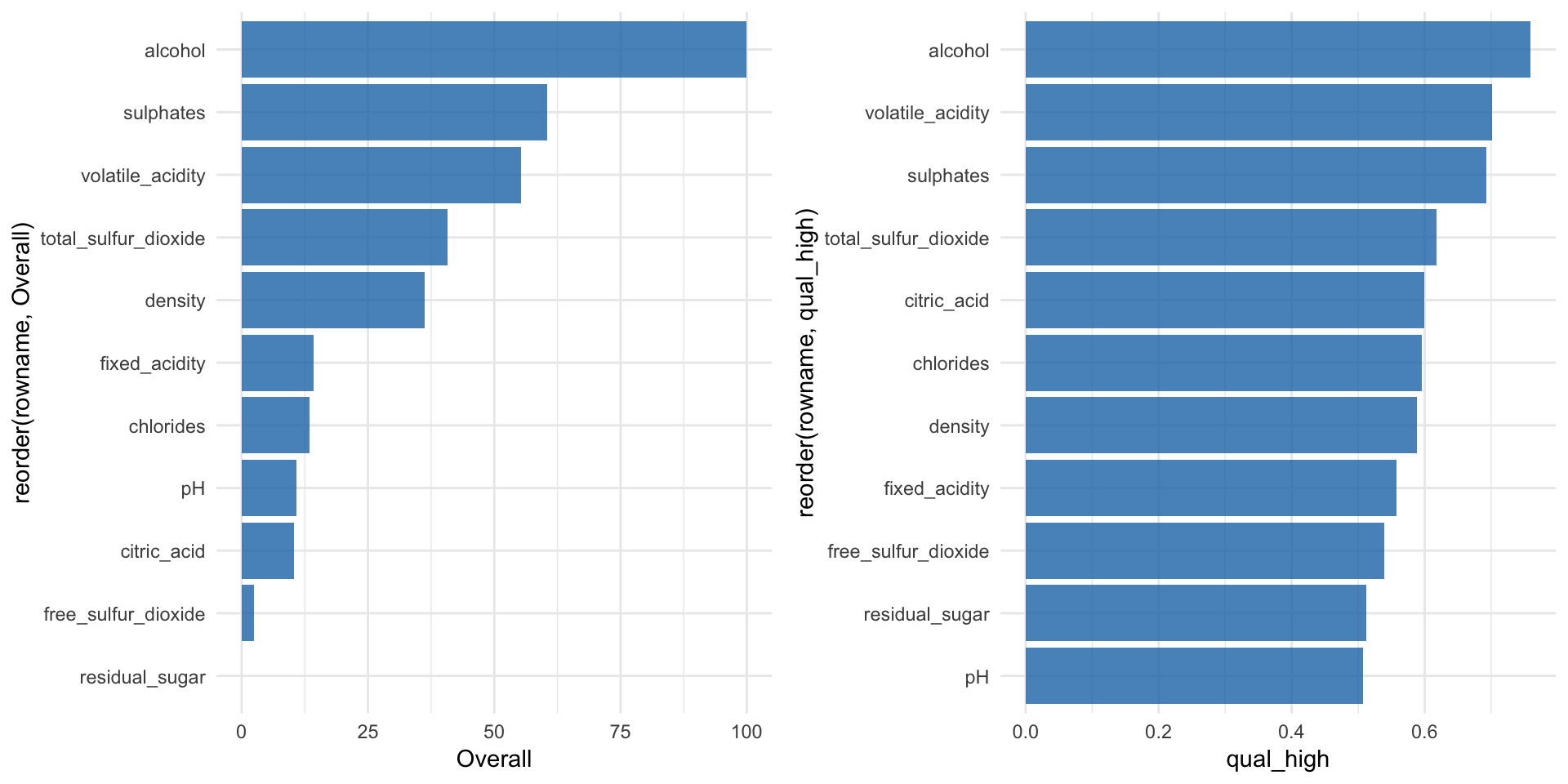
rownames\_to\_column() %>%

ggplot(aes(x = reorder(rowname, qual\_high), y = qual\_high)) +

geom\_bar(stat = "identity", fill = "#1F77B4", alpha = 0.8) +

coord\_flip()

grid.arrange(p1, p2, ncol = 2, widths = c(0.5, 0.5))



**iml**

The iml package combines a number of methods for explaining/interpreting machine learning model, like

* Feature importance
* Partial dependence plots
* Individual conditional expectation plots (ICE)
* Tree surrogate
* LocalModel: Local Interpretable Model-agnostic Explanations (similar to [lime](https://shirinsplayground.netlify.com/2018/06/keras_fruits_lime/))
* Shapley value for explaining single predictions

In order to work with iml, we need to adapt our data a bit by removing the response variable and the creating a new predictor object that holds the model, the data and the class labels.

“The iml package uses R6 classes: New objects can be created by calling Predictor$new().”

Library(iml)

X <- wine\_train %>%

select(-quality) %>%

as.data.frame()

predictor <- Predictor$new(rf\_model, data = X, y = wine\_train$quality)

str(predictor)

## Classes 'Predictor', 'R6' <Predictor>

## Public:

## class: NULL

## clone: function (deep = FALSE)

## data: Data, R6

## initialize: function (model = NULL, data, predict.fun = NULL, y = NULL, class = NULL)

## model: train, train.formula

## predict: function (newdata)

## prediction.colnames: NULL

## prediction.function: function (newdata)

## print: function ()

## task: classification

## Private:

## predictionChecked: FALSE

**Partial Dependence & Individual Conditional Expectation plots (ICE)**

Now we can explore some of the different methods. Let’s start with [partial dependence plots](https://christophm.github.io/interpretable-ml-book/pdp.html) as we had already looked into feature importance.

“Besides knowing which features were important, we are interested in how the features influence the predicted outcome. The Partial class implements partial dependence plots and individual conditional expectation curves. Each individual line represents the predictions (y-axis) for one data point when we change one of the features (e.g. ‘lstat’ on the x-axis). The highlighted line is the point-wise average of the individual lines and equals the partial dependence plot. The marks on the x-axis indicates the distribution of the ‘lstat’ feature, showing how relevant a region is for interpretation (little or no points mean that we should not over-interpret this region).”

|  |
| --- |
| ```{r, echo = FALSE, message = FALSE} |
|  | knitr::opts\_chunk$set( |
|  | comment = "#>", |
|  | fig.width = 7, |
|  | fig.height = 7, |
|  | fig.align = "center" |
|  | ) |
|  | options( |
|  | tibble.print\_min = 4L, |
|  | tibble.print\_max = 4L |
|  | ) |
|  | ``` |
|  |  |
|  |  |
|  | Machine learning models usually perform really well for predictions, but are not interpretable. |
|  | The iml package provides tools for analysing any black box machine learning model: |
|  |  |
|  | \* Feature importance: Which were the most important features? |
|  |  |
|  | \* Feature effects: How does a feature influence the prediction? (Accumulated local effects, partial dependence plots and individual conditional expectation curves) |
|  |  |
|  | \* Explanations for single predictions: How did the feature values of a single data point affect its prediction? (LIME and Shapley value) |
|  |  |
|  | \* Surrogate trees: Can we approximate the underlying black box model with a short decision tree? |
|  |  |
|  | \* The iml package works for any classification and regression machine learning model: random forests, linear models, neural networks, xgboost, etc. |
|  |  |
|  | This document shows you how to use the iml package to analyse machine learning models. |
|  |  |
|  |  |
|  |  |
|  | **## Data: Boston Housing** |
|  |  |
|  | We'll use the `MASS::Boston` dataset to demonstrate the abilities of the iml package. This dataset contains median house values from Boston neighbourhoods. |
|  |  |
|  | ```{r} |
|  | data("Boston", package = "MASS") |
|  | head(Boston) |
|  | ``` |
|  |  |
|  | **## Fitting the machine learning model** |
|  |  |
|  | First we train a randomForest to predict the Boston median housing value: |
|  |  |
|  | ```{r, message = FALSE} |
|  | set.seed(42) |
|  | library("iml") |
|  | library("randomForest") |
|  | data("Boston", package = "MASS") |
|  | rf <- randomForest(medv ~ ., data = Boston, ntree = 50) |
|  | ``` |
|  |  |
|  | **## Using the iml Predictor() container** |
|  |  |
|  | We create a `Predictor` object, that holds the model and the data. The iml package uses R6 classes: New objects can be created by calling `Predictor$new()`. |
|  |  |
|  | ```{r} |
|  | X <- Boston[which(names(Boston) != "medv")] |
|  | predictor <- Predictor$new(rf, data = X, y = Boston$medv) |
|  | ``` |
|  |  |
|  | **## Feature importance** |
|  |  |
|  | We can measure how important each feature was for the predictions with `FeatureImp`. The feature importance measure works by shuffling each feature and measuring how much the performance drops. For this regression task we choose to measure the loss in performance with the mean absolute error ('mae'), another choice would be the mean squared error ('mse'). |
|  |  |
|  | Once we create a new object of `FeatureImp`, the importance is automatically computed. |
|  | We can call the `plot()` function of the object or look at the results in a data.frame. |
|  |  |
|  | ```{r} |
|  | imp <- FeatureImp$new(predictor, loss = "mae") |
|  | library("ggplot2") |
|  | plot(imp) |
|  | imp$results |
|  | ``` |
|  |  |
|  | **## Feature effects** |
|  |  |
|  | Besides knowing which features were important, we are interested in how the features influence the predicted outcome. |
|  | The `FeatureEffect` class implements accumulated local effect plots, partial dependence plots and individual conditional expectation curves. |
|  | The following plot shows the accumulated local effects (ALE) for the feature 'lstat'. |
|  | ALE shows how the prediction changes locally, when the feature is varied. |
|  | The marks on the x-axis indicates the distribution of the 'lstat' feature, showing how relevant a region is for interpretation (little or no points mean that we should not over-interpret this region). |
|  |  |
|  | ```{r} |
|  | ale <- FeatureEffect$new(predictor, feature = "lstat") |
|  | ale$plot() |
|  | ``` |
|  |  |
|  | If we want to compute the partial dependence curves on another feature, we can simply reset the feature: |
|  |  |
|  | ```{r} |
|  | ale$set.feature("rm") |
|  | ale$plot() |
|  | ``` |
|  |  |
|  | **## Measure interactions** |
|  |  |
|  | We can also measure how strongly features interact with each other. |
|  | The interaction measure regards how much of the variance of $f(x)$ is explained by the interaction. |
|  | The measure is between 0 (no interaction) and 1 (= 100% of variance of $f(x)$ due to interactions). |
|  | For each feature, we measure how much they interact with any other feature: |
|  |  |
|  | ```{r} |
|  | interact <- Interaction$new(predictor) |
|  | plot(interact) |
|  | ``` |
|  |  |
|  | We can also specify a feature and measure all it's 2-way interactions with all other features: |
|  |  |
|  | ```{r} |
|  | interact <- Interaction$new(predictor, feature = "crim") |
|  | plot(interact) |
|  | ``` |
|  |  |
|  | You can also plot the feature effects for all features at once: |
|  |  |
|  | ```{r} |
|  | effs <- FeatureEffects$new(predictor) |
|  | plot(effs) |
|  | ``` |
|  |  |
|  |  |
|  |  |
|  | **## Surrogate model** |
|  | Another way to make the models more interpretable is to replace the black box with a simpler model - a decision tree. |
|  | We take the predictions of the black box model (in our case the random forest) and train a decision tree on the original features and the predicted outcome. |
|  | The plot shows the terminal nodes of the fitted tree. |
|  | The maxdepth parameter controls how deep the tree can grow and therefore how interpretable it is. |
|  |  |
|  | ```{r} |
|  | tree <- TreeSurrogate$new(predictor, maxdepth = 2) |
|  | plot(tree) |
|  | ``` |
|  |  |
|  | We can use the tree to make predictions: |
|  |  |
|  | ```{r} |
|  | head(tree$predict(Boston)) |
|  | ``` |
|  |  |
|  | **## Explain single predictions with a local model** |
|  |  |
|  | Global surrogate model can improve the understanding of the global model behaviour. |
|  | We can also fit a model locally to understand an individual prediction better. |
|  | The local model fitted by `LocalModel` is a linear regression model and the data points are weighted by how close they are to the data point for wich we want to explain the prediction. |
|  |  |
|  | ```{r} |
|  | lime.explain <- LocalModel$new(predictor, x.interest = X[1, ]) |
|  | lime.explain$results |
|  | plot(lime.explain) |
|  | ``` |
|  |  |
|  | ```{r} |
|  | lime.explain$explain(X[2, ]) |
|  | plot(lime.explain) |
|  | ``` |
|  |  |
|  | **## Explain single predictions with game theory** |
|  |  |
|  | An alternative for explaining individual predictions is a method from coalitional game theory named Shapley value. |
|  | Assume that for one data point, the feature values play a game together, in which they get the prediction as a payout. |
|  | The Shapley value tells us how to fairly distribute the payout among the feature values. |
|  |  |
|  | ```{r} |
|  | shapley <- Shapley$new(predictor, x.interest = X[1, ]) |
|  | shapley$plot() |
|  | ``` |
|  |  |
|  | We can reuse the object to explain other data points: |
|  |  |
|  | ```{r} |
|  | shapley$explain(x.interest = X[2, ]) |
|  | shapley$plot() |
|  | ``` |
|  |  |
|  | The results in data.frame form can be extracted like this: |
|  |  |
|  | ```{r} |
|  | results <- shapley$results |
|  | head(results) |
|  | ``` |

We can look at individual features, like the alcohol or pH and plot the curves:

pdp\_obj <- Partial$new(predictor, feature = "alcohol")

pdp\_obj$center(min(wine\_train$alcohol))

glimpse(pdp\_obj$results)

## Observations: 51,240

## Variables: 5

## $ alcohol <dbl> 8.400000, 8.742105, 9.084211, 9.426316, 9.768421, 10.1...

## $ .class <fct> qual\_high, qual\_high, qual\_high, qual\_high, qual\_high,...

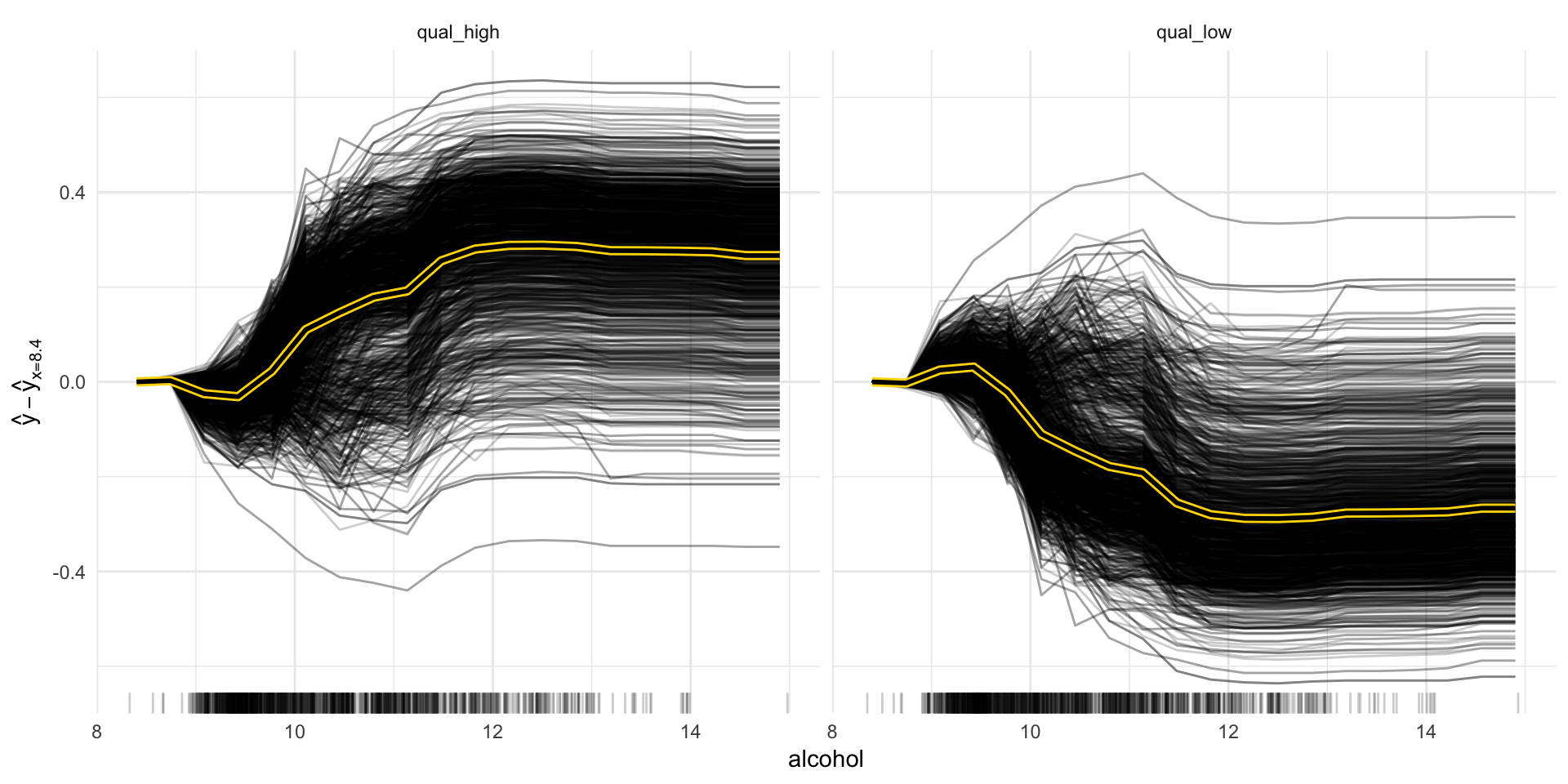
## $ .y.hat <dbl> 0.00000000, 0.00259375, -0.02496406, -0.03126250, 0.02...

## $ .type <chr> "pdp", "pdp", "pdp", "pdp", "pdp", "pdp", "pdp", "pdp"...

## $ .id <int> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA...

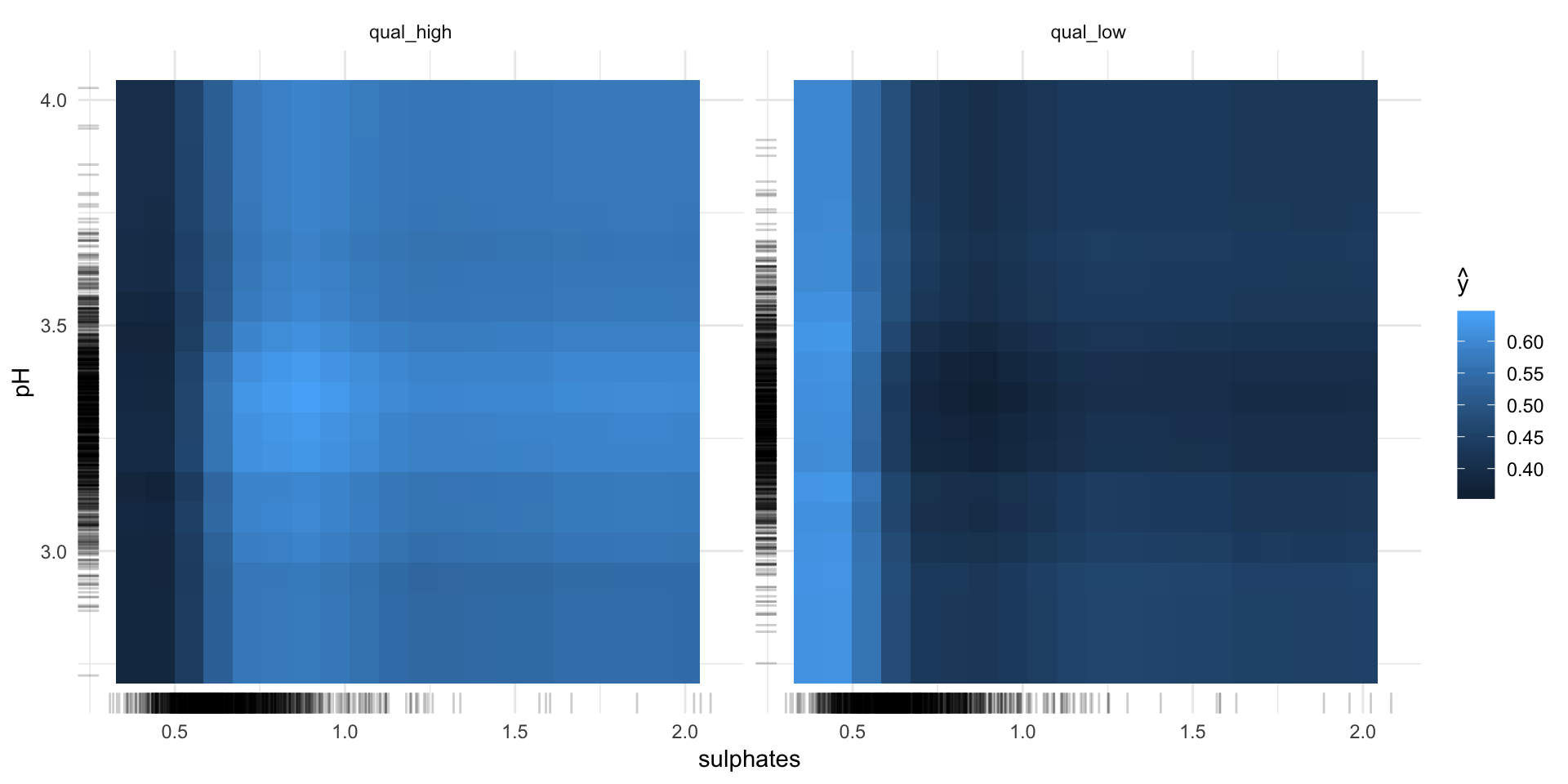
“The partial dependence plot calculates and plots the dependence of f(X) on a single or two features. It’s the aggregate of all individual conditional expectation curves, that describe how, for a single observation, the prediction changes when the feature changes.” iml help for Partial

pdp\_obj$plot()



pdp\_obj2 <- Partial$new(predictor, feature = c("sulphates", "pH"))

pdp\_obj2$plot()



**Feature interaction**

“Interactions between features are measured via the decomposition of the prediction function: If a feature j has no interaction with any other feature, the prediction function can be expressed as the sum of the partial function that depends only on j and the partial function that only depends on features other than j. If the variance of the full function is completely explained by the sum of the partial functions, there is no interaction between feature j and the other features. Any variance that is not explained can be attributed to the interaction and is used as a measure of interaction strength. The interaction strength between two features is the proportion of the variance of the 2-dimensional partial dependence function that is not explained by the sum of the two 1-dimensional partial dependence functions. The interaction measure takes on values between 0 (no interaction) to 1.” iml help for Interaction

interact <- Interaction$new(predictor, feature = "alcohol")

All of these methods have a plot argument. However, since I am writing this for an article, I want all my plots to have the same look. That’s why I’m customizing the plots I want to use in my article as shown below.

#plot(interact)

interact$results %>%

ggplot(aes(x = reorder(.feature, .interaction), y = .interaction, fill = .class)) +

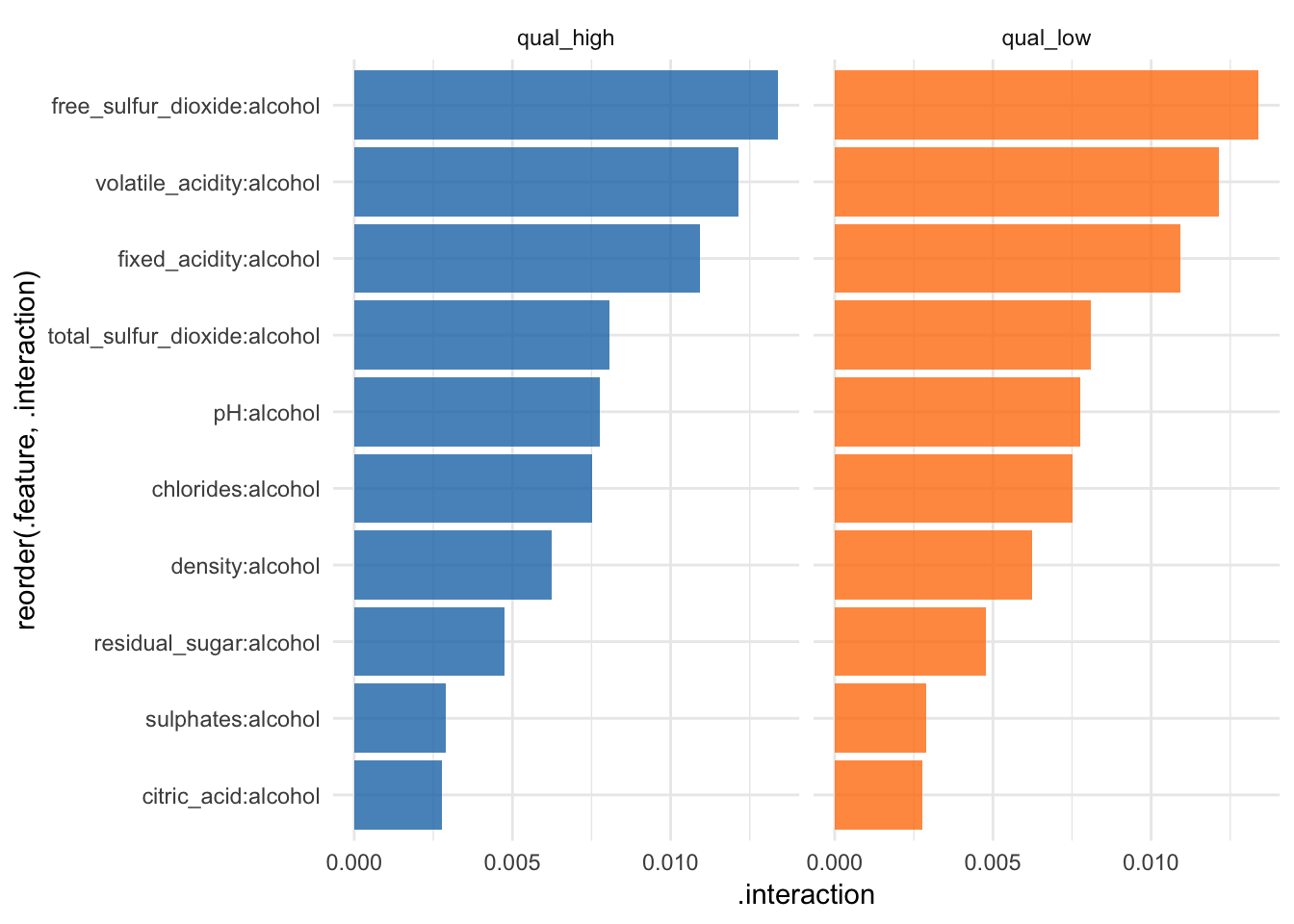
facet\_wrap(~ .class, ncol = 2) +

geom\_bar(stat = "identity", alpha = 0.8) +

scale\_fill\_tableau() +

coord\_flip() +

guides(fill = FALSE)



**Tree Surrogate**

The tree surrogate method uses decision trees on the predictions.

“A conditional inference tree is fitted on the predicted from the machine learning model and the data. The partykit package and function are used to fit the tree. By default a tree of maximum depth of 2 is fitted to improve interpretability.” iml help for TreeSurrogate

tree <- TreeSurrogate$new(predictor, maxdepth = 5)

The R^2 value gives an estimate of the goodness of fit or how well the decision tree approximates the model.

tree$r.squared

## [1] 0.4571756 0.4571756

#plot(tree)

tree$results %>%

mutate(prediction = colnames(select(., .y.hat.qual\_high, .y.hat.qual\_low))[max.col(select(., .y.hat.qual\_high, .y.hat.qual\_low),

ties.method = "first")],

prediction = ifelse(prediction == ".y.hat.qual\_low", "qual\_low", "qual\_high")) %>%

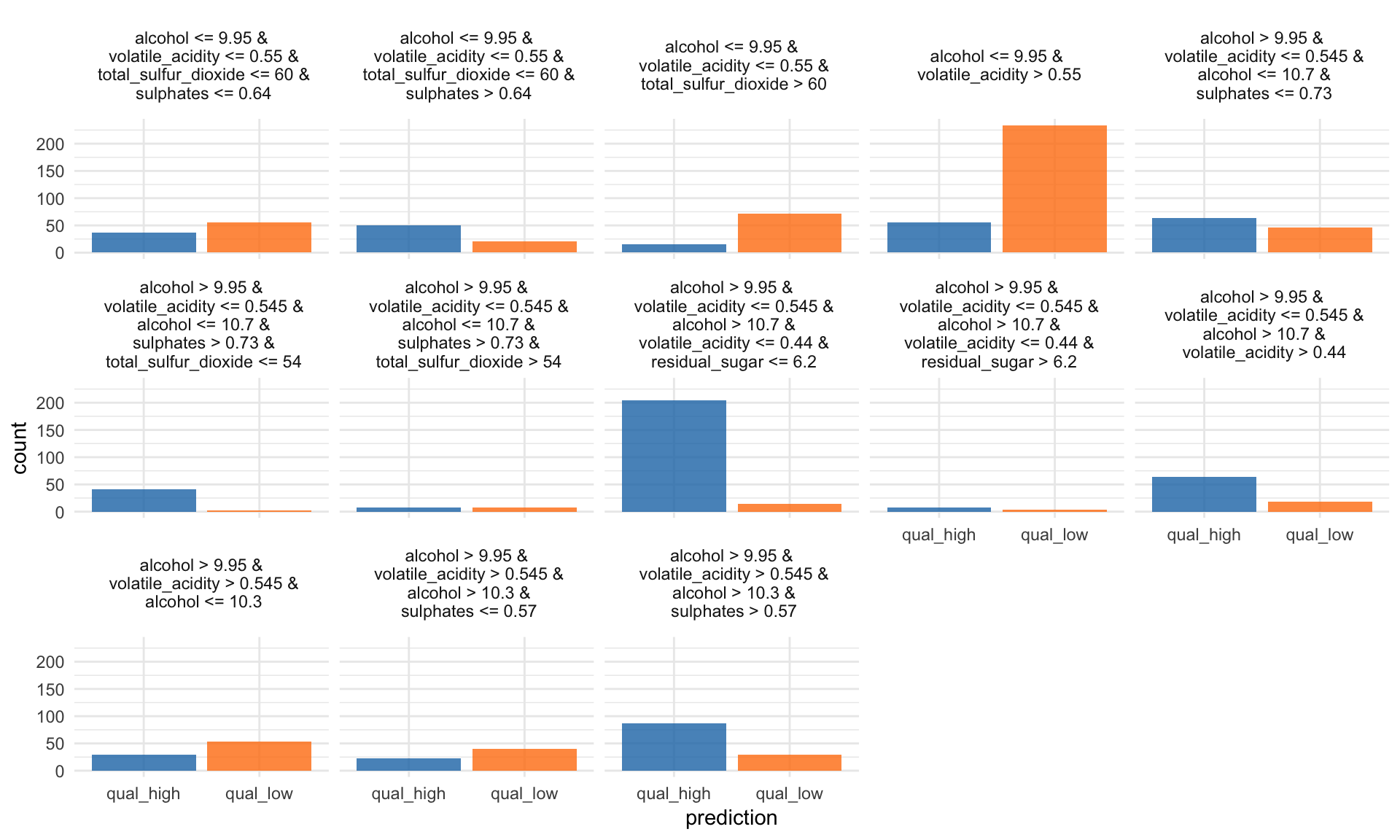
ggplot(aes(x = prediction, fill = prediction)) +

facet\_wrap(~ .path, ncol = 5) +

geom\_bar(alpha = 0.8) +

scale\_fill\_tableau() +

guides(fill = FALSE)



**LocalModel: Local Interpretable Model-agnostic Explanations**

LocalModel is a implementation of the LIME algorithm from Ribeiro et al. 2016

According to the LIME principle, we can look at individual predictions. Here, for example on the first row of the test set:

X2 <- wine\_test[, -12]

i = 1

lime\_explain <- LocalModel$new(predictor, x.interest = X2[i, ])

lime\_explain$results

## beta x.recoded effect x.original feature

## 1 -0.7653408409 0.7 -0.53573859 0.7 volatile\_acidity

## 2 -0.0006292149 34.0 -0.02139331 34 total\_sulfur\_dioxide

## 3 0.2624431667 9.4 2.46696577 9.4 alcohol

## 4 0.7653408409 0.7 0.53573859 0.7 volatile\_acidity

## 5 0.0006292149 34.0 0.02139331 34 total\_sulfur\_dioxide

## 6 -0.2624431667 9.4 -2.46696577 9.4 alcohol

## feature.value .class

## 1 volatile\_acidity=0.7 qual\_high

## 2 total\_sulfur\_dioxide=34 qual\_high

## 3 alcohol=9.4 qual\_high

## 4 volatile\_acidity=0.7 qual\_low

## 5 total\_sulfur\_dioxide=34 qual\_low

## 6 alcohol=9.4 qual\_low

#plot(lime\_explain)

p1 <- lime\_explain$results %>%

ggplot(aes(x = reorder(feature.value, -effect), y = effect, fill = .class)) +

facet\_wrap(~ .class, ncol = 2) +

geom\_bar(stat = "identity", alpha = 0.8) +

scale\_fill\_tableau() +

coord\_flip() +

labs(title = paste0("Test case #", i)) +

guides(fill = FALSE)

… or for the sixth row:

i = 6

lime\_explain$explain(X2[i, ])

p2 <- lime\_explain$results %>%

ggplot(aes(x = reorder(feature.value, -effect), y = effect, fill = .class)) +

facet\_wrap(~ .class, ncol = 2) +

geom\_bar(stat = "identity", alpha = 0.8) +

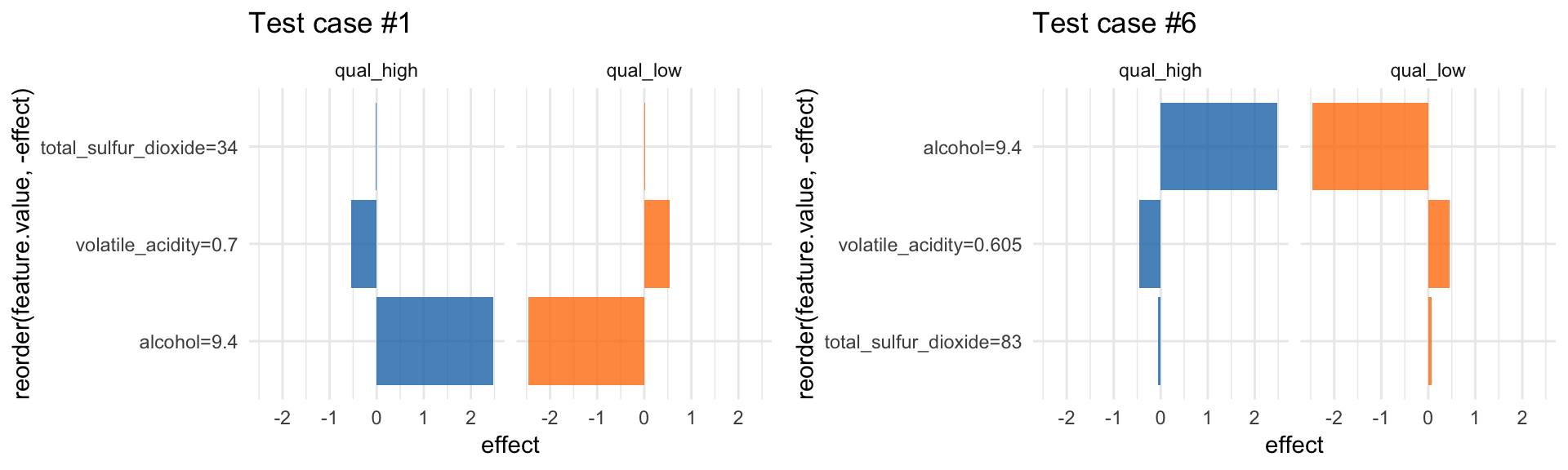
scale\_fill\_tableau() +

coord\_flip() +

labs(title = paste0("Test case #", i)) +

guides(fill = FALSE)

grid.arrange(p1, p2, ncol = 2)



**Shapley value for explaining single predictions**

Another way to interpret individual predictions is whit Shapley values.

“Shapley computes feature contributions for single predictions with the Shapley value, an approach from cooperative game theory. The features values of an instance cooperate to achieve the prediction. The Shapley value fairly distributes the difference of the instance’s prediction and the datasets average prediction among the features.” iml help for Shapley

shapley <- Shapley$new(predictor, x.interest = X2[1, ])

head(shapley$results)

## feature class phi phi.var

## 1 fixed\_acidity qual\_high -0.01100 0.003828485

## 2 volatile\_acidity qual\_high -0.16356 0.019123037

## 3 citric\_acid qual\_high -0.02318 0.005886472

## 4 residual\_sugar qual\_high -0.00950 0.001554939

## 5 chlorides qual\_high -0.01580 0.002868889

## 6 free\_sulfur\_dioxide qual\_high 0.00458 0.001250044

## feature.value

## 1 fixed\_acidity=7.4

## 2 volatile\_acidity=0.7

## 3 citric\_acid=0

## 4 residual\_sugar=1.9

## 5 chlorides=0.076

## 6 free\_sulfur\_dioxide=11

#shapley$plot()

shapley$results %>%

ggplot(aes(x = reorder(feature.value, -phi), y = phi, fill = class)) +

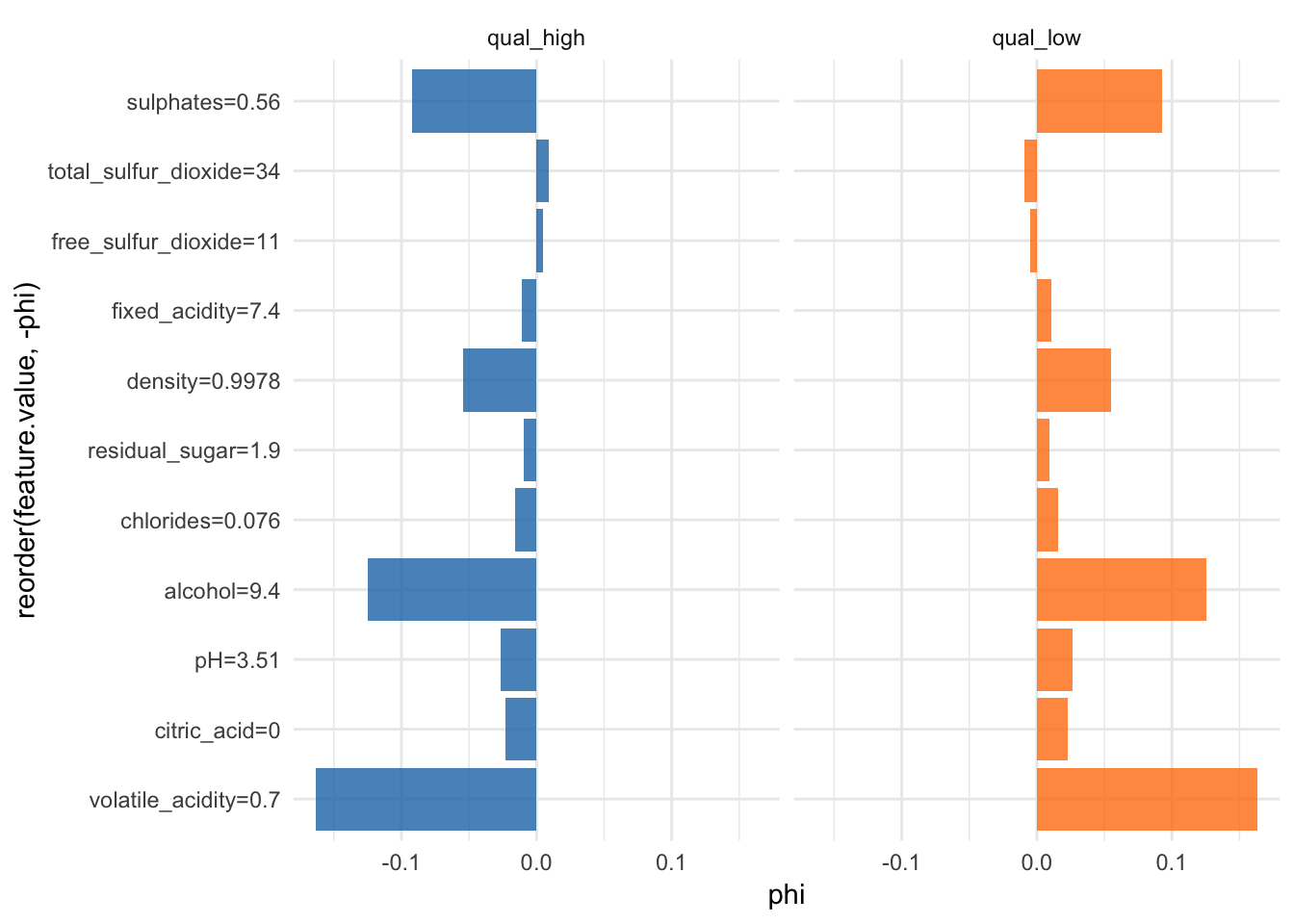
facet\_wrap(~ class, ncol = 2) +

geom\_bar(stat = "identity", alpha = 0.8) +

scale\_fill\_tableau() +

coord\_flip() +

guides(fill = FALSE)



**breakDown**

Another package worth mentioning is breakDown. It provides

“Model agnostic tool for decomposition of predictions from black boxes. Break Down Table shows contributions of every variable to a final prediction. Break Down Plot presents variable contributions in a concise graphical way. This package work for binary classifiers and general regression models.”

The broken() function decomposes model predictions and outputs the contributions of each feature to the final prediction.

library(breakdown)

predict.function <- function(model, new\_observation) {

predict(model, new\_observation, type="prob")[,2]

}

predict.function(rf\_model, X2[1, ])

## [1] 0.966

br <- broken(model = rf\_model,

new\_observation = X2[1, ],

data = X,

baseline = "Intercept",

predict.function = predict.function,

keep\_distributions = TRUE)

br

## contribution

## (Intercept) 0.000

## + alcohol = 9.4 0.138

## + volatile\_acidity = 0.7 0.097

## + sulphates = 0.56 0.060

## + density = 0.9978 0.038

## + pH = 3.51 0.012

## + chlorides = 0.076 0.017

## + citric\_acid = 0 0.026

## + fixed\_acidity = 7.4 0.048

## + residual\_sugar = 1.9 0.014

## + free\_sulfur\_dioxide = 11 0.016

## + total\_sulfur\_dioxide = 34 0.034

## final\_prognosis 0.501

## baseline: 0.4654328

The plot function shows the average predictions and the final prognosis:

#plot(br)

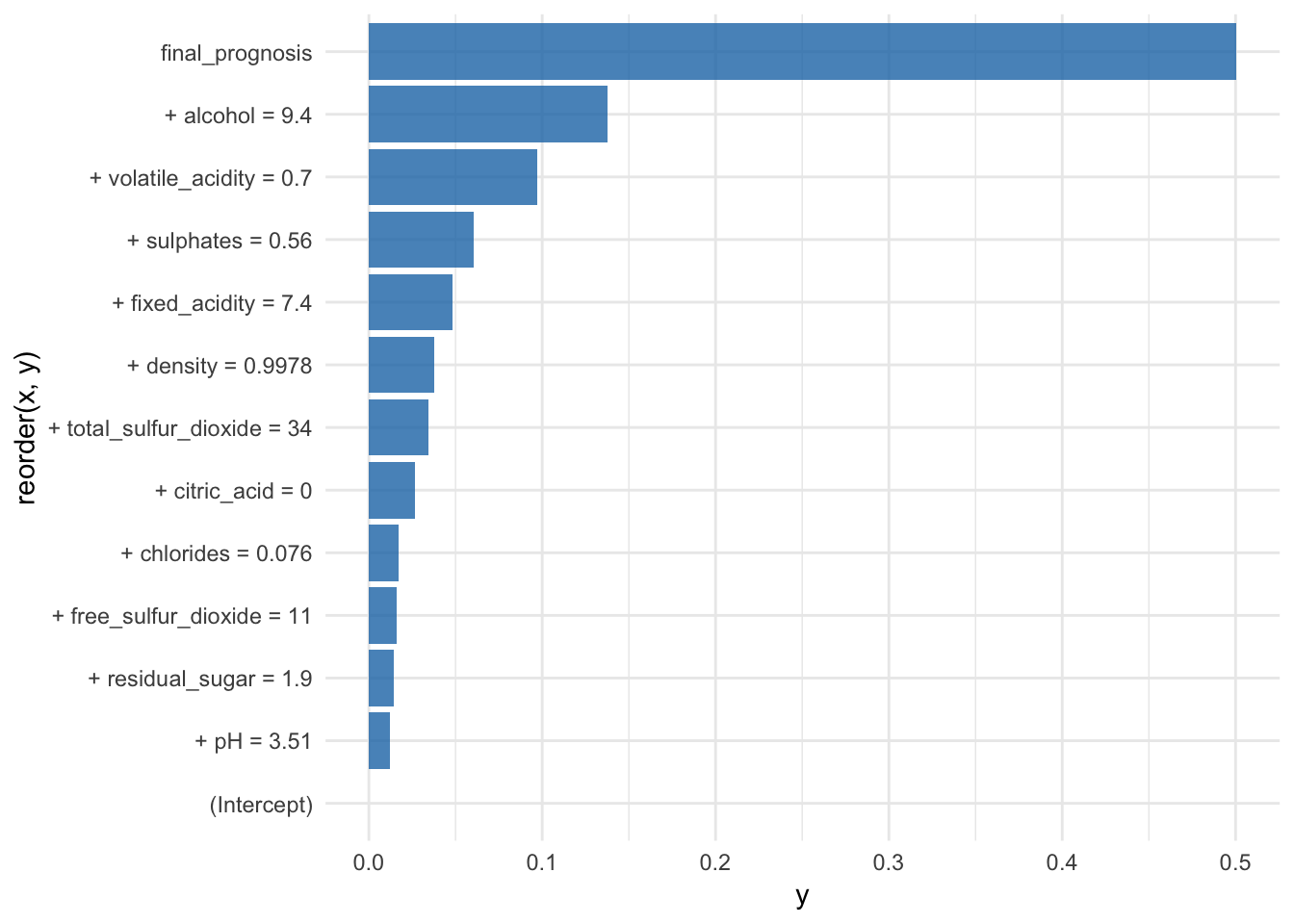
data.frame(y = br$contribution,

x = br$variable) %>%

ggplot(aes(x = reorder(x, y), y = y)) +

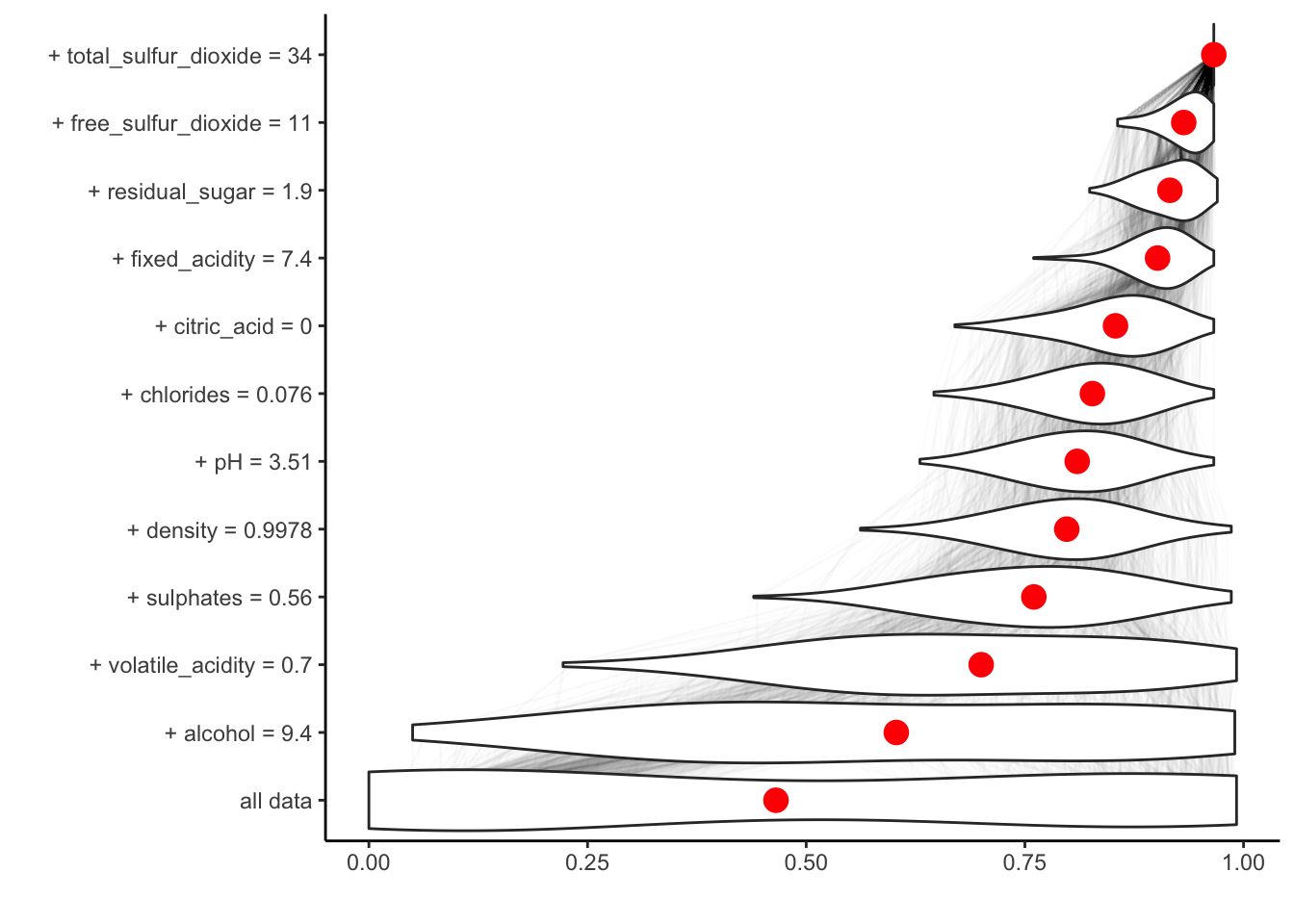
geom\_bar(stat = "identity", fill = "#1F77B4", alpha = 0.8) +

coord\_flip()



If we set keep\_distributions = TRUE, we can plot these distributions of partial predictions, as well as the average.

plot(br, plot\_distributions = TRUE)



**DALEX: Descriptive mAchine Learning EXplanations**

The third package I want to showcase is DALEX, which stands for Descriptive mAchine Learning EXplanations and contains a collection of functions that help with interpreting/explaining black-box models.

“Machine Learning (ML) models are widely used and have various applications in classification or regression. Models created with boosting, bagging, stacking or similar techniques are often used due to their high performance, but such black-box models usually lack of interpretability. DALEX package contains various explainers that help to understand the link between input variables and model output. The single\_variable() explainer extracts conditional response of a model as a function of a single selected variable. It is a wrapper over packages ‘pdp’ and ‘ALEPlot’. The single\_prediction() explainer attributes parts of a model prediction to particular variables used in the model. It is a wrapper over ‘breakDown’ package. The variable\_dropout() explainer calculates variable importance scores based on variable shuffling. All these explainers can be plotted with generic plot() function and compared across different models.”

We first create an explain object, that has the correct structure for use with the DALEX package.

p\_fun <- function(object, newdata){predict(object, newdata = newdata, type = "prob")[, 2]}

yTest <- as.numeric(wine\_test$quality)

explainer\_classif\_rf <- DALEX::explain(rf\_model, label = "rf",

data = wine\_test, y = yTest,

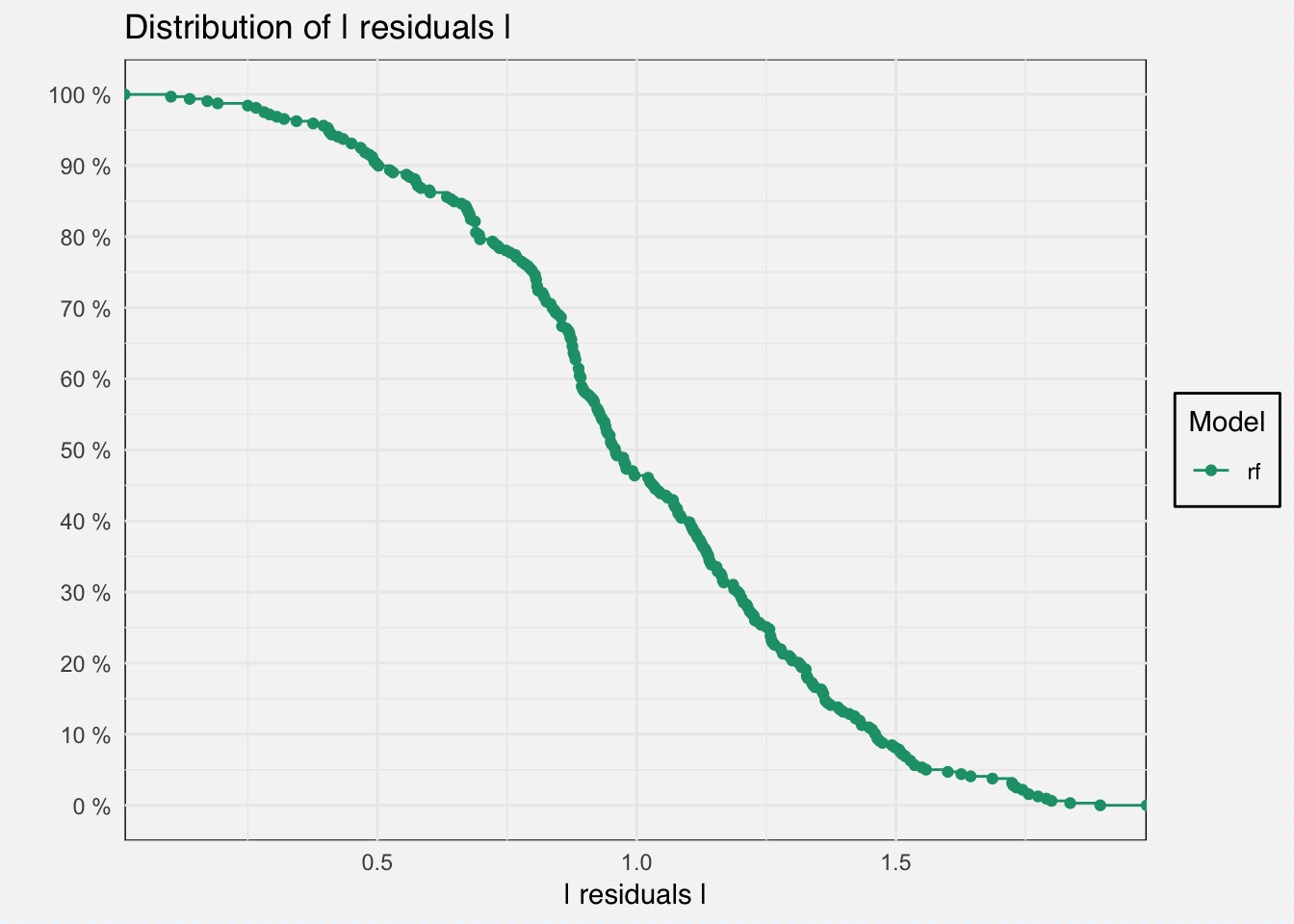
predict\_function = p\_fun)

**Model performance**

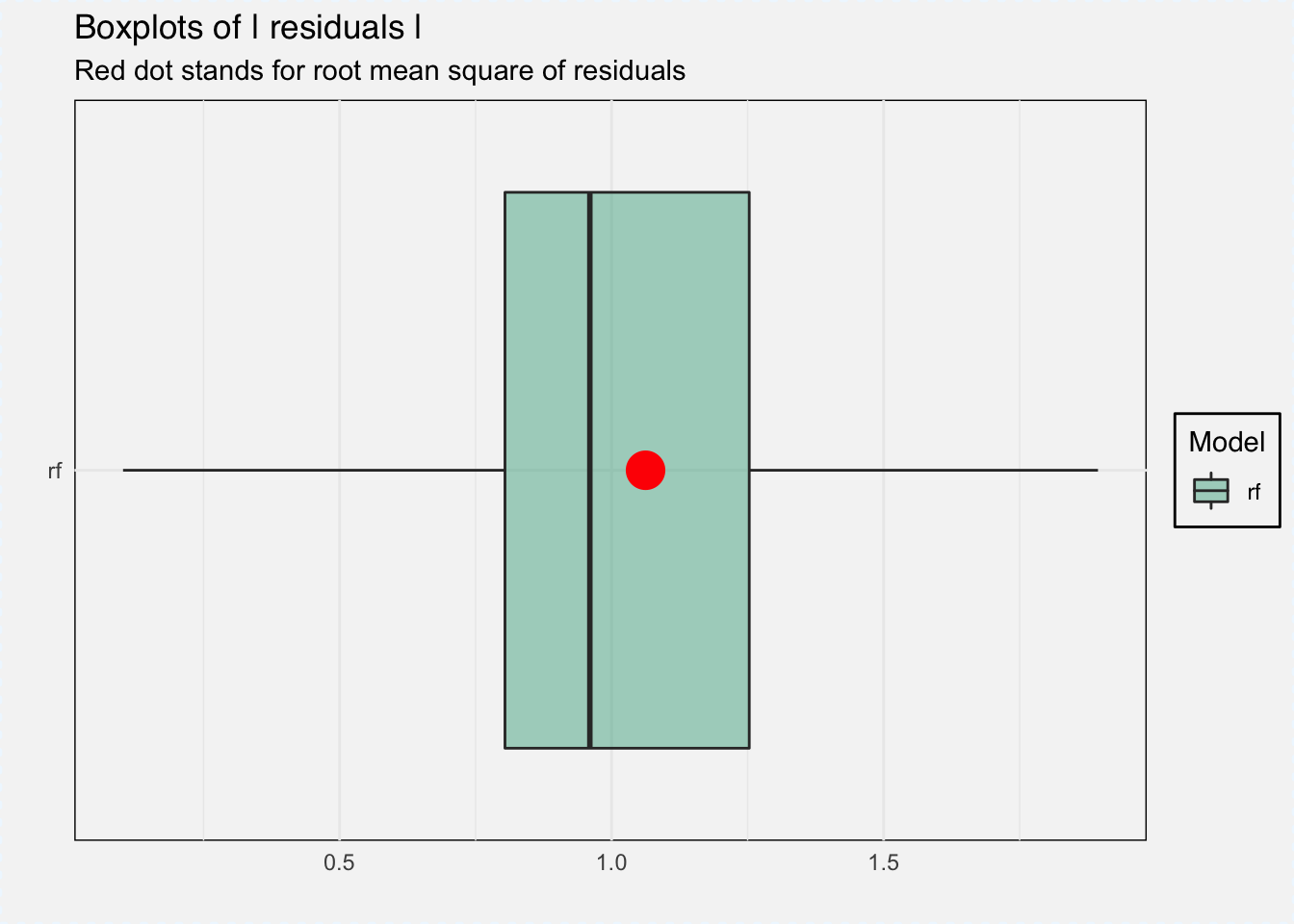
With DALEX we can do several things, for example analyze model performance as the distribution of residuals.

mp\_classif\_rf <- model\_performance(explainer\_classif\_rf)

plot(mp\_classif\_rf)



plot(mp\_classif\_rf, geom = "boxplot")

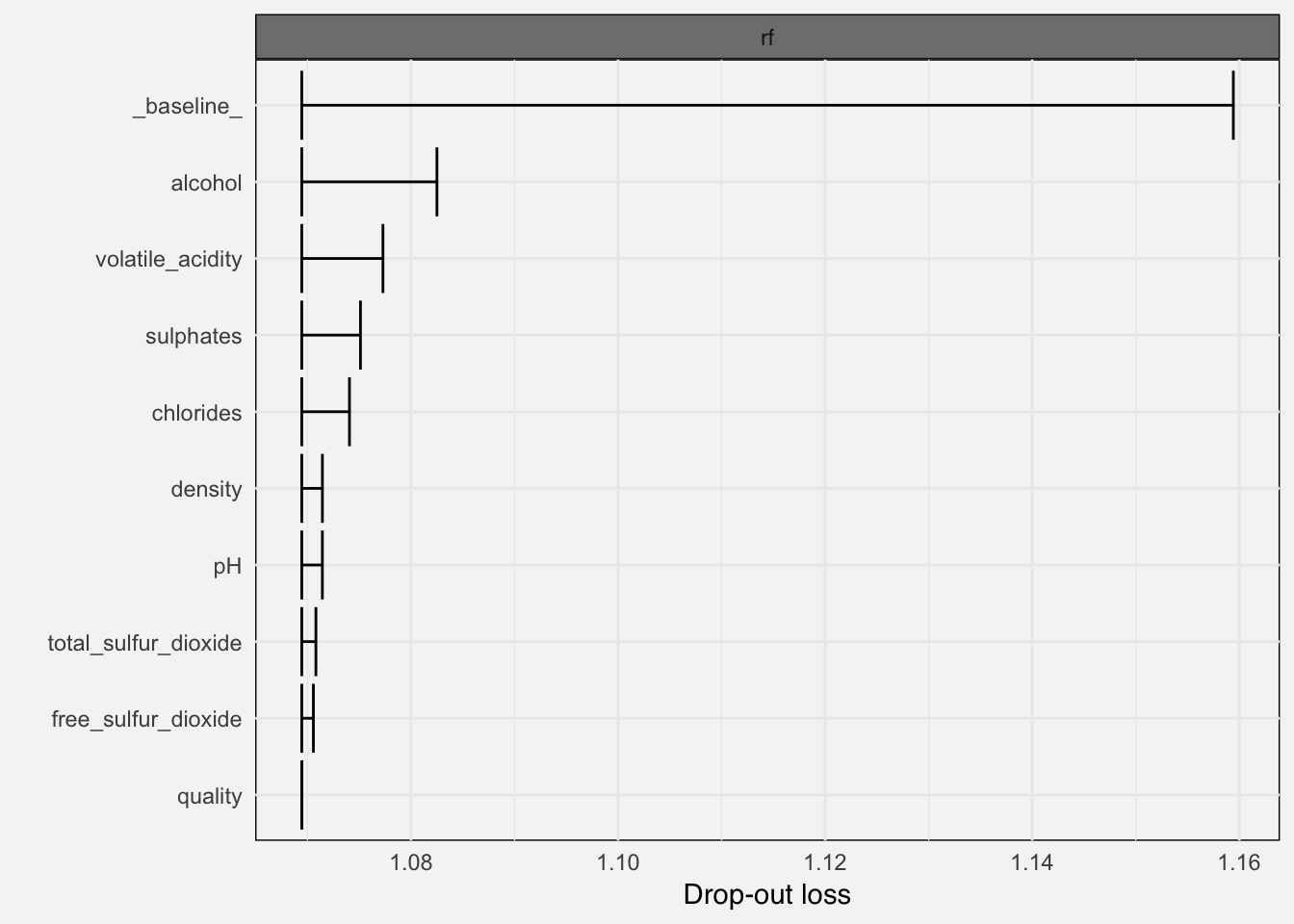


**Feature importance**

Feature importance can be measured with variable\_importance() function, which gives the loss from variable dropout.

vi\_classif\_rf <- variable\_importance(explainer\_classif\_rf, loss\_function = loss\_root\_mean\_square)

plot(vi\_classif\_rf)



**Variable response**

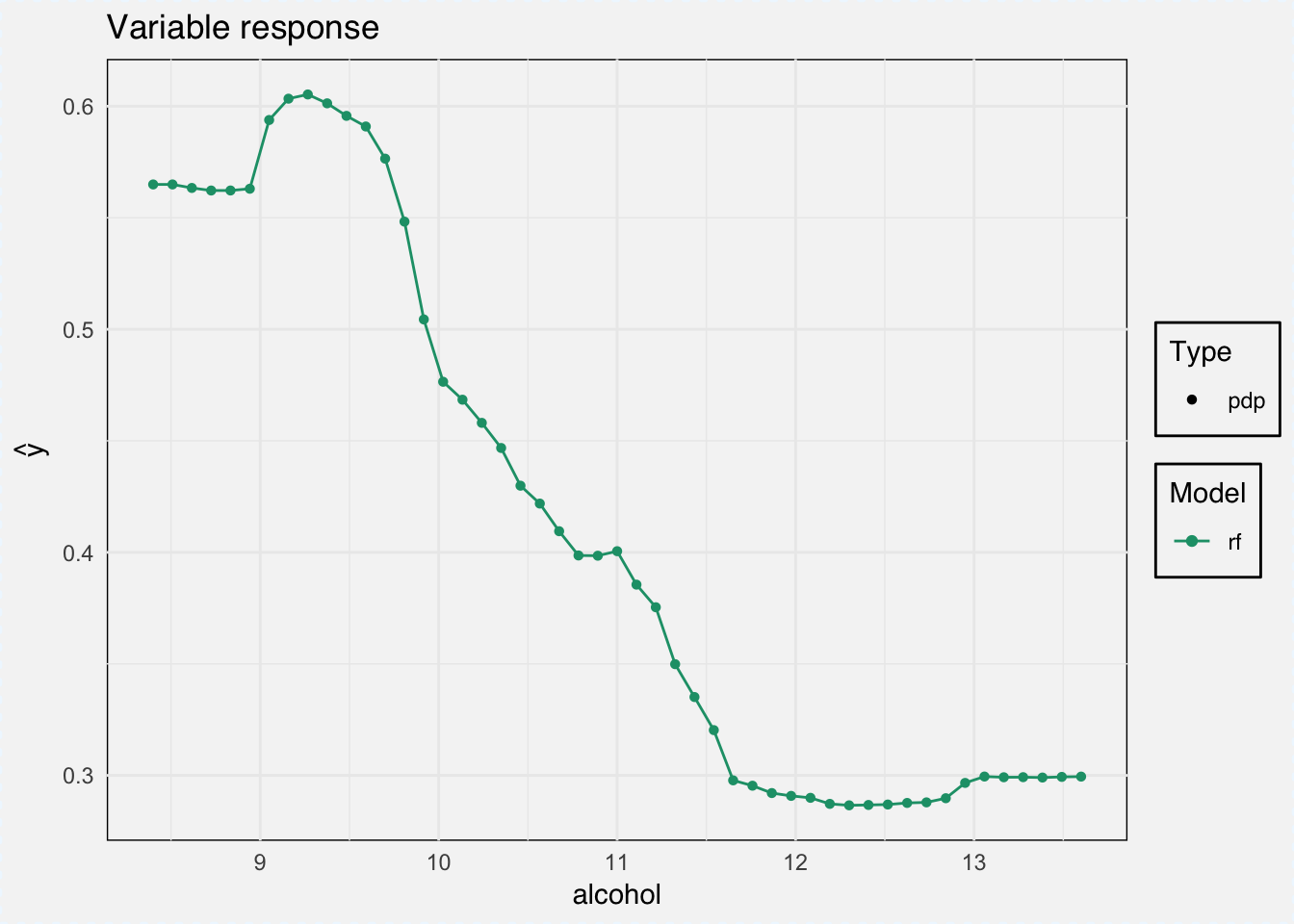
And we can calculate the marginal response for a single variable with the variable\_response() function.

“Calculates the average model response as a function of a single selected variable. Use the ‘type’ parameter to select the type of marginal response to be calculated. Currently for numeric variables we have Partial Dependency and Accumulated Local Effects implemented. Current implementation uses the ‘pdp’ package (Brandon M. Greenwell (2017). pdp: An R Package for Constructing Partial Dependence Plots. The R Journal, 9(1), 421–436.) and ‘ALEPlot’ (Dan Apley (2017). ALEPlot: Accumulated Local Effects Plots and Partial Dependence Plots.)” DALEX help for variable\_response

As type we can choose between ‘pdp’ for Partial Dependence Plots and ‘ale’ for Accumulated Local Effects.

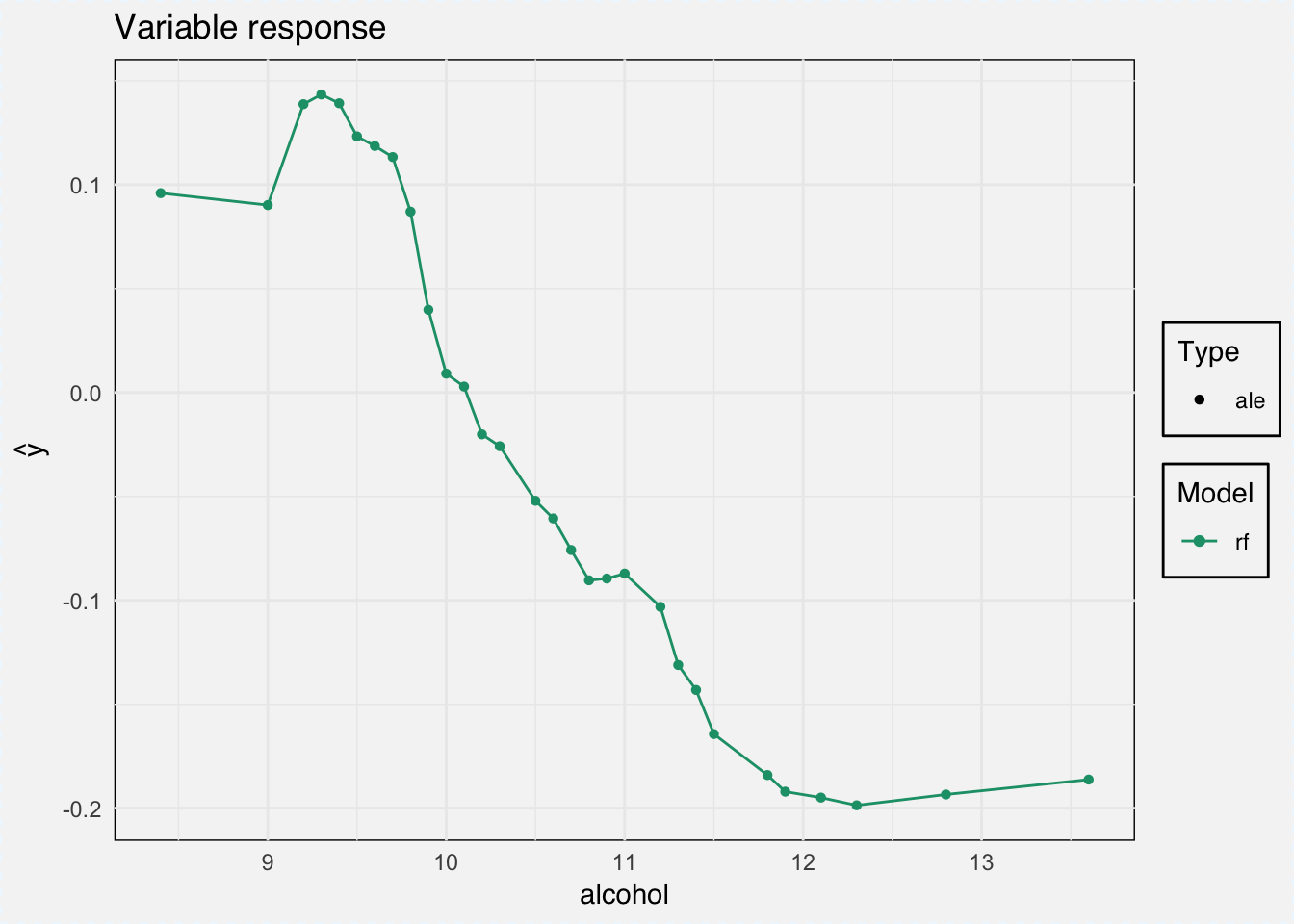
pdp\_classif\_rf <- variable\_response(explainer\_classif\_rf, variable = "alcohol", type = "pdp")

plot(pdp\_classif\_rf)



ale\_classif\_rf <- variable\_response(explainer\_classif\_rf, variable = "alcohol", type = "ale")

plot(ale\_classif\_rf)



sessionInfo()

## R version 3.5.1 (2018-07-02)

## Platform: x86\_64-apple-darwin15.6.0 (64-bit)

## Running under: macOS High Sierra 10.13.6

##

## Matrix products: default

## BLAS: /Library/Frameworks/R.framework/Versions/3.5/Resources/lib/libRblas.0.dylib

## LAPACK: /Library/Frameworks/R.framework/Versions/3.5/Resources/lib/libRlapack.dylib

##

## locale:

## [1] de\_DE.UTF-8/de\_DE.UTF-8/de\_DE.UTF-8/C/de\_DE.UTF-8/de\_DE.UTF-8

##

## attached base packages:

## [1] grid stats graphics grDevices utils datasets methods

## [8] base

##

## other attached packages:

## [1] gower\_0.1.2 glmnet\_2.0-16 foreach\_1.4.4 Matrix\_1.2-14

## [5] bindrcpp\_0.2.2 DALEX\_0.2.3 breakDown\_0.1.6 iml\_0.5.1

## [9] ggthemes\_3.5.0 ggridges\_0.5.0 gridExtra\_2.3 caret\_6.0-80

## [13] lattice\_0.20-35 forcats\_0.3.0 stringr\_1.3.1 dplyr\_0.7.6

## [17] purrr\_0.2.5 readr\_1.1.1 tidyr\_0.8.1 tibble\_1.4.2

## [21] ggplot2\_3.0.0 tidyverse\_1.2.1

##

## loaded via a namespace (and not attached):

## [1] readxl\_1.1.0 backports\_1.1.2 plyr\_1.8.4

## [4] lazyeval\_0.2.1 sp\_1.3-1 splines\_3.5.1

## [7] AlgDesign\_1.1-7.3 digest\_0.6.15 htmltools\_0.3.6

## [10] gdata\_2.18.0 magrittr\_1.5 checkmate\_1.8.5

## [13] cluster\_2.0.7-1 sfsmisc\_1.1-2 Metrics\_0.1.4

## [16] recipes\_0.1.3 modelr\_0.1.2 dimRed\_0.1.0

## [19] gmodels\_2.18.1 colorspace\_1.3-2 rvest\_0.3.2

## [22] haven\_1.1.2 xfun\_0.3 crayon\_1.3.4

## [25] jsonlite\_1.5 libcoin\_1.0-1 ALEPlot\_1.1

## [28] bindr\_0.1.1 survival\_2.42-3 iterators\_1.0.9

## [31] glue\_1.2.0 DRR\_0.0.3 gtable\_0.2.0

## [34] ipred\_0.9-6 questionr\_0.6.2 kernlab\_0.9-26

## [37] ddalpha\_1.3.4 DEoptimR\_1.0-8 abind\_1.4-5

## [40] scales\_0.5.0 mvtnorm\_1.0-8 miniUI\_0.1.1.1

## [43] Rcpp\_0.12.17 xtable\_1.8-2 spData\_0.2.9.0

## [46] magic\_1.5-8 proxy\_0.4-22 foreign\_0.8-70

## [49] spdep\_0.7-7 Formula\_1.2-3 stats4\_3.5.1

## [52] lava\_1.6.2 prodlim\_2018.04.18 httr\_1.3.1

## [55] yaImpute\_1.0-29 RColorBrewer\_1.1-2 pkgconfig\_2.0.1

## [58] nnet\_7.3-12 deldir\_0.1-15 labeling\_0.3

## [61] tidyselect\_0.2.4 rlang\_0.2.1 reshape2\_1.4.3

## [64] later\_0.7.3 munsell\_0.5.0 cellranger\_1.1.0

## [67] tools\_3.5.1 cli\_1.0.0 factorMerger\_0.3.6

## [70] pls\_2.6-0 broom\_0.4.5 evaluate\_0.10.1

## [73] geometry\_0.3-6 yaml\_2.1.19 ModelMetrics\_1.1.0

## [76] knitr\_1.20 robustbase\_0.93-1 pdp\_0.6.0

## [79] randomForest\_4.6-14 nlme\_3.1-137 mime\_0.5

## [82] RcppRoll\_0.3.0 xml2\_1.2.0 compiler\_3.5.1

## [85] rstudioapi\_0.7 e1071\_1.6-8 klaR\_0.6-14

## [88] stringi\_1.2.3 highr\_0.7 blogdown\_0.6

## [91] psych\_1.8.4 pillar\_1.2.3 LearnBayes\_2.15.1

## [94] combinat\_0.0-8 data.table\_1.11.4 httpuv\_1.4.4.2

## [97] agricolae\_1.2-8 R6\_2.2.2 bookdown\_0.7

## [100] promises\_1.0.1 codetools\_0.2-15 boot\_1.3-20

## [103] MASS\_7.3-50 gtools\_3.8.1 assertthat\_0.2.0

## [106] CVST\_0.2-2 rprojroot\_1.3-2 withr\_2.1.2

## [109] mnormt\_1.5-5 expm\_0.999-2 parallel\_3.5.1

## [112] hms\_0.4.2 rpart\_4.1-13 timeDate\_3043.102

## [115] coda\_0.19-1 class\_7.3-14 rmarkdown\_1.10

## [118] inum\_1.0-0 ggpubr\_0.1.7 partykit\_1.2-2

## [121] shiny\_1.1.0 lubridate\_1.7.4