University of Warsaw
Faculty of Mathematics, Informatics and Mechanics

Aleksandra Paluszyńska
Student no. 320255

Structure mining and knowledge extraction from random forest with applications to The Cancer Genome Atlas project

Master's thesis
in MATHEMATICS
in the field of APPLIED MATHEMATICS

Supervisor:
dr hab. Przemysław Biecek
Instytut Matematyki Stosowanej i Mechaniki

July 2017

Supervisor's statement

Hereby I confirm that the presented thesis was prepared under my supervision and that it fulfils the requirements for the degree of Master of Mathematics.

Date Supervisor's signature

Author's statement

Hereby I declare that the presented thesis was prepared by me and none of its contents was obtained by means that are against the law.

The thesis has never before been a subject of any procedure of obtaining an academic degree.

Moreover, I declare that the present version of the thesis is identical to the attached electronic version.

Date Author's signature

Abstract

The thesis discusses various approaches to interpreting black boxes, i.e. predictive models with extremely complicated structure. In particular, I consider random forests that often produce accurate predictions, which are not easily explained in terms of relative importance of input variables.

As information on how the explanatory variables influence prediction is often critical in applications, I introduce a new R package, randomForestExplainer, that calculates new and existing measures of variable importance in random forests. More importantly, the package proposes new ways of visualizing relative importance of variables and provides a wrap-up function that summarizes a given random forest in various ways.

Finally, I demonstrate the usage of the package in an analysis of the Cancer Genome Atlas data concerning glioblastoma cancer. My data set has many more variables than observations, which leads to shallow trees in the forest and makes the assessment of variable importance particularly hard. Thus, using this example I show how the package works in such problematic settings.

Keywords

random forests, structure mining, visualization, variable importance, feature importance

Thesis domain (Socrates-Erasmus subject area codes)

11.2 Statistics
11.3 Informatics, Computer Science

Subject classification

68 Computer Science
68Q Theory of Computing
68Q32 Computational learning theory
62 Statistics
62P Applications
62P10 Applications to biology and medical sciences

Tytuł pracy w języku polskim

Analiza struktury i ekstrakcja wiedzy z lasów losowych z przykşadami zastosowañ dla danych The Cancer Genome Atlas

Contents

Introduction

In recent years the use of predictive models, which predict an outcome using a set of inputs, has become more widespread than ever. Abundance of data and ample computing power create opportunities for the emergence of new fields that rely on predictive modelling, such as speech or image recognition, which are inherently different from the established ones and may require different kinds of models. Traditionally the focus of statistical modelling was to understand the data-generating process so predictive ability of the model was a sort of useful byproduct of the analysis.

Nowadays, many applications primarily call for accurate predictions due to the following: first, financial rewards may be driven by prediction accuracy (e.g., accurate facial recognition boosts productivity of a security firm) and second, the nature of the data-generating process may be too complex or not interesting enough to be analyzed as such (e.g., how the structure of pixels in an image shapes the probability of a face appearing in it).

Whenever a statistical model is used for prediction, a different sort of model assessment is necessary than when building it in order to explain some natural phenomenon. Similarly, whenever a model built for the sole purpose of prediction is to be used in the decision-making process, there is a pressing need of explaining how the prediction is made so that it can be trusted. If the model in question is a black box, i.e. it is so complex that there is no natural way of interpreting it, an explanation has to be provided by what we call structure mining and knowledge extraction.

In this thesis I focus on random forests, one class of predictive black boxes that is recognized as capable of accurate prediction using relatively little computing power in comparison to other methods (e.g., gradient boosting [5]). As a committee of decision trees, each built on a bootstrap sample of the data, random forests do not provide any natural way of measuring which variables drive the prediction. To address this issue I introduce a new R package `randomForestExplainer` that aims at explaining a forest in terms of distinguishing the variables that are essential to its performance.

The package provides a few functions that calculate a set of measures of variable importance including the ones already in use and some new ones such as the number of times a predictor was used to split the root node of a tree in the forest. Most importantly, `randomForestExplainer` offers a wide variety of possibilities for visualizing the forest that help to understand its structure and assess importance of variables. Finally, it includes a wrapper function that utilizes all capabilities of the package for a given random forest and produces a HTML report summarizing the results. This allows the user to familiarize herself with the explanatory insight that the package offers for her forest without getting to know the details about how to produce each plot or data frame.

I demonstrate the capabilities of `randomForestExplainer` for a forest predicting whether a patient survives a year from diagnosis of glioblastoma cancer. The data come from The Cancer Genome Atlas Project [18] and contain 16117 predictors for 125 observations – such a high number of variables relative to observations leads to shallow trees in the forest and this

in turn makes the assessment of variable importance particularly hard. As such problems are recently ever more common in practice, the point of using this data set is to show how the new package manages them.

The thesis is structured as follows: in Chapter 1 I discuss predictive models in general, present general methods of structure mining, formally define random forests and existing methods of measuring variable importance in them. Then, in Chapter 2 I present how each `randomForestExplainer` function works, what are its parameters and how it should be used. Finally, Chapter 3 contains a step-by-step application of the new package to the glioblastoma data set.

Chapter 1

Structure mining of predictive models

1.1. Predictive models

In this section we first define our problem as either a regression or classification task and introduce its mathematical structure. Then, we present two main approaches to solving the problem: through a data model that requires certain assumptions concerning the data-generating process or with an algorithmic model that treats this process as a black box.

1.1.1. The problem

Lets assume that we observe pairs $(y, \mathbf{x})$, where y will be called the *response variable* and $\mathbf{x}$ is a vector of length p of predictors. Moreover, we assume that y is a realization of random variable Y, while $\mathbf{x}$ is a realization of random variable $\mathbf{X}$. We face the following problem: given observed inputs $\mathbf{x}$ we want to predict the expected value of Y. Usually, we assume that Y depends on $\mathbf{X}$ and the expected value of Y can be expressed as follows:

$$\mathbb{E}(Y|\mathbf{X} = \mathbf{x}) = f(\mathbf{x}), \tag{1.1}$$

where f is a function $f : \mathbb{R}^p \to \mathcal{Y}$ that we call a *predictive model*. Moreover, if $\mathcal{Y} = \mathbb{R}$ (i.e., the response is measured on a quantitative scale and is thus called *quantitative*) we call f a *regression model* and if $\mathcal{Y} = \{G_1, G_2, \ldots, G_K\}$ is a set of K possible groups to which the observation belongs, we use the term *classification model*. Fortunately, in our case this distinction will not matter much, as the models we discuss work equally well under both settings.

Throughout the paper we assume that we observe an n-element training sample

$$\mathbf{Z} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\}. \tag{1.2}$$

Using this knowledge we want to predict y for a given vector of values $\mathbf{x}$; we denote the prediction as $\hat{y}(\mathbf{x})$. For simplicity, we will use the shorter notation $\hat{y}$.

1.1.2. Data models

Until recently the main approach to predicting $\hat{y}$ from a set of inputs was to assume an underlying data model. Breiman [3] refers to this as the data modelling culture in which one assumes that the data are independently drawn from some distribution whose expectation can be denoted as

$$\mathbb{E}(Y|\mathbf{X} = \mathbf{x}) = f(\mathbf{x}, \boldsymbol{\beta}), \tag{1.3}$$

where f is a known function and $\boldsymbol{\beta}$ is the vector of unknown parameters. Under such assumptions one can estimate the unknown parameters using the available data.

In [15] Schmueli refers to this approach as *explanatory modelling* as it is often used for testing the validity of causal theoretical models in which the variables in $\mathbf{X}$ cause the effect measured by Y. It is worth noting that this is most appropriate in settings where the underlying theory is important, sometimes to such an extent that it is given as a justification for the causal interpretation of the model. Furthermore, as Schmueli points out, this focus on theory is usually present at all stages of explanatory modelling including the choice of independent variables, model selection and validation. This leads to the conclusion that data models are indeed most useful when explaining a phenomenon is the focus and not necessarily when the sole purpose of modelling is prediction (this point is often made in the literature, most notably by Breiman [3]). More importantly, if the assumed model is not an accurate approximation of the natural processes it may lead to wrong conclusions.

To give an example, *generalized linear models* are a class of widely-used data models that can be represented by the following equation:

$$\mathbb{E}(Y|\mathbf{X} = \mathbf{x}) = g^{-1}(\mathbf{x}\boldsymbol{\beta}), \tag{1.4}$$

where g is usually referred to as the *link function*. The simplest examples of link functions are: the identity function for linear regression, the logistic function for logistic regression and the natural logarithm for Poisson regression. Regardless of the link function chosen, the parameters $\boldsymbol{\beta} = (\beta_0, \beta_1, \ldots, \beta_p)^T$, of which the first one is a constant and the others correspond to p independent variables, can be directly interpreted as measures of relative importance of those variables in the Y-generating stochastic process. Moreover, using classical statistics we can test various hypotheses concerning these parameters (e.g., whether the true parameter is different than zero) and with some additional assumptions we can interpret the estimated relationship as causal.

1.1.3. Algorithmic models

In contrast to the data modelling culture, the algorithmic modelling culture (as called by Breiman in [3]) focuses on predictive accuracy of models. Although it also takes input variables $\mathbf{X}$ and tries to find a function $f(\mathbf{X})$ that best predicts Y it rejects the idea that we can restrict the search to some class of functions as in (1.4). The core assumption here is that the data-generating process is complex and unknown so, instead of approximating it, we use an algorithm that based on available data imitates natural outcomes most accurately. The resulting model is usually not interpretable due to a very complicated structure and is thus called a *black box*.

The focus of algorithmic models on prediction makes them a valuable tool whenever the mechanism of the data-generating process is not of main interest or there is no theory supporting an explanatory model. The first scenario often appears in the industry, whenever decisions rely on accurate predictions of the future (e.g., weather forecast) and not on understanding what drives them. The latter scenario is ever more common in emerging fields such as speech, image and handwriting recognition [3]. In general, algorithmic models offer greater predictive accuracy than simple explanatory models as they are designed for the sole purpose of prediction.

Algorithmic models also offer a solution to two major problems with data models: the multiplicity of good models and the curse of dimensionality [3]. The first of them concerns the fact that even a slight perturbation of the data or an alteration of the set of models considered (e.g., with or without some variable) may lead to major changes in the final model, regardless

of whether it is a data or an algorithmic one. One solution is to average over a set of competing models in order to stabilize the result – this is often introduced as an inherent part of the model-building process of algorithmic models and is usually a minor complication of an inherently complex black box[1].

Finally, data models cannot usually include too many explanatory variables (e.g., no more than the number of data points in the case of linear regression), so the dimension of the analyzed data set often has to be reduced prior to modelling, which leads to loss of information. Conversely, in algorithmic modelling it is usually the case that only computation power limits the numer of input variables and nowadays this is not too big of a restriction. Clearly, this not only improves the aforementioned predictive accuracy but also eliminates the problem of variable selection as all variables and even many of their transformations can be included in the black box. Breiman [3] calls this the *blessing of dimensionality* in contrast to the curse of dimensionality inherent to explanatory modelling. As in this paper our main focus is prediction accuracy and our motivating example is the high-dimensional problem of predicting whether a patient will die of cancer from The Cancer Genome Project data, from now on we will only consider algorithmic models.

1.2. Structure mining

We already discussed the fact that having a black box tuned to our problem (of either the regression or classification type) gives us a good chance of predicting the desired outcome variable accurately. However, we cannot justify the prediction in any way, e.g. we have no way of knowing which input variables drive it, as our model has an inherently complicated, uninterpretable structure. This may be a big problem in applications, where such predictions must be at least partially explained to the decision-makers so they will trust them and, as a result, use them.

To address this issue, we need to analyze the structure of our black box and in this section we present some existing methods of doing this. We call it *structure mining* as it aims at discovering patterns and extracting knowledge from the vast and complex structure of the black box. Sánchez, Rocktäschel, Riedel and Singh in [14] list three main approaches to this: *pedagogical*, which analyzes the behavior of the model, *decompositional*, which aims at dividing the model into simpler parts to inspect each of them in turn and *eclectic* – a mix of the two. In this section we focus on pedagogical methods, as they usually do not require any additional assumptions about the black box and we would like to start our discussion of black boxes from a general perspective.

1.2.1. Local Interpretable Model-agnostic Explanations

The paper [13] by Riberio, Singh and Guerstin introduces a technique called *Local Interpretable Model-agnostic Explanations* (LIME). Its main idea is to approximate the black box $f(\mathbf{x})$ by an interpretable model $\xi(\mathbf{x})$ in the small neighborhood of the actual prediction, as measured by $\pi_{\mathbf{x}}(\mathbf{z})$ – the structure of the black box is not explicitly considered. We present the method following [13] with slight modifications due to notation differences.

Let us define $\pi_{\mathbf{x}}(\mathbf{z})$ as the proximity measure between the input values for our observation $\mathbf{x}$ and $\mathbf{z}$, which defines locality around $\mathbf{x}$. Also, let $\xi : \mathbb{R}^p \to \mathcal{Y}$ be a model such that $\xi \in \mathfrak{H}$

[1] It is worth noting that this can also be done in the case of data models, but usually complicates the interpretation. Also, data models are generally sensitive to inclusion/exclusion of some explanatory variables and there is no simple way in determining a set of models (different in this respect) over which to average.

where $\mathfrak{H}$ is a class of potentially interpretable models in contrast to our uninterpretable black box f. We denote the measure of complexity of ξ as $\Omega(\xi)$ – if it is too high, ξ may be hard to interpret (e.g., linear regression with too many parameters). Finally, let $\mathcal{L}(f, \xi, \pi_{\mathbf{x}})$ measure how unfaithfully ξ approximates f locally, as defined by $\pi_{\mathbf{x}}$.

LIME aims at producing an *explanation* $\xi(\mathbf{x})$ of the black box f, which is both interpretable by humans (i.e., $\Omega(\xi)$ is sufficiently low) and locally faithful (i.e., gives similar results as the black box in the vicinity of a given observation). This is obtained by the following:

$$\xi(\mathbf{x}, f) = \arg\min_{h \in \mathfrak{H}} \{\mathcal{L}(f, h, \pi_{\mathbf{x}}) + \Omega(h)\}. \tag{1.5}$$

Note that this gives many possibilities of proceeding, depending on what class of interpretable models we consider, etc. As in [13] we focus on sparse linear models and use perturbation of inputs to search for them.

In order to learn the behavior of our black box under varying inputs we approximate $\mathcal{L}(f, \xi, \pi_{\mathbf{x}})$ by optimizing it on a perturbed sample drawn from the neighbourhood of $\mathbf{x}$. In [13] the authors propose the following procedure: draw the number of elements of the perturbed sample K with equal probabilities from the set $\{1, 2, \ldots, n\}$. Then, uniformly draw K elements of our sample $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ to form the perturbed sample $\{\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_K\}$, so for all $i \in \{1, 2, \ldots, K\}$ there exists a $j \in \{1, 2, \ldots, n\}$ such that $\mathbf{z}_i = \mathbf{x}_j$. After repeating this procedure r times we obtain a set of perturbed samples $\mathcal{Z}$ so we can optimize the equation (1.5) over this set to get $\xi(\mathbf{x})$.

For example, we might define $\mathfrak{H}$ as linear models, i.e. $\xi(\mathbf{z}) = \mathbf{z}^T\boldsymbol{\beta}$, and use locally weighted square loss function to measure the accuracy of approximation:

$$\mathcal{L}(f, \mathbf{z}^T\boldsymbol{\beta}, \pi_{\mathbf{x}}) = \sum_{\mathbf{z} \in \mathcal{Z}} \pi_{\mathbf{x}}(\mathbf{z})[f(\mathbf{z}) - \mathbf{z}^T\boldsymbol{\beta}]^2, \quad \pi_{\mathbf{x}}(\mathbf{z}) = \exp\left\{-\frac{\operatorname{dist}(\mathbf{x}, \mathbf{z})^2}{\sigma^2}\right\}, \tag{1.6}$$

where $\pi_{\mathbf{x}}$ is the exponential kernel with width σ, defined on some distance function dist(). To ensure that the explanation is interpretable we may restrict the number of non-zero entries of $\boldsymbol{\beta}$ to M, i.e. set $\Omega(\xi) = \|\boldsymbol{\beta}\|_0$ and add an additional restriction that $\Omega(\xi) \leq M$.

1.2.2. Visualization of the model

Wickham, Cook and Hofman recognize visualization as a powerful tool for explaining statistical models and propose three main strategies of doing so (see [19]): visualizing the model in the data space, analyzing a collection of models instead of a single one and exploring the model-fitting process. It is worth noting that, apart from approaching the issue of structure mining without any assumptions concerning the model as was the case for LIME, Wickham, Cook and Hofman focus on visual presentation of the model, to make it appealing to a broader audience and enhance its understanding. We summarize their main points in [19] by addressing each of the aforementioned strategies in turn.

When using visual model descriptions we usually want to explain either what the model looks like or how well it fits the data. In the first case we consider changes to the model caused by those of its parameters or changes to parameters caused by those in the data. When it comes to exploring model fitness, we can compare the shapes of the model and data, identify the regions of good or bad fit and ones in which it can be improved. All this leads to our better understanding of the underlying process and improves out ability to meaningfully criticize the model.

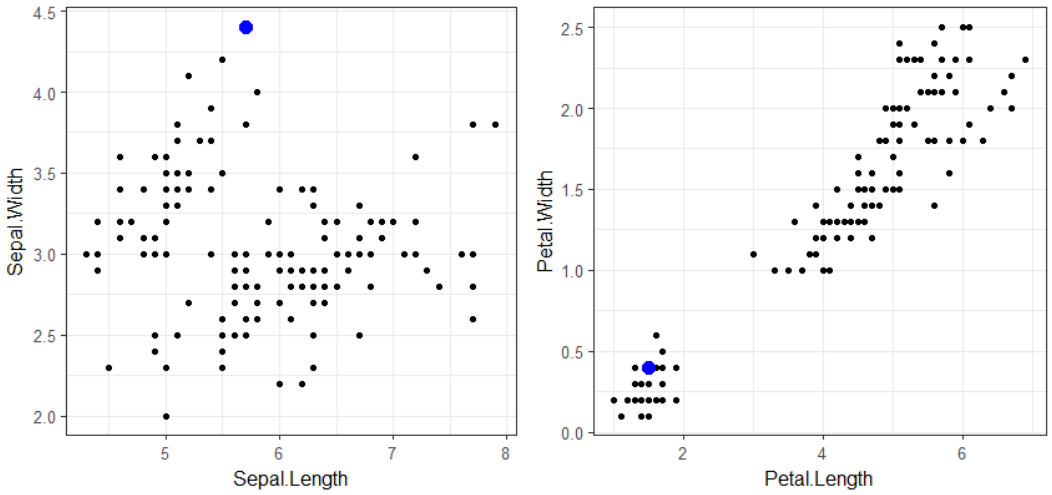


Figure 1.1: Illustration of linked brushing for the `iris` data. In each plot the dimensions correspond to a pair of quantitative variables from the data set. A seemingly outlying observation in the first plot is highlighted on both of them leading to a conclusion that it is atypical only in one of the four dimensions presented. [Source: own research.]

Model in the data space

The first strategy of model visualization concerns displaying the model in the data space – this is not as easy as showing data in the model space (e.g., plotting predicted vs. observed values) as the data space typically has many more dimensions than the model space. Therefore, the following techniques of visualizing high-dimensional objects come in handy:

1. *The grand tour* facilitates looking at many projections of the data: having our p-dimensional object we randomly and smoothly rotate between its different 2-dimensional projections until we see every possible view (or decide to stop). This is useful for identifying outliers or clusters of the data and may be also modified in such a way, that subsequent projections are selected based on some index of their interestingness (it is then called *the guided tour*).

2. *Linked brushing* facilitates looking at many sections of the data: we use a brush to color a selected set of observations in a series of plots, each showing the data from a different perspective (see Figure 1.1). The most useful version of this is an interactive one in which the user selects an observation in any of the plots leading to it being highlighted in all of them.

Equipped with these methods we can easily display our model in the data space by sampling from this space and computing a prediction for each sample. Then, we plot these samples using either the grand tour or linked brushing with the color or size of the points corresponding to the predicted value.

Collections of models

The second strategy stems from the idea that a single model is not as informative as collections of them, whereas a common approach is to select the best model from such a collection and ignore the rest. However, some of the discarded models may explain the data almost as well

as the best one but lead to substantially different interpretations, presumably valuable when we try to understand the underlying process. To take this into account, one should visualize the entire collection of models.

Unfortunately, we are usually able to visualize only a few models from the collection in their entirety, so it is essential to somehow summarize them – this is most easily done with descriptive statistics that report model quality. Usually, these are specific to the class of models considered (e.g., R^2, information criteria or degrees of freedom for regression models), but there are more universal ones like various summaries of residuals or, in the case of parameter estimation, statistics describing the distribution of estimates across models. In any case, if we have l models, n observations and p variables, we can think of descriptive statistics on five different levels [19]:

- model level: l observations, each corresponding to one model, e.g. prediction accuracy,
- model-estimate level: $l \times p$ observations, e.g. coefficient estimates,
- estimate level: p observations, e.g. summary of estimates across models,
- model-observation level: $l \times n$ observations, e.g. influence measures,
- observation level: n observations, e.g. average of residuals across models.

Having calculated descriptive statistics, we can use linked brushing for visualization in order to connect statistics from different levels across plots.

The model-fitting process

Finally, the third visualization strategy aims at exploring the model-fitting process whenever possible, instead of only looking at the final result. Wickham, Cook and Hofman note that this may be particularly valuable if the algorithm used for model-fitting is iterative, because then we observe how the consecutive versions of the model improve using their natural ordering. When it comes to visualization we can think of two main approaches exploiting this ordering: time series plots (i.e., time on the x-axis and a summary statistic on the y-axis) or movies depicting the evolution of the model (i.e., many static plots capturing different iterations of the model stringed together). Moreover, we could fit the model multiple times using different starting points for the algorithm to explore optima reached, as many algorithms only guarantee convergence to a local optimum (see Figure 1.2).

In summary, visualization is an important tool not only in data exploration but also in the process of building statistical models. Therefore, in this paper we aim at explaining black boxes with a strong emphasis on visualization.

1.3. Random forests

In this section we focus our attention on one class of predictive black boxes: random forests. We start by presenting their construction and motivation behind it to then discuss their properties in more detail. Finally, we address the issue of structure mining in this context by presenting various measures of importance of explanatory variables included in the model. Throughout the section we rely heavily on the work of Hastie, Tibshirani and Friedman [5] and occasionally refer to the original work of Breiman [2] that laid the foundations of random forests.

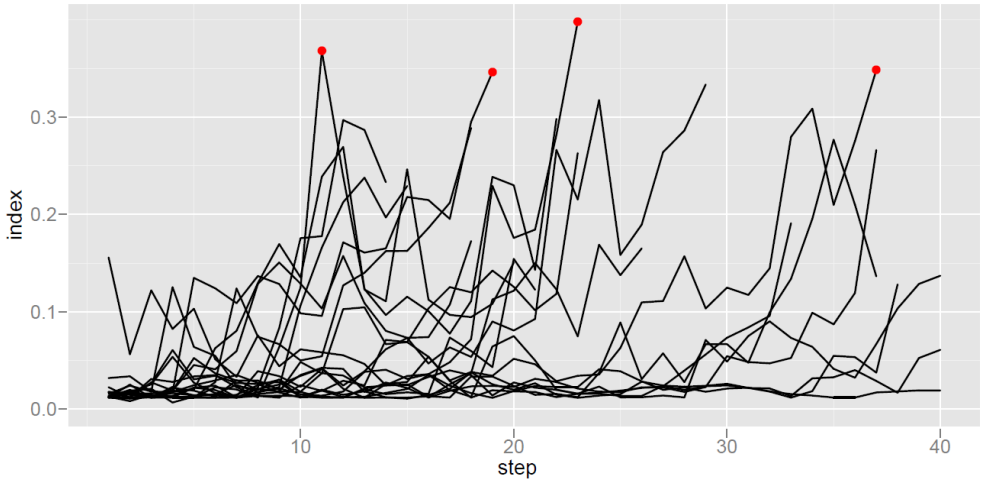


Figure 1.2: The model-fitting process of simulated annealing with 20 random starts. The quality of the model in any particular time is measured by a clumpiness index that is high when a division of the data into distinct groups emerges. Red points indicate four highest values. [Source: Wickham, Cook and Hofman [19], Figure 14.]

1.3.1. Construction of a random forest

Random forest is a collection of trees, each built on a different bootstrap sample of the data, from which an aggregate prediction is calculated [2]. This definition makes it natural to present the construction of a random forest by first discussing its individual elements, decision trees, and then considering their whole collection.

Decision trees

First, we need to introduce terminology connected to decision trees. In mathematics a *tree* is defined as an undirected, acyclic and connected graph or, equivalently, an undirected graph in which any two vertices are connected by exactly one path [10]. In the context of trees the vertices are called *nodes* and edges – *branches*. When it comes to decision trees it is convenient to use directed trees so that they can be depicted in such a way that on top we have a unique starting node, referred to as the *root*, which has only outgoing branches so that consecutive branches and nodes are below it. If a branch starts at a node the latter is called a *parent* and the node to which the branch points is called a *daughter*. Nodes without daughters are referred to as *leaves*.

A *decision tree* is a model that partitions the p-dimensional space generated by predictors into L hypercubes R_l for $l = 1, 2, \ldots, L$ called *decision regions* and fits a simple model of Y (such as a constant) on each of them separately. Thus, the model has the following form[2]:

$$f(\mathbf{x}) = \sum_{l=1}^{L} c_l \mathbb{1}(\mathbf{x} \in R_l), \tag{1.7}$$

where c_l is some constant, $\bigcup_{l=1}^{L} R_l = \mathbb{R}^p$ and $R_l \cap R_k = \varnothing$ for all $l \neq k$. This model can be

[2]For a multi-class classification problem the notation is slightly different – instead of $\mathbb{E}(Y|\mathbf{X}) = f(\mathbf{X})$ we write $\mathbb{E}(Y = k|\mathbf{X}) = f_k(\mathbf{X})$ for $k = 1, 2, \ldots, K$ corresponding to the K possible values of Y so in (1.7) the constant is indexed over both l and k.

characterized by a tree: each node that is not a leaf corresponds to a hyperplane perpendicular to the axis of one of the predictors that divides the hypercube in two. In other words, at every non-terminal node one predictor is chosen according to which the remaining subset of observations is split (at the root we split the whole training sample). This concept is illustrated in Figure 1.3.

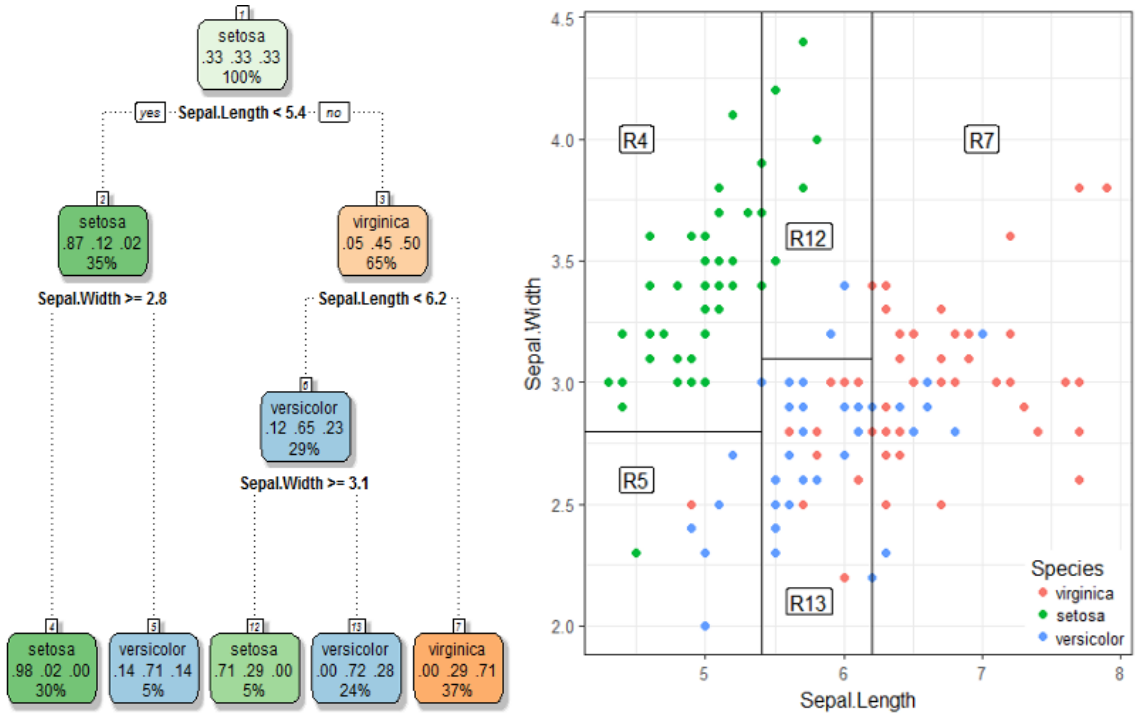


Figure 1.3: Decision tree (left) and corresponding decision regions (right) for a 3-class classification problem on the `iris` data. The tree is meant to predict the Species of iris flowers using their sepal length and sepal width (both measured in centimeters). The tree has 9 nodes – 5 of them are leaves corresponding to decision regions. Each node is marked by: the name of the majority class, fractions that quantify the distribution of classes in the rectangle split by the node and the percentage of all observations contained in this rectangle. Labels of regions correspond to labels of nodes and, since not only leaves are labeled, these are not consecutive integers. [Source: own research.]

It is worth noting that we only consider binary partitions that correspond to trees such that at most two branches can start at any of the nodes. This is because splitting each node into more than two groups at each stage fragments the data too quickly and is thus not a good general strategy [5]. Also, as a series of binary splits can lead to the same partitioning as a multiway split, the latter can be de facto present if really needed.

We discussed how a decision tree looks like and next we will explain how to *grow it*, i.e. how to fit it to the data. To do this we follow the CART (*classification and regression trees*) methodology for growing trees. First, assume that we already have a partition of the predictor space into L regions $R_1, R_2, \ldots, R_L$, each corresponding to one of the terminal nodes of our tree. In the case of regression we find the constants in equation (1.7) by minimizing the sum of squares $\sum(y_i - \hat{y}_i)^2$, with the prediction given by $f(\mathbf{X})$. Naturally, in region R_l the best estimate of $\hat{y}_i$ is just the average of y_i:

$$\hat{c}_l = \frac{1}{N_l} \sum_{i:\mathbf{x}_i \in R_l} y_i, \tag{1.8}$$

where $N_l = |\{i : \mathbf{x}_i \in R_l\}|$ is the number of observations in region R_l. In the case of classification in order to minimize the number of misclassified observations, we classify all observations from region R_l to the majority class $k(l) = \arg\max_k \hat{p}_{lk}$, where $\hat{p}_{lk}$ is the estimate of $\mathbb{E}(Y = k|\mathbf{X})$, i.e. the proportion of observations in leave l that belong to class k:

$$\hat{p}_{lk} = \frac{1}{N_l} \sum_{i:\mathbf{x}_i \in R_l} \mathbb{1}(y_i = k). \tag{1.9}$$

Now we return to the question of how to find the best binary partition of the predictor space. We start at the root (with all the data) and consider splitting it on variable X_j at the split point s which defines a pair of half-planes corresponding to the left and right daughter of the root[3]:

$$R_L(j, s) = \{\mathbf{X} : X_j \leq s\}, \quad R_R(j, s) = \{\mathbf{X} : X_j > s\}. \tag{1.10}$$

We pick X_j and s such that they solve:

$$\min_{j,s} \left[\min_{c_L} N_L Q_L(T) + \min_{c_R} N_R Q_R(T) \right], \tag{1.11}$$

where $Q_l(T)$ measures the *impurity* of node l and is weighted by the number of observations in this node. In general, $Q_l(T)$ should measure how different are observations in node l as we are interested in finding such a split that each node contains observations as similar as possible so predicting the same value of response for them as in equation (1.7) will be accurate. In practice different measures of node impurity can be used and we need to select one suitable to our problem. For regression one usually uses the sum of squares:

$$Q_l(T) = \frac{1}{N_l} \sum_{i:\mathbf{x}_i \in R_l} (y_i - \hat{c}_l)^2. \tag{1.12}$$

As a result, the inner minimization problems given in (1.11) are solved by averages of the response in both of the nodes as discussed before. The outer minimization can then be easily solved as follows: we scan through X_j for $j = 1, 2, \ldots, p$ and for each variable find the optimal split s to then determine the best pair (j, s). After finding the best split at the root we repeat the above procedure to its daughters using their subsets of our sample and so on.

In the case of classification one usually uses one of the following node impurity measures:

- misclassification error: $\frac{1}{N_l} \sum_{i:\mathbf{x}_i \in R_l} \mathbb{1}(y_i \neq k(l)) = 1 - \hat{p}_{lk(l)}$,
- Gini index: $\sum_{k \neq k'} \hat{p}_{lk} \hat{p}_{lk'} = \sum_{k=1}^{K} \hat{p}_{lk}(1 - \hat{p}_{lk})$,
- cross-entropy or deviance: $-\sum_{k=1}^{K} \hat{p}_{lk} \log \hat{p}_{lk}$.

The above measures are all similar except for the fact that the misclassification error is not differentiable so may be problematic when it comes to numerical optimization. In the end, the inner optimization problems are solved by (1.9) and to solve (1.11) we proceed as in the regression case.

[3]Note that this approach works only when the values of predictor X_j have a natural order, i.e. are either quantitative or qualitative and ordinal. Otherwise, inequalities with X_j are meaningless so we define the half-planes in (1.10) in the following way: $R_1(j, C) = \{\mathbf{X} : X_j \in C\}$ and $R_2(j, C) = \{\mathbf{X} : X_j \notin C\}$ for $C \subset \mathcal{C}$, where $\mathcal{C}$ is the set of values taken by X_j. Of course, there are $2^{|\mathcal{C}|-1} - 1$ possible partitions of the set $\mathcal{C}$ into C and C^c so the computations become prohibitive for large $|\mathcal{C}|$.

A committee of trees

The main advantage of decision trees is that they can capture complex nonlinear relations found in the data due to their hierarchical structure – the split of a set of observations in one node is conditional on all splits occurring before, which is equivalent to modelling composite interactions. If grown sufficiently deep (i.e., such that each region of the predictor space contains observations with the same or very similar values of y_i) trees produce estimates with low bias but high variance, as the structure of a tree is very sensitive to minor changes in the data [5].

Consequently, decision trees are natural candidates for averaging that could reduce variance and at the same time keep bias low. *Bagging* or *bootstrap aggregation* is designed to do just that: after drawing B bootstrap samples $\mathbf{Z}^{*b}$ of our training data (i.e., samples of the same size as our data that are drawn uniformly with replacement) we grow a tree f^{*b} on each of them and aggregate the result by taking the average in the case of regression and deciding by a majority vote in the case of classification. As all trees are identically distributed the expectation of any one of them is the same as the expectation of their average, so bagging does not increase bias. The remaining question is whether it reduces the variance.

If the prediction of tree f^{*b} is a random variable T_b and $T_1, T_2, \ldots, T_B$ are identically distributed with variance σ^2 and the correlation between any two of them $\rho := \mathrm{Cov}(T_i, T_j)/\sigma^2$ (for $i \neq j$) is positive, the variance of their average is equal to:

$$\begin{aligned}
\mathrm{Var}\left(\frac{1}{B}\sum_{b=1}^{B} T_b\right) &= \frac{1}{B^2}\left(B\,\mathrm{Var}(T_1) + \sum_{i\neq j}\mathrm{Cov}(T_i, T_j)\right) \\
&= \frac{1}{B^2}\left(B\sigma^2 + \sum_{i=1}^{B}\sum_{j\neq i}^{B}\rho\sqrt{\mathrm{Var}(T_i)\,\mathrm{Var}(T_j)}\right) \\
&= \frac{1}{B^2}\left(B\sigma^2 + B(B-1)\rho\sigma^2\right) = \rho\sigma^2 + \frac{1-\rho}{B}\sigma^2. \qquad (1.13)
\end{aligned}$$

Thus, even as B goes to infinity and the second term disappears the variance of our estimator is determined by the size of correlation between bagged trees. Therefore, further variance reduction can be achieved by reducing the correlation between trees in such a way that this is not offset by the increase in their individual variance. This idea is implemented in random forests that use the following modification of bagging decision trees: when finding the optimal split at a given node we only consider a random subset of predictors. We can summarize the algorithm of growing a random forest as follows[4]:

1. Draw a bootstrap sample $\mathbf{Z}^{*b}$ of size n from the training sample $\mathbf{Z}$.
2. Grow a decision tree T_b on the bootstrap sample by repeating the following steps for each node starting at the root until the terminal nodes contain observations with as similar values of y_i as possible:
   (a) Draw independently and uniformly r out of p predictors, where $r \ll p$.
   (b) For each selected variable find the best split point.
   (c) Split the node into two daughter nodes using such variable and its optimal split that impurity of new nodes is lowest.

[4]It is worth noting that although in this paper we only consider regression and classification problems, random forests are also used in survival analysis where they are called *random survival forests* [9].

3. Repeat steps 1. and 2. for $b = 1, 2, \ldots, B$ producing the ensemble of tees $\{T_b\}_{b=1}^{B}$.

4. Using predictions of individual trees predict the response for a new observation $\mathbf{x}$:

   Regression: $\hat{y} = \frac{1}{B}\sum_{b=1}^{B} T_b(\mathbf{x})$,
   Classification: $\hat{y} = \text{majority vote}\{T_b(\mathbf{x})\}_{b=1}^{B}$.

In practice, random forests to some extent reduce the variance of noisy estimates obtained by decision trees while keeping the bias relatively low. Even more importantly, they are very simple to use as they only require two tuning parameters r and B – the choice of the latter does not change much as long as it is sufficiently big (see section 1.3.2). The other one is usually set by default to $\lfloor p/3 \rfloor$ for regression and $\lfloor \sqrt{p} \rfloor$ for classification [5].

1.3.2. Analysis using random forests

Summary statistics

Due to their structure random forests provide summary statistics at three levels [19]:

1. *Tree-level:* as each tree is grown on its own bootstrap sample $\mathbf{Z}^{*b}$ it has its own test set called the *out-of-bag* (OOB) sample[5] composed of observations that were not selected into $\mathbf{Z}^{*b}$. Using the OOB sample for each tree we can compute an unbiased estimate of our prediction error that is almost identical to the one obtained by n-fold cross-validation [5] so this additional procedure is no longer necessary. Also, we can grow the trees of our forest until this error stabilizes meaning that B is sufficiently big.

2. *Variable-level:* we can asses the importance of each predictor by randomly permuting it and observing the drop in accuracy of prediction of the forest.

3. *Observation-level:* for each observation the forest produces a distribution of predictions across all its trees so that we can identify observations for which prediction was unusually hard or easy as measured by how many trees predicted the response correctly.

These statistics make random forests interesting subjects of structure mining as one can come up with dedicated techniques of knowledge extraction exploiting the structure of a forest and not just rely on the general approaches presented in section 1.2 – the "box" is not that "black" after all. In particular, we will discuss the variable-level summary statistics in greater detail in section 1.3.3.

Missing data

When it comes to missing data, random forests (and decision trees in general) offer two approaches in addition to the usual ones of either discarding observations that have missing values or filling in those values (e.g., with the mean for quantitative, median for qualitative and ordinal, and mode for multinomial variables). The first approach is to create a separate category "missing value" so this information can be used for splitting. Of course, this is only

[5] Observe that on average around one third of the sample is not selected to the bootstrap sample [10]. This stems from a simple calculation: we independently draw n observations from an n-element sample and if n is large enough we use the following approximation:

$$\mathbb{P}((\mathbf{x}_i, y_i) \notin \mathbf{Z}^{*b}) = \left(1 - \frac{1}{n}\right)^n \xrightarrow{n\to\infty} \frac{1}{e} \approx 0.368.$$

possible for qualitative variables but its combination with data imputation could be used for quantitative ones as well: we could fill in the missing values using the mean and create a separate binary variable that points to the observations that had those values missing and include it as an additional predictor.

The second approach, distinctive for tree-based models, is to use *surrogate splits* whenever a value of the split variable is missing for an observation. Consider a split of variable X_j at the value s_j – if the value of X_j is missing for an observation i we choose another predictor $X_{j'}$ and its split point $s_{j'}$ such that this new (surrogate) split best mimics the original one. Note that we can form an ordered list of surrogate splits for each node in case more predictor values are missing for some observations. To sum up, this approach exploits correlation between predictors to make up for missing data so it will work best when this correlation is indeed high.

Imbalanced data

Another issue that may arise when using random forests concerns classification problems on *imbalanced data* meaning that at least one of the categories of the response variable is observed only for a small number of observations. This is quite common in applications and is often accompanied by the need of correct classification of the rare event (e.g., predicting default or fraud) which may not even appear in many bootstrap samples to which we fit decision trees when building a forest so we may observe a bias in prediction towards more-common classes. Moreover, the algorithm treats all misclassifications the same, whereas in such cases we are often mainly interested in correct classification of the rare category.

In [4] Chen, Liaw and Breiman propose two ways of dealing with imbalanced data:

1. *Weighted random forest* (WRF) places a heavier penalty on misclassifying the minority class – we assign a weight to each class (a larger one is given to the rare one) to then use them to weight the Gini criterion used for finding splits and finally to weight the shares of each class at terminal nodes to determine prediction of a tree.

2. *Balanced random forest* (BRF) modifies the way of forming the bootstrap sample in the following way: a bootstrap sample is drawn from the minority case and then the same number of cases is drawn from the majority case. This is an implementation of such *down-sampling* (i.e., reducing the size of the majority class in the bootstrap sample) that no information is lost as all observations from the majority class can appear in the bootstrap sample.

The authors corroborate the usefulness of the above methods with experiments on various data sets and both performed better than most other methods considered but none was found to be unequivocally superior [16].

1.3.3. Importance of variables in a forest

In this paper we aim at explaining the structure of a random forest with a particular emphasis on the importance of variables. We recognize the fact that in applications it is often critical to point to the variables that drive our prediction (otherwise it could be discarded as untrustworthy, see detailed discussion in [13]). To rise to this challenge we first review existing approaches of measuring variable importance in a random forest.

Perturbation of predictors

The first approach to assessing importance of predictors in a random forest was proposed by Breiman in his paper [2] introducing the method and is still widely used today. It adopts the idea that if a variable plays a role in predicting our response, then perturbing it on the OOB sample should decrease prediction accuracy of our forest on this sample. Therefore, taking each variable in turn, one can perturb its values and calculate the resulting average decrease of predictive accuracy of the trees – such a measure is sometimes referred to as VIMP, which stands for *variable importance*.

Since the first was introduced, two additional methods of permuting a predictor X_j have been proposed and together with Breiman's approach[6] they form the following list:

(A) *Permutation of the input* – the values of X_j in the OOB sample are randomly permuted.

(B) *Random node assignment* – in each tree to assign the terminal value for an observation follow split rules of the tree and if a split on X_j occurs than with equal probabilities choose the left or right daughter of this split.

(C) *Opposite node assignment* – proceed similarly as in (B) but instead of randomly assigning the left or right daughter choose the opposite one than would be normally chosen based on the value of X_j for this observation.

All of the above measures are implemented in R: (A) is available in the benchmark package `randomForest`, whereas the newer package `randomForestSRC` allows for choosing among all three perturbation methods. However, there are not a lot of theoretical results on the topic – some are given in [7] in the case of regression and for method (B) modified in the following way: if a split on X_j occurs than a random daughter is assigned not only at this split but also for all subsequent ones even if they do not use X_j. The authors of the paper note that this is similar to methods (A)-(C) because for all four methods VIMP is directly affected by the location of the first split on X_j. On the other hand, the modification differs substantially from (A)-(C) importance measures as neither of them would be affected by an early split on a non-informative X_j that was selected for the split only due to chance as no informative predictor was selected to the r variables considered for the split. Nevertheless, the results presented in [7] are instructive, in particular the authors prove that nodes closer to the root contribute more to prediction error – this motivates the usefulness of *minimal depth*, a measure of variable importance introduced in the next point.

Generally, perturbation of inputs as a measure of variable importance in random forests has been widely criticized in the literature [1]. Most notably, when it comes to qualitative predictors, VIMP measures are biased in favor of those that can take more categories. The authors of [17], who corroborate this criticism with simulations, ascribe this bias partly to the CART algorithm of growing trees – as a solution they propose building random forests using conditional inference trees [6] and incorporating a bias-correction into VIMP calculations (they implement this approach in the R package `cforest`).

Minimal depth

As mentioned before, variables used for splitting close to the root tend to be important as measured by VIMP. This has been proven in [7] using the concept of maximal subtrees, defined as follows: for each predictor X_j we call T_{X_j} an X_j-*subtree* of our tree T if the root of T_{X_j} is

[6] The Breiman's approach is sometimes called the Breiman-Cutler permutation VIMP.

split using X_j. T_{X_j} is a *maximal X_j-subtree* if it is not a subtree of a larger T_{X_j}-subtree. We illustrate this concept in Figure 1.4.

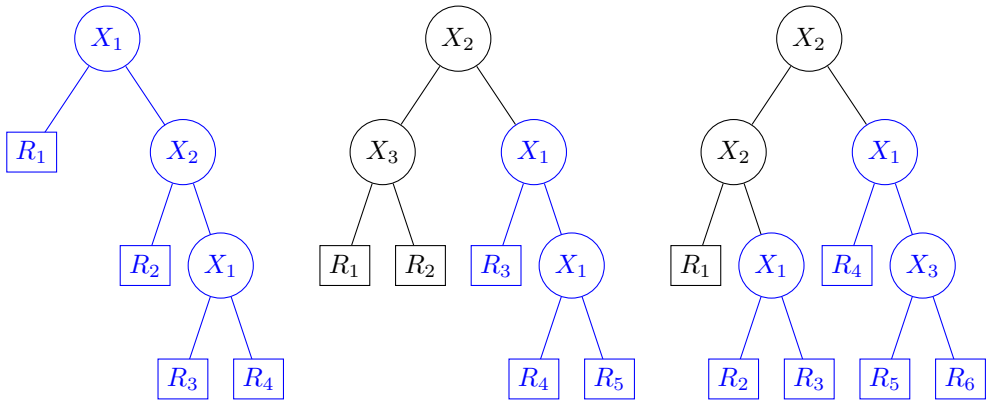


Figure 1.4: Illustration of the concept of maximal subtrees. Maximal X_1-subtrees are highlighted in blue. In the first tree X_1 splits the root so the maximal X_1-subtree is the whole tree. In the second tree the maximal X_1-subtree contains an X_1-subtree that is not maximal. In the third tree there are two maximal X_1-subtrees from which one is closer to the root than the other. [Source: own research.]

Maximal subtrees are a useful tool in exploring random forests; looking at how close they are to the root of a tree can help in assessing importance of a given variable. This approach has been developed in [8] and, in contrast to perturbation of inputs, aims at exploring the structure of the forest instead of treating it like an impenetrable black box[7].

The first-order statistic for a maximal subtree measures the distance (measured by the number of edges) from the root of the whole tree to the closest root of a maximal X_j-subtree for a given X_j. This is denoted as D_{X_j} and we call it the *minimal depth* of X_j in the tree. $D_{X_j} = d$ means that X_j is used for splitting for the first time at a node that is d edges away from the root. Obviously, D_{X_j} is nonnegative and can be at most equal to the depth of the tree, defined as the maximal distance between its root and any terminal node. The idea of using minimal depth as a measure of variable importance comes from the observation that nodes closer to the root contain on average more observations than the ones further away. In other words, variables used for splitting high in the tree more often divide big parts of the population and thus play a greater role in sorting the observations.

Minimal depth is a random variable with known distribution derived in [8]. Let $D(T) \geq 1$ be the (fixed) depth of tree T and assume that l_d, the number of nodes at depth d, is equal to 2^d for all $d = 1, 2, \ldots, D(T)$ (this is a restrictive assumption so relaxing it leads to slightly different results, see [8]). Then

$$\mathbb{P}(D_{X_j} = d) = \left[\prod_{i=0}^{d-1}(1 - \pi_{X_j,i}\theta_{X_j,i})^{l_i}\right][1 - (1 - \pi_{X_j,d}\theta_{X_j,d})^{l_d}], \quad 0 \leq d \leq D(T) - 1, \quad (1.14)$$

where

- $\pi_{X_j,i} := \pi_{X_j,i}(t)$ and $\theta_{X_j,i} := \theta_{X_j,i}(t)$ depend only on the depth of the node t,

[7] Note that according to the division mentioned in section 1.2 up until now we only discussed pedagogical methods of structure mining, whereas exploring maximal subtrees is an inherently decompositional method, as it decomposes the forest into trees and examines maximal subtrees in each of them.

- $\pi_{X_j,i}(t)$ is the probability that X_j is selected as one of the r candidates for splitting node t at depth i, assuming there is no maximal X_j-subtree with depth lower than i,
- $\theta_{X_j,i}(t)$ is the probability that X_j splits the node t with depth i, assuming that X_j is a candidate for splitting t and there is no maximal X_j-subtree with depth lower than i.

It is important to realize that the sum of probabilities (1.14) over d is bounded between 0 and 1 but does not equal 1 when no maximal X_j-subtree exists. In such a case we set D_{X_j} to the depth of the tree:

$$\mathbb{P}(D_{X_j} = D(T)) = 1 - \sum_{d=1}^{D(T)-1} \mathbb{P}(D_{X_j} = d). \tag{1.15}$$

Now we can easily calculate the minimal depth of variables X_1, X_2, X_3 in trees presented in Figure 1.4:

- X_1 has minimal depth 0 in the first tree and 1 in the others,
- X_2 has minimal depth 1 in the first tree and 0 in the others,
- X_3 is not used for splitting in the first tree so its minimal depth there is 3, it is equal to 1 in the second tree and 2 in the third.

It must be remembered that the concept of minimal depth strongly relies on the existence of maximal subtrees with respect to predictors whose importance we want to measure. Therefore, in problems with small n and big p we observe a "ceiling effect" of minimal depth as the trees cannot be grown deep enough to sufficiently differentiate the statistic between variables[8].

Node impurity

The third popular approach to measuring variable importance in random forests is based on node impurity measures that are used for growing trees (see section 1.3.1). Recall that at each split potential improvement of node purity is calculated for all candidate variables and we select for splitting the one which maximizes this improvement. Therefore, we can measure importance of each predictor by calculating its improvement in the split-criterion accumulated over all trees. This idea is easily implemented as one only needs to store the information on node purity calculated in the tree-growing process and aggregate it for each predictor.

In the case of classification problems, in which Gini index is the most widely used impurity measure, many studies found that measuring variable importance with this index induces bias in cases where the values of a predictor cluster into well separated groups regardless of whether the predictor is qualitative of quantitative [1]. Some procedures of correcting this bias have been proposed: e.g., permutation importance (PIMP) which normalizes the biased importance measure using a permutation test and additionally reports a p-value for each variable (see [1]).

[8] One solution to this issue has been proposed in [8]: RSF-Variable Hunting, which is a regularized algorithm for random survival forests.

Chapter 2

Functionality of the R package `randomForestExplainer`

In this chapter I introduce my new R package `randomForestExplainer` devoted to exploring importance of variables in a random forest. The package is designed for forests built with the `randomForest` package, which I consider the most popular implementation of the random forest algorithm. Functionality of my package is mainly concerned with visualization and for that purpose it uses `ggplot2` and two other packages that build upon it: `GGally` and `ggrepel`. For data processing and aggregation I use `data.table`, `dplyr`, `dtplyr`, `DT` and `reshape2`. Finally, for creating automatic summaries of forests in the form of HTML documents I use `rmarkdown`.

Each section of this chapter is devoted to discussing a different functionality of the package[1]. Every section starts with general ideas that I implement, then lists the functions to finally describe in detail how they work. For brevity I present my functions using pseudocode – the R code can be found in the online repository of the package[2]. I divide most of my functions into three groups:

1. *Data-generating* – the function takes as its main argument a `randomForest` object and returns data that are potentially useful to the user, e.g. importance measures.

2. *Plotting* – the function takes as its main argument a result of some data-generating function and plots it using `ggplot2`.

3. *Auxiliary* – the function is a building block of a function from group 1. or 2. and is not directly available to users of the package.

Note that this division separates data from plots so when the user wants to visualize some result, she usually has to generate appropriate data first. This may seem onerous, but it is useful as the user only needs to generate the data once (and for large random forests this may be time-consuming) to then plot it multiple times while adjusting graphical parameters.

Apart from describing how the functions work I illustrate the plotting ones using a random forest build on the `iris` data that predicts the `Species` of an iris flower using its four numerical characteristics: sepal and petal length and width. I grow the forest using the `randomForest::randomForest` function with option `localImp` set to `TRUE`.

[1] The division into sections corresponds to the division of R code that makes up `randomForestExplainer` into separate files.

[2] See https://github.com/geneticsMiNIng/BlackBoxOpener/tree/master/randomForestExplainer.

2.1. Minimal depth distribution

In section 1.3.3 I introduced the concept of minimal depth and discussed its usefulness. In practice, calculating minimal depth is implemented in the `randomForestSRC` package, which in addition to what `randomForest` does allows for building random survival forests. To compare variables, mean minimal depth over all trees is calculated, leading to a ranking of predictors from lowest (best) to highest (worst) mean minimal depth.

In my opinion looking only at the mean is not always sufficient and there is much to be gained from analyzing the whole distribution of minimal depth. I implement this idea in `randomForestExplainer` using the following functions:

(1) `calculate_tree_depth` (auxiliary) takes a data frame describing a single decision tree and adds to it information about depth of each node,

(2) `min_depth_distribution` (data-generating) for each tree in a forest the function computes depth of predictors using (1), gathers results for all trees in one data frame and computes the minimum of depth of each variable in each tree,

(3) `min_depth_count` (auxiliary) takes the result of (2) and counts the instances of each minimal depth for each variable and the number of trees in which each variable occurred; it also computes the mean tree depth in the forest,

(4) `get_min_depth_means` (auxiliary) takes results of (2) and (3), and calculates mean minimal depth for each variable in one of three possible ways specified by the user,

(5) `plot_min_depth_distribution` (plotting) takes the result of (2) to plot the discrete distribution of minimal depth obtained with (3) for a certain number of variables with the lowest mean minimal depth as given by (4); mean values are also added to the plot.

In this section I describe each of the above functions in turn.

2.1.1. Calculate the distribution

Let `forest` be my random forest produced by the package `randomForest` with option `localImp = TRUE`. To get a data frame with split information on the b-th tree, where variables are coded with their labels instead of numbers, I use the function `randomForest::getTree(forest, k = b, labelVar = TRUE)` and store the result in `frame`.

Each row in `frame` corresponds to a node in my tree and the first two entries point to the left and right daughter of this node using names of their respective rows; the third entry is the name of the predictor used for splitting. Conveniently, the rows in `frame` are ordered in such a way that daughters are always below their parents. This leads to a very simple construction of the function `calculate_tree_depth` described in Algorithm 2.1.

Algorithm 2.1: Calculating depth of nodes in a single tree

```
Function: calculate_tree_depth(frame)
let r ← number of rows of frame
let depth ← vector of length r
let depth[1] ← 0
for i = 2, 3, ..., r
    begin
    find j such that: left daughter[j] = i or right daughter[j] = i
    let depth[i] ← depth[j] + 1
    end
```

```
Return: frame with depth as a new column
```

Once I am able to calculate depth of nodes in a single tree it is easy to do that for the whole forest by taking all its trees in turn and applying `calculate_tree_depth`. Then, for each tree and predictor used in it for splitting I calculate the minimum of its depth. This simple procedure is implemented in the function `min_depth_distribution` described in Algorithm 2.2 – the function takes `forest` as its argument and returns a data frame with the whole distribution of minimal depth.

Algorithm 2.2: Calculating minimal depth in every tree of a forest

```
Function: min_depth_distribution(forest)
let forest_table ← empty data frame
for b = 1, 2, ..., B
    begin
    let frame ← getTree(forest, k = b, labelVar = TRUE)
    add calculate_tree_depth(frame) to forest_table
    end
group forest_table by: tree, split variable
let min_depth_frame ← min(depth) in each group
Return: min_depth_frame
```

To prepare the data frame `min_depth_frame` containing the distribution for plotting and averaging, I use the function `min_depth_count` (Algorithm 2.3) that returns a list with three elements: a data frame with frequency counts of each minimal depth for each variable, a data frame that for each variable gives the number of trees in which it was used for splitting and the mean depth of a tree in the forest.

Algorithm 2.3: Count the trees in which each variable had a given minimal depth

```
Function: min_depth_count(min_depth_frame)
/* Count instances of each minimal depth */
group min_depth_frame by: variable, minimal_depth
let min_depth_count ← count observations in each group
/* Count number of trees in which each variable occured */
group min_depth_count by: variable
let occurrences ← sum(count) in each group
/* Calculate mean depth of a tree */
group min_depth_frame by: tree
let mean_tree_depth ← mean[max(minimal_depth) + 1 in each group]
Return: list of min_depth_count, occurrences, mean_tree_depth
```

2.1.2. Mean minimal depth

Although my main goal in this section is to look at the whole distribution of minimal depth, calculating its mean is useful as it gives a simple ranking of variables. Such a ranking can be used for selecting the subset of variables of which the minimal depth distribution we wish to plot. Moreover, while considering the whole distribution, I should also look at the means to see whether they indeed provide insufficient information about my variables.

I propose two ways of calculating mean minimal depth in addition to the one described in section 1.3.3. All three approaches differ in he way they treat missing values that appear when a variable is not used for splitting in a tree. They can be described as follows:

(I) Filling missing values: the minimal depth of a variable in a tree that does not use it for splitting is equal to the mean depth of trees[3] in the forest [8].

(II) Restricting the sample: to calculate the mean minimal depth only $\tilde{B}$ out of B observations are considered, where $\tilde{B}$ is equal to the maximum number of trees in which any variable was used for splitting. Remaining missing values for variables that were used for splitting less than $\tilde{B}$ times are filled in as in (I).

(III) Ignoring missing values: mean minimal depth is calculated using only non-missing values.

Obviously, the results obtained using each of those approaches differ whenever many values are missing. One notable example of this occurs when the data contain many variables but few observations (large p, small n) as this leads to shallow trees, each using only a fraction of all p variables for splitting.

The main advantage of calculating mean minimal depth using (I) is the fact that it increases the mean for variables that are not frequently used for splitting. However, when the majority of values are missing the means will be strongly pulled towards mean depth of trees in the forest leading to small variability of the mean between predictors and problems with interpretation of these inflated values (in section 1.3.3 I referred to this as the "ceiling effect"). The second approach aims at reducing this effect by imputing only as many observations as necessary to use samples of the same size for all variables to compute the mean. One drawback of both (I) and (II) is that the value used for filling missing values is somewhat artificial.

Finally, approach (III) to calculating mean minimal depth only uses available observations and discards missing data. This removes the concern of filling missing values with something that is not really variable depth and therefore distorts interpretability of the mean. However, a major drawback of this approach is that low values of mean minimal depth obtained in this way do not necessarily mean that a variable is important. On the contrary: when a variable is used for splitting only once in a forest and this happens at the root (e.g. because no important variables were selected as candidates for the split) its mean minimal depth will be equal to zero, the lowest possible value.

In my opinion each of the three methods may be useful in applications so whenever I calculate the minimal depth I ask the user to specify the parameter `mean_sample` as equal to one of the following: `"all_trees"`, `"top_trees"`, `"relevant_trees"` corresponding to methods (I), (II) and (III) (I set `"top_trees"` as the default).

After obtaining `min_depth_frame` using the function `min_depth_distribution` and saving the result of `min_depth_count` as `count_list` I need to calculate means of minimal depth of my variables as described above. To do this I create the function `get_min_depth_means` described in Algorithm 2.4.

Algorithm 2.4: Calculate means of minimal depth in one of three ways

```
Function: get_min_depth_means(min_depth_frame, count_list, mean_sample)
if mean_sample = "all_trees"
    begin
    for j such that: minimal_depth[j] is missing
        begin
        let minimal_depth[j] ← mean_tree_depth
        end
```

[3]Note that the depth of a tree is equal to the length of the longest path from root to leave in this tree. This equals the maximum depth of a variable in this tree plus one, as leaves are by definition not split by any variable.

```
    group min_depth_frame by: variable
    let min_depth_means ← mean(minimal_depth) in each group
    end
if mean_sample = "top_trees"
    begin
    for j such that: minimal_depth[j] is missing
        begin
        let count[j] ← count[j] − min(count[all j])
        let minimal_depth[j] ← mean_tree_depth
        end
    group min_depth_count by: variable
    let min_depth_means ← mean(minimal_depth, weights = count) in each group
    end
if mean_sample = "relevant_trees"
    begin
    for j such that: minimal_depth[j] is missing
        begin
        remove observation j
        end
    group min_depth_frame by: variable
    let min_depth_means ← mean(minimal_depth) in each group
    end
Return: min_depth_means
```

2.1.3. Plot the distribution

Finally, I create the function `plot_min_depth_distribution` for plotting the minimal depth distribution (Algorithm 2.5). Its arguments are:

- `min_depth_frame` produced by the data-generating function `min_depth_distribution`,
- `k` – the maximal number of variables with lowest mean minimal depth to be included in the plot,
- `min_no_of_trees` – the minimal number of trees in which a variable has to be used for splitting to be used for plotting,
- `mean_sample` – the type of sample on which to calculate mean minimal depth,
- `mean_scale` – logical: should the mean minimal depth be rescaled so that its minimum and maximum are equal to 0 and 1, respectively?
- `mean_round` – integer: number of digits to which the displayed mean minimal depth should be rounded,
- `main` – the title of the plot.

The function plots the discrete distribution of minimal depth for at most `k` variables with lowest mean minimal and in addition displays this statistic (see Figure 2.1). As I mentioned before, in some applications it might be the case that the minimal depth distribution is dominated by missing values. Then, including missing values in the plot could obscure the rest of the distribution. To avoid that for each variable we only plot the number of missing values in addition to the minimal number of missing values ($B - \tilde{B}$).

Algorithm 2.5: Plot the distribution of minimal depth

```
Function: plot_min_depth_distribution(min_depth_frame, k, min_no_of_trees,
    mean_sample, mean_scale, mean_round, main)
let count_list ← min_depth_count(min_depth_frame)
let min_depth_means ← get_min_depth_means(min_depth_frame, count_list, mean_sample)
let data ← min_depth_count with means from min_depth_means
/* Subtract B − B̃ from missing values count */
for j such that: minimal_depth[j] is missing
    begin
    let count[j] ← count[j] − min(count[all j])
    end
/* Rescale means of minimal depth */
if mean_scale = TRUE
    begin
    let mean_minimal_depth ← (mean_minimal_depth − min(mean_minimal_depth)) / (max(mean_minimal_depth) − min(mean_minimal_depth))
    end
/* Keep variables that appeared often enough */
for j such that: no_of_occurrences[j] < min_no_of_trees
    begin
    remove observation j
    end
/* Select k best variables */
order data with increasing mean_minimal_depth
keep at most k first rows of data
Plot: y-axis: variable, x-axis: count, color: minimal_depth
    add labels with mean_minimal_depth rounded to mean_round digits
    add main as title
```

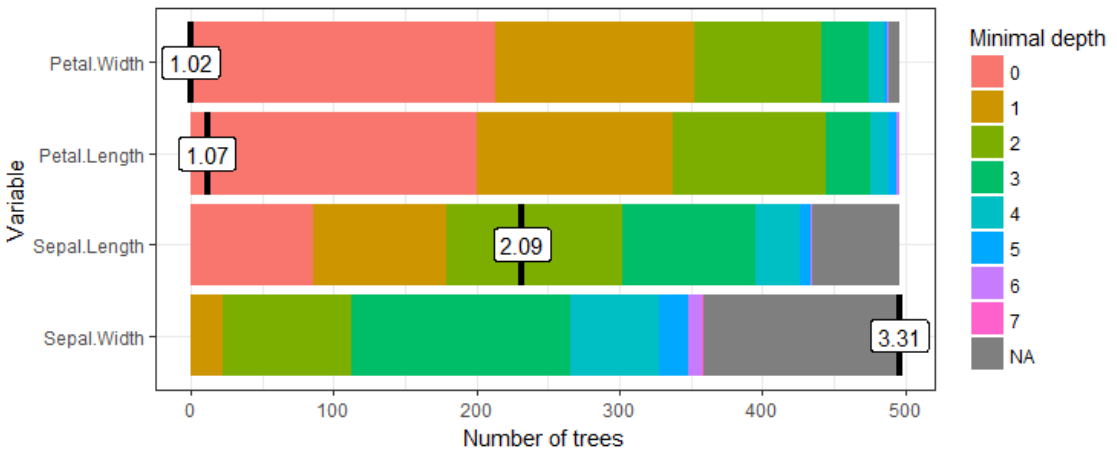


Figure 2.1: The distribution of minimal depth for the `iris` forest with `min_depth_frame` as main argument, `main = NULL` and remaining parameters set to their default values: `k = 10`, `min_no_of_trees = 0`, `mean_sample = "top_trees"`, `mean_scale = FALSE`, `mean_round = 2`. [Source: own research.]

2.2. Variable importance

In this section I introduce the part of `randomForestExplainer` functionality that calculates and plots various variable importance measures using the following functions:

(1) `measure_importance` (data-generating) takes `forest` and generates a data frame containing various variable importance measures,

(2) `important_variables` takes the result of (1) and returns names of up to `k` top variables according to the sum of rankings based on specified importance measures,

(3) `plot_multi_way_importance` (plotting) takes the result of (1) and plots two or three variable importance measures against each other (two corresponding to y and x coordinates and the optional third to the size or color of points),

(4) `plot_importance_ggpairs` (plotting) takes the result of (1) and plots all pairs of selected variable importance measures against each other,

(5) `plot_importance_rankings` (plotting) takes the result of (1) and plots all pairs of rankings based on selected variable importance measures against each other.

First, I need to calculate various variable importance measures. In Algorithm 2.6 I describe the function `measure_importance` that takes our `forest`, gathers split information on all its trees in a `forest_table` and calculates the following importance measures for each variable X_j:

(a) `accuracy_decrease` (classification) – mean decrease of prediction accuracy after X_j is permuted,

(b) `gini_decrease` (classification) – mean decrease in the Gini index of node impurity (i.e. increase of node purity) by splits on X_j,

(c) `mse_increase` (regression) – mean increase of mean squared error after X_j is permuted,

(d) `node_purity_increase` (regression) – mean node purity increase by splits on X_j, as measured by the decrease in sum of squares,

(e) `mean_minimal_depth` – mean minimal depth calculated in one of three ways specified by the parameter `mean_sample`, as described in section 2.1.2,

(f) `no_of_trees` – total number of trees in which a split on X_j occurs,

(g) `no_of_nodes` – total number of nodes that use X_j for splitting (it is usually equal to `no_of_trees` if trees are shallow),

(h) `times_a_root` – total number of trees in which X_j is used for splitting the root node (i.e., the whole sample is divided into two based on the value of X_j),

(i) `p_value` – p-value for the one-sided binomial test using the following distribution:

$$Bin(\texttt{no\_of\_nodes},\ \mathbb{P}(\text{node splits on } X_j)), \tag{2.1}$$

where I calculate the probability of split on X_j as if X_j was uniformly drawn from the r candidate variables

$$\mathbb{P}(\text{node splits on } X_j) = \mathbb{P}(X_j \text{ is a candidate}) \cdot \mathbb{P}(X_j \text{ is selected}) = \frac{r}{p} \cdot \frac{1}{r} = \frac{1}{p}. \tag{2.2}$$

This test tells me whether the observed number of successes (number of nodes in which X_j was used for splitting) exceeds the theoretical number of successes if they were random (i.e. following the binomial distribution (2.1)).

Measures (a)-(d) are calculated by the `randomForest` package (see [12]) so need only to be extracted from my `forest` object if option `localImp = TRUE` was used for growing the forest (I assume this is the case). Note that measures (a) and (c) are based on the decrease in predictive accuracy of the forest after perturbation of the variable, (b) and (d) are based on changes in node purity after splits on the variable and (e)-(i) are based on the structure of the forest.

Algorithm 2.6: Calculate variable importance measures

```
Function: measure_importance(forest, mean_sample)
/* Extract randomForest importance measures */
if forest type = "classification"
    begin
    let accuracy_decrease ← MeanDecreaseAccuracy from forest importance
    let gini_decrease ← MeanDecreaseGini from forest importance
    let vimp_frame ← accuracy_decrease and gini_decrease
    end
if forest type = "regression"
    begin
    let mse_increase ← %IncMSE from forest importance
    let node_purity_increase ← IncNodePurity from forest importance
    let vimp_frame ← mse_increase and node_purity_increase
    end
/* Calculate structure importance measures */
let forest_table ← empty data frame
for b = 1, 2, ..., B
    begin
    let frame ← getTree(forest, k = b, labelVar = TRUE)
    add calculate_tree_depth(frame) to forest_table
    end
group forest_table by: tree, split variable
let min_depth_frame ← min(depth) in each group
min_depth ← get_min_depth_means(min_depth_frame, min_depth_count(min_depth_frame),
    mean_sample)
group forest_table by: split variable
let no_of_nodes ← count observations in each group
group min_depth_frame by: variable
let no_of_trees ← count observations in each group
/* Calculate p-value */
let p_value ← p-value of right-sided binomial test:
    number of successes: no_of_nodes
    number of trials: sum(no_of_nodes)
    probability of success: 1/p
Return: data frame with vimp_frame, min_depth, no_of_nodes, no_of_trees, p_value
```

After calculating all variable importance measures I might be interested in finding a number of top variables according to one or more of those measures (e.g., to restrict further analysis to this subset of variables). I propose a simple solution of doing that: selecting variables with the lowest sum of rankings (`index`), each based on one of the importance measures of interest. This is implemented in the function `important_variables` (Algorithm 2.7) that takes as its argument the result of `measure_importance`, a vector of importance measures to be used,

the number of top variables to select and a `ties_action` parameter. The last specifies which variables should be selected if a problematic tie occurs, i.e. when the `k`-th top variable has sum of rankings q and this is equal to that of the `k`+1-th variable. Three possible values of this parameter are:

- `none` – no variable with `index`=q is included in the result so it may be shorter than `k`,
- `all` – all variables with `index`=q are included in the result so it may be longer than `k`,
- `draw` – a uniformly drawn subset of variables with `index`=q appears in the result so it will be exactly of length `k`.

Algorithm 2.7: Select `k` most important variables in a forest

```
Function: important_variables(importance_frame, k, measures, ties_action)
let rankings ← rank variables according to measures
let index ← sum(rankings)
let vars ← min(k, p) variables with lowest index
let q ← index[j] for j such that: index[j] = max(index[l ∈ vars])
if [#(j such that: index[j] = q) > 1] and [#(j such that: index[j] ≤ q) > k]
    begin
    if ties_action = "none" then let vars ← j such that: index[j] < q
    if ties_action = "all" then let vars ← j such that: index[j] ≤ q
    if ties_action = "draw"
        begin
        let vars ← j such that: index[j] < q
        add to vars: uniformly draw k−#(vars) j's such that: index[j] = q
        end
    end
Return: vars (a vector of variable names)
```

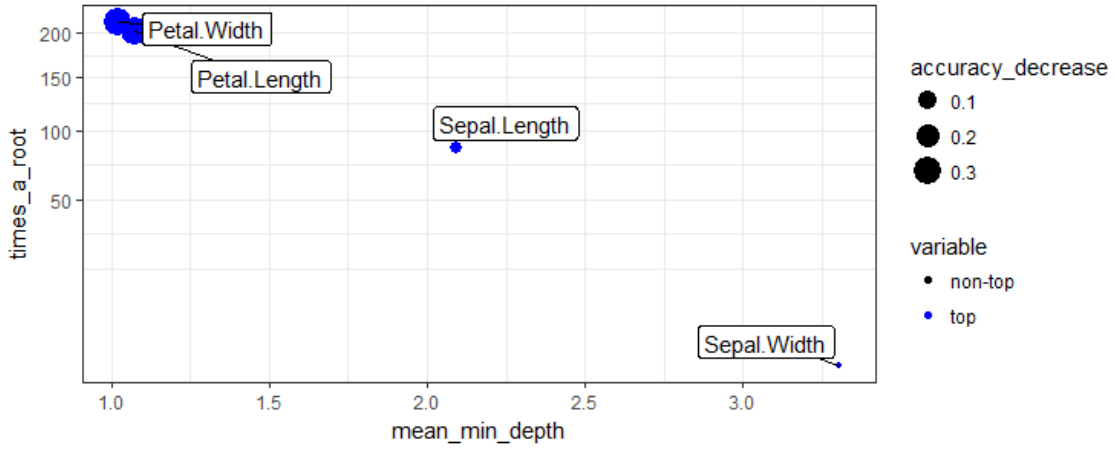


Figure 2.2: The multi-way importance plot for the `iris` forest with `importance_frame` as main argument, `size_measure` = `"accuracy_decrease"`, `main` = `NULL` and the remaining parameters set to their default values: `x_measure` = `"mean_min_depth"`, `y_measure` = `"times_a_root"`, `min_no_of_trees` = 0, `no_of_labels` = 10. [Source: own research.]

Regardless of the type of my random forest, the result of `measure_importance` contains seven importance measures for the predictors, which opens a lot of possibilities for visual-

ization. I propose three plotting functions in turn – in all of them, in addition to tuning parameters discussed below, the user can supply a character string `main` to be used as title.

The first plotting function produces what I call a *multi-way importance plot* that shows three selected measures using a scatter plot with size or color of points varying according to the third measure (see Figure 2.2). The corresponding function `plot_multi_way_importance` takes the following parameters:

- `importance_frame` – the result of `measure_importance`,
- `x_measure`, `y_measure`, `size_measure` – each is a string containing one of importance measures contained in `importance_frame`; `size_measure` is optional,
- `min_no_of_trees` – the minimal number of trees in which a variable has to be used for splitting to appear in the plot,
- `no_of_labels` – the number of top variables, according to all measures plotted, to be labeled (we allow for more labels in case of ties).

The resulting plot is slightly different depending on which measures are being plotted. If `no_of_nodes`, `no_of_trees` or `times_a_root` are used as either `x_measure` or `y_measure` the corresponding axis uses the square root scale so that differences in high values are clearly visible (as these are usually the most interesting). When it comes to `size_measure`: if it is set to `mean_min_depth` its scale is reversed so that bigger points correspond to smaller values (as variables with smaller mean minimal depth are considered better). Furthermore, if `p_value` is the `size_measure`, then instead of varying the size of points I vary their color after turning `p_value` into a qualitative variable informing of the level of significance at which each predictor is significant – this modification makes the plot a lot easier to analyze.

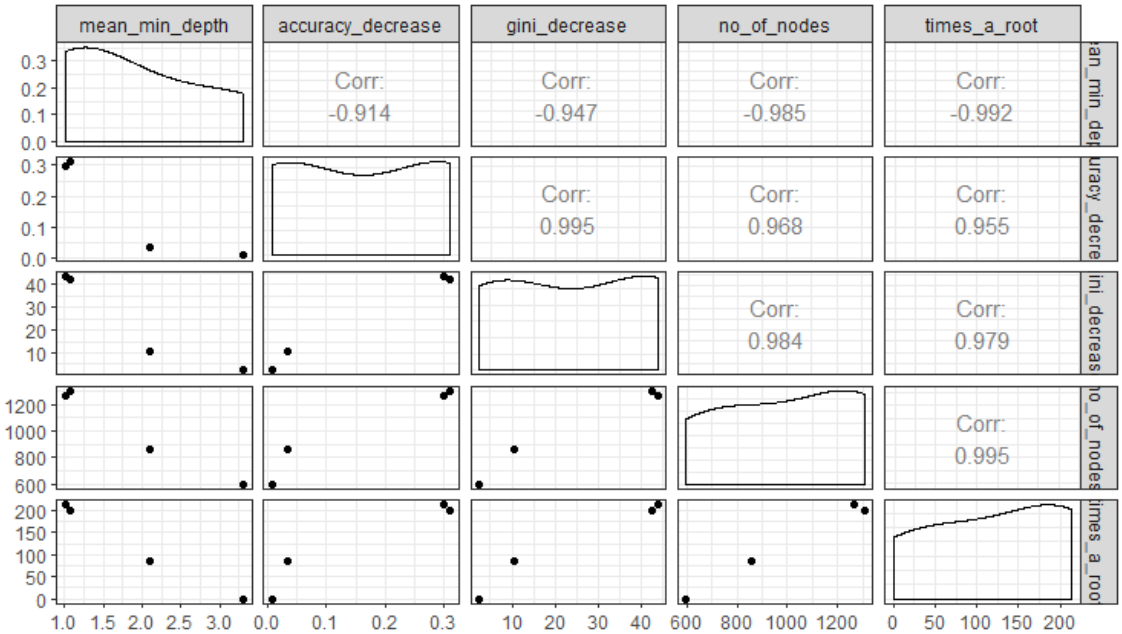


Figure 2.3: Pairwise comparisons of importance measures of variables for the `iris` forest with `importance_frame` as main argument, `main = NULL` and the `measures` argument set to its default `names(importance_frame)[c(2, 4, 5, 3, 7)]`. [Source: own research.]

The main problem with the multi-way importance plot is that it only shows three measures at a time and one of them is represented by size (or color) so differences in it are not always clearly visible. To address this issue I propose plotting more measures pairwise against each other with the function `plot_importance_ggpairs` that simply uses the function `ggpairs` from package `GGally` to selected measures of importance (argument `measures`) from the `importance_frame` supplied by the user (see Figure 2.3). In addition to scatter plots for all pairs of importance measures the resulting plot includes pairwise correlation between them as well as a density plot for each measure.

Finally, if the distribution of an importance measure is atypical (e.g., strongly skewed), then pairwise scatter plots can be obscure and uninformative as many points will overlap. An alternative approach is to use rankings of variables with respect to importance measures instead of raw measures as the former are more or less (depending on the number of ties) uniformly distributed. I implement this approach in the function `plot_importance_rankings` that takes the same arguments as `plot_importance_ggpairs` but plots rankings of measures with fitted LOESS curves that are helpful whenever numerous points fill the whole plot (see Figure 2.4).

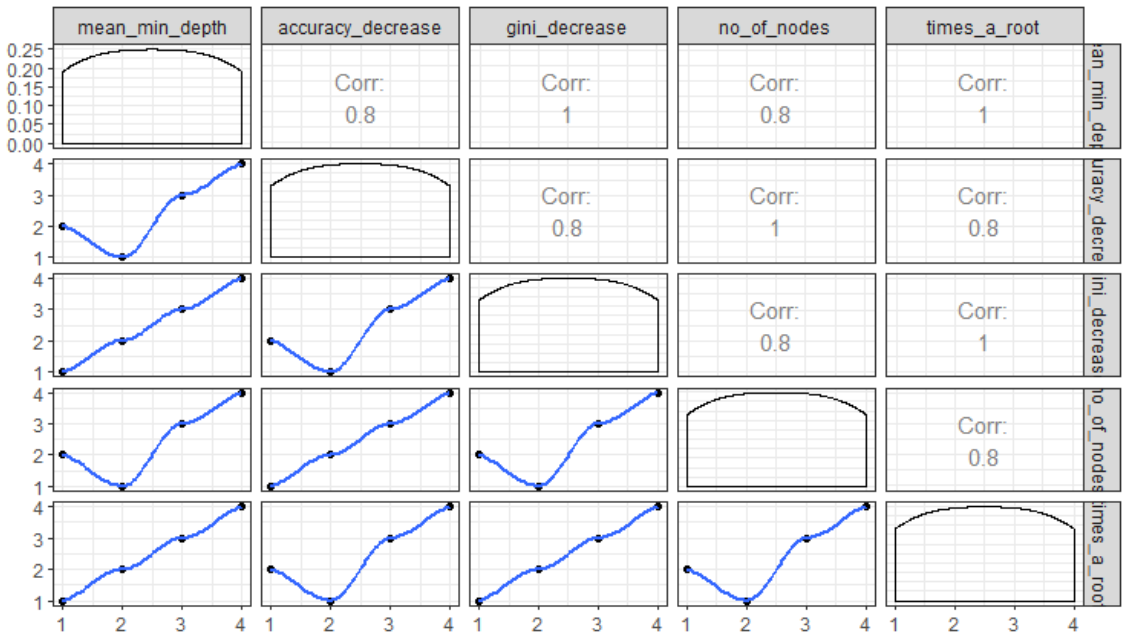


Figure 2.4: Pairwise comparisons of rankings according to importance measures of variables for the `iris` forest with `importance_frame` as main argument, `main = NULL` and the `measures` argument set to its default `names(importance_frame)[c(2, 4, 5, 3, 7)]`. [Source: own research.]

2.3. Interactions of variables

In this section I present `randomForestExplainer` functions that investigate the role of interactions of variables in order to find the most important ones in the forest. On one hand, the need for such analysis is pressing as random forests are composed out of trees and decision trees explore the data precisely through interactions. On the other hand, although I already gave some ideas of how to analyze single predictors it is much less clear of how to approach

their interactions. Usually, this is done by adjusting the concept of importance of single variables such that it would allow for interactions. I do this in the case of minimal depth by introducing conditional minimal depth. In addition, I examine the prediction of our random forest for a grid of values of an interaction of two numerical variables. I do this using the following functions:

(1) `conditional_depth` (auxilliary) takes a data frame describing a single decision tree (after depths of nodes have been calculated with `calculate_tree_depth`), a vector of conditioning variables and returns the data frame with information on depth of nodes conditional on each of the conditioning variables,

(2) `min_depth_interactions_values` (auxilliary) takes a random forest, a vector of conditioning variables and uses (1) to compute conditional depth in the whole forest to then calculate the minimum for each variable in each tree and return it in a list together with mean depths of maximal subtrees with respect to each conditioning variable,

(3) `min_depth_interactions` (data-generating) takes a random forest and a vector of conditioning variables, uses (2) to compute conditional minimal depth in all trees and mean depths of maximal subtrees, and returns a data frame with mean conditional minimal depth for each interaction calculated in one of three ways,

(4) `plot_min_depth_interactions` (plotting) takes the result of (3) and plots the conditional mean minimal depth for a given number of most frequent interactions together with mean minimal depth of the second variable in the interaction, for comparison,

(5) `plot_predict_interaction` uses the forest and data on which it was trained to plot its prediction for each point on a grid for two numerical predictors with values of other predictors sampled from their empirical distributions.

It is worth noting that exploring interactions is usually computationally intensive due to many possibilities (for p variables there are $1/2p(p-1)$ possible interactions of two variables) so I restrict my analysis to interactions of two variables.

2.3.1. Conditional minimal depth

Conditional minimal depth is a simple modification of minimal depth: for a conditioning variable X_j and a second variable X_l I define *conditional minimal depth of X_l with respect to X_j* as the minimal depth of X_l in the maximal X_j-subtree closest to the root of the whole tree minus one (my definition is based on the ideas presented in [8]). I subtract one in order to make the values of conditional minimal depth comparable to the unconditional ones: the latter range from zero, whereas without subtracting one the former would range from one as the root of any X_j-subtree is by definition split on X_j.

Obviously, conditional minimal depth is not symmetric: if X_j is the conditioning variable I analyze X_j-subtrees and look for splits on X_l in them, whereas if I exchange the roles of these variables I would be examining X_l-subtrees. As an example consider the simple decision trees in Figure 1.4 and variables X_2 and X_3. In the second and third trees X_3 is always below X_2 so conditional minimal depth of X_3 with respect to X_2 is equal to 1 and 2, respectively. On the other hand, the conditional minimal depth of X_2 with respect to X_3 is equal to the mean depth of maximal X_3-subtrees in the forest as there are no splits on X_2 in X_3-subtrees.

This leads to the conclusion that analyzing conditional minimal depth makes much more sense if the conditioning variables are often used for splitting in the forest, because then I

have many maximal subtrees to consider and ample possibilities for other variables being used for splitting in them. Therefore, I select a vector of important predictors `vars` to use as conditioning variables – I can do that for example with the function `important_variables`.

The function `conditional_depth` (Algorithm 2.8) takes as arguments a vector of conditioning variables and the result of `calculate_tree_depth` with empty columns that have names corresponding to conditioning variables. It returns the latter with these columns filled with values of conditional minimal depth of each variable present in the tree with respect to each conditioning variable.

Algorithm 2.8: Calculating conditional depth of nodes in a single tree

```
Function: calculate_tree_depth(frame, vars)
let r ← number of rows of frame
/* Get the position of roots of maximal subtrees */
group frame by: split variable
let index ← number of row with min(depth) in each group
for j ∈ vars
    begin
    let start ← index for j
    let df ← rows of frame from start to r
    /* Depth of the root */
    (column j of df)[1] ← 0
    for i = 2, 3, ..., (r−start+1)
        begin
        find l such that: left daughter[l] = i or right daughter[l] = i
        let (column j of df)[i] ← (column j of df)[l] + 1
        end
    update corresponding entries of frame with df
    end
replace 0 in vars columns of frame with missing values
Return: frame with vars columns filled
```

To calculate the conditional minimal depth in the whole forest I use the function presented in Algorithm 2.9, `min_depth_interactions_values`. This function returns a list of which the first element contains conditional minimal depth values for all interactions and the second is a vector with mean depth of maximal subtrees of each variable in `vars` that are necessary for calculating mean conditional minimal depth using approaches (I) or (II) from section 2.1.2, which include filling missing values.

Algorithm 2.9: Calculating conditional minimal depth in a forest

```
Function: min_depth_interactions_values(forest, vars)
let interactions ← empty data frame
/* Calculate unconditional depth */
for b = 1, 2, ..., B
    begin
    let frame ← getTree(forest, k = b, labelVar = TRUE)
    add calculate_tree_depth(frame) to interactions
    end
add columns with names vars to interactions
/* Fill in conditional depth */
for b = 1, 2, ..., B
    begin
    let frame ← interactions rows for tree = b
    calculate_tree_depth(frame, vars)
```

```
    end
let result ← empty data frame
let tree_depths ← empty vector of the same length as vars
for j ∈ vars
    begin
    /* Calculate minimal depths */
    group interactions by: tree, split variable
    let column j of result ← min(column j of interactions) in each group
    /* Calculate mean depths of maximal subtrees */
    group interactions by: tree
    let tree_depths[j] ← mean[max(column j of interactions) + 1 in each group]
    end
/* Subtract one for comparability with unconditional depth */
let result ← result − 1
Return: list of result, tree_depths
```

Finally, I create the function `min_depth_interactions` to calculate mean conditional minimal depth using the method specified in the same way as for single variables: by the parameter `mean_sample`. In addition, for every interaction the function returns the unconditional mean minimal depth of the second variable in the interaction (as opposed to the conditioning one) so that a comparison between the mean depth of this variable in whole trees and subtrees can be made. The method for calculating the last statistic is specified with the `uncond_mean_sample` parameter. In Algorithm 2.10 I describe how the function works without going into the details of processing the result such that instead of a matrix with entry (i, j) corresponding to the mean conditional minimal depth of variable in row i with respect to variable in column j I get a data frame with each interaction in a single row (this is easily done in R).

Algorithm 2.10: Calculating means of conditional minimal depth in a forest

```
Function: min_depth_interactions(forest, vars, mean_sample, uncond_mean_sample)
let result, tree_depths ← elements of min_depth_interactions_values(forest, vars)
group result by: variable
/* Calculate means for relevant variables */
let means ← mean(each j ∈ vars) in each group
let occurrences ← (count non-missing observations of each j ∈ vars) in each group
if mean_sample ≠ "relevant_trees"
    begin
    let non_occurrences ← B − occurrences
    replace missing values in means by 0
    end
if mean_sample = "top_trees"
    begin
    let B̃ ← min(non_occurrences)
    let non_occurrences ← by column: non_occurrences − B̃
    for j ∈ vars
        begin
        let column j of means ← (column j of means · column j of occurrences
                    + tree_depths[j] · column j of non_occurrences)1/(B − B̃)
        end
if mean_sample = "all_trees"
    for j ∈ vars
        begin
        let column j of means ← (column j of means · column j of occurrences
                    + tree_depths[j] · column j of non_occurrences)1/B
```

```
        end
    end
Return: data frame with means and occurrences of interactions in rows appended with
    unconditional minimal depth of second variable calculated on sample given by
    uncond_mean_sample
```

The function `plot_min_depth_interactions` plots the data on mean conditional minimal depth obtained by `min_depth_interactions` in the following way. It produces a bar plot for at most `k` (parameter) most frequent interactions ordered by decreasing frequency from left to right, where the height of the bar corresponds to the mean conditional minimal depth of each interaction and the minimum of this statistic across all interactions is given by a red line (see Figure 2.5). In addition, the unconditional mean minimal depth of the second variable in the interaction is marked by a point so that it can be compared with the height of the bar.

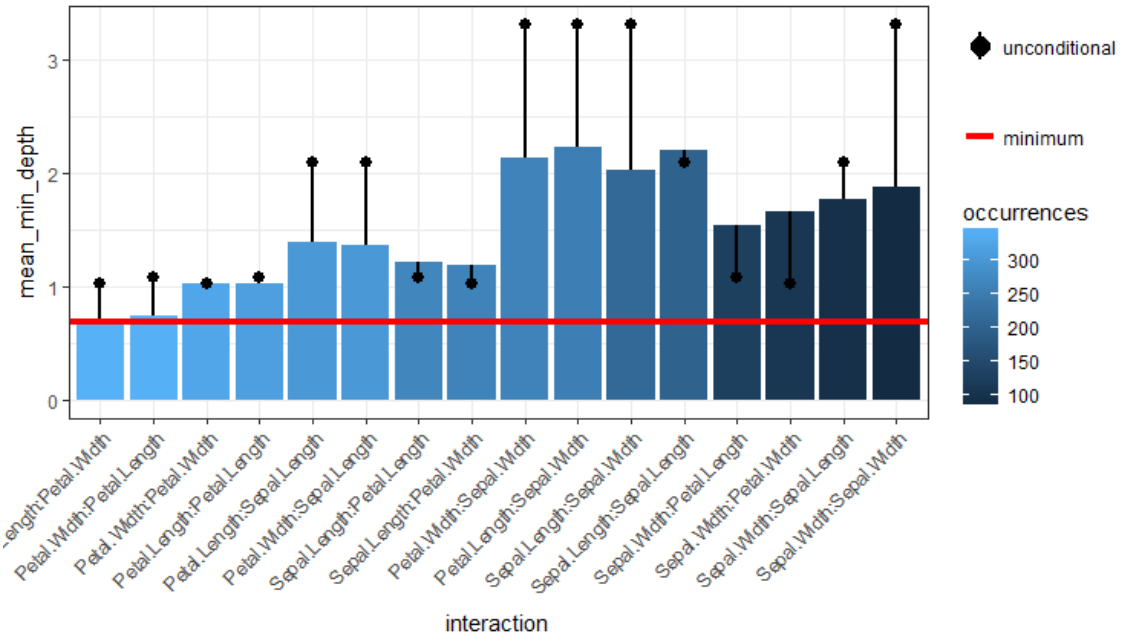


Figure 2.5: Mean conditional minimal depth for the `iris` forest with `interactions_frame` as main argument, `main = NULL` and the remaining parameter `k` set to its default value of 30. [Source: own research.]

2.3.2. Prediction on a grid

So far I have analyzed interactions in terms of where splits corresponding to them appear in the forest. However, one may also be interested in how the prediction of the forest depends on values of both variables that make up the interaction. A simple idea of addressing this question is to calculate the forest prediction for many combinations of values of these variables and to present it on a two-dimensional plot with color of points reflecting the predicted value.

To do this, the function `plot_predict_interaction` creates a 100×100 grid for quantitative predictors `variable1`, `variable2` such that the grid ranges from the minimum to maximum of each variable and is equally spaced in each dimension (the size of the grid can be changed with the `grid` parameter). Then, remaining predictors are one by one sampled from their empirical distributions – to do that original data used for training the forest has to

be passed to the function as argument called `data`. Finally, predicted values for the new data are calculated: in the case of regression the response is predicted, whereas for classification – the probabilities of each class (even though they sum up to 1 we include all of them as any could be of interest to the user). The final result is a plot (multiple plots for classification) with `variable1` on the x-axis, `variable2` on the y-axis and color of the points corresponding to predicted values (see Figure 2.6).

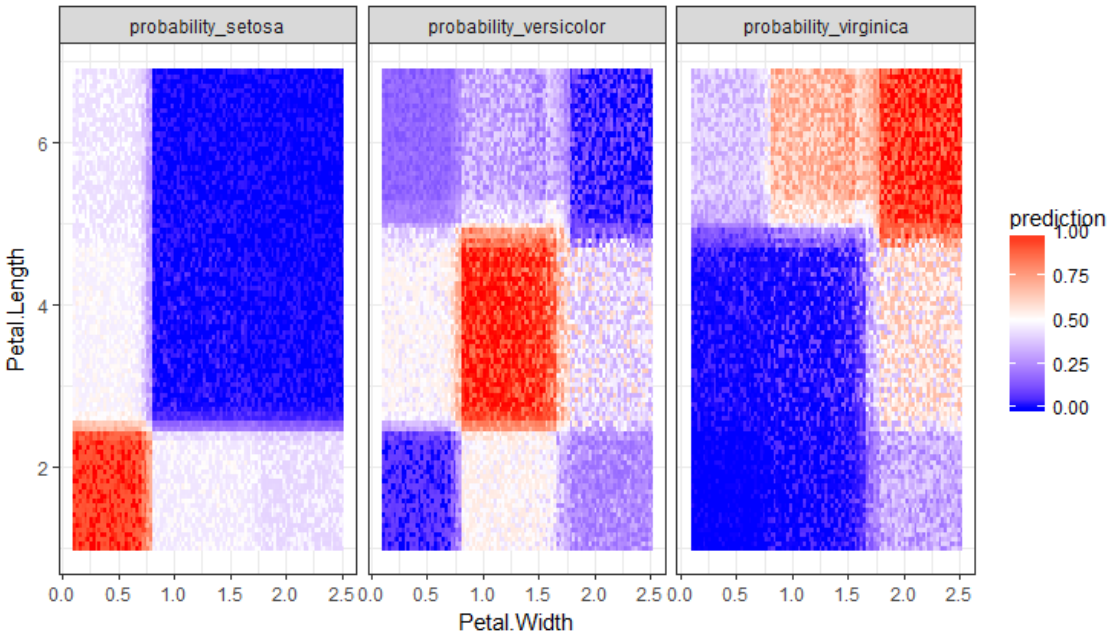


Figure 2.6: Prediction of the `iris` forest on a grid with the following set of parameters: `forest = forest, data = iris, variable1 = "Petal.Width", variable2 = "Petal.Length", main = NULL` and the `grid` parameter set to its default value of 100. [Source: own research.]

2.4. Explain the forest

As I have persistently argued in this chapter, `randomForestExplainer` allows us to generate various data and plots of which each looks at predictors used by the forest from a different perspective. Of course, as random forests are inherently complex, there is no one right way to extract knowledge from them and it is up to the user to decide which plot or variable importance measure is most convincing in a specific application. To do that, the user would probably want to briefly examine all `randomForestExplainer` plots to help her decide which seems the most promising and explore it in more detail (e.g., create similar plots with slightly different parameters). In order to facilitate this, I propose a wrapper function `explain_forest` that uses all capabilities of `randomForestExplainer` and renders a HTML document that includes resulting plots and interactive tables of data for the user to explore.

The function `explain_forest` takes the following arguments:

(a) `forest` – a `randomForest` object created with the option `localImp = TRUE`,

(b) `interactions` – logical: should interactions of variables be considered (the default value is `FALSE` as exploring interactions is computationally intensive)?

(c) `data` – the data frame on which `forest` was trained,

(d) `vars` – a vector of names of predictors with respect to which interactions should be considered; if equal to `NULL` a default of at most 15 conditioning variables will be selected using the function `important_variables`,

(e) `no_of_pred_plots` – the number of most frequent interactions of numeric variables for which prediction on a grid will be plotted, the default is 3,

(f) `measures` – a vector of names of importance measures to be plotted by functions `plot_importance_ggpairs` and `plot_importance_rankings`, by default the following are used: mean minimal depth, number of nodes, number of trees in which the variable splits the root and both measures calculated by the `randomForest` package.

Note that parameters (c)-(e) are redundant if `interactions` = `FALSE` as they are needed only by functions dedicated for analyzing interactions of variables. The default values of parameters allow the user to get the HTML report simply by passing the forest to the function. If one wants interactions to be considered as well, then apart from setting `interactions` to `TRUE` the argument `data` needs to be specified.

The function `explain_forest` is very simple: it creates a new environment to which it passes the values of all parameters and uses `rmarkdown::render` to produce the HTML report using a template from the package directory with objects from the new environment. The report is saved to the working directory in a file called `Your_forest_explained.html`.

Clearly, the crucial element in rendering the document is the R markdown template that is used. The template starts by listing the details of the `forest` (i.e., the formula used, the number of trees, the parameter r and information on prediction accuracy). Then, data for plotting are generated using functions `min_depth_distribution`, `measure_importance` and `min_depth_interactions` if interactions are to be considered. Further, the distribution of minimal depth is plotted and an interactive table with all importance measures is generated followed by two multi-way importance plots: the first contains structure measures (mean minimal depth, number of nodes and number of trees in which the variable splits the root), the second contains the increase in node purity measure, decrease in prediction accuracy after permutation and the p-value. Finally, both plots comparing all `measures` pairwise appear: the first uses raw measures and the second – rankings.

A conditional part of the template is used only if `interactions` = `TRUE`. Then, mean conditional minimal depth is plotted for up to 30 most frequent interactions and an interactive table containing the data concerning interactions is given. Finally, up to `no_of_pred_plots` most frequent interactions of numerical variables are plotted with `plot_predict_interaction`. Apart from all the aforementioned plots and tables, the template contains brief comments that help understand the results. For example output of the function see Appendix B.

All in all, the `randomForestExplainer` is a useful tool for exploring the importance of variables in a random forest. In my opinion the emphasis on visualization is a major advantage of the package, as well as the wrapper function that allows the user to see all plots without getting to know the details of how to create them. A list of functions of the package available to the user together with arguments and their default values can be found in Appendix A.

Chapter 3

Application to The Cancer Genome Atlas data

In this chapter we use `randomForestExplainer` to explore a random forest built on data concerning glioblastoma brain cancer, generated by the TCGA Research Network [18]. The forest is meant to predict whether a patient survived one year after the diagnosis using data on expression of various genes. This is a typical example of a problem with large p and small n as the data set contains many more genes than patients.

3.1. The data and random forest

Our data come from the R package `RTCGA` [11] and contain observations for 125 patients that have been diagnosed with glioblastoma. There are 16118 columns in the data: `death1y` (1 if the patient died within a year from diagnosis, 0 otherwise), `Cluster` (subtype of the tumor), `age` (age in years at the time of diagnosis) and 16115 columns containing numerical measurements of gene expression (names of columns are those of corresponding genes). Our aim is to build a random forest that predicts `death1y` and to single out genes whose expression drives the prediction.

Before we proceed, we have to clarify the term *gene expression* in order to understand what we are actually modelling. It is worth noting that the definition of a *gene* depends on context and in our analysis a *gene* is a place on the DNA. In cell cycle different places of DNA transcribe with different speed and efficiency. We treat the amount of transcripts that are related to a given place as a random variable. In a single experiment we have *gene expression* which is its realization. [16].

Moreover, we need to resolve the issue of missing data before we start our analysis. In section 1.3.2 we discussed three ways of dealing with missing data in random forests: imputing missing values, creating a "missing value" category for factors and using surrogate splits. We do not apply any of these approaches here as we do not wish to somewhat arbitrarily impute data, all our missing values are numeric gene expressions and surrogate splits are very computationally intensive and thus not implemented in the `randomForest` package. Therefore, in order to use all of our 125 observations we simply discard the 1989 out of 16117 predictors that contain missing values, which still leaves a plethora of variables for our analysis.

Finally, we use the `randomForest::randomForest` function to train a forest of 10000 trees, with option `localImp = TRUE` and $r = \lfloor\sqrt{14128}\rfloor = 118$ candidate variables for each split. The reason for growing such a big forest is that we are interested in distinguishing important predictors – in that case the more trees in a forest the better, as they create more opportunities

for good variables to be chosen for splitting (i.e., increase the sample on which statistics such as mean minimal depth are calculated).

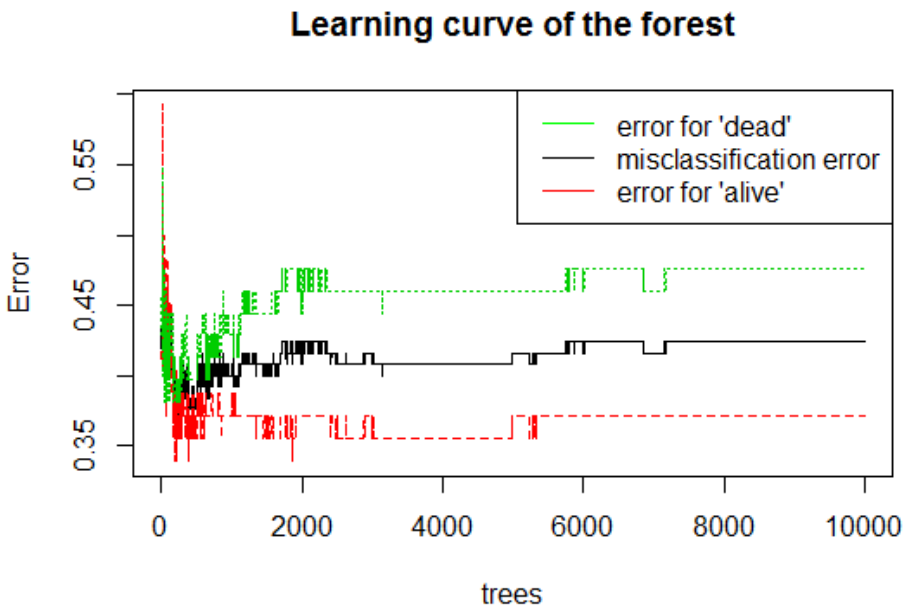


Figure 3.1: The OOB prediction error for each class and total OOB misclassification error of the random forest depending on the number of trees it consists of. [Source: own research.]

In Figure 3.1 we can see the learning curve of our forest, i.e. the evolution of out-of-bag error when the number of trees increases. Clearly, this error is minimal for around 500 trees and stabilizes at around 0.4 at 2000 trees. With 10000 trees the OOB estimate of our error rate is 0.42 and the forests classifies correctly 39 out of 62 surviving patients and 33 out of 63 deceased patients. We conclude that this is a sensible predictor and when it comes to prediction accuracy we could obtain similar results with a lower number of trees.

3.2. Distribution of minimal depth

To obtain the distribution of minimal depth we pass our forest (stored in the object `forest`) to the function `min_depth_distribution` and store the result in `min_depth_frame` which contains three columns: `tree` (tree number), `variable` (name of the predictor) and `minimal_depth`. Then we apply `plot_min_depth_distribution` to `min_depth_frame` and obtain Figure 3.2. Note that the x-axis ranges from zero trees to the maximum number of trees in which any variable was used for splitting which is in this case equal to 85 and is reached by the variable `C8orf58`. This means, that each predictor was used for splitting in less than 1% of all trees, which is caused by a low number of observations that leads to shallow trees.

The ordering of variables in Figure 3.2 by their mean minimal depth seems accurate when we look at the distribution of minimal depth (e.g., one could argue whether `C8orf58` is indeed better than `SLC17A9` as the latter is more often the root but the former is used for splitting in many more trees). However, this is not always the case. For example, we can calculate the mean minimal depth only using non-missing observations (`mean_sample = "relevant_trees"`) and require that only variables present in at least 60 trees be considered (i.e., set `min_no_of_trees` to 60, note that this is fulfilled by all variables in Figure 3.2).

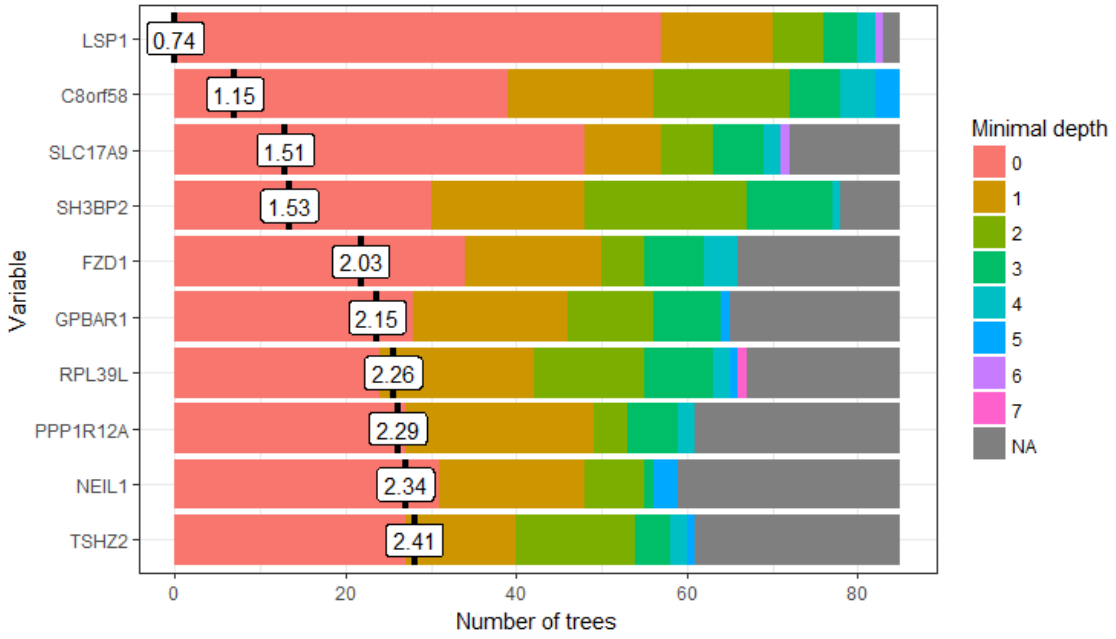


Figure 3.2: The distribution of minimal depth for top ten variables according to mean minimal depth under default settings. The mean is calculated using top trees (method (II), only $\tilde{B} - B$ missing values are imputed) and is scaled to fit the range of the x-axis. [Source: own research.]

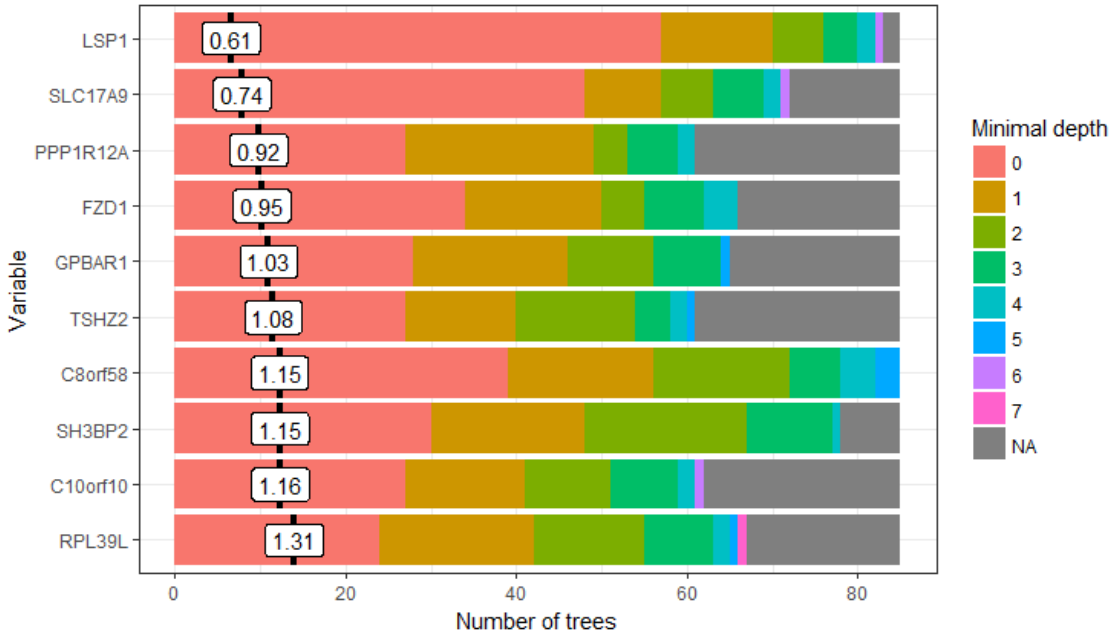


Figure 3.3: The distribution of minimal depth for top ten variables according to mean minimal depth calculated using only relevant trees (method (III), no missing values imputation) with the additional requirement that the variable is used for splitting in at least 60 trees. [Source: own research.]

Some form of the last requirement is in this case necessary to avoid selecting variables that have been by chance used for splitting once at the root. These settings result in Figure 3.3.

Clearly, method (III) of calculating mean minimal depth does not penalize missing observations (when a variable is not used for splitting in a tree) and the plotting function only requires the threshold of 60 trees to be fulfilled. This approach leads to the variable `PPP1R12A` appearing as third best even though it is not only present in about 25% less trees than `C8orf58` but also much less often splits the root node. `C8orf58` on the other hand has minimal depth of 5 in some trees and this contributes to higher mean minimal depth – in our opinion being used for splitting at depth 5 in a tree is better than not being used at all.

Regardless of the exact parameters used in `plot_min_depth_distribution`, looking at the whole distribution of minimal depth offers a lot more insight into the role that a predictor plays in a forest in contrast to looking only at the mean, especially as it can be calculated in more than one way.

3.3. Various variable importance measures

To further explore variable importance measures we pass our forest to `measure_importance` function and save the result as `importance_frame` – it contains 14128 rows, each corresponding to a predictor, and 8 columns of which one stores the variable names and the rest store the variable importance measures. Although this data frame can be useful in itself, we simply pass it to `randomForestExplainer` plotting functions.

In Figure 3.4 we present the result of `plot_multi_way_importance` for the default values of `x_measure` and `y_measure`, where the size of points reflects the number of nodes split on the variable (`size_measure = "no_of_nodes"`). In order not to obscure the picture with too many points we only plot those that correspond to variables used for splitting at least in 30 trees (`min_no_of_trees` = 30); in our case this requirement restricts the sample to 145 out of 14128 observations. By default `no_of_labels` = 10 top variables in the plot are highlighted in blue and labeled – these are selected using the function `important_variables`, i.e. using the sum of rankings based on importance measures used in the plot (more variables may be labeled if ties occur, as is the case here: 11 variables are highlighted).

Observe the marked negative relation between `times_a_root` and `mean_min_depth` in Figure 3.4 (though it looks linear it is not as the y-axis uses the square root scale). It is necessary to remember that the plot excludes the worst predictors and including them could make the aforementioned relation look a lot different. However, if we are interested in the relation of the depicted importance measures only among variables that were used for splitting in at least 30 trees then our observation remains valid and we can conclude that using either of those two importance measures is sufficient as they are highly (negatively) correlated.

In Figure 3.5 we present the multi-way importance plot for a different set of importance measures: decrease in predictive accuracy after permutation (x-axis), decrease in the Gini index (y-axis) and levels of significance (color of points). As in the previous plot, the two measures used as coordinates are strongly correlated, but in this case this is somewhat more surprising as one is connected to the structure of the forest and the other to its prediction, whereas in the previous plot both measures reflected the structure. Also, Figure 3.5 includes all 14128 observations and we can see that the most pronounced differences appear among top variables. Moreover, the p-value criterion is evidently not very selective – indeed, 1015 predictors are significant at the 1% significance level. Interestingly, all 10 top variables in Figure 3.5 are also top variables in Figure 3.4 so we can be fairly confident that they are truly crucial to predictive ability of our forest.

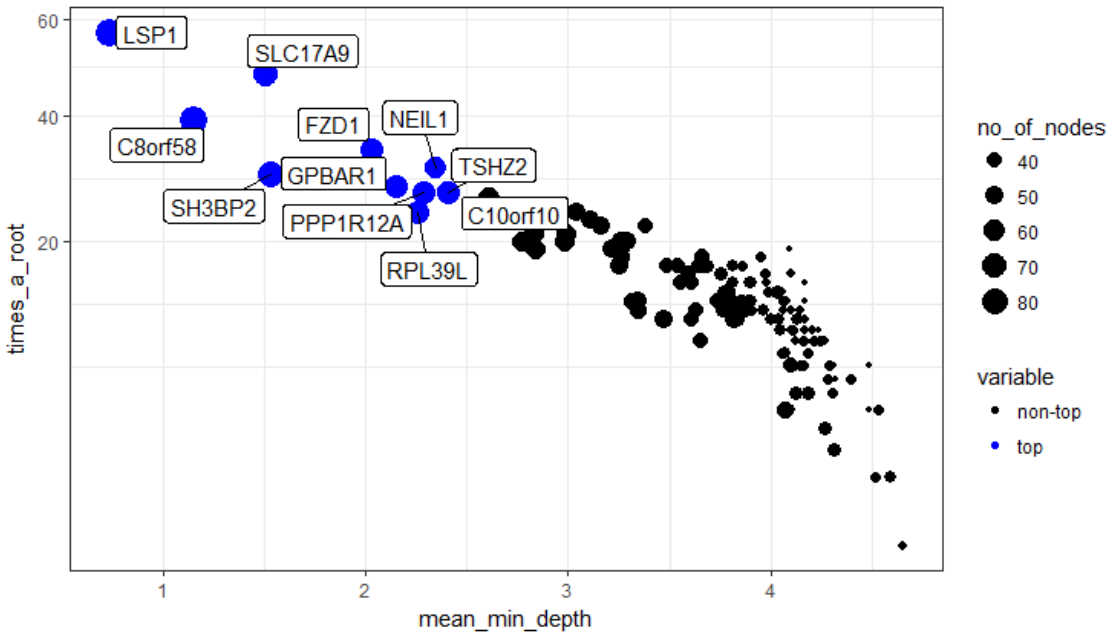


Figure 3.4: Multi-way importance plot for variables that were used for splitting in at least 30 trees. The x and y-axis measure mean minimal depth and number of splits at root of a variable and the size of points corresponds to the total number of nodes split on that variable. [Source: own research.]

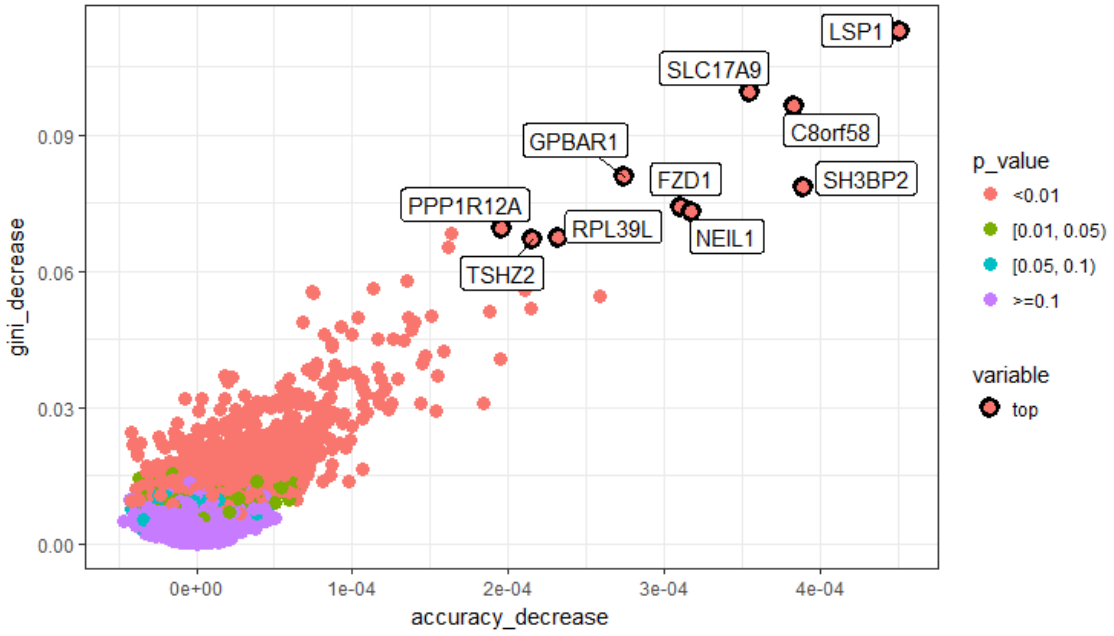


Figure 3.5: Multi-way importance plot for all variables. The x and y-axis measure decrease in accuracy when the variable is permuted and decrease in the Gini measure of node impurity resulting from splits on the variable, respectively. The color corresponds to variable significance which is derived from the number of nodes that split on it. [Source: own research.]

Generally, the multi-way importance plot offers a wide variety of possibilities so it can be hard to select the most informative one. One idea of overcoming this obstacle is to first explore relations between different importance measures to then select three that least agree with each other and use them in the multi-way importance plot to select top variables. The first is easily done by plotting selected importance measures pairwise against each other using `plot_importance_ggpairs` as in Figure 3.6. One could of course include all seven measures in the plot but by default p-value and the number of trees are excluded as both carry similar information as the number of nodes.

In Figure 3.6 we can see that all depicted measures are highly correlated (of course the correlation of any measure with mean minimal depth is negative as the latter is lowest for best variables), but some less than others. Notably, the points in plots including `accuracy_decrease` and `times_a_root` are most dispersed, so these may be the measures one should select. Moreover, regardless of which measures we compare, there is always only a handful of points that stand out.

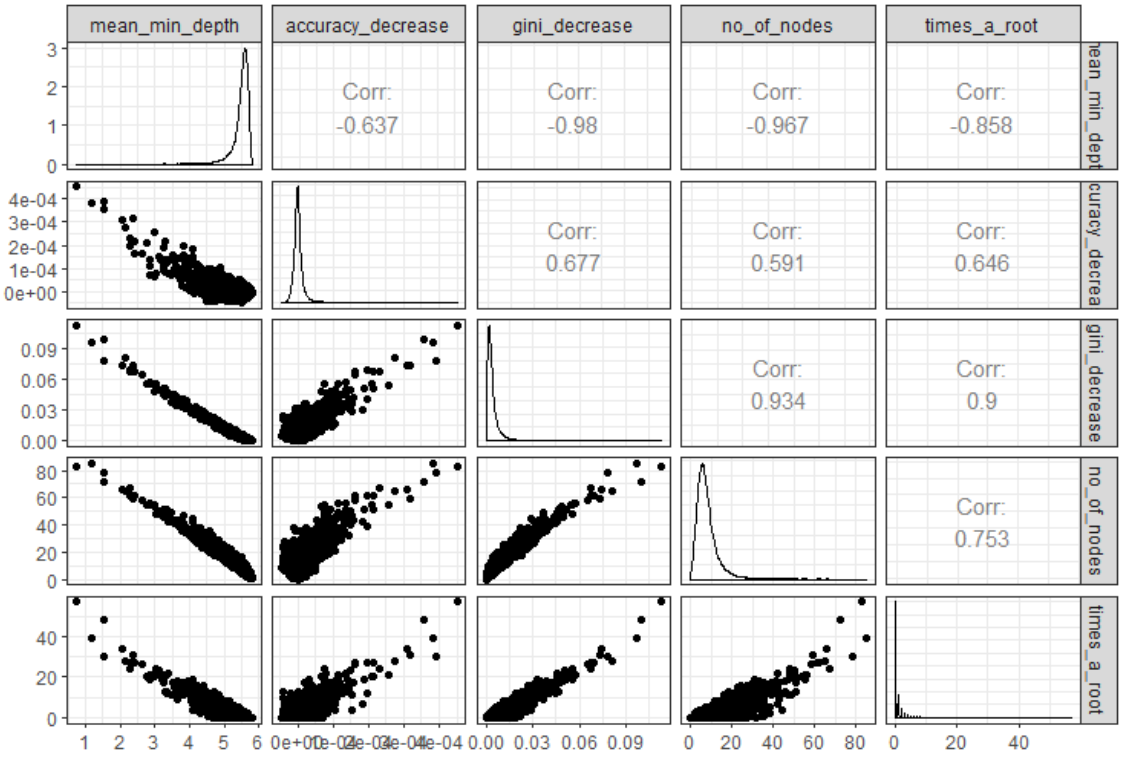


Figure 3.6: Pairwise comparison of five importance measures: below the diagonal each pair of measures is plotted against each other, pairwise correlation coefficients are given above the diagonal and density estimates are depicted on the diagonal. [Source: own research.]

In addition to scatter plots and correlation coefficients, Figure 3.6 also depicts density estimate for each importance measure – all of which are in this case very skewed. An attempt to eliminate this feature by plotting rankings instead of raw measures is implemented in the function `plot_importance_rankings` whose result is presented in Figure 3.7.

Density estimates in Figure 3.7 show that skewness was eliminated only for two out of five importance measures: `accuracy_decrease` and `gini_decrease`. The skewness of ranking distributions for other measures is most likely caused by frequent ties: e.g. 10370 predictors

were ranked last according to the `times_a_root` measure, as they were never used for splitting at the root. In general, ties are much less frequent for importance measures computed by the `randomForest` package also because they are measured on a continuous scale in contrast to the discrete one used for root, node and tree counts.

When comparing the rankings in Figure 3.7 we can see that for `accuracy_decrease` and `gini_decrease` they more or less agree for observations with low ranks according to those measures but strongly disagree for variables ranked higher according to `accuracy_decrease`. Also, observe that `times_a_root` and `no_of_nodes` differentiate much more between variables with low ranks than rankings according to other measures.

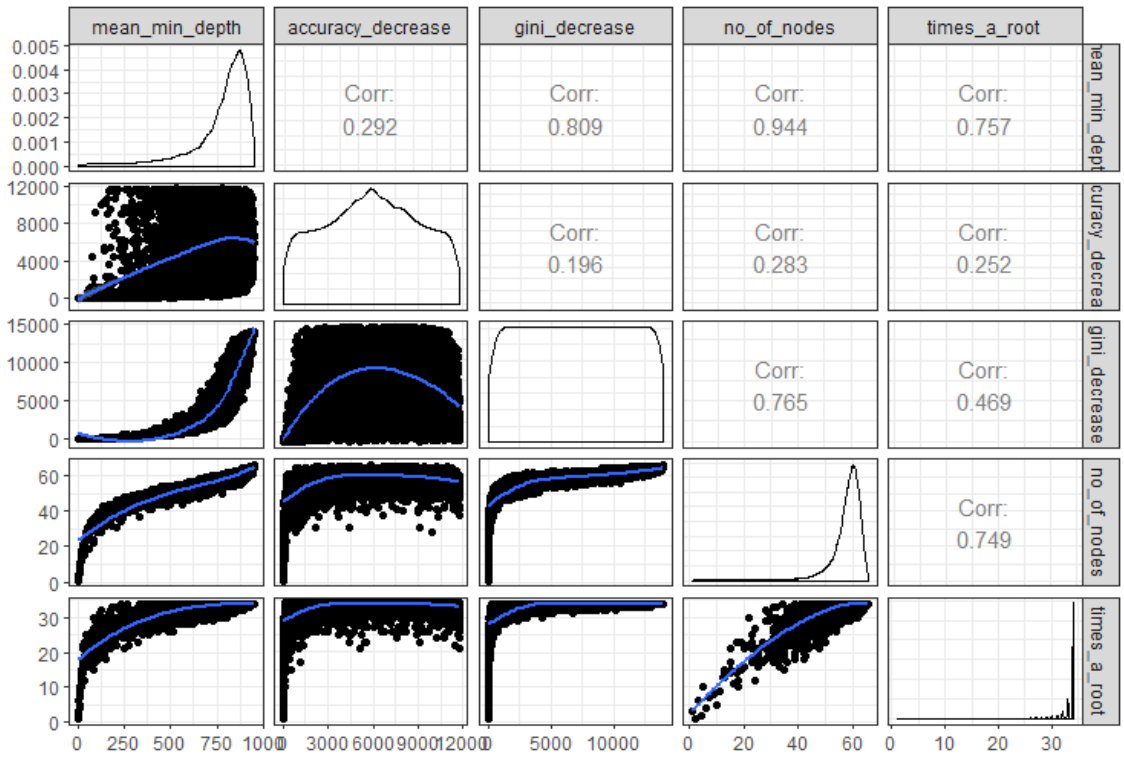


Figure 3.7: Pairwise comparison of rankings according to five importance measures: below the diagonal each pair of rankings is plotted against each other, pairwise correlation coefficients for rankings are given above the diagonal and density estimates are depicted on the diagonal. [Source: own research.]

3.4. Variable interactions

After selecting a set of most important variables we can investigate interactions with respect to them, i.e. splits appearing in maximal subtrees with respect to one of the variables selected. To extract the names of 20 most important variables according to both the mean minimal depth and number of trees in which a variable appeared, we pass our `importance_frame` to the function `important_variables` with parameter `k` $=$ 20 and selected importance measures passed to the argument `measures`. After using the resulting vector of predictor names together with the forest as arguments of `min_depth_interactions` we obtain a data frame containing the mean conditional minimal depth for each interaction, its number of occurrences and the

unconditional mean minimal depth of the second variable in the interaction (for both types of mean minimal depth we use the `"top_trees"` option).

Unfortunately, as we have many predictors and little observations, the trees in our forest are shallow and interactions are rare. Specifically, only two interactions appeared 3 times, 153 – two times and 7767 only one time. Thus, the mean conditional minimal depth is in this case meaningless, as it is calculated on up to three observations. We conclude, that to properly analyze interactions we need to grow another random forest with either increased number of trees or increased r (`mtry`) parameter so that more candidate variables will be tried at each split.

We choose to grow a new forest with $r = \lfloor p/3 \rfloor = 4709$ (this is the default r for regression) and after repeating the above calculations we save the result of `min_depth_interactions` as `interactions_frame`. In the new forest 16 of the interactions considered appeared at least 10 times in the forest, of which the most frequent, `SLC17A9:IFIT2` (this corresponds to splits on `IFIT2` in maximal subtrees of `SLC17A9`), a total of 29 times, which is a lot better than before and offers some ground for interaction analysis.

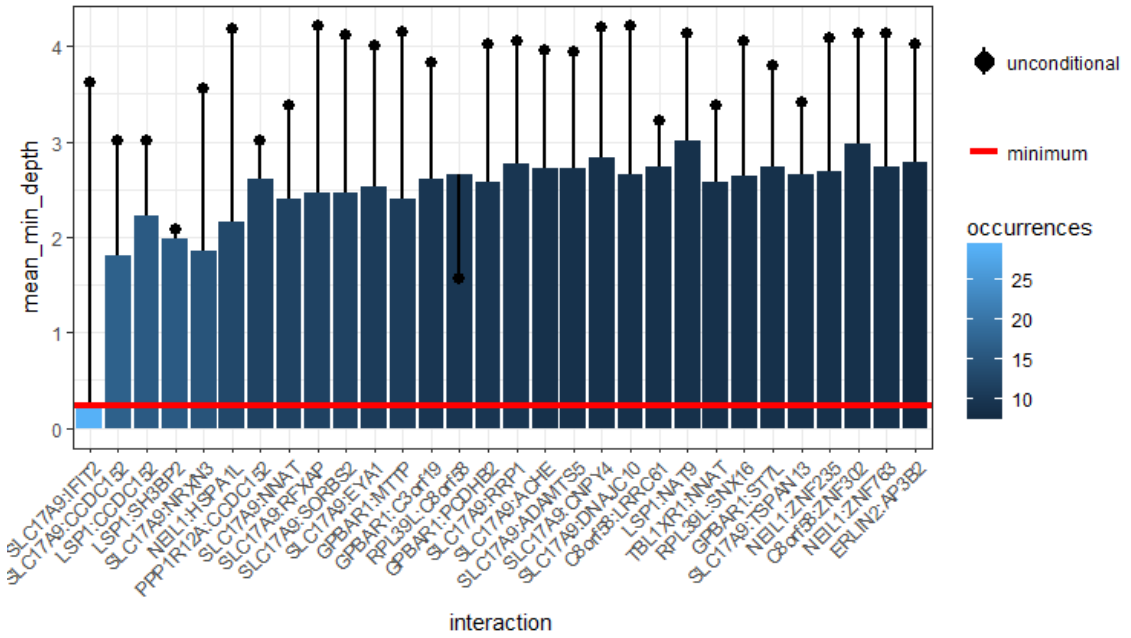


Figure 3.8: 30 most frequent interactions ordered by decreasing frequency. The bars report mean conditional minimal depth and the points – the unconditional mean minimal depth for the second variable in the interaction. Both are calculated using method (II) (`mean_sample` = `"top_trees` and `uncond_mean_sample` = `"top_trees`). [Source: own research.]

In Figure 3.8 we present the result of passing our `interactions_frame` to the function `plot_min_depth_interactions` with the default number of most frequent interactions plotted (`k` = 30). An interesting thing about the interactions depicted is that they usually do not include top 20 predictors used as conditioning variables as the non-conditioning variable (only 8 out of 30 do), which suggests that variables that are important in maximal subtrees with respect to top predictors are not important in whole trees. This is actually what we would expect from an interaction: some variable is only important conditional on another one and this in most pronounced in the case of `SLC17A9:IFIT2`, as `IFIT2` has mean minimal depth over 3.5 but in maximal subtrees of `SLC17A9` it is often used for splitting immediately after

the root.

Note that in Figure 3.8 the most frequent interaction has a lot lower mean conditional minimal depth than the rest – this is due to calculating the mean using method (II) that penalizes interactions that occur less frequently than the most frequent one. Of course, one can switch between methods (I), (II) and (III) of calculating the mean both of the conditional and unconditional minimal depth but each of them has its drawbacks as discussed in section 2.1.2 and we favour method (II). However, as `plot_min_depth_interactions` plots interactions by decreasing frequency the major drawback of calculating the mean only for relevant variables vanishes as interactions appearing only once but with conditional depth 0 will not be included in the plot anyway. Thus, we repeat the computation of means using method (III) and report the result in Figure 3.9.

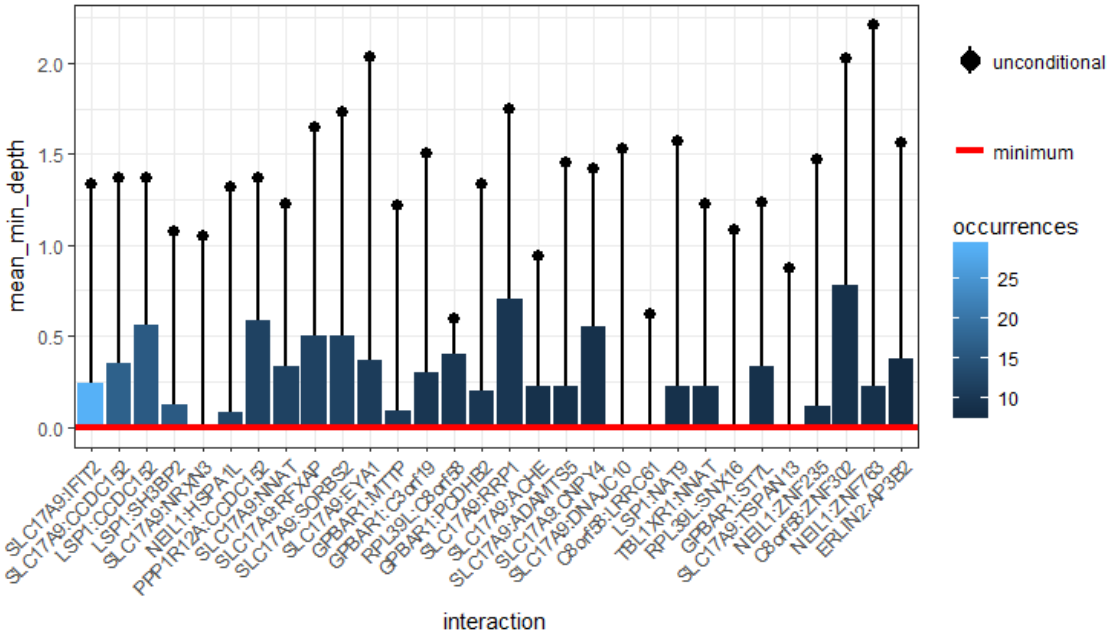


Figure 3.9: 30 most frequent interactions ordered by decreasing frequency. The bars report mean conditional minimal depth and the points – the unconditional mean minimal depth for the second variable in the interaction. Both are calculated using method (III) (`mean_sample` = `"relevant_trees"` and `uncond_mean_sample` = `"relevant_trees"`). [Source: own research.]

In Figure 3.9 we see, that all interactions have mean conditional minimal depth lower than 1 meaning that the split on second variable in the interaction is often the daughter of the root, which is by definition split on the first variable. This is not surprising as trees in our forest are shallow and there are not many possibilities for splits deep in maximal subtrees. From this plot we can also infer that, apart from the first one, the fourth, fifth and sixth interaction seem important, with mean close to zero and number of occurrences equal to 16, 15 and 13, respectively.

To further investigate the interactions mentioned above (`SLC17A9:IFIT2`, `LSP1:SH3BP2`, `SLC17A9:NRXN3` and `NEIL1:HSPA1L`) we use the function `plot_predict_interaction` to plot prediction of our forest on a grid of values for the components of each interaction. Unfortunately, neither of the plots offers any insight into how simultaneous changes of both variables in the interaction shape the prediction. We report the plot for `SLC17A9:IFIT2` in Figure 3.10 as an example – the other three look similar.

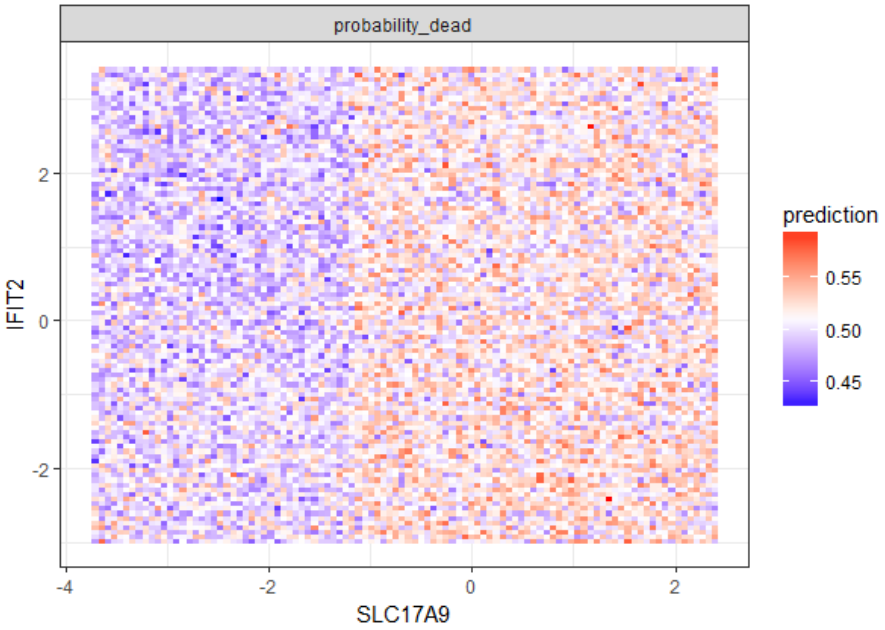


Figure 3.10: Prediction of the forest on a 100×100 grid of values of variables `SLC17A9` and `IFIT2`. The range of both axes corresponds to the ranges of the respective variables in our data set. [Source: own research.]

Clearly, higher values of `SLC17A9` lead to higher predicted probabilities of the patient dying within a year from diagnosis, but this effect does not change with the variable `IFIT2`. All things considered, this is not surprising – interaction of these two variables appears in only 29 out of 10000 trees in our forest so it cannot influence the prediction much. This is, as is frequency of interactions, a consequence of the structure of our high-p-low-n problem. We already modified our forest to be able to consider interactions at all, and we would have to do more in order to investigate their effect on forest prediction. A possible solution would be to build a forest using only a subset of our predictors, for example 50 top ones and all components of interactions that appeared at least five times in the forest.

3.5. Explain the forest

Instead of producing each plot separately, exploring the details of how each of them is made and what parameters need to be specified, we could have just passed out data, forest and `interactions = TRUE` to the function `explain_forest`. As in our analysis we mostly relied on default values of parameters, so using the wrapper function would produce an HTML document with all of the plots we discussed except Figure 3.3 and 3.9 as they were duplicates with slightly changed parameters. Moreover, the resulting HTML document would contain interactive versions of both `importance_frame` and `interactions_frame` so that the user could sort variables and interactions according to different importance measures or easily find information concerning predictors of interest.

As we do not want to duplicate our results, we do not report the document produced by `explain_forest` for our forest. However, in Appendix B we include such documents for two other data sets. The first concerns breast cancer and the problem there is to classify the

observations into one of five groups corresponding to cancer subtypes on the basis of gene expression data. The second data set contains information on students and their performance on the PISA cognitive test. We build a forest that predicts the quantitative score in mathematics of the students based on their various characteristics. Obviously, we modified somewhat the HTML documents obtained with the function `explain_forest` in order to fit the format of our paper. Notably, instead of interactive tables we show snippets of their non-interactive counterparts and move them to the beginning of the document to take less space.

Summary

As I have shown throughout this thesis, random forests, though inherently complex, allow for at least two approaches of knowledge extraction: first, one can analyze how the prediction of the forest changes after various modifications of the data (e.g., inputs permutation) and second, one can come up with a number of useful summary statistics concerning the structure of a single tree and then aggregate them over the whole forest.

In particular, I have thoroughly discussed the minimal depth statistic and various ways of calculating its mean, each substantially different if the forest consists of shallow trees and variables are often not used for splitting in a tree at all. Such considerations lead me to believe that it may be useful to investigate the whole distribution of minimal depth in the forest instead of just looking at its mean.

To address this and other issues I provide new ways of visualizing a random forest in R by introducing the `randomForestExplainer` package that calculates various statistics measuring the importance of variables and their interactions in a forest. Although the package mainly utilizes existing variable importance measures with a few simple extensions, it possesses a wide variety of graphical capabilities, all based on the `ggplot2` package. Out of 11 functions of the new package, six produce plots of which each presents predictors used in the forest from a different point of view. In my opinion, this focus on visualization is the main advantage of the new package.

Furthermore, the flagship function `explain_forest` gives the user immediate insight into the results that can be obtained by the package for a given forest without almost any need of getting to know how the package works. The resulting HTML document presents the results of all plotting functions of `randomForestExplainer` in turn, under default settings, allowing the user to immediately asses the usefulness of each of them in the particular application.

When it comes to the example of predicting if the patient will survive one year from diagnosis of glioblastoma cancer, I found that out of a multitude of variables measuring gene expression only a small group seems driving the prediction of the forest. Moreover, this finding is consistent across plots using different variable importance measures for comparison. My analysis also helps in distinguishing which predictors are almost entirely useless when it comes to predicting the outcome so an analysis excluding them could be performed.

All in all, I believe that the ideas presented in this thesis can potentially help explain predictions given by random forests. In other words, `randomForestExplainer` will hopefully lighten the random forest black box so that it will be of some more accessible shade of grey.

Appendix A

List of functions available in `randomForestExplainer`

Below we list all functions available in the package `randomForestExplainer`. For each function we include all its parameters and their default values, if any, and explain the nature of objects that have to be passed to the function as parameters that have no default value:

(1) `min_depth_distribution(forest)` – the only argument is a `randomForest` object,

(2) `plot_min_depth_distribution(min_depth_frame, k = 10, min_no_of_trees = 0, mean_sample = "top_trees", mean_scale = FALSE, mean_round = 2, main = "Distribution of minimal depth and its mean")` – the first argument is a result of function (1),

(3) `measure_importance(forest, mean_sample = "top_trees")` – the first argument is a `randomForest` object grown with option `localImp = TRUE`,

(4) `important_variables(importance_frame, k = 15, measures = names(importance_frame)[2:5], ties_action = "all")` – the first argument is a result of function (3),

(5) `plot_multi_way_importance(importance_frame, x_measure = "mean_min_depth", y_measure = "times_a_root", size_measure = NULL, min_no_of_trees = 0, no_of_labels = 10, main = "Multi-way importance plot")` – the first argument is a result of function (3),

(6) `plot_importance_ggpairs(importance_frame, measures = names(importance_frame)[c(2, 4, 5, 3, 7)], main = "Relations between measures of importance")` – the first argument is a result of function (3),

(7) `plot_importance_rankings(importance_frame, measures = names(importance_frame)[c(2, 4, 5, 3, 7)], main = "Relations between rankings according to different measures")` – the first argument is a result of function (3),

(8) `min_depth_interactions(forest, vars, mean_sample = "top_trees", uncond_mean_sample = mean_sample)` – the first argument is a `randomForest` object, the second one is a vector of names of (possibly all) predictors,

(9) `plot_min_depth_interactions(interactions_frame, k = 30, main = paste0("Mean minimal depth for ", paste0(k, " most frequent interactions")))` – the first argument is a result of function (8),

(10) `plot_predict_interaction(forest, data, variable1, variable2, grid = 100, main = paste0("Prediction of the forest for different values of ", paste0( variable1, paste0(" and ", variable2))))` – the first argument is a `randomForest` object, the second one is a data frame on which `forest` was trained, two subsequent ones are names of predictors,

(11) `explain_forest(forest, interactions = FALSE, data = NULL, vars = NULL, no_of_pred_plots = 3, pred_grid = 100, measures = if (forest$type == "classification") c("mean_min_depth", "accuracy_decrease", "gini_decrease", "no_of_nodes", "times_a_root") else c("mean_min_depth", "mse_increase", "node_purity_increase", "no_of_nodes", "times_a_root"))` – the first argument is a `randomForest` object.

Appendix B

Additional examples

In this appendix we present documents produced by the `randomForestExplainer` function `explain_forest` applied to random forests grown on two different data sets.

B.1. Multi-label classification: breast cancer data

The first data set comes from The Cancer Genome Atlas Project [18] and contains observations on 100 patients that have been diagnosed with breast cancer. We build a random forest that aims at predicting the subtype of cancer (one of five: `Basal`, `Her2`, `LumA`, `LumB` or `Normal`) using 730 predictors containing numerical gene expression. Thus, this problem is similar to our glioblastoma example presented in Chapter 3 as it is also a classification problem with many more predictors than observations leading to a forest with shallow trees.

We report the document produced by `randomForestExplainer` on pages 58–62.

B.2. Regression: PISA data

The second data set comes from the `PISA2012lite` package and is a subset of the PISA 2012 OECD database which contains information on students' performance on various cognitive tests and their personal characteristics. We use a relatively small sample that contains 12473 observations on students from four members of the Visegrad Group (i.e., Czech Republic, Hungary, Poland and Slovak Republic). We build a random forest that aims at predicting the quantitative score in mathematics `PV1MATH` using 12 characteristics of a student such as her gender or index of wealth of her family. Clearly, this is a typical regression problem with n considerably bigger than p.

We report the document produced by `randomForestExplainer` on pages 63–67.

A graphical summary of your random forest

randomForestExplainer

July 08, 2017

Details of your forest

```
##
## Call:
##  randomForest(formula = SUBTYPE ~ ., data = brca, ntree = 10000,      localImp = TRUE)
##                Type of random forest: classification
##                      Number of trees: 10000
## No. of variables tried at each split: 27
##
##         OOB estimate of  error rate: 19%
## Confusion matrix:
##        Basal Her2 LumA LumB Normal class.error
## Basal     23    0    0    0      0        0.00
## Her2       3    0    1    2      0        1.00
## LumA       0    0   47    0      0        0.00
## LumB       0    0    9   11      0        0.45
## Normal     1    0    2    1      0        1.00
```

Importance measures

Below you can see how measures of importance for variables in the forest look like:

```
##   variable mean_min_depth no_of_nodes accuracy_decrease gini_decrease
## 1    AANAT       5.772162          89      4.542379e-05    0.03006395
## 2   AARSD1       5.640127         152      1.617355e-04    0.05163999
## 3     AATF       5.540638         181      1.314132e-04    0.05696943
##   no_of_trees times_a_root   p_value
## 1          89            1 1.0000000
## 2         152            0 0.9955646
## 3         181            0 0.6579973
```

Minimal depth interactions frame

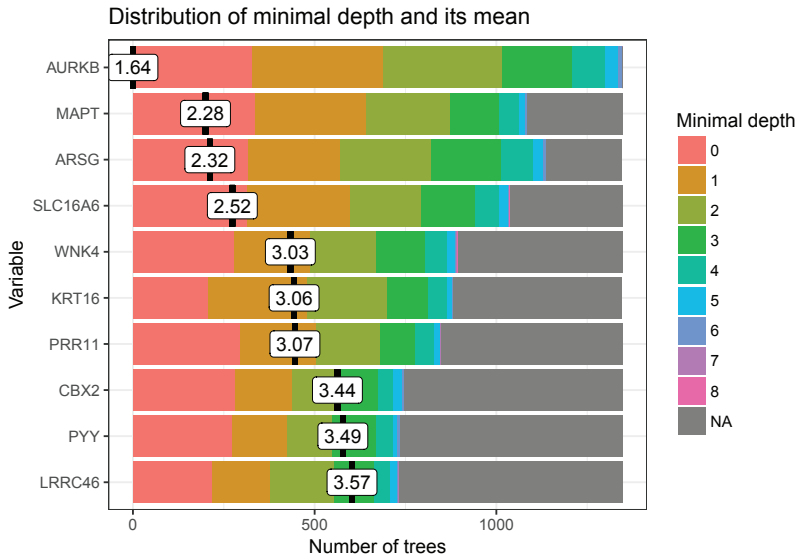
The data used for plotting mean minimal depth for interactions is in the following table:

```
##   variable root_variable mean_min_depth occurrences interaction
## 1    AANAT          ARSG       3.573993           1  ARSG:AANAT
## 2    AANAT         AURKB       3.346270          11 AURKB:AANAT
## 3    AANAT          CBX2       3.820170           3  CBX2:AANAT
##   uncond_mean_min_depth
## 1              5.772162
## 2              5.772162
## 3              5.772162
```


Distribution of minimal depth

The plot below shows the distribution of minimal depth among the trees of your forest. Note that:

- the mean of the distribution is marked by a vertical bar with a value label on it (the scale for it is different than for the rest of the plot),
- the scale of the X axis goes from zero to the maximum number of trees in which any variable was used for splitting.



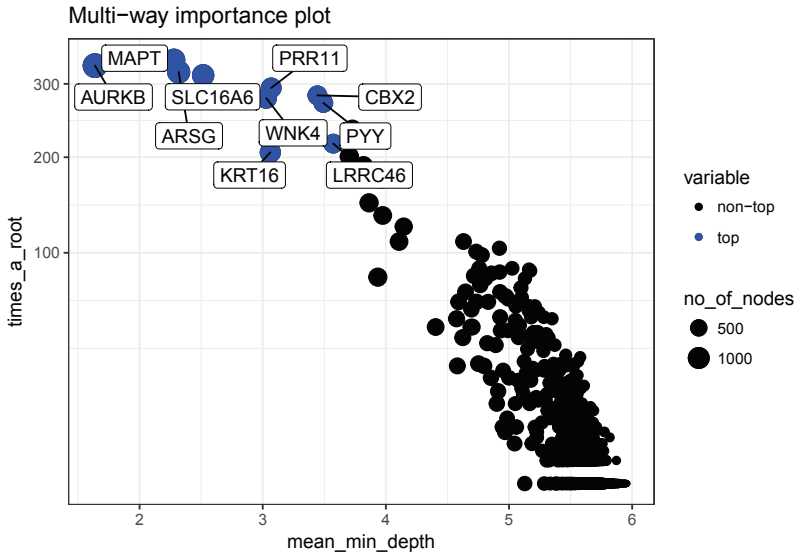
Minimal depth for a variable in a tree equals to the depth of the node which splits on that variable and is the closest to the root of the tree. If it is low than a lot of observations are divided into groups on the basis of this variable

Multi-way importance plot

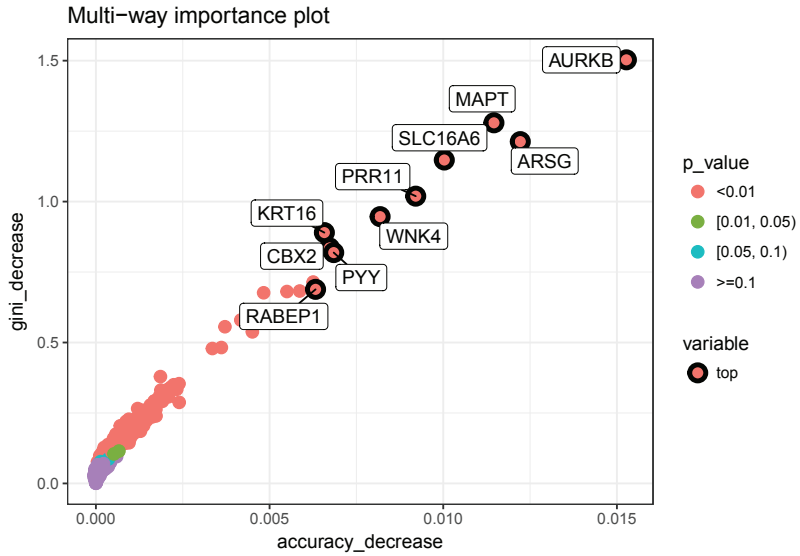
The multi-way importance plot shows the relation between three measures of importance and labels 10 variables which scored best when it comes to these three measures (i.e. for which the sum of the ranks for those measures is the lowest).

The first multi-way importance plot focuses on three importance measures that derive from the structure of trees in the forest:

- mean depth of first split on the variable,
- number of trees in which the root is split on the variable,
- the total number of nodes in the forest that split on that variable.

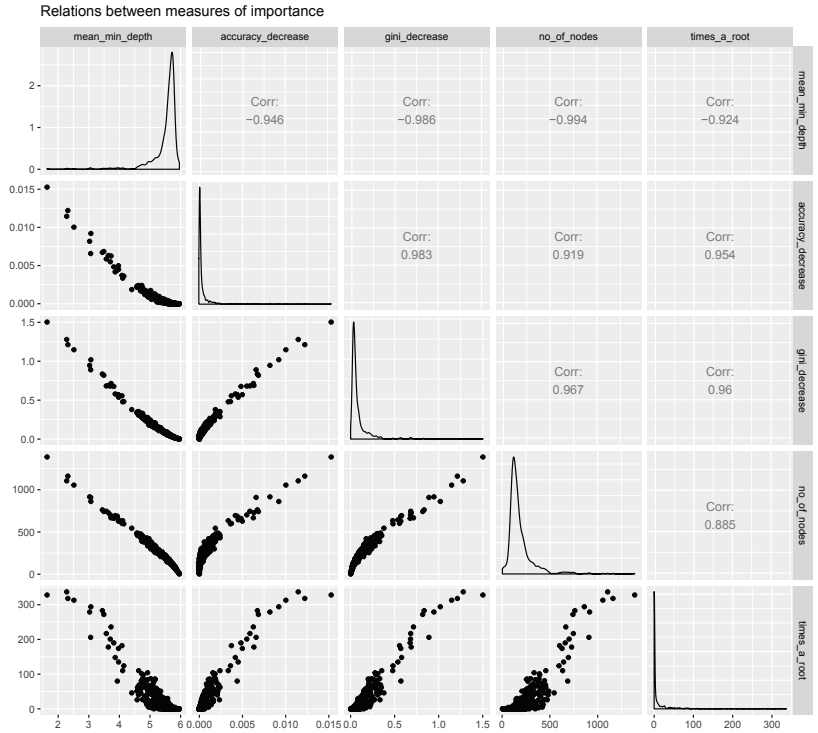


The second multi-way importance plot shows two importance measures that derive from the role a variable plays in prediction: with the additional information on the p-value based on a binomial distribution of the number of nodes split on the variable assuming that variables are randomly drawn to form splits (i.e. if a variable is significant it means that the variable is used for splitting more often than would be the case if the selection was random).



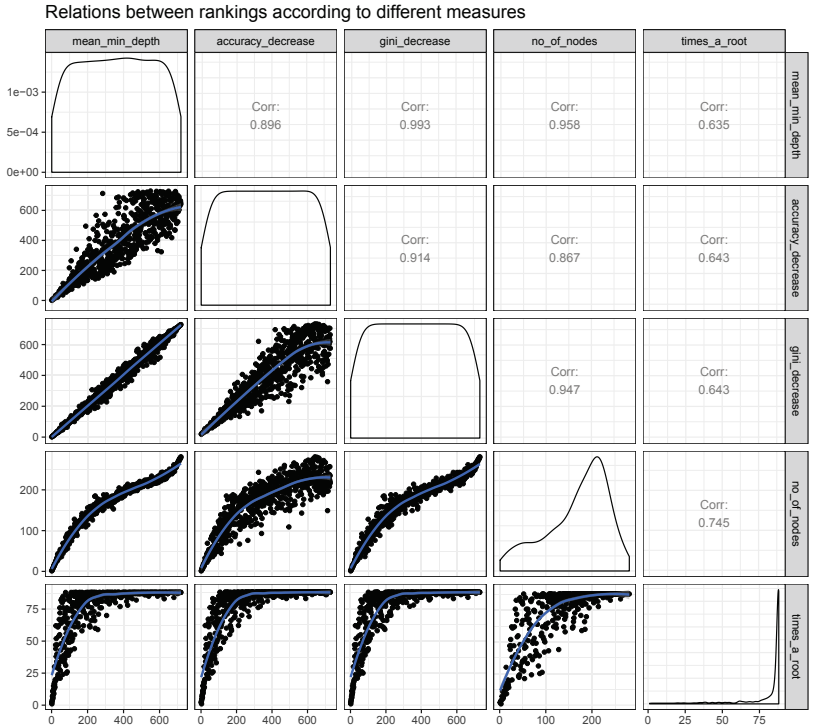
Compare importance measures

The plot below shows bilateral relations between the following importance measures: , if some variables are strongly related to each other it may be worth to consider focusing only on one of them.



Compare rankings of variables

The plot below shows bilateral relations between the rankings of variables according to chosen importance measures. This approach might be useful as rankings are more evenly spread than corresponding importance measures. This may also more clearly show where the different measures of importance disagree or agree.



Variable interactions

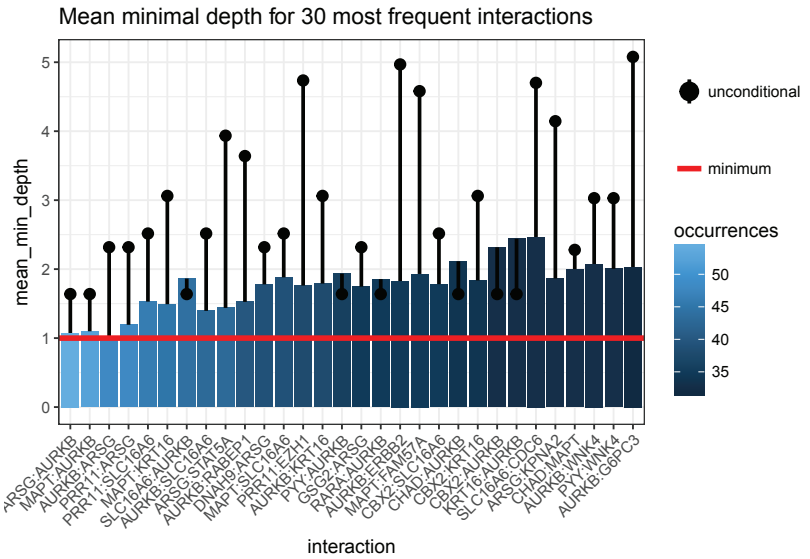
Conditional minimal depth

The plot below reports 30 top interactions according to mean of conditional minimal depth – a generalization of minimal depth that measures the depth of the second variable in a tree of which the first variable is a root (a subtree of a tree from the forest). In order to be comparable to normal minimal depth 1 is subtracted so that 0 is the minimum.

For example value of 0 for interaction x:y in a tree means that if we take the highest subtree with the root splitting on x then y is used for splitting immediately after x (minimal depth of x in this subtree is 1). The values presented are means over all trees in the forest.

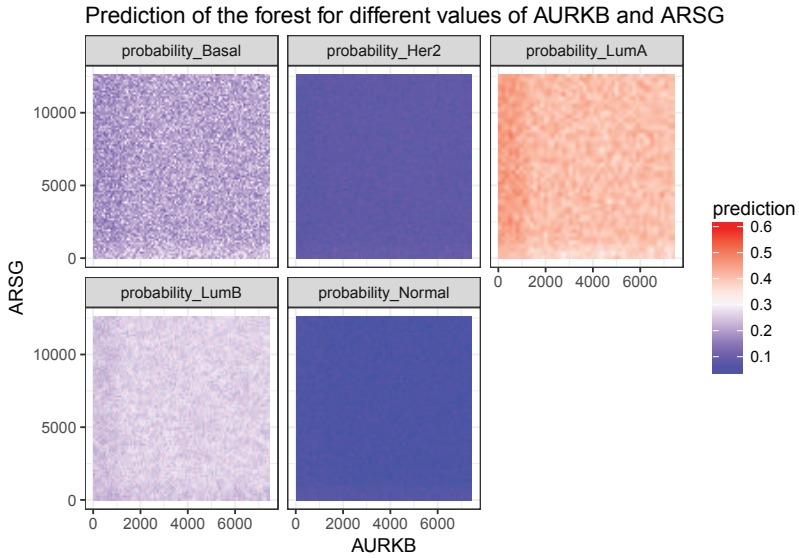
Note that:

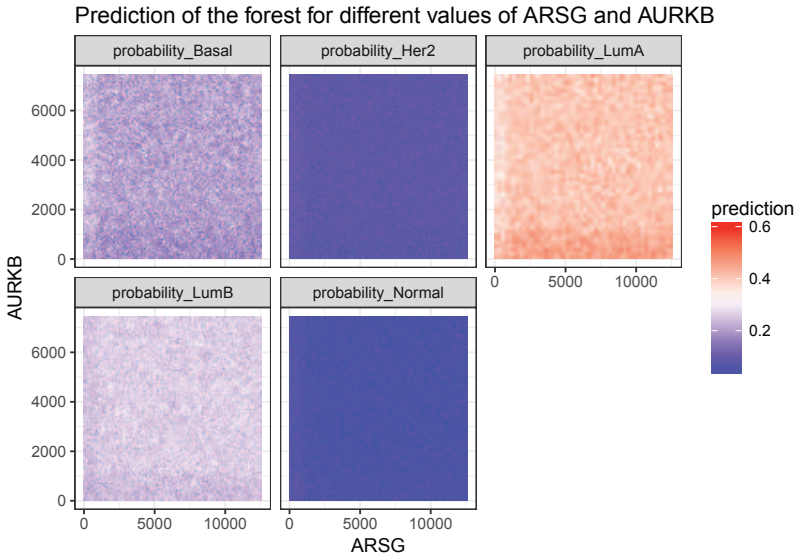
- the plot shows only 30 interactions that appeared most frequently,
- the horizontal line shows the minimal value of the depicted statistic among interactions for which it was calculated,
- the interactions considered are ones with the following variables as first (root variables): and all possible values of the second variable.

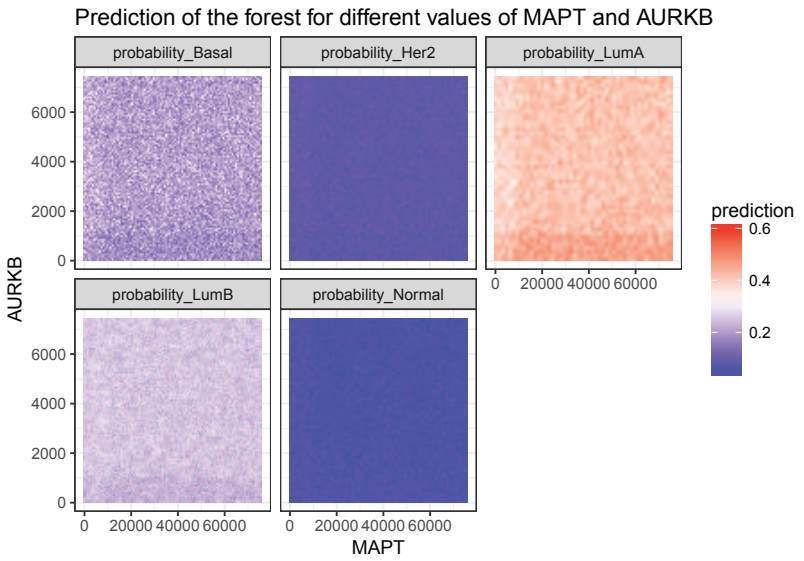


Prediction on a grid

The plots below show predictions of the random forest depending on values of components of an interaction (the values of remaining predictors are sampled from their empirical distribution) for up to 3 most frequent interactions that consist of two numerical variables.


Prediction of the forest for different values of AURKB and ARSG
probability_Basal
probability_Her2
probability_LumA
probability_LumB
probability_Normal
ARSG
AURKB
prediction
0.6
0.5
0.4
0.3
0.2
0.1
0
5000
10000
2000
4000
6000


Prediction of the forest for different values of ARSG and AURKB
probability_Basal
probability_Her2
probability_LumA
probability_LumB
probability_Normal
AURKB
ARSG
prediction
0.6
0.4
0.2
0
2000
4000
6000
5000
10000


Prediction of the forest for different values of MAPT and AURKB
probability_Basal
probability_Her2
probability_LumA
probability_LumB
probability_Normal
AURKB
MAPT
prediction
0.6
0.4
0.2
0
2000
4000
6000
20000
40000
60000

A graphical summary of your random forest

randomForestExplainer

July 08, 2017

Details of your forest

```
##
## Call:
##  randomForest(formula = PV1MATH ~ ., data = pisa, localImp = TRUE)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
##           Mean of squared residuals: 7293.245
##                     % Var explained: 18.91
```

Importance measures

Below you can see how measures of importance for variables in the forest look like:

```
##   variable mean_min_depth no_of_nodes mse_increase node_purity_increase no_of_trees times_a_root
## 1      CNT          1.414       62408     1477.227              5861251         500          109
## 2   ISCEDO          1.216       19526     1340.554              6915233         500          155
## 3     OECD             NA           0        0.000                    0           0            0
##        p_value
## 1 1.472617e-51
## 2 1.000000e+00
## 3 1.000000e+00
```

Minimal depth interactions frame

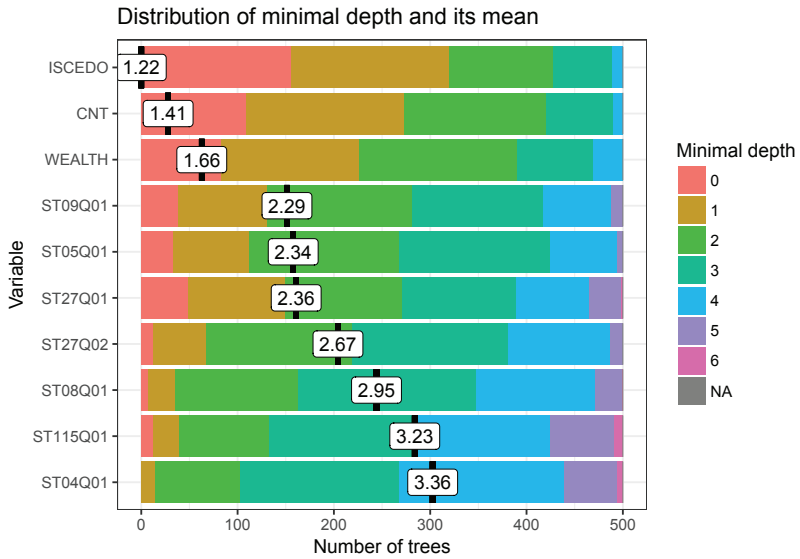
The data used for plotting mean minimal depth for interactions is in the following table:

```
##   variable root_variable mean_min_depth occurrences interaction uncond_mean_min_depth
## 1      CNT           CNT       3.574100         485     CNT:CNT                 1.414
## 2      CNT        ISCEDO       1.892594         490  ISCEDO:CNT                 1.414
## 3      CNT          OECD            NaN           0    OECD:CNT                 1.414
```


Distribution of minimal depth

The plot below shows the distribution of minimal depth among the trees of your forest. Note that:

- the mean of the distribution is marked by a vertical bar with a value label on it (the scale for it is different than for the rest of the plot),
- the scale of the X axis goes from zero to the maximum number of trees in which any variable was used for splitting.



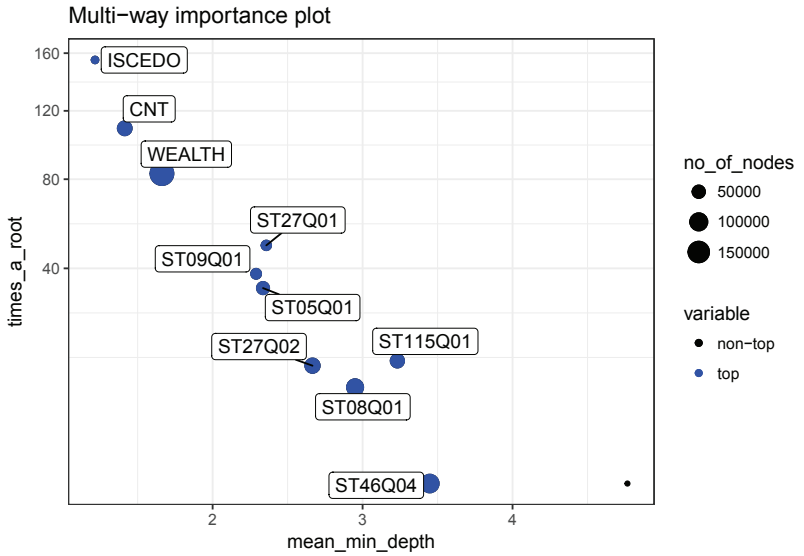
Minimal depth for a variable in a tree equals to the depth of the node which splits on that variable and is the closest to the root of the tree. If it is low than a lot of observations are divided into groups on the basis of this variable

Multi-way importance plot

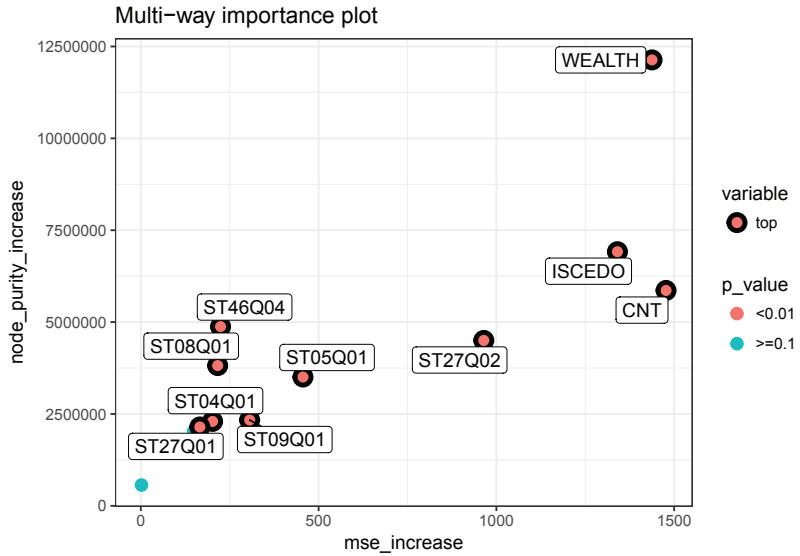
The multi-way importance plot shows the relation between three measures of importance and labels 10 variables which scored best when it comes to these three measures (i.e. for which the sum of the ranks for those measures is the lowest).

The first multi-way importance plot focuses on three importance measures that derive from the structure of trees in the forest:

- mean depth of first split on the variable,
- number of trees in which the root is split on the variable,
- the total number of nodes in the forest that split on that variable.

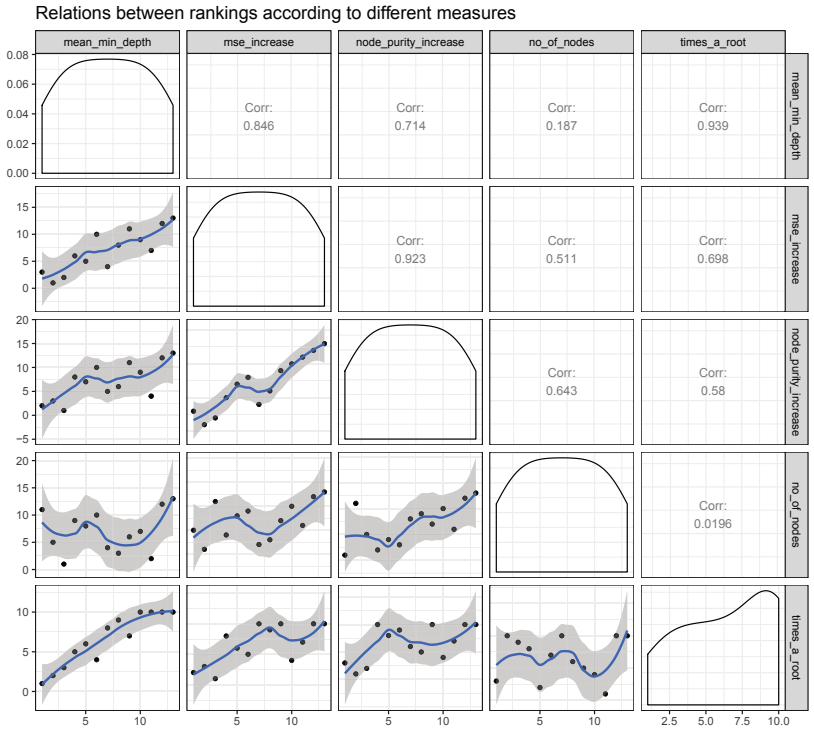


The second multi-way importance plot shows two importance measures that derive from the role a variable plays in prediction: with the additional information on the p-value based on a binomial distribution of the number of nodes split on the variable assuming that variables are randomly drawn to form splits (i.e. if a variable is significant it means that the variable is used for splitting more often than would be the case if the selection was random).



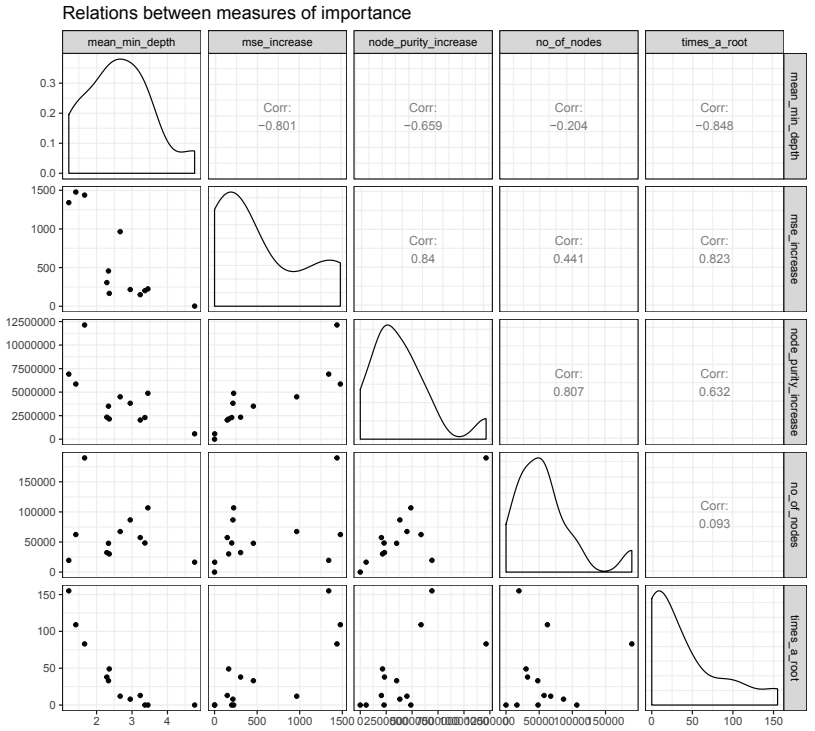
Compare rankings of variables

The plot below shows bilateral relations between the rankings of variables according to chosen importance measures. This approach might be useful as rankings are more evenly spread than corresponding importance measures. This may also more clearly show where the different measures of importance disagree or agree.



Compare importance measures

The plot below shows bilateral relations between the following importance measures: , if some variables are strongly related to each other it may be worth to consider focusing only on one of them.



Variable interactions

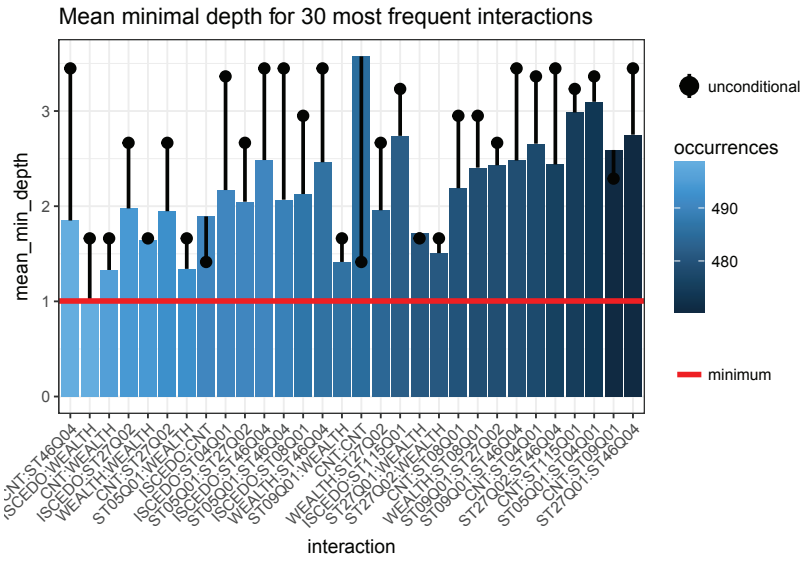
Conditional minimal depth

The plot below reports 30 top interactions according to mean of conditional minimal depth – a generalization of minimal depth that measures the depth of the second variable in a tree of which the first variable is a root (a subtree of a tree from the forest). In order to be comparable to normal minimal depth 1 is subtracted so that 0 is the minimum.

For example value of 0 for interaction x:y in a tree means that if we take the highest subtree with the root splitting on x then y is used for splitting immediately after x (minimal depth of x in this subtree is 1). The values presented are means over all trees in the forest.

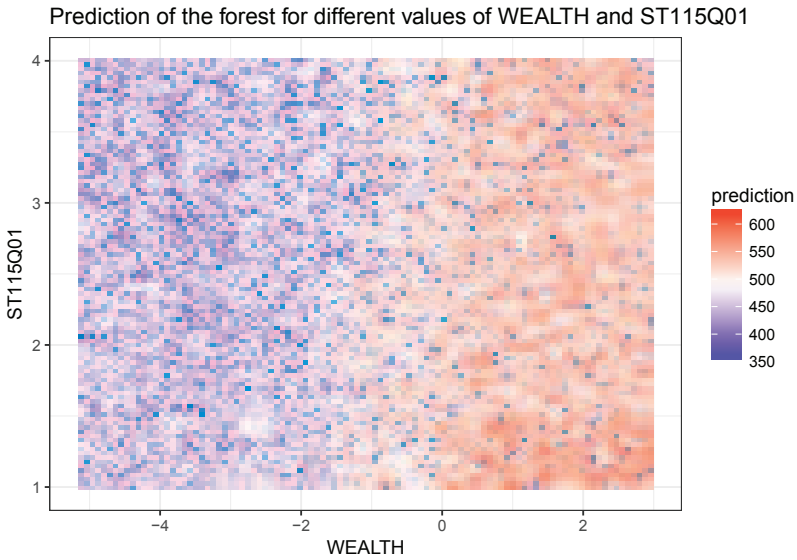
Note that:

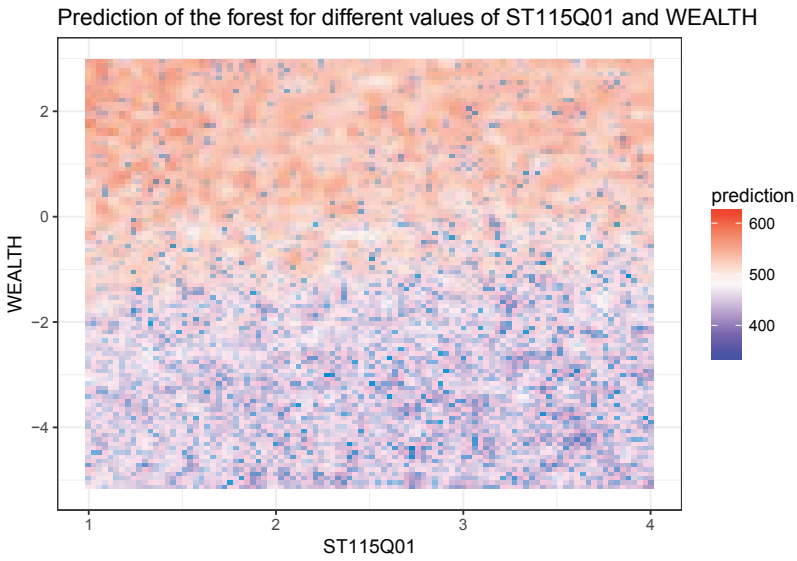
- the plot shows only 30 interactions that appeared most frequently,
- the horizontal line shows the minimal value of the depicted statistic among interactions for which it was calculated,
- the interactions considered are ones with the following variables as first (root variables): and all possible values of the second variable.



Prediction on a grid

The plots below show predictions of the random forest depending on values of components of an interaction (the values of remaining predictors are sampled from their empirical distribution) for up to 3 most frequent interactions that consist of two numerical variables.


Prediction of the forest for different values of WEALTH and ST115Q01
ST115Q01
WEALTH
prediction
600
550
500
450
400
350


Prediction of the forest for different values of ST115Q01 and WEALTH
WEALTH
ST115Q01
prediction
600
500
400

List of Figures

List of Algorithms

Bibliography

[1] André Altmann, Laura Tolosi, Oliver Sander, and Thomas Lengauer. Permutation importance: a corrected feature importance measure. *Bioinformatics*, 26(10):1340–1347, 2010.

[2] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[3] Leo Breiman. Statistical modeling: The two cultures (with comments and a rejoinder by the author). *Statistical Science*, 16(3):199–231, 2001.

[4] Chao Chen, Andy Liaw, and Leo Breiman. Using random forest to learn imbalanced data. Technical report, Department of Statistics, University of Berkeley, 2004.

[5] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2016.

[6] Torsten Hothorn, Kurt Hornik, and Achim Zeileis. Unbiased recursive partitioning: A conditional inference framework. *Journal of Computational and Graphical Statistics*, 15(3):651–674, 2006.

[7] Hemant Ishwaran. Variable importance in binary regression trees and forests. *Electronic Journal of Statistics*, 1:519–537, 2007.

[8] Hemant Ishwaran, Udaya B. Kogalur, Eiran Z. Gorodeski, Andy J. Minn, and Michael S. Lauer. High-dimensional variable selection for survival data. *Journal of the American Statistical Association*, 105(489):205–217, 2010.

[9] Hemant Ishwaran, Udaya B. Kogular, Eugene H. Blackstone, and Michael S. Lauer. Random survival forests. *The Annals of Applied Statistics*, 2(3):841–860, 2008.

[10] Jacek Koronacki and Jan Ćwik. *Statystyczne systemy uczące się*. Akademicka Oficyna Wydawnicza EXIT, Warszawa, 2nd edition, 2008.

[11] Marcin Kosiński and Przemysław Biecek. RTCGA: The cancer genome atlas data integration. *R package version 1.2.5*, 2016.

[12] Andy Liaw and Matthew Wiener. Classification and Regression by randomForest. *R News*, 2(3):18–22, 2002.

[13] Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. "Why Should I Trust You?": Explaining the predictions of any classifier. In *KDD '16 Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1135–1144, 2016.

[14] Ivan Sanchez, Tim Rocktaschel, Sebastian Riedel, and Sameer Singh. Towards extracting faithful and descriptive representations of latent variable models. In *AAAI Spring Syposium on Knowledge Representation and Reasoning (KRR): Integrating Symbolic and Neural Approaches*, 2015.

[15] Galit Shmueli. To explain or to predict? *Statistical Science*, 25(3):289–310, 2010.

[16] Katarzyna Sobiczewska. Analiza metod uczenia maszynowego wykorzystywanych w budowaniu sygnatur genetycznych. Master's thesis, Politechnika Warszawska, 2016.

[17] Carolin Strobl, Anne-Laure Boulesteix, Achim Zeileis, and Torsten Hothorn. Bias in random forest variable importance measures: Illustrations, sources and a solution. *BMC Bioinformatics*, 8(25):1471–2105, 2007.

[18] John Weinstein, Eric Collission, Gordon Mills, Kenna Mills Shaw, Brad Ozenberger, Kyle Ellrott, Ilya Shmulevich, Chris Sander, and Joshua Stuart. Cancer genome atlas research network. *Nature Genetics*, 45:1113–1120, 2013.

[19] Hadley Wickham, Dianne Cook, and Heike Hofmann. Visualizing statistical models: Removing the blindfold. *Statistical Analysis and Data Mining*, 8(4):203–225, 2015.