Journal of Statistical Software

July 2018, Volume 85, Issue 12. doi: 10.18637/jss.v085.i12

rmcfs: An R Package for Monte Carlo Feature Selection and Interdependency Discovery

Michał Dramiński
IPI PAN

Jacek Koronacki
IPI PAN

Abstract

We describe the R package **rmcfs** that implements an algorithm for ranking features from high dimensional data according to their importance for a given supervised classification task. The ranking is performed prior to addressing the classification task per se. This R package is the new and extended version of the MCFS (Monte Carlo feature selection) algorithm where an early version was published in 2005. The package provides an easy R interface, a set of tools to review results and the new ID (interdependency discovery) component. The algorithm can be used on continuous and/or categorical features (e.g., gene expression and phenotypic data) to produce an objective ranking of features with a statistically well-defined cutoff between informative and non-informative ones. Moreover, the directed ID graph that presents interdependencies between informative features is provided.

Keywords: MCFS-ID, feature selection, high-dimensional problems, Java, R, ID graph.

1. Introduction

In the area of feature ranking and selection for high-dimensional supervised classification, a very significant progress has been achieved in the past two decades. For a brief account, up to 2002, see Dudoit and Fridlyand (2003) and for an extensive survey and somewhat later developments see Saeys, Inza, and Larrañaga (2007). Without coming to details let us note that feature selection can be *wrapped* around the classifier construction or directly built (*embedded*) into the classifier construction, and not performed prior to addressing the classification task per se by *filtering* out noisy features first and keeping only informative ones for building a classifier. An early and successful method with embedded feature selection included, not mentioned by Saeys *et al.* (2007), was developed by Tibshirani and others (see Tibshirani, Hastie, Narasimhan, and Chu 2002 and Tibshirani, Hastie, Narasimhan, and Chu

2003). More recently and within non-filter approaches, a Bayesian technique of automatic relevance determination, the use of support vector machines, and the use of ensembles of classifiers, all these either alone or in combination, have proved promising. For further details see Li, Campbell, and Tipping (2002), Lu, Devos, Suykens, Arús, and Huffel (2007), Chrysostomou, Chen, and Liu (2008) and the literature therein.

Moreover, the last developments by the late Leo Breiman deserve special attention. In his random forests (RFs), he proposed to make use of the so-called variable (i.e., feature) importance for feature selection. Determination of the importance of the variable is not necessary for random forest construction, but it is a subroutine performed in parallel to building the forest (cf. Breiman and Cutler 2008). Ranking features by variable importance can thus be considered to be a by-product of building the classifier. At the same time, nothing prevents one from using such variable importances within, say, the embedded approach; cf., e.g., Díaz-Uriarte and De Andres (2006). In any case, feature selection by measuring variable importance in random forests should be seen as a very promising method, albeit under one proviso. Namely, the problem with variable importance as originally defined is that it is biased towards variables with many categories; cf. Strobl, Boulesteix, Zeileis, and Hothorn (2007), Archer and Kimes (2008), Nicodemus, Malley, Strobl, and Ziegler (2010). Accordingly, proper debiasing is needed, in order to obtain true ranking of features; cf. Strobl, Boulesteix, Kneib, Augustin, and Zeileis (2008). And, however sound such debiasing may be, it incurs much additional computational cost. For an excellent and recent survey on RFs, their properties and capabilities, see Ziegler and König (2014).

Most recently, much work has been done to: (i) give embedded feature selection procedures, in particular those used within RFs (whether biased or unbiased), a clear statistical meaning; and (ii) better understand RFs' capability of discovering interdependencies between features; cf. Paul and Dupont (2015) (see also Huynh-Thu, Saeys, Wehenkel, and Geurts (2012) and the literature therein for (i)) and Wright, Ziegler, and König (2016) and the literature therein for (ii).

In 2005, a novel, effective and reliable method for ranking features according to their importance for a given supervised classification task has been introduced by Dramiński, Koronacki, and Komorowski (2005). The method, which relies on a Monte Carlo approach to select informative features and was fully developed in Dramiński, Rada-Iglesias, Enroth, Wadelius, Koronacki, and Komorowski (2008) as Monte Carlo feature selection algorithm or MCFS, is capable of incorporating interdependencies between features. It bears some remote similarity to the RF methodology, but differs entirely in the way feature ranking is performed. Specifically, our method does not require debiasing (cf. Dramiński, Kierczak, Koronacki, and Komorowski (2010)) and is conceptually simpler. A more important and newer result is that it provides explicit information about interdependencies among features (cf. Dramiński *et al.* (2010), where an early version of the MCFS algorithm with an interdependency discovery, or ID, component has been introduced; see also Kierczak, Ginalski, Dramiński, Koronacki, Rudnicki, and Komorowski (2009) and Kierczak, Dramiński, Koronacki, and Komorowski (2010)).

In this paper, we present an R (R Core Team 2018) package that implements the most recent version of the MCFS-ID algorithm. In particular, the ideas from Dramiński *et al.* (2010) have been substantially expanded in Dramiński, Dąbrowski, Diamanti, Koronacki, and Komorowski (2016) by providing not only the ranking of features but also the directed ID graph that presents interdependencies between informative features. Within our approach, discovering

interdependencies builds on identifying features which "cooperate" in determining that some samples belong to one class, other samples to another class, still others to still a further class and so on. It is worthwhile to emphasize that this is completely different from the usual approach which aims at finding features that are similar in some sense. Instead, it can be said that our way to discover interdependencies between features amounts to determining multidimensional dependency between the classes and sequences of features. In this sense, we are in fact interested in *contextual (or predictive) interdependencies between features*, since the dependency in question requires the context of class information.

Let us emphasize that we do not aim at classification. While in our approach we heavily rely on using classifiers, we do not use them for the classification task per se. Indeed, we use classifiers only to: (i) rank features according to their importance with respect to their discriminative power to distinguish between classes; (ii) discover interdependencies between features. Given the top features found in step (i), one can later use them for classification by any classifier, but this is neither required nor of our interest. And clearly, step (ii) is aimed at something vastly different from sheer solving the classification task.

The procedure is particularly well suited to dealing with high-dimensional data including "small n large p problems", i.e., those with a small number of objects (records, samples) versus several orders of magnitude greater number of features for each object.

The procedure from Dramiński *et al.* (2008) and Dramiński *et al.* (2016) for Monte Carlo feature selection and interdependency discovery is briefly recapitulated in Section 2. In Section 3, an overview of the **rmcfs** package (Dramiński and Koronacki 2018) and its main R functions is provided. In Section 4 we demonstrate the use of the package **rmcfs** by performing the MCFS algorithm and building the ID graph for simulated data. We close with concluding remarks in Section 5.

2. MCFS-ID algorithm

2.1. Feature selection – the MCFS part

We begin with a brief recapitulation of our MCFS; see Dramiński *et al.* (2008), which can be consulted for details as well as rationale and statistical validation of our approach to feature selection.

We consider a particular feature to be important, or informative, if it is likely to take part in the process of classifying samples into classes "more often than not". This "readiness" of a feature to take part in the classification process, termed relative importance of a feature, is measured via intensive use of classification trees. In the main step of the procedure, we estimate relative importance of features by constructing thousands of trees for randomly selected subsets of features.

More precisely, out of all d features, s subsets of m features are randomly selected, m being fixed and $m \ll d$, and for each subset of features, t trees are constructed and their performance is assessed (one can easily see that the procedure is essentially the same as that of the random subspace method, the fact the authors were not aware of at the time they wrote their proposal; cf. Ho 1998).

Each of the t trees in the inner loop is trained and evaluated on different, randomly selected

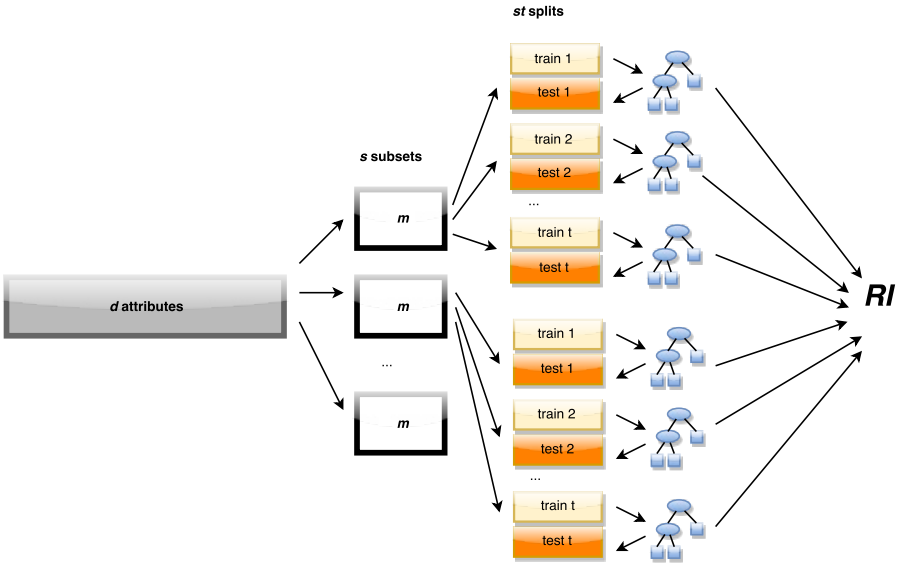

Figure 1: Block diagram of the main step of the MCFS procedure.

training and test sets that come from a split of the full set of training data into two subsets: each time, out of all n samples, 2/3 of the samples are drawn at random for training in such a way as to preserve proportions of classes from the full set of training data, and the remaining samples are used for testing. See Figure 1 for a block diagram of the procedure.

The relative importance of feature g_k, RI_{g_k}, is defined as:

$$\mathrm{RI}_{g_k} = \sum_{\tau=1}^{s \cdot t} \mathrm{wAcc}_\tau^u \sum_{n_{g_k}(\tau)} \mathrm{GR}(n_{g_k}(\tau)) \left(\frac{\text{no. in } n_{g_k}(\tau)}{\text{no. in } \tau} \right)^v, \tag{1}$$

where summation is over all $s \cdot t$ trees and, within each τth tree, over all nodes $n_{g_k}(\tau)$ of that tree on which the split is made on feature g_k, wAcc_τ stands for the weighted accuracy of the τ's tree, $\mathrm{GR}(n_{g_k}(\tau))$ stands for gain ratio for node $n_{g_k}(\tau)$, (no. in $n_{g_k}(\tau)$) denotes the number of samples in node $n_{g_k}(\tau)$, (no. in τ) denotes the number of samples in the root of the τth tree, and u and v are fixed positive reals (now set to 1 by default; cf. Dramiński *et al.* 2010). The normalizing factor (no. in τ), which has the same value for all τ, has been included mainly for computational reasons.

With u and v set to 1, there are three parameters, m, s and t to be set by an experimenter. Note that, overall, $s \cdot t$ trees are constructed and evaluated in the main step of the procedure. Both s and t should be sufficiently large, so that each feature has a chance to appear in many different subsets of features and randomness due to inherent variability in the data is properly accounted for. The choice of subset size m of features selected for each series of t experiments should take into account the trade-off between the need to prevent informative features from being masked too severely by the relatively most important ones and the natural requirement that s be not too large. Indeed, the smaller m, the smaller the chance of masking the occurrence of a feature. However, a larger s is then needed, since all features should have a high chance of being selected into many subsets of the features. For classification problems of dimension d ranging from several thousands to hundreds of thousands, we have found that

taking m equal to a few hundreds (say, $m = 300 - 500$) and t equal to maximum 20 (even $t = 5$ usually suffices) is a good choice in terms of reliability and overall computational cost of the procedure. Finally, for a given m, s can be made a running parameter of the procedure, and the procedure executed for increasing s until the rankings of top scoring $p\%$ features prove (almost) the same for successive values of the s.

2.2. Determining the cutoff value

The above procedure provides one with a ranking of features. However, this ranking as such does not enable one to discern between informative and non-informative features. A cutoff between these two types of features is needed and we propose to determine it by one of 5 different methods (one of them being still under development). Note that finding the cutoff point is related to separating features with large enough RI values from the rest.

Available methods:

- Critical angle: The critical angle method is based on the plot of the features' RIs in decreasing order of size, with the corresponding features equally spaced along the abscissa. The plot can be seen as piecewise linear function, where each linear segment joins two neighboring RIs. Roughly speaking, the cutoff (placed on the abscissa) corresponds to this point on the plot where the slope of consecutive segments changes significantly and lastingly from large to small.

- k-means: The method is based on clustering the RI values into two clusters by the k-means algorithm. It sets the cutoff where the two clusters are separated. This method is quite valuable when data contains a subset of very informative features.

- Permutations (max RI): The method consists of permuting the decision attribute at least 20 times and running the MCFS algorithm for each permutation. The set of the maximal RIs from all these experiments is assumed approximately normally distributed and a critical value based on the the one-sided (upper-tailed) Student's t test (at 95% significance level) is provided. A feature is declared informative if its RI in the original ranking (without any permutation) exceeds the obtained critical value. A more detailed description of this method is included in Dramiński *et al.* (2010).

- Permutations (z score): The method consists of permuting the decision attribute at least 30 times and running the MCFS algorithm for each permutation. For each feature, a separate distribution of the obtained RI values is constructed. For each feature, its RI in the original ranking (without any permutation) is compared against the corresponding distribution of RIs from the experiments with permuted decision attribute (the z test is used this time). This method has been described and studied in Bornelöv and Komorowski (2016). It gives similar results to the previous one but needs more MCFS runs. Along with that, however, it gives a separate p value for each feature.

- Contrast attributes: The method consists of permuting all d features to obtain the so-called contrast features, including them into the dataset and running the MCFS. Given RIs of the contrast features, a statistical test can be used to find a cutoff between the informative and non-informative original features. This method is under development and is currently off.

Given our experience and taking into account its sound statistical foundation, we recommend most the permutations (max RI) method. However, it may sometimes prove very conservative in the sense that it may give a highly limited set of informative features. On the other hand, if there are no meaningful features, the method will discover this fact and the set of informative features will prove empty. Finally, one has to admit that using the permutations method requires much caution. Actually, proper significance level should be chosen adaptively, depending on the data under consideration. Instead of adding this kind of complexity to the way the cutoff is determined, we set (by default) the final cutoff value to be the mean of all the obtained cutoffs.

2.3. Interdependency discovery – the ID part

Once features are ranked by the MCFS procedure, a natural issue to be raised concerns possible interdependencies among the informative features. In this subsection, we describe shortly a way to present such interdependencies in the form of a directed graph.

The interdependencies among features are often modeled using interactions, similarly as in experimental design and analysis of variance. Perhaps the most widely used approach to recognizing interdependencies is finding correlations between features or finding groups of features that behave in some sense similarly across samples. A classical bioinformatics example of this problem is finding co-regulated features, most often genes or, rather more precisely, their expression profiles. Searching groups of similar features is usually done with the help of various clustering techniques, frequently specially tailored to a task at hand. See Smyth, Yang, and Speed (2003), Hastie, Tibshirani, Botstein, and Brown (2001), Saeys *et al.* (2007), Gyenesei, Wagner, Barkow-Oesterreicher, Stolte, and Schlapbach (2007) and the literature therein. Our approach to interdependency discovery is significantly different in that we focus on identifying features that "cooperate" in determining that a sample belongs to a particular class. The initial idea was presented in Dramiński *et al.* (2010) but the current version provides directed ID graphs and is based on an improved interdependency measure (cf. Dramiński *et al.* 2016).

Our ID graph is based on aggregating information provided by all the $s \cdot t$ trees (see Figure 1). However, let us begin by noting that, for a single classification tree, a set of decision rules defining each class is provided by this tree's paths. Each decision rule is produced as a conjunction of conditions imposed on the particular features which in fact point to conditional interdependencies between the features. Our trust in the decision rules that are learned by any single rule-based classifier, and thus in the discovered interdependencies, is naturally limited. Moreover, the classifier is trained on just one training set and therefore our conclusions are necessarily dependent on the classifier and are conditional upon the training set. In the case of classification trees, the problem is aggravated by their high variance, i.e., their tendency to provide varying results even for slightly different training sets. Accordingly, in order to overcome these problems and provide more objective results, the MCFS-ID procedure rests on building thousands of classification trees.

To see how an ID graph is built, let us recall that each node in each of the multitude of classification trees represents a feature on which a split is made. Now, for each node in each classification tree all its antecedent nodes can be taken into account along the path to which the node belongs; note that each node in a tree has only one parent and thus for the given node we simply consider its parent, then the parent of the parent and so on. In practice,

Algorithm 1 ID graph building procedure.

$w[n_\delta \to n] = 0$
for $\tau_k \in T$ **do**
  for $n \in \tau_k$ **do**
    for $\delta \in D$ **do**
      $n_\delta = \delta$th antecedent of n
      $w[n_\delta \to n] = w[n_\delta \to n] + \mathrm{GR}(n) \left(\frac{\text{no. in } n}{\text{no. in } n_\delta} \right)$
    end for
  end for
end for

the maximum possible depth of such analysis, i.e., the number of antecedents considered, if available before the tree's root is attained, is set to some predetermined value, which is the procedure's parameter (its default value being 5). For each pair [*antecedent node* → *given node*] we add one directed edge to our ID graph from *antecedent node* to *given node*. Let us emphasize again that a node is equated with the feature it represents and thus any directed edge found is in fact an edge joining two uniquely determined features in a directed way. The edges are found along the paths in all the $s \cdot t$ MCFS-ID trees. Clearly, the same edge can appear more than once even in a single tree.

The strength of the interdependence between two nodes, actually two features, connected by a directed edge, termed ID weight of a given edge, or ID weight for short, is equal to the gain ratio (GR) in the given node multiplied by the fraction of objects in the given node and the antecedent node. Thus, for node $n_k(\tau)$ in τth tree, $\tau = 1, \ldots, s \cdot t$, and its antecedent node $n_i(\tau)$, ID weight of the directed edge from $n_i(\tau)$ to $n_k(\tau)$, denoted $w[n_i(\tau) \to n_k(\tau)]$, is equal to

$$w[n_i(\tau) \to n_k(\tau)] = \mathrm{GR}(n_k(\tau)) \left(\frac{\text{no. in } n_k(\tau)}{\text{no. in } n_i(\tau)} \right), \tag{2}$$

where $\mathrm{GR}(n_k(\tau))$ stands for gain ratio for node $n_k(\tau)$, (no. in $n_k(\tau)$) denotes the number of samples in node $n_k(\tau)$ and (no. in $n_i(\tau)$) denotes the number of samples in node $n_i(\tau)$.

The final ID graph is based on the sums of all ID weights for each pair [*antecedent node* → *given node*]; i.e., for each directed edge found, its ID weights are summed over all occurrences of this edge in all paths of all MCFS classification trees. For a given edge, it is this sum of ID weights which becomes the ID weight of this edge in the final ID graph. The pseudo code in Algorithm 1 describes the calculation. T denotes the set of all $s \cdot t$ trees and $D = \{1, 2, \ldots, \mathit{depth}\}$ with *depth* being the predetermined number of antecedents considered.

Note that an edge $n_i \to n_k$ from node n_i to node n_k is directed as is the edge (if found) from n_k to n_i, $(n_k \to n_i)$. Interestingly, in most cases of ID graphs, we find that one of such two edges is dominating, i.e., has a much larger ID weight than the other. Whenever it happens, it means that not only n_i and n_k form a sound partial decision (a part of a conjunction rule) but also that their succession in the directed rule is not random.

The ID graph is a way to present interdependencies that follow from all of the MCFS classification trees. Each path in a tree represents a decision rule and by analyzing all tree paths we in fact analyze decision rules to find the most frequently observed features that along with other features form good decision rules. The ID graph thus presents some patterns that

frequently occur in thousands of classification trees built by the MCFS procedure.

In sum, an ID graph provides a general roadmap that not only shows all the most variable attributes that allow for efficient classification of the objects but, moreover, it points to possible interdependencies between the attributes and, in particular, to a hierarchy between pairs of attributes. High differentiation of the values of ID weights in the ID graph gives strong evidence that some interdependencies between some features are much stronger than others and that they create some patterns/paths calling for interpretation based on background knowledge.

Let us conclude this subsection with a short comparison of our approach to discovering interdependencies between features and that based on the RFs (cf. Wright *et al.* 2016). Most importantly, we take full advantage of the Monte Carlo mechanism, which makes direct examination of possible interactions superfluous. Indeed, nothing like, e.g., pairwise permutation importance for pairs of features needs to be calculated. Moreover, due to the form of (2), in particular since the summands in the ID weights of pairs of features do not depend in any way on prediction performance of the trees involved in calculation of these summands, we can hope for no masking of interaction effects measured by ID weights by marginal effects (as yet, this last claim has not been confirmed by a thorough simulation study). And, last but not least, ours are directed interactions.

Clearly, only ID graphs for strong interdependencies (interactions) are of interest. It is readily seen that such graphs can equally easily be constructed for discovering two-way interactions when each of the two features involved or only one of them, or even none of them is of large relative importance, i.e., gives a strong marginal effect (except for the XOR interaction of two features with no marginal effects).

It should be noticed, however, that the ID weights are calculated in a way which makes them incomparable with relative importances of individual features. To put it otherwise, both measures give us only rankings (from the greatest to the smallest) of, respectively: features w.r.t. their relative importance; and strengths of pairwise interactions, albeit without direct information on predictive ability of any interaction.

3. The R package rmcfs

The R package **rmcfs** is available from the Comprehensive R Archive Network (CRAN) at `https://CRAN.R-project.org/package=rmcfs`. It contains 13 user functions and one example dataset borrowed from Alizadeh *et al.* (2000). All of them are described as part of the standard R package documentation and their description is also available through the R built-in help system. Here we will focus only on the major **rmcfs** functionality covered by the following set of functions:

- `mcfs()` performs Monte Carlo feature selection and interdependence discovery (MCFS-ID) on a given dataset. It uses C4.5 trees (`J48`) as implemented in **Weka** 3-6-10, see Hall, Frank, Holmes, Pfahringer, Reutemann, and Witten (2009).
- `plot` S3 method for class '`mcfs`' objects. It plots various aspects of the MCFS-ID result.
- `print` S3 method for class '`mcfs`' objects. It prints basic information of the MCFS-ID results: top features, cutoff values, confusion matrix obtained for $s \cdot t$ trees and classification rules obtained by the *RIPPER* (`JRip`) algorithm.

- `build.idgraph()` constructs the ID graph based on the result returned by the `mcfs` function.
- `plot` S3 method for '`idgraph`' objects visualizing the ID graph.

3.1. Function `mcfs`

The function `mcfs` can be used to build feature ranking, find the cutoff and evaluate classification performance of a set of top features. It is used as:

```
mcfs(formula, data, projections = "auto", projectionSize = "auto",
  featureFreq = 150, splits = 5, splitSetSize = 1000, balance = "auto",
  cutoffMethod = c("permutations", "criticalAngle", "kmeans", "mean"),
  cutoffPermutations = 20, buildID = TRUE, finalRuleset = TRUE,
  finalCV = TRUE, finalCVSetSize = 1000, finalCVRepetitions = 3,
  seed = NA, threadsNumber = 2)
```

and takes the following arguments:

- `formula`: specifies the decision attribute and the relation between class and other attributes (e.g., `class ~ .`).
- `data`: defines the input `data.frame` containing all features with decision attribute included.
- `projections`: defines the number of subsets with randomly selected features. This parameter is usually set to a few thousands and is denoted in Equation 1 as s. By default it is set to `"auto"` and then the value is based on the size of the input dataset and the `featureFreq` parameter.
- `projectionSize`: defines the number of features in one subset. It can be defined by an absolute value (e.g., 100 denotes 100 randomly selected features) or by a fraction of input attributes (e.g., 0.05 denotes 5% of input features). This parameter is denoted in Equation 1 as m. If it is set to `"auto"` then `projectionSize` equals $\sqrt{d}$, where d is the number of input features. Minimum number of input features in one subset is 1.
- `featureFreq`: determines how many times each input feature should be randomly selected when `projections = "auto"`. By default each feature should be drawn 150 times.
- `splits`: defines the number of splits of each subset. This parameter is denoted in Equation 1 as t and by default set to 5.
- `splitSetSize`: determines whether to limit the input dataset size. It helps to speed up the computation for datasets with a large number of objects. If the parameter is larger than 1, it determines the number of objects that are drawn at random for each of the $s \cdot t$ decision trees. If `splitSetSize = 0` then `mcfs` uses all objects in each iteration.

- `balance`: determines the way to balance classes. It should be set to 2 or higher if the input dataset contains heavily unbalanced classes. Each subset s will contain all the objects from the least frequent class and randomly selected set of objects from each of the remaining classes. This option helps to select features that are important for discovering a relatively rare class. The parameter defines the maximal imbalance ratio. If the ratio is set to 2, then subset s will contain the number of objects from each class (but the least frequent one) proportional to the square root of the class size $\sqrt{\text{size}(c)}$. If `balance = 0` then balancing is turned off. If `balance = 1` it is turned on but does not change the size of classes. Default value is `"auto"`.
- `cutoffMethod`: determines the final cutoff method. Default value is `"permutations"`.
- `cutoffPermutations`: determines the number of permutation runs. It needs at least `cutoffPermutations = 20` for a statistically significant result. Minimum is 3; 0 turns off the permutation method.
- `buildID`: if `= TRUE`, interdependencies discovery is turned on and all ID graph edges are collected.
- `finalRuleset`: if `= TRUE`, classification rules (by *RIPPER* algorithm) are created on the basis of the final set of features.
- `finalCV`: if `= TRUE`, it runs cross validation (CV) experiments on the final set of features. The following set of classifiers is used: C4.5, NB, SVM, kNN, logistic regression and RIPPER.
- `finalCVSetSize`: limits the number of objects used in the final CV experiment. For each CV repetition, the objects are selected randomly from the uniform distribution.
- `finalCVRepetitions`: defines the number of repetitions of the CV experiment. The more repetitions, the more stable is the result.
- `seed`: seed for the random number generator in Java. By default the seed is random. Replication of the result is possible only if `threadsNumber = 1`.
- `threadsNumber`: number of threads to use in computation. More threads need more CPU cores and also memory usage is a bit higher. It is recommended to set this value equal to or less than the CPU available cores.

Function `mcfs` produces an S3 object of class '`mcfs`' that is a list of the following components:

- `data`: input `data.frame` limited to the top important features set.
- `target`: decision attribute name.
- `RI`: `data.frame` that contains all features with relevance score sorted from the most relevant to the least relevant. This is the ranking of features.
- `ID`: `data.frame` that contains features interdependencies as graph edges. It can be converted into a graph object using the `build.idgraph` function.
- `distances`: `data.frame` that contains convergence statistics of subsequent projections.

- `cmatrix`: confusion matrix obtained from all $s \cdot t$ decision trees.
- `cutoff`: `data.frame` that contains cutoff values obtained by the following methods: `criticalAngle`, `kmeans`, `permutations` (max RI) and the last one representing the `mean` value based on all cutoff values.
- `cutoff_value`: the number of features chosen as informative by the method defined by parameter `cutoffMethod`.
- `cv_accuracy`: `data.frame` that contains the classification results obtained by cross validation performed on `cutoff_value` features. This `data.frame` exists if `finalCV = TRUE`.
- `permutation`: `data.frame` that contains the results of the permutation experiments: all RI values obtained from all permutation experiments; RI obtained for reference MCFS experiment (i.e, the experiment on the original data); p values from the Anderson-Darling normality test applied separately for each feature to the `cutoffPermutations` RI set; p values from the Student-t test applied separately for each feature to the `cutoffPermutations` RI vs. reference RI. All these p values are related to the permutation (z score) method (see Section 2.2). This `data.frame` exists if `cutoffPermutations > 0`.
- `jrip`: classification rules (produced by the *RIPPER* algorithm) and related CV statistics obtained for `cutoff_value` features for this algorithm.
- `params`: all settings used by MCFS-ID.
- `exec_time`: execution time of MCFS-ID.

Admittedly, one can become overwhelmed by the sheer number of input parameters but in most applications the `mcfs` function can be used with the set of default/recommended parameters (as in Section 4).

3.2. Plot method for 'mcfs' objects

The plot method for '`mcfs`' objects is defined as follows:

```
plot(x,
  type = c("ri", "id", "distances", "features", "cv", "cmatrix", "heatmap"),
  size = NA, ri_permutations = c("max", "all", "sorted", "none"),
  diff_bars = TRUE, features_margin = 10,
  cv_measure = c("wacc", "acc", "pearson", "MAE", "RMSE", "SMAPE"),
  heatmap_norm = c("none", "norm", "scale"),
  heatmap_fun = c("median", "mean"), heatmap_colors = c("white", "red"),
  cex = 1, ...)
```

and takes the following arguments:

- `x`: a '`mcfs`' S3 object, e.g., the result of the MCFS-ID experiment returned by `mcfs` function.

- `type`:
  - `"ri"`: plots top features set with their RIs as well as max RI obtained from the permutation experiments. Red color denotes important features.
  - `"id"`: plots top ID values obtained from the MCFS-ID.
  - `"distances"`: plots distances (convergence diagnostics of the algorithm) between subsequent feature rankings obtained during the MCFS-ID experiment.
  - `"features"`: plots top features set along with their RI. It is a horizontal barplot that shows important features in red color and non important in gray.
  - `"cv"`: plots cross validation results based on the top features.
  - `"cmatrix"`: plots the confusion matrix obtained on all $s \cdot t$ trees.
  - `"heatmap"`: plots heatmap results based on top features. Only numeric features can be presented on the heatmap.
- `size`: number of features to plot.
- `ri_permutations`: if `type = "ri"` and `ri_permutations = "max"`, then it additionally shows horizontal lines that correspond to max RI values obtained from each single permutation experiment.
- `diff_bars`: if `type = "ri"` or `type = "id"` and `diff_bars = TRUE`, then it shows the difference values for RI or ID values.
- `features_margin`: if `type = "features"`, then it determines the size of the left margin of the plot.
- `cv_measure`: if `type = "cv"`, then it determines the type of accuracy shown in the plot: weighted or unweighted/balanced accuracy (`"wacc"` or `"acc"`). If the target attribute is numeric it is possible to review one of the following prediction quality measures: `"pearson"`, `"MAE"`, `"RMSE"`, `"SMAPE"`.
- `heatmap_norm`: if `type = "heatmap"`, then it defines the type of input data normalization: `"none"` – without any normalization, `"norm"` – normalization within range $[-1, 1]$, `"scale"` – standardization/centering by mean and standard deviation.
- `heatmap_fun`: if `type = "heatmap"`, then it determines the calculation of `"mean"` or `"median"` within the class to be shown as heatmap color intensity.
- `heatmap_colors`: if `type = "heatmap"`, then it defines low and high colors on the heatmap.
- `cex`: size of fonts.
- `...`: additional plotting parameters.

3.3. Function `build.idgraph`

The function to build ID graph is defined as follows:

```
build.idgraph(mcfs_result, size = NA, size_ID = NA, self_ID = FALSE,
  outer_ID = FALSE, orphan_nodes = FALSE, size_ID_mult = 3,
  size_ID_max = 100)
```

and takes the following arguments:

- `mcfs_result`: results returned by the `mcfs` function.
- `size`: number of top features to select. If `size = NA`, then `size` is defined by the `mcfs_result$cutoff_value` parameter.
- `size_ID`: number of interdependencies (edges in ID graph) to be included. If `size_ID = NA`, then parameter `size_ID` is defined by multiplication of `size_ID_mult` · `size`.
- `self_ID`: if `self_ID = TRUE`, then include self-loops from ID graph.
- `outer_ID`: if `outer_ID = TRUE`, then include include all interactions between a feature from the top set features (defined by `size` parameter) with any other feature.
- `orphan_nodes`: if `orphan_nodes = TRUE`, then include all nodes, even if they are not connected to any other node (isolated nodes).
- `size_ID_mult`: if `size_ID_mult = 3` there will be 3 times more edges than features (nodes) present in the ID graph. It works only if `size = NA` and `size_ID = NA`.
- `size_ID_max`: maximum number of interactions to be included in the ID graph (the upper limit).

It produces an S3 '`idgraph`'/'`igraph`' object that can be plotted in R, exported to graphML (in XML format) or saved as CSV or RDS files.

3.4. Plot method for '`idgraph`' objects

The plot method for '`idgraph`' objects is defined as follows:

```
plot(x, label.dist = 0.5, cex = 1)
```

and takes the following arguments:

- `graph`: '`idgraph`'/'`igraph`' S3 object representing feature interdependencies. This object is produced by the `build.idgraph` function.
- `label.dist`: space between the node's label and the corresponding node in the plot.
- `cex`: size of fonts.

4. Example

First of all make sure you have Java installed on your computer and then install package **rmcfs** from the CRAN repository:

```
R> install.packages("rmcfs")
```

Before loading the package, set the Java parameters (we set the maximum size of memory allocated by Java at 2 GB) and then load the package.

```
R> options(java.parameters = "-Xmx2g")
R> library("rmcfs")
```

After this the R environment is ready to run the example below. Note that slightly different results than those reported might be obtained using the replication material because of the algorithm's implementation and the random number generator in the Java version used.

4.1. The artificial data

To review and understand the MCFS-ID algorithm we suggest to create and use an extraordinarily simple example dataset. It consists of objects from 3 classes, A, B and C, that contain 40, 20 and 10 objects, respectively (70 objects altogether). For each object, we create 6 binary features (*A1*, *A2*, *B1*, *B2*, *C1* and *C2*) that are "ideally" or "almost ideally" correlated with the *class* feature. If an object's *'class'* equals *'A'*, then its features *A1* and *A2* are set to class value *'A'*; otherwise *A1 = A2 = 0*. If an object's *'class'* is *'B'* or *'C'*, we proceed analogously, but we introduce some random corruption to 2 observations from class *'B'* and to 4 observations from class *'C'*: in the former case, for each of the two observations and both attributes *B1/B2*, we randomly replace their value *'B'* by '0' and in the latter case, again for each of the four observations and both attributes *C1/C2*, we randomly replace their value *'C'* by '0'. The data also contains additional 500 random numerical features with uniformly $[0, 1]$ distributed values. Thus we end up with 6 nominal important features (3 pairs with different levels of importance for classification) and 500 randomly distributed ones.

```
R> set.seed(0)
R> class <- c(rep("A", 40), rep("B", 20), rep("C", 10))
R> A <- B <- C <- rep("0", length(class))
R> A[class == "A"] <- "A"
R> B[class == "B"] <- "B"
R> C[class == "C"] <- "C"
R> rnd <- runif(length(class))
R> B[class == "B"][rnd[class == "B"] <= (sort(rnd[class == "B"]))[2]] <- "0"
R> C[class == "C"][rnd[class == "C"] <= (sort(rnd[class == "C"]))[4]] <- "0"
R> d <- data.frame(matrix(runif(500 * length(class)), ncol = 500))
R> d <- cbind(d, data.frame(A1 = A, A2 = A, B1 = B, B2 = B, C1 = C, C2 = C,
+    class))
```

Once the input data is created, we can review, e.g., the last few columns and rows of it:

```
R> d[50:70, 497:507]
```

```
         X497       X498       X499       X500 A1 A2 B1 B2 C1 C2 class
50 0.20466809 0.56092946 0.95223478 0.70965238  0  0  B  B  0  0     B
51 0.11404115 0.29816202 0.04477183 0.03345120  0  0  B  B  0  0     B
```

```
52 0.02938875 0.21069288 0.20364813 0.94183044  0  0  B  B  0  0     B
53 0.49005094 0.12659981 0.22824847 0.81885672  0  0  B  B  0  0     B
54 0.12656223 0.71003559 0.12826985 0.57745653  0  0  B  B  0  0     B
55 0.84182080 0.02133940 0.98461836 0.18093354  0  0  B  B  0  0     B
56 0.24221154 0.32765974 0.36310019 0.78545327  0  0  0  0  0  0     B
57 0.49214327 0.12442556 0.61966539 0.58656922  0  0  B  B  0  0     B
58 0.74438287 0.81923355 0.87711323 0.45680401  0  0  B  B  0  0     B
59 0.06014164 0.95796935 0.81141427 0.65610882  0  0  B  B  0  0     B
60 0.34402144 0.46230511 0.98609784 0.65171840  0  0  B  B  0  0     B
61 0.01665507 0.35895054 0.92959504 0.66303803  0  0  0  0  C  C     C
62 0.53326256 0.54672994 0.94477699 0.53642020  0  0  0  0  C  C     C
63 0.92763065 0.15789376 0.33911702 0.37232281  0  0  0  0  0  0     C
64 0.42980634 0.15969091 0.11927503 0.61437815  0  0  0  0  C  C     C
65 0.20276997 0.29115349 0.23746162 0.79282321  0  0  0  0  0  0     C
66 0.76193072 0.18045207 0.13270288 0.26172409  0  0  0  0  C  C     C
67 0.12037553 0.13130389 0.68910685 0.99061888  0  0  0  0  0  0     C
68 0.88857332 0.95222348 0.07483475 0.94459348  0  0  0  0  C  C     C
69 0.20723421 0.42698189 0.77679508 0.49228194  0  0  0  0  C  C     C
70 0.15157902 0.08401528 0.94564002 0.03452377  0  0  0  0  0  0     C
```

The number of objects in each class is equal to:

```
R> table(d$class)

 A  B  C
40 20 10
```

Package **rmcfs** includes the function `artificial.data` that creates the above example dataset.

```
R> d <- artificial.data(rnd_features = 500, corruption = c(0, 2, 4),
+    seed = 0)
```

4.2. MCFS-ID on artificial data

Let us run `mcfs` on the created data. If the given CPU is a HT (hyper-threading) quad core, we recommend setting up to 8 threads for processing because the efficiency gain of multithreaded processing is limited by the number of physical CPU cores. However, in the example below, we use 1 thread to get perfect reproducibility, since the implemented multithreaded mechanism is asynchronous. For a standard PC and 1 thread, it may take around 2 minutes, but we observe nearly linear dependence between number of threads running on physical cores and speed of calculation (see Section 4.3). Parameter `cutoffPermutations` is set to 5 for this simple example dataset, but for real data we recommend the default 20.

```
R> result <- mcfs(class ~ ., d, cutoffPermutations = 5, seed = 1,
+    threadsNumber = 1)
R> result$exec_time

Time difference of 2.622107 mins
```

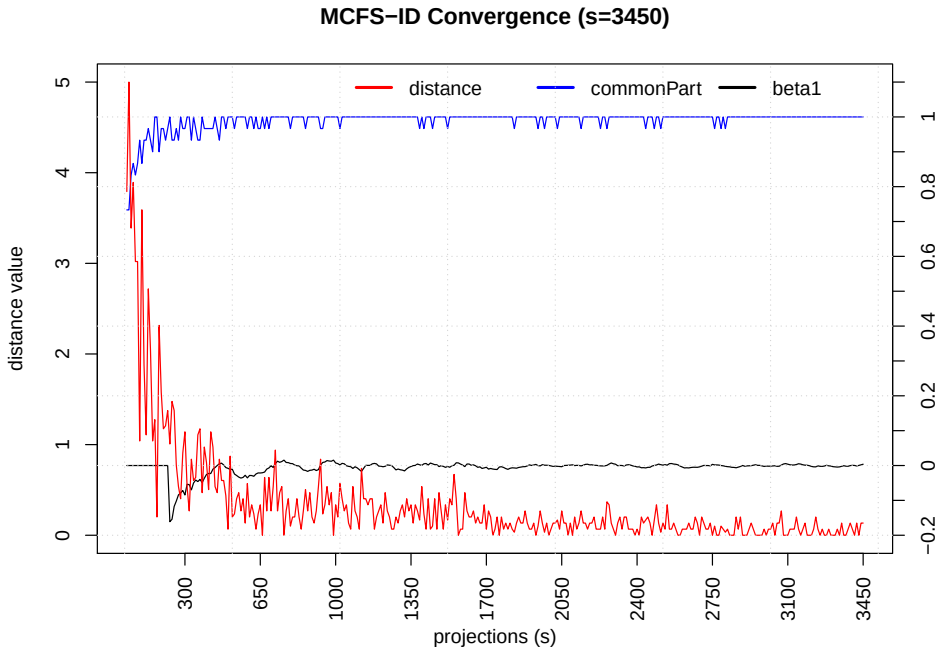

Figure 2: Distance function and common part.

After successfully running the MCFS-ID algorithm we can check convergence of the algorithm (see Figure 2). The distance function shows the difference between two consecutive rankings – zero means no changes between two rankings (see the left y-axis). The common part gives the fraction of features that overlap for two different rankings (see the right y-axis). The ranking stabilizes after a number of iterations: the distance tends to zero and the common part tends to 1. beta1 shows the slope of the tangent of a smoothed distance function. If beta1 tends to 0 (the right y-axis) then the distance is given by a flat line.

```
R> plot(result, type = "distances")
```

Now we can check the cutoff value for various methods and review the result for the default one.

```
R> result$cutoff
```

```
         method      minRI size    minID
1 criticalAngle 0.02667534   22       NA
2        kmeans 0.33329248    6       NA
3  permutations 0.07494005    6 5.029552
4          mean 0.03540977   11       NA
```

```
R> result$cutoff_value
```

```
[1] 6
```

By default, the final cutoff value is equal to the value obtained from the method based on permutations of the decision attribute. The mean obtained from all methods is in our example

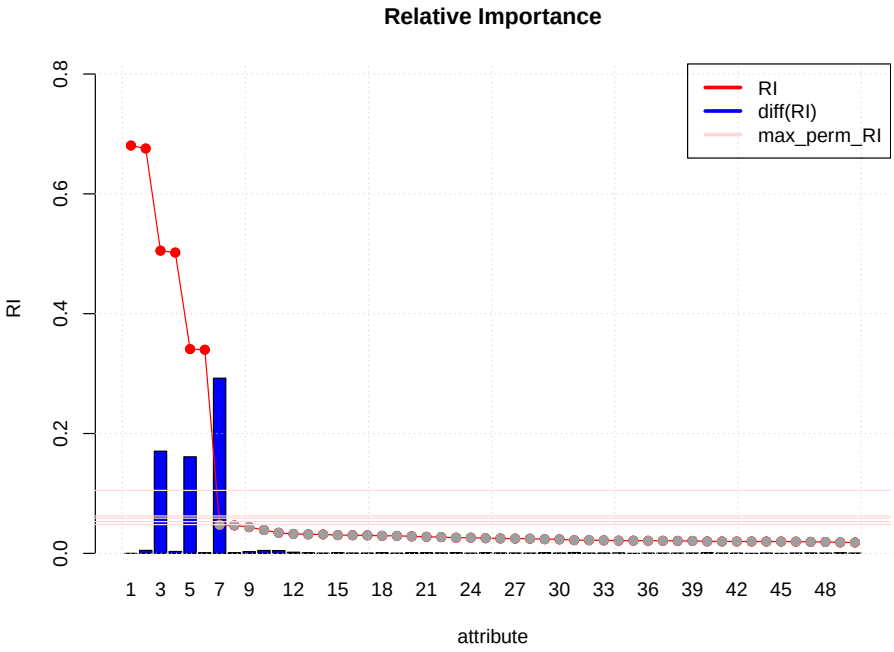

Figure 3: RIs and maximal RIs from the permutation experiments for the top 50 features.

equal to 11. The mean cutoff value is larger than the actual number of informative features since the critical angle method takes into consideration only the shape of the RI distribution. The input parameter `cutoffMethod` for the `mcfs` function determines the method which we want to use as the oracle. Since the cutoff value is determined by `result$cutoff_value`, the user may change it accordingly. This value is used by the `plot` method.

We may plot RI values in decreasing order for the top, e.g., 50 features. See Figure 3. The line with red/gray dots gives RI values, the blue vertical barplot gives the difference δ between consecutive RI values. Informative features are separated from non-informative ones by the cutoff value and are presented in the plot as red and gray dots, respectively. We can also view all maximal RIs obtained from all permutation experiments (parameter `plot_permutations = TRUE`) – Figure 3 shows these 5 maximal RIs as horizontal red lines. The distribution of the max RI values determines the cutoff value.

```
R> plot(result, type = "ri", size = 50, plot_permutations = TRUE)
```

Similarly, we can plot ID weights in decreasing order for the top, e.g., 50 ID graph edges.

```
R> plot(result, type = "id", size = 50)
```

Now, we can review labels and RIs of the top features. The resulting plot is presented in Figure 4. One can see that all six features are highly important and their RIs are much higher than those of other features. The set of informative features is flagged in red in the plot. Features *A1* and *A2* have substantially higher RI values because they "ideally" separate the largest class and in most cases they appear in the root node. The second level of a tree may be determined by either of *B1*, *B2*, *C1* and *C2*, but in our case features *B1/B2* are less corrupted by "noise" and hence they are much more informative than features *C1/C2*. Notice

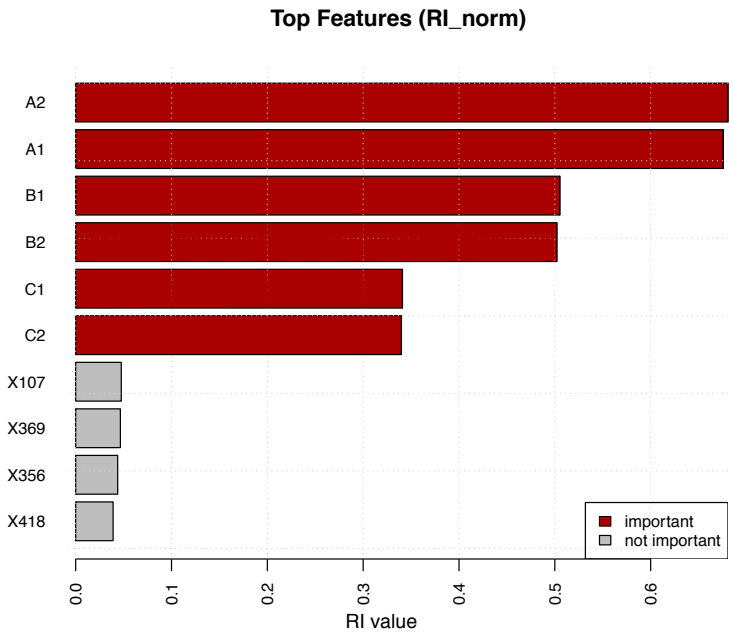

Figure 4: Top features selected by MCFS-ID.

that all pairs (*A1/A2*, *B1/B2*, *C1/C2*) have similar RIs and all the resulting importance levels are consistent with the corruption level introduced to our artificial data.

```
R> plot(result, type = "features", size = 10)
```

Finally, we can build and visualize the ID graph. By default, we plot all the edges that connect the 6 informative features as defined by `result$cutoff_value`.

```
R> gid <- build.idgraph(result)
R> plot(gid, label_dist = 1)
```

In the ID graph, as seen in Figure 5, some additional information is conveyed with the help of suitable graphical means. The color intensity of a node is proportional to the corresponding feature's RI. The size of a node is proportional to the number of edges related to this node. The width and level of darkness of an edge is proportional to the ID weight of this edge. Since we would like to review only the strongest ID weights let us plot the ID graph with only the 12 top edges (`size_ID = 12`).

```
R> gid <- build.idgraph(result, size_ID = 12)
R> plot(gid, label_dist = 1)
```

In Figure 6, the top 6 features along with top 12 ID weights are presented. Notice that the two most important features, *A1* and *A2*, point to all other ones, while *B1* and *B2* point to *C1* and *C2*. The directions of edges reproduce paths in decision trees. If features *A1*, *A2* determine the root nodes, then the leafs are *B1*, *B2*, *C1* or *C2*. If features *B1*, *B2* determine the root nodes then the leafs are *C1*, *C2*. Interestingly and correctly, the ID graph (in both Figures 5 and 6) does not show any connection between identical features (e.g., between *A1*

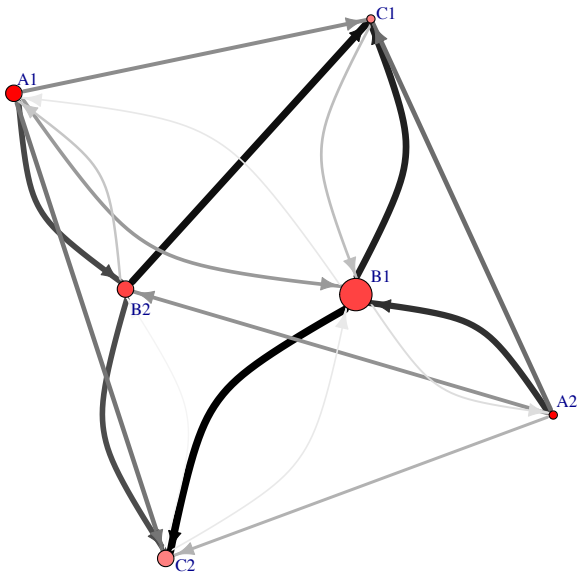

Figure 5: ID graph for artificial data.

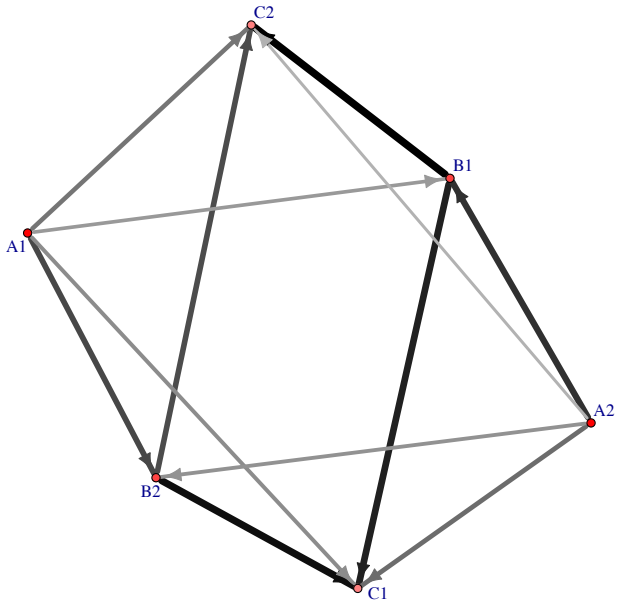

Figure 6: ID graph for artificial data, limited to top 6 features and top 12 ID weights.

and *A2*). Indeed, such features do not cooperate in distinguishing between classes, since they are the same!

For the top features set, when the execution of MCFS-ID has been finished, the procedure runs 10-fold cross validation (CV) on 6 different classifiers (see Figure 7). Each CV is repeated 3 times (defined by the `finalCVRepetitions` parameter) and the mean value of accuracy and weighted accuracy are gathered. Since the weighted accuracy is equal to the mean over all true positive rates (TPR), it is more meaningful for datasets with unbalanced classes. In our example, given the first two features from the ranking, *A1* and *A2*, only 66% weighted

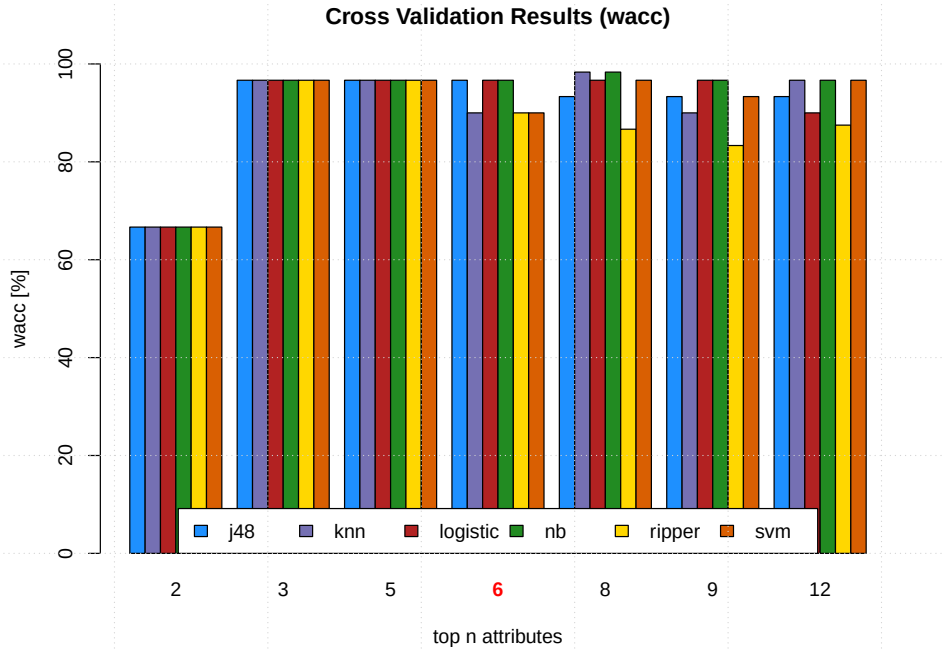

Figure 7: Cross validation results for the top features. The selected cutoff value is flagged by red color on the x-axis.

accuracy can be achieved, since only class A can then be separated and the remaining objects can be labeled as B which perfectly classifies 2 out of 3 classes (and hence wacc = 66%). For 6 and more top features the accuracy depends on an algorithm and a given CV experiment. The CV plot presents the result for `result$cutoff_value` features (red label on the x-axis) and its multiples of (0.25, 0.5, 0.75, 1, 1.25, 1.5, 2).

```
R> plot(result, type = "cv", measure = "wacc")
```

The RIPPER algorithm is a rule-based classifier and thus it provides the user with classification rules. Function `print()` provides in particular the rules and their classification ability.

```
R> print(result)
```

```
[...]

#################################
JRIP classification rules created on top 6 features:
JRIP rules:
===========

(A1 = 0) and (B1 = 0) => class=C (12.0/2.0)
(A1 = 0) => class=B (18.0/0.0)
 => class=A (40.0/0.0)
```

```
Number of Rules : 3

RIPPER CV Result (10 folds repeated 3 times)
Confusion Matrix
                        predicted
class         B         C         A
B         54.0         6.0         0.0
C         2.0         26.0         2.0
A         0.0         0.0         120.0

Accuracy = 0.9523
WeightedAccuracy = 0.9222

True Positive Rate
        A: 1.0
        B: 0.9
        C: 0.8666
False Positive Rate
        A: 0.0222
        B: 0.0133
        C: 0.0333
```

4.3. MCFS-ID on real data

In order to illustrate the practical usability of the algorithm we use the Arcene dataset downloaded from the UCI Machine Learning Repository at https://archive.ics.uci.edu/ml/datasets/Arcene (Dua and Karra Taniskidou 2017). Arcene's task is to distinguish cancer versus normal patterns from mass-spectrometric data. This is a two-class classification problem with 10000 continuous input variables and 100 objects used for training. This dataset was one of 5 datasets of the NIPS 2003 feature selection challenge; see Guyon, Gunn, Ben-Hur, and Dror (2005). After running MCFS-ID with its default parameters we obtain the ranking of features that we use for prediction on the validation set. Starting from 1 up to 200 top features we repeatedly train a SVM and apply it on 100 new objects (unseen during the feature selection process and training of the model) from the validation dataset. For over 100 top features classification accuracy is almost 99% and for over 175 top features it achieves 100% (see Figure 8); this almost or strictly perfect accuracy follows from the task's low complexity and small size of the validation set. Unfortunately, a larger test set with class labels is unavailable).

It should be expected that some combination of the top features from the ranking and the features that reveal the strongest two-way interdependencies (interactions) with those top ones may comprise a very good set of features to classify on. Note that such strong interdependencies, measured by the ID weights, may occur between pairs of features each of which is of high relative importance and, perhaps equally or almost equally likely, between features only one of which has a high value of RI. Indeed, a feature with low value of RI may prove a good helper of the other feature to increase predictive power of the latter through a directed interaction with the former. In order to confirm this claim we first select 200 features with

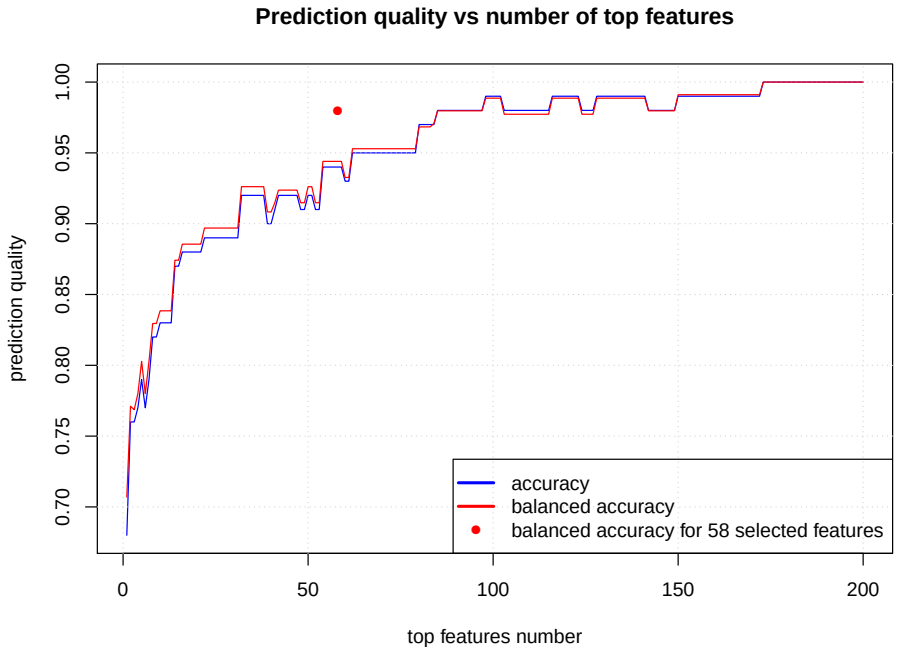

Figure 8: Prediction result on the validation set vs. the number of top features used to train the SVM classifier.

the highest RI and then build the ID graph for the 50 edges with the highest ID weights and such that each edge connects a feature from the aforementioned set of 200 features with any other feature (see Figure 9). The concatenation of the features present in the ID graph with the top 10 features results in 58 features with 9 of them present both in the top 10 and in the graph:

```
[1] "V1184" "V5473" "V698"  "V8502" "V8806" "V7197" "V4290" "V1476" "V7899"
[10] "V9743" "V7542" "V4183" "V9050" "V3206" "V6292" "V4352" "V2448" "V4557"
[19] "V893"  "V504"  "V4542" "V2227" "V2278" "V7696" "V4970" "V8623" "V9082"
[28] "V130"  "V5112" "V2080" "V3170" "V9213" "V2057" "V4895" "V7530" "V7014"
[37] "V8976" "V8904" "V4322" "V1046" "V4513" "V1322" "V9104" "V5417" "V9358"
[46] "V4738" "V4301" "V8132" "V9319" "V787"  "V9477" "V3956" "V3014" "V9884"
[55] "V8506" "V7319" "V2247" "V7748"
```

This obtained set of features is used to train a SVM model and determine its classification quality on the validation set. For the 58 features we obtain an accuracy of 98%.

To verify the convergence of our MCFS heuristic procedure we run the algorithm three times on the same training data – each time with a different seed. All three runs are made using the same default parameters as before. Having obtained all three rankings, we compare the three sets of top 100 features. As many as 92 features are present in all three rankings, regardless of the starting seed. Below we present the top twenty features from the first run and their positions in the rankings for the remaining two runs.

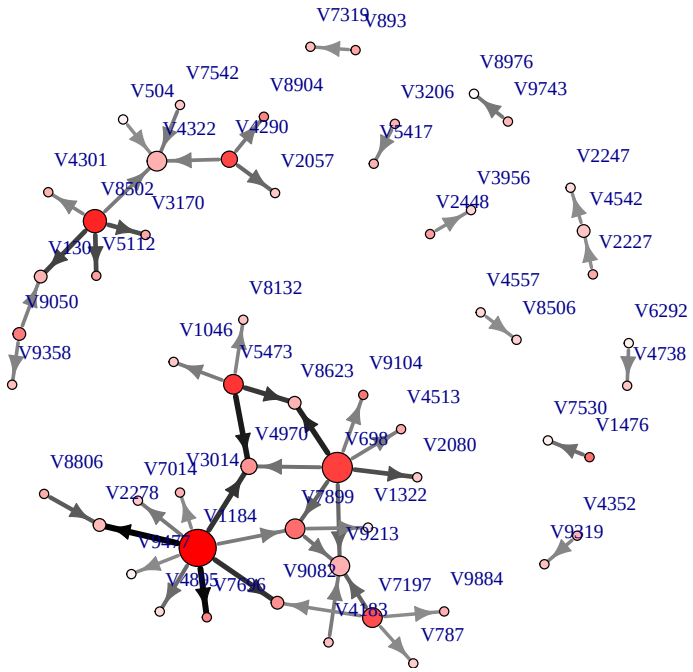

Figure 9: ID graph created for the Arcene dataset. Top 50 edges, each connecting a feature from the top 200 with another one.

	attribute	position_1	position_2	position_3	RI_1	RI_2	RI_3
1	V1184	1	1	1	0.1627335	0.1667576	0.1608779
2	V8502	2	2	2	0.1393340	0.1379264	0.1378373
3	V5473	3	3	3	0.1260721	0.1306319	0.1259599
4	V698	4	4	4	0.1234093	0.1245912	0.1246813
5	V7197	5	6	5	0.1178760	0.1110732	0.1172902
6	V4290	6	5	6	0.1062811	0.1192953	0.1164607
7	V7899	7	10	7	0.0962338	0.0928261	0.0991367
8	V9104	8	7	9	0.0928820	0.0974560	0.0942104
9	V7748	9	8	8	0.0910649	0.0958219	0.0944055
10	V9050	10	12	10	0.0883471	0.0800851	0.0888312
11	V1476	11	9	11	0.0861567	0.0929645	0.0859365
12	V8904	12	14	15	0.0803787	0.0763632	0.0754650
13	V5015	13	11	12	0.0787790	0.0827997	0.0831588
14	V6928	14	17	14	0.0782698	0.0740738	0.0805239
15	V86	15	16	16	0.0764142	0.0744112	0.0750206
16	V7696	16	15	19	0.0744866	0.0752617	0.0705253
17	V436	17	13	13	0.0733441	0.0780559	0.0812494
18	V6986	18	24	27	0.0726183	0.0676880	0.0655208
19	V3339	19	19	18	0.0718252	0.0691084	0.0720879
20	V9082	20	28	24	0.0715442	0.0659364	0.0686405

To analyze the scalability of our current implementation we run the `mcfs` procedure for a multiple number of threads for three different Intel CPUs machines. The result presented

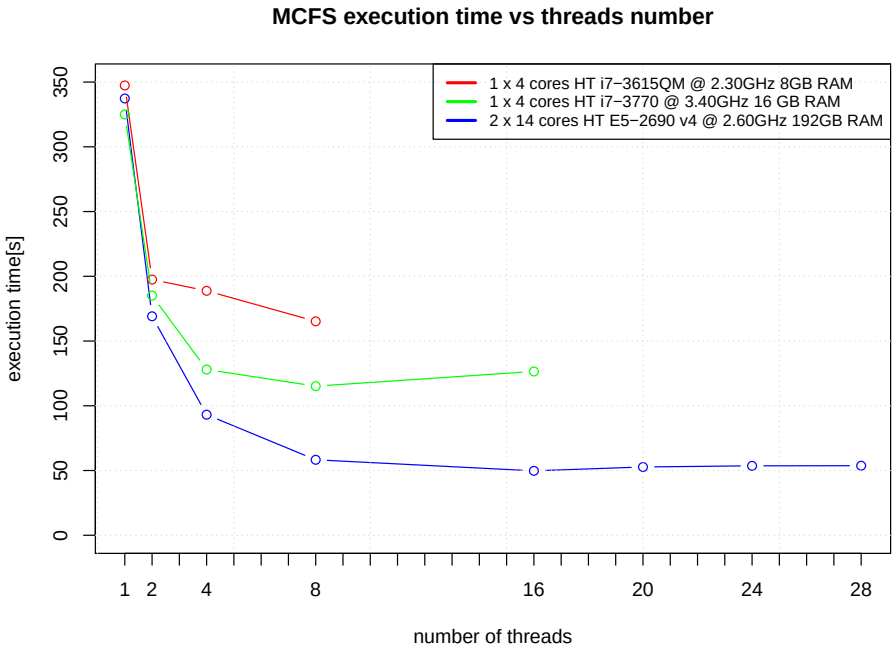

Figure 10: Execution time vs. number of threads (Arcene data).

in Figure 10 shows an initially nearly linear relation between time and threads number: time(th) = time($th = 1$)/th, where th is the number of threads, for a 14 cores CPU. Experiments show that the gain in speed highly depends on the architecture of the given CPU. Since hyper-threading (HT) technology is based on utilizing the same physical core by two threads simultaneously, it can only give a boost to the speed when the CPU is not well utilized by a single thread. For a multicore system (2×14), we noticed that there is no gain in speed from 16 threads on. We profiled the application to find possible concurrency between threads and locks for critical section execution but it has proved not to be the case. There are many other issues that can affect a Java multithreading program such as: memory bandwidth, L3 cache size limitation, or frequent garbage collector execution (see Qian, Li, Srisa-an, Jiang, and Seth 2015). However multithreading allows to run a calculation 7 times faster by using 8 threads vs. 1 on a 14 cores CPU (49 sec vs. 340 sec respectively).

5. Summary

In this paper, we presented the **rmcfs** package that can be successfully used for feature selection and interdependencies discovery. Here, we discussed one extraordinarily simple usage example and only one example using a real-life dataset, but the MCFS-ID algorithm has proven in past studies to be a very useful technique to limit the number of features and select the informative ones in various real high-dimensional problems; see Dramiński *et al.* (2008), Kierczak *et al.* (2009), Kierczak *et al.* (2010), Khaliq, Leijon, Belák, and Komorowski (2015), Enroth, Bornelöv, Wadelius, and Komorowski (2012), Dramiński *et al.* (2016). The ranking of features can be used to review features one by one (starting from the top) even in the case of *big data*. Using MCFS, one can build rule sets on top features and review their meaning; see Bornelov, Marillet, and Komorowski (2014). The ID part presents interdependencies be-

tween features in a consistent and comprehensive way. It shows not simple correlations but nonlinear relations between features that may reveal causality in the data (to be inferred from or confirmed by suitable background knowledge); see Dramiński *et al.* (2016). The novelty of the paper lies in presenting: (i) a concise and comprehensive description of an R package that automatically implements proper (default) settings of all crucial MCFS-ID parameters to obtain reliable results for data of any given size; (ii) ways of determining the cutoff value between informative and non-informative features; (iii) a scalability analysis of the current implementation in package **rmfcs**; (iv) an illustration of using discovered interdependencies (interactions) to take advantage in classification tasks.

Acknowledgments

We thank our close collaborators, Jan Komorowski, Michal J. Dabrowski, Klev Diamanti, Marcin Kierczak and Marcin Kruczyk, who have built on the MCFS-ID, most notably by providing a host of new insights and results within the area of bioinformatics. Our thanks also go to Julian Zubek who wrote some pieces of the R code. Last but not least, we thank the anonymous reviewers for most valuable comments and insightful suggestions.

References

Alizadeh AA, Eisen MB, Davis RE, Ma C, Lossos IS, Rosenwald A, Boldrick JC, Sabet H, Tran T, Yu X, Powell JI, Yang L, Marti GE, Moore T, Hudson Jr J, Lu L, Lewis DB, Tibshirani R, Sherlock G, Chan WC, Greiner TC, Weisenburger DD, Armitage JO, Warnke R, Levy R, Wilson W, Grever MR, Byrd JC, Botstein D, Brown PO, Staudt LM (2000). "Distinct Types of Diffuse Large B-Cell Lymphoma Identified by Gene Expression Profiling." *Nature*, **403**(6769), 503–511. doi:10.1038/35000501.

Archer KJ, Kimes RV (2008). "Empirical Characterization of Random Forest Variable Importance Measures." *Computational Statistics & Data Analysis*, **52**(4), 2249–2260. doi:10.1016/j.csda.2007.08.015.

Bornelöv S, Komorowski J (2016). "Selection of Significant Features Using Monte Carlo Feature Selection." In S Matwin, J Mielniczuk (eds.), *Challenges in Computational Statistics and Data Mining*, volume 605 of *Studies in Computational Intelligence*, pp. 25–38. Springer-Verlag, Cham. doi:10.1007/978-3-319-18781-5_2.

Bornelov S, Marillet S, Komorowski J (2014). "**Ciruvis**: A Web-Based Tool for Rule Networks and Interaction Detection Using Rule-Based Classifiers." *BMC Bioinformatics*, **15**(139). doi:10.1186/1471-2105-15-139.

Breiman L, Cutler A (2008). *Random Forests – Classification/Clustering Manual*. URL http://www.math.usu.edu/~adele/forests/cc_home.htm.

Chrysostomou K, Chen SY, Liu X (2008). "Combining Multiple Classifiers for Wrapper Feature Selection." *International Journal of Data Mining, Modelling and Management*, **1**(1), 91–102. doi:10.1504/ijdmmm.2008.022539.

Díaz-Uriarte R, De Andres SA (2006). "Gene Selection and Classification of Microarray Data Using Random Forest." *BMC Bioinformatics*, **7**(3). doi:10.1186/1471-2105-7-3.

Dramiński M, Dąbrowski MJ, Diamanti K, Koronacki J, Komorowski J (2016). "Discovering Networks of Interdependent Features in High-Dimensional Problems." In N Japkowicz, J Stefanowski (eds.), *Big Data Analysis: New Algorithms for a New Society*, Studies in Big Data, pp. 285–304. Springer-Verlag. doi:10.1007/978-3-319-26989-4_12.

Dramiński M, Kierczak M, Koronacki J, Komorowski J (2010). "Monte Carlo Feature Selection and Interdependency Discovery in Supervised Classification." In *Advances in Machine Learning II*, volume 263 of *Studies in Computational Intelligence*, pp. 371–385. Springer-Verlag. doi:10.1007/978-3-642-05179-1_17.

Dramiński M, Koronacki J (2018). ***rmcfs**: The MCFS-ID Algorithm for Feature Selection and Interdependency Discovery*. R package version 1.2.13, URL https://CRAN.R-project.org/package=rmcfs.

Dramiński M, Koronacki J, Komorowski J (2005). "A Study on Monte Carlo Gene Screening." In *Intelligent Information Processing and Web Mining*, volume 31 of *Advances in Soft Computing*, pp. 349–356. Springer-Verlag. doi:10.1007/3-540-32392-9_36.

Dramiński M, Rada-Iglesias A, Enroth S, Wadelius C, Koronacki J, Komorowski HJ (2008). "Monte Carlo Feature Selection for Supervised Classification." *Bioinformatics*, **24**(1), 110–117. doi:10.1093/bioinformatics/btm486.

Dua D, Karra Taniskidou E (2017). "UCI Machine Learning Repository." URL http://archive.ics.uci.edu/ml/.

Dudoit S, Fridlyand J (2003). "Classification in Microarray Experiments." In T Speed (ed.), *Statistical Analysis of Gene Expression Microarray Data*, volume 1, pp. 93–158. Chapman & Hall/CRC. doi:10.1201/9780203011232.ch3.

Enroth S, Bornelöv S, Wadelius C, Komorowski J (2012). "Combinations of Histone Modifications Mark Exon Inclusion Levels." *PloS ONE*, **7**(1), e29911. doi:10.1371/journal.pone.0029911.

Guyon I, Gunn SR, Ben-Hur A, Dror G (2005). "Result Analysis of the NIPS 2003 Feature Selection Challenge." In LK Saul, Y Weiss, L Bottou (eds.), *Advances in Neural Information Processing Systems 17*, pp. 545–552. MIT Press. URL http://papers.nips.cc/paper/2728-result-analysis-of-the-nips-2003-feature-selection-challenge.pdf.

Gyenesei A, Wagner U, Barkow-Oesterreicher S, Stolte E, Schlapbach R (2007). "Mining Co-Regulated Gene Profiles for the Detection of Functional Associations in Gene Expression Data." *Bioinformatics*, **23**(15), 1927–1935. doi:10.1093/bioinformatics/btm276.

Hall M, Frank E, Holmes G, Pfahringer B, Reutemann P, Witten IH (2009). "The **Weka** Data Mining Software: An Update." *ACM SIGKDD Explorations Newsletter*, **11**(1), 10–18. doi:10.1145/1656274.1656278.

Hastie T, Tibshirani R, Botstein D, Brown P (2001). "Supervised Harvesting of Expression Trees." *Genome Biology*, **2**(research0003.1). doi:10.1186/gb-2001-2-1-research0003.

Ho TK (1998). "The Random Subspace Method for Constructing Decision Forests." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **20**(8), 832–844. doi:10.1109/34.709601.

Huynh-Thu VA, Saeys Y, Wehenkel L, Geurts P (2012). "Statistical Interpretation of Machine Learning-Based Feature Importance Scores for Biomarker Discovery." *Bioinformatics*, **28**(13), 1766–1774. doi:10.1093/bioinformatics/bts238.

Khaliq Z, Leijon M, Belák S, Komorowski J (2015). "A Complete Map of Potential Pathogenicity Markers of Avian Influenza Virus Subtype H5 Predicted from 11 Expressed Proteins." *BMC Microbiology*, **15**(128). doi:10.1186/s12866-015-0465-x.

Kierczak M, Dramiński M, Koronacki J, Komorowski J (2010). "Computational Analysis of Molecular Interaction Networks Underlying Change of HIV-1 Resistance to Selected Reverse Transcriptase Inhibitors." *Bioinformatics and Biology Insights*, **4**, 137–146. doi:10.4137/bbi.s6247.

Kierczak M, Ginalski K, Dramiński M, Koronacki J, Rudnicki W, Komorowski J (2009). "A Rough Set-Based Model of HIV-1 Reverse Transcriptase Resistome." *Bioinformatics and Biology Insights*, **3**, 109–127. doi:10.4137/bbi.s3382.

Li Y, Campbell C, Tipping M (2002). "Bayesian Automatic Relevance Determination Algorithms for Classifying Gene Expression Data." *Bioinformatics*, **18**(10), 1332–1339. doi:10.1093/bioinformatics/18.10.1332.

Lu C, Devos A, Suykens JAK, Arús C, Huffel SV (2007). "Bagging Linear Sparse Bayesian Learning Models for Variable Selection in Cancer Diagnosis." *IEEE Transactions on Information Technology in Biomedicine*, **11**(3), 338–347. doi:10.1109/titb.2006.889702.

Nicodemus KK, Malley JD, Strobl C, Ziegler A (2010). "The Behaviour of Random Forest Permutation-Based Variable Importance Measures under Predictor Correlation." *BMC Bioinformatics*, **11**(110). doi:10.1186/1471-2105-11-110.

Paul J, Dupont P (2015). "Inferring Statistically Significant Features from Random Forests." *Neurocomputing*, **150**(Part B), 471–480. doi:10.1016/j.neucom.2014.07.067.

Qian J, Li D, Srisa-an W, Jiang H, Seth S (2015). "Factors Affecting Scalability of Multithreaded Java Applications on Manycore Systems." In *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 167–168. IEEE.

R Core Team (2018). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

Saeys Y, Inza I, Larrañaga P (2007). "A Review of Feature Selection Techniques in Bioinformatics." *Bioinformatics*, **23**(19), 2507–2517. doi:10.1093/bioinformatics/btm344.

Smyth GK, Yang YH, Speed T (2003). "Statistical Issues in cDNA Microarray Data Analysis." In MJ Brownstein, KA B (eds.), *Functional Genomics*, volume 224 of *Methods and Protocols*, pp. 111–136. Humana Press. doi:10.1385/1-59259-364-x:111.

Strobl C, Boulesteix AL, Kneib T, Augustin T, Zeileis A (2008). "Conditional Variable Importance for Random Forests." *BMC Bioinformatics*, **9**(307). doi:10.1186/1471-2105-9-307.

Strobl C, Boulesteix AL, Zeileis A, Hothorn T (2007). "Bias in Random Forest Variable Importance Measures: Illustrations, Sources, and a Solution." *BMC Bioinformatics*, **8**(25). doi:10.1186/1471-2105-8-25.

Tibshirani R, Hastie T, Narasimhan B, Chu G (2002). "Diagnosis of Multiple Cancer Types by Nearest Shrunken Centroids of Gene Expressions." *Proceedings of the National Academy of Sciences of the United States of America*, **99**(10), 6567–6572. doi:10.1073/pnas.082099299.

Tibshirani R, Hastie T, Narasimhan B, Chu G (2003). "Class Prediction by Nearest Shrunken Centroids, with Applications to DNA Microarrays." *Statistical Science*, **18**(1), 104–117. doi:10.1214/ss/1056397488.

Wright MN, Ziegler A, König IR (2016). "Do Little Interactions Get Lost in Dark Random Forests?" *BMC Bioinformatics*, **17**(145). doi:10.1186/s12859-016-0995-8.

Ziegler A, König IR (2014). "Mining Data with Random Forests: Current Options for Real-World Applications." *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, **4**(1), 55–63. doi:10.1002/widm.1114.

Affiliation:

Michał Dramiński
Institute of Computer Science
Polish Academy of Sciences
Jana Kazimierza 5, 01-248 Warsaw, Poland
E-mail: michal.draminski@ipipan.waw.pl
URL: http://www.ipipan.eu/staff/m.draminski/

Journal of Statistical Software http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
July 2018, Volume 85, Issue 12 *Submitted:* 2016-03-30
doi:10.18637/jss.v085.i12 *Accepted:* 2017-08-07