Contents lists available at ScienceDirect

Computational Statistics and Data Analysis

journal homepage: www.elsevier.com/locate/csda

Benchmark for filter methods for feature selection in high-dimensional classification data[☆]

Andrea Bommert[a,*], Xudong Sun[b], Bernd Bischl[b], Jörg Rahnenführer[a], Michel Lang[a]

[a] *Department of Statistics, TU Dortmund University, 44221 Dortmund, Germany*
[b] *Department of Statistics, Ludwig-Maximilians-Universität München, Ludwigstr. 33, 80539 München, Germany*

ARTICLE INFO

Article history:
Received 27 July 2018
Received in revised form 24 June 2019
Accepted 15 July 2019
Available online 19 September 2019

Keywords:
Feature selection
Filter methods
High-dimensional data
Benchmark

ABSTRACT

Feature selection is one of the most fundamental problems in machine learning and has drawn increasing attention due to high-dimensional data sets emerging from different fields like bioinformatics. For feature selection, filter methods play an important role, since they can be combined with any machine learning model and can heavily reduce run time of machine learning algorithms. The aim of the analyses is to review how different filter methods work, to compare their performance with respect to both run time and predictive accuracy, and to provide guidance for applications. Based on 16 high-dimensional classification data sets, 22 filter methods are analyzed with respect to run time and accuracy when combined with a classification method. It is concluded that there is no group of filter methods that always outperforms all other methods, but recommendations on filter methods that perform well on many of the data sets are made. Also, groups of filters that are similar with respect to the order in which they rank the features are found. For the analyses, the R machine learning package *mlr* is used. It provides a uniform programming API and therefore is a convenient tool to conduct feature selection using filter methods.

1. Introduction

Feature selection has become increasingly important for data analysis, machine learning, and data mining. Especially for high-dimensional data sets, it is necessary to filter out the irrelevant and redundant features by choosing a suitable subset of relevant features in order to avoid over-fitting and tackle the curse of dimensionality. With respect to data sets from the bioinformatics domain, feature selection often allows identifying the features that are important for biological processes of interest.

When fitting a statistical model for such high-dimensional data sets, one needs to decide first which feature selection method to use. As many of them exist, this is not an easy decision. Regarding the comparison of different methods, benchmark studies have gained increasing attention in the machine learning community (Fernández-Delgado et al., 2014).

In this paper, we benchmark state-of-the-art feature selection techniques on high-dimensional data sets. We compare 22 filter methods from different toolboxes on 16 high-dimensional classification data sets from various domains. We investigate which methods select the features of a data set in a similar order. Additionally, we search for optimal methods

[☆] The R source code for the analyses of this article is publicly available at https://github.com/bommert/filter-benchmark-paper.
[*] Corresponding author.
E-mail address: bommert@statistik.tu-dortmund.de (A. Bommert).

https://doi.org/10.1016/j.csda.2019.106839

with respect to predictive accuracy and run time. The results of our analyses show three groups of similar filter methods as well as several filter methods that are not very similar to any other filter method. Some of the filter methods seem to work better than others, but which filter methods perform best depends on the data set. There is no one-fits-all solution, but some recommendations can be made on the choice of filter methods.

In the past decades, many feature selection methods have been proposed. The methods can be categorized into three classes (Guyon and Elisseeff, 2003): *Filter methods* rank features by calculating a score for each feature independent of a model. Either the m features with the highest scores or all the features whose scores exceed a threshold τ are selected (with $m \in \mathbb{N}$ or $\tau \in \mathbb{R}$ being pre-specified). For many filter methods, the score calculation can be done in parallel. Lazar et al. (2012) give an extensive overview of existing filter methods. *Wrapper methods* (Kohavi and John, 1997) consider subsets of the set of all features. For each of the subsets, a supervised learning (e.g. classification) model is fitted. The subsets are evaluated by a performance measure calculated on the resulting model (e.g. classification accuracy). Wrapper methods include simple approaches like greedy sequential searches (Kittler, 1978), but also more elaborate algorithms like recursive feature elimination (Huang et al., 2018) as well as evolutionary and swarm intelligence algorithms for feature selection (Yang and Honavar, 1998; Xue et al., 2016; Brezočnik et al., 2018). *Embedded methods* include the feature selection in the model fitting process. Examples for predictive methods that perform embedded feature selection are Lasso regression (Tibshirani, 1996), tree based methods like classification and regression trees (Breiman et al., 1984) or random forests (Breiman, 2001), and gradient boosting (Biau et al., 2019). There are many overview papers that describe in detail, categorize, and suggest how to evaluate existing feature selection methods, e.g. Guyon and Elisseeff (2003), Liu and Yu (2005), Saeys et al. (2007), Tang et al. (2014), Chandrashekar and Sahin (2014), Hira and Gillies (2015), Jović et al. (2015), Li et al. (2018), Cai et al. (2018) and Venkatesh and Anuradha (2019).

There are also several papers in which feature selection methods are compared. In many of these, the feature selection methods are combined with classification methods in order to assess the predictive performance of the selected features. Liu et al. (2002) compare filter methods based on two gene expression data sets, counting the number of misclassified samples. Bolón-Canedo et al. (2013) analyze the classification accuracy of different filter, wrapper, and embedded methods on several artificial data sets. Bolón-Canedo et al. (2014) and Inza et al. (2004) compare filter methods with respect to classification accuracy based on microarray data sets. Forman (2003) and Aphinyanaphongs et al. (2014) conduct extensive comparisons based on text classification data sets. They analyze filter and wrapper methods, respectively. Darshan and Jaidhar (2018) compare filter methods with respect to classification accuracy on malware detection data. Liu (2004) and Peng et al. (2005) study filter methods on large data sets, analyzing the predictive accuracy with respect to the number of features to be chosen. Dash and Liu (1997) and Sánchez-Maroño et al. (2007) use small artificial data sets to assess whether the correct features are chosen. Dash and Liu (1997) compare different feature selection methods while Sánchez-Maroño et al. (2007) consider filter methods only. Wah et al. (2018) compare filter and wrapper methods on large simulated data sets with respect to the correctness of the chosen features. Additionally, they conduct comparisons with respect to classification accuracy on real data sets. Xue et al. (2015) comprehensively compare filter and wrapper methods with respect to classification accuracy and run time, considering each of the two objectives separately. Most of the data sets on which the comparison is based contain a small or medium number of features.

In some papers, only filter methods whose scores are based on similar concepts are compared. Both Meyer et al. (2008) and Brown et al. (2012) compare several filter methods that are based on mutual information. Meyer et al. (2008) analyze the accuracy and the run time of the methods separately. Additionally, they take into account theoretical properties and look at the percentages of correctly identified features on artificial data. Brown et al. (2012) assess the similarity of the filter methods with respect to feature subsets of size 10. Moreover, they analyze the classification accuracy with respect to the number of chosen features and search for Pareto optimal methods considering the accuracy and feature selection stability. Hall (1999) conducts an extensive study of correlation based feature selection. The author analyzes the classification accuracy based on real data sets as well as the choosing of relevant or irrelevant features based on artificial data sets.

Several authors introduce new feature selection methods and compare them in a benchmark study to competing approaches. Zhu et al. (2007) and Mohtashami and Eftekhari (2019) present new wrapper methods. The comparisons they conduct for the new methods also include filter methods. The authors assess the classification accuracy of the methods on several data sets. Zhu et al. (2007) also consider the number of features for which the best performance is achieved. Yu and Liu (2004), Fleuret (2004), and Ke et al. (2018) present new filter methods and conduct a comparison of their new methods with established feature selection methods. All of them consider the classification accuracy of the feature selection methods and the run time separately. Hoque et al. (2018) develop an ensemble filter method that aggregates the scores of several filter methods and compare it to the single filter methods. They compare the methods with respect to classification accuracy considering both small and high-dimensional data sets. Ghosh et al. (2019) create an ensemble filter method for guiding an evolutionary algorithm. This algorithm is compared to evolutionary algorithms guided by single filter methods with respect to classification accuracy. The comparison is based on high-dimensional data sets and small numbers of features to be chosen.

None of the above studies conduct an extensive comparison of feature selection methods on high-dimensional data with respect to both accuracy and run time jointly. However, for the analysis of high-dimensional data sets it is crucial to know which feature selection methods are suitable. For high-dimensional data, it is necessary to perform feature selection. Also, it is more difficult to perform feature selection than for low-dimensional data. The decision on which features should be selected is more complicated due to the curse of dimensionality. Moreover, the feature selection method must be fast to compute because of the large number of features.

In this work, we compare 22 filter methods on high-dimensional classification data sets. The filter methods compared by us are representatives of the most prominent general concepts for filter methods. These classes of filter methods are univariate statistical tests, univariate predictive performance indicators, feature variance, random forest importance and information theoretic measures. We use the R package *mlr* as a basis for our experiments. *mlr* is a comprehensive package for machine learning and a standard in the R community. It provides a unified platform that can also be used to concatenate filter methods with predictive methods. All of the compared filter methods have been integrated into *mlr* and are ready to use. We focus on the comparison of filter methods based on the following considerations. Most wrapper methods, evolutionary feature selection algorithms and recursive feature elimination methods are computationally infeasible for high-dimensional data sets. Embedded methods require that a certain predictive model is used. Most filter methods, however, are fast to calculate and can be combined with any kind of predictive method, even methods with embedded feature selection, see Bommert et al. (2017). We extensively compare different toolboxes with filter methods available in R (R Core Team, 2017). To the best of our knowledge, our approaches of considering the accuracy and the run time jointly as well as considering the similarity of the ranking of all features have not been analyzed yet by researchers in a filter comparison study.

The remainder of this paper is organized as follows: In Section 2, we explain a variety of filter methods. In Section 3, we describe the setup of the experiments we conduct to compare the filter methods and analyze their results. Section 3.3 investigates the similarity of the filter methods based on the feature ranking for several data sets and on their scaling behavior. In Section 3.4, we search for optimal filter methods with respect to predictive accuracy and run time. Section 4 contains a summary and the conclusions of our work.

2. Filter methods

All filter methods described in this section are applicable for classification data sets with numeric features. Some of the methods also work for categorical features. We present two kinds of filter methods: Most filter methods calculate a score for all features and then select the features with the highest scores. Some filter methods, however, select features iteratively in a greedy forward fashion. For these filters, in each iteration the feature with the maximal score is selected but the scores of different iterations are not comparable. Notation-wise, we consider a data set with n instances of the p features $X_1, \ldots, X_p$ and class variable Y. The filter methods in Sections 2.1–2.3 are univariate, i.e. they do not consider interactions between the p features. Most of the filter methods in Sections 2.4 and 2.5 are multivariate.

2.1. Univariate statistical tests

Filter *anova.test* performs for each feature an analysis of variance where the class variable is explained by the feature. The value of the F statistic is used as the score. The higher the F statistic, the more different are the mean values of the corresponding feature between the classes. For each feature X_k, the filter score is defined as

$$J_{\text{anova.test}}(X_k) = \frac{\sum_{i=1}^{l} n_i \left(\bar{x}_{i\bullet}^{(k)} - \bar{x}_{\bullet\bullet}^{(k)}\right)^2 / (l-1)}{\sum_{i=1}^{l} \sum_{j=1}^{n_i} \left(x_{ij}^{(k)} - \bar{x}_{i\bullet}^{(k)}\right)^2 / (n-l)}, \tag{1}$$

where l denotes the number of classes of Y and $x_{ij}^{(k)}$, $i \in \{1, \ldots, l\}, j \in \{1, \ldots, n_i\}$, denote the observed values of feature X_k for instances of class i. $\bar{x}_{i\bullet}^{(k)} = \frac{1}{n_i} \sum_{j=1}^{n_i} x_{ij}^{(k)}$ is the mean value of X_k in class i and $\bar{x}_{\bullet\bullet}^{(k)} = \frac{1}{n} \sum_{i=1}^{l} \sum_{j=1}^{n_i} x_{ij}^{(k)}$ is the mean value of X_k of all instances in the data set (Rasch et al., 2011, pp. 241 ff.).

Both filters *limma* and *sam* perform a moderated version of the F test. The basic idea of both approaches is to stabilize the estimation of the variance by using information about the variation of all features. For *limma*, this is done with a linear regression model (Smyth, 2004). For *sam*, the observed statistics are compared to the expected values of the statistics based on permutations (Tusher et al., 2001). Both methods are popular for gene expression data analysis. For more details about the two methods, the interested reader is referred to Smyth (2004) and Tusher et al. (2001).

Filter *kruskal.test* applies for each feature a Kruskal–Wallis rank sum test which is the non-parametric equivalent of the analysis of variance. Analogously, the test statistic is used as the filter score and higher values of this test statistic mean that the values of the corresponding feature differ more between the classes. The filter score for feature X_k is

$$J_{\text{kruskal.test}}(X_k) = \frac{12}{n(n+1)} \sum_{i=1}^{l} \frac{1}{n_i} \left(R_i^{(k)} - \frac{n_i(n+1)}{2}\right)^2, \tag{2}$$

where n_i denotes the number of instances in class i, $i \in \{1, \ldots, l\}$, and $R_i^{(k)} = \sum_{j=1}^{n_i} \text{rank}\left(x_{ij}^{(k)}\right)$ is the sum of the ranks of the observations of X_k belonging to class i among all observed values for X_k (Kruskal and Wallis, 1952).

Filter *chi.squared* performs for each feature a χ^2 test of independence between a dichotomized transformation of this feature and the class variable. The value of the χ^2 statistic is used as the score

$$J_{\text{chi.squared}}(X_k) = \sum_{i=1}^{l} \sum_{j=1}^{s} \frac{\left(o_{ij}^{(k)} - e_{ij}^{(k)}\right)^2}{e_{ij}^{(k)}}. \tag{3}$$

$o_{ij}^{(k)}$ denotes the observed number of instances with class i, $i \in \{1, \ldots, l\}$, and a value of X_k of the jth category, $j \in \{1, \ldots, s\}$. $e_{ij}^{(k)} = \frac{1}{n} \cdot \sum_{i=1}^{l} o_{ij}^{(k)} \cdot \sum_{j=1}^{s} o_{ij}^{(k)}$ is the expected number of instances with class i and a value of X_k of the jth category under the assumption of independence (Rasch et al., 2011, pp. 221). The higher the value of the χ^2 statistic, the higher the dependency between the corresponding feature and the class variable. To calculate this test, continuous features have to be discretized. For the implementation in the toolbox *FSelector* (Romanski and Kotthoff, 2016), this is done with the MDL method, see Section 2.6.

2.2. Univariate predictive performance

The score of the *auc* filter represents the classification accuracy when each feature is used directly and separately for class prediction. For each feature X_k, the following prediction rule for the class variable Y is used: $\widehat{Y} = \mathbb{I}_{[c,\infty)}(X_k)$, with $\mathbb{I}$ denoting the indicator function. The receiver operating curve displays the sensitivity and specificity of a classification rule for all choices of a threshold c, see Sammut and Webb (2011). The area under the receiver operating curve (AUC) of the classification rule $\widehat{Y} = \mathbb{I}_{[c,\infty)}(X_k)$ is used to measure how well X_k separates the target variable. An AUC value of 1 means that there is a threshold c for which the prediction rule is perfectly accurate. The value 0 indicates that there is a threshold c for which the rule predicts all labels wrongly which implies that X_k can achieve perfect classification with the rule $\widehat{Y} = \mathbb{I}_{(-\infty,c)}(X_k)$. A value of 0.5 is the worst possible in this application. The value 0.5 is attained, e.g. when the feature and the class variable are independent. Therefore,

$$J_{\text{auc}}(X_k) = |0.5 - \text{AUC}|\,, \tag{4}$$

where AUC is calculated for the classification rule $\widehat{Y} = \mathbb{I}_{[c,\infty)}(X_k)$, is used as the AUC filter score. This filter is only applicable for two-class data sets.

The idea of the simple association rule filter *oneR* (Romanski and Kotthoff, 2016) is to predict the class based on the value of a single feature. For this, continuous features have to be discretized in advance. For the implementation in the toolbox *FSelector* (Romanski and Kotthoff, 2016), this is done using the MDL method, see Section 2.6. The score of a feature X_k is calculated in the following way: Let $V^{(k)}$ denote the set of possible values for feature X_k. Also, for each value $v \in V^{(k)}$, let $n_{vi}^{(k)}$ denote the number of instances with $X_k = v$ and class i, $i \in \{1, \ldots, l\}$. A simple classification rule for instances with $X_k = v$ is predicting the class i with the highest count $n_{vi}^{(k)}$. The proportion of correctly classified instances by this rule without conducting any resampling is used as filter score:

$$J_{\text{oneR}}(X_k) = \frac{1}{n} \sum_{v \in V^{(k)}} \max_{i \in \{1,\ldots,l\}} n_{vi}^{(k)}. \tag{5}$$

The higher the accuracy of the rule, the more the feature X_k is considered as suitable for univariate class prediction.

Filter *univariate.model.score* (Bischl et al., 2016) fits a model for each feature in which only this feature is used to predict the class variable. Based on these models, the predictive performance of all features is assessed and the resulting filter score is

$$J_{\text{univariate.model.score}}(X_k) = \text{accuracy of univariate predictive model that only uses } X_k. \tag{6}$$

The implementation in *mlr* (Bischl et al., 2016) uses per default a classification tree with default hyper parameters (Therneau et al., 2017), measures the predictive performance by the accuracy (see Section 2.9) and performs a train–test split with ratio 2:1 in order to avoid overestimation of the accuracy. These specifications can be changed.

2.3. Variance

The *variance* filter uses the variance of a feature as its score

$$J_{\text{variance}}(X_k) = \frac{1}{n-1} \left(x_i^{(k)} - \frac{1}{n} \sum_{i=1}^{n} x_i^{(k)} \right)^2. \tag{7}$$

$x_i^{(k)}$, $i \in 1, \ldots, n$, denote the observed values of feature X_k. The idea of this filter is to get rid of features that only consist of noise and therefore have very little variation. This filter only makes sense for data where the features are measured on the same scale and have not been scaled to unit variance.

2.4. Random forest importance

Random forests are bagging ensembles with trees as base learners (Izenman, 2013, pp. 536 ff.). There are two popular ways of measuring feature importance based on a random forest: permutation importance and impurity importance. To calculate the permutation importance, the out of bag (oob) instances for each tree, i.e. the instances that were not used for fitting this tree, are considered. For the oob instances of each tree, feature X_k is permuted. Then the permuted instances are classified by the corresponding trees. The resulting classification accuracy (see Section 2.9) is compared to the classification accuracy without permuting feature X_k. The score of a permutation importance filter is the decrease in classification accuracy from original oob instances to permuted instances (Izenman, 2013, pp. 542 ff.). Features that are important for class prediction cause a large decrease in accuracy as their relevant information is not available when the feature is permuted. Filter *cforest.importance* is a permutation importance filter of a random forest with conditional inference trees as base learners. Filter *permutation* is a permutation importance filter for a forest of classification trees.

$$\begin{aligned} J_{\text{permutation}}(X_k) = J_{\text{cforest.importance}}(X_k) = {} & \text{accuracy for original oob instances} \\ & - \text{accuracy for oob instances with permuted values of } X_k. \end{aligned} \tag{8}$$

Filter *impurity* considers the node impurities of the trees. A node containing only instances of one class is called pure, a node with many instances of different classes is considered as impure. For each node in each tree of the forest, the impurity – before and after the split is made – is measured. This can be done for example with the Gini index. The filter score of feature X_k is the mean decrease in impurity due to the splits based on X_k (Izenman, 2013, pp. 542 ff.)

$$J_{\text{impurity}}(X_k) = \frac{\sum_{i \in N^{(k)}} (\text{impurity before node } i - \text{impurity after node } i)}{\left|N^{(k)}\right|}, \tag{9}$$

with $N^{(k)}$ denoting the set of nodes in the random forest in which a split based on X_k is made. A feature that is important for class prediction causes on average large decreases in impurity.

2.5. Mutual information

Let X and Y be two discrete variables with respective (empirical) probability mass function p. Then the entropy of Y is defined as

$$H(Y) = -\sum_y p(y) \log_2 (p(y)) \tag{10}$$

and the conditional entropy of Y given X is given by

$$H(Y|X) = \sum_x p(x) H(Y|X=x) = \sum_x p(x) \left(-\sum_y p(y|x) \log_2 (p(y|x)) \right). \tag{11}$$

The entropy measures the uncertainty of the variable. When all possible values occur with about the same probability, the entropy is high. If the probabilities of occurrence are very different from each other, the entropy is low. The mutual information of two variables is defined as

$$I(Y; X) = H(Y) - H(Y|X). \tag{12}$$

It can be interpreted as the decrease in uncertainty about Y conditional on knowing X. Considering the symmetry property $I(Y; X) = I(X; Y)$ it can also be seen as the amount of information shared by X and Y. There exist several filter methods that are based on this mutual information, see Hall (1999) and Brown et al. (2012) for more information. Continuous features have to be discretized before applying these filters. In the following, we describe filters from two different toolboxes, which calculate similar scores but differ in the way they discretize the features. The filters from the toolbox *FSelectorRcpp* (Zawadzki and Kosinski, 2017) use the MDL method for discretization, while the filters from the toolbox *praznik* (Kursa, 2018) perform a discretization into equally spaced intervals. For both discretization methods see Section 2.6.

Filter *info.gain* (Zawadzki and Kosinski, 2017) uses

$$J_{\text{info.gain}}(X_k) = I(Y; X_k), \tag{13}$$

the reduction of uncertainty about the class variable Y due to feature X_k, as the score for this feature.

The score of filter *gain.ratio* (Zawadzki and Kosinski, 2017) is

$$J_{\text{gain.ratio}}(X_k) = \frac{I(Y; X_k)}{H(X_k)}, \tag{14}$$

the ratio of the mutual information and the entropy of feature X_k. Out of two features with the same information about Y, this filter favors the feature with the smaller entropy, e.g. the feature that attains less different values. The reason for dividing by the entropy is to balance out the bias of the mutual information towards selecting features that take on many different values like credit card numbers or similar.

For filter *sym.uncert* (Zawadzki and Kosinski, 2017), the score

$$J_{\text{sym.uncert}}(X_k) = \frac{2 \cdot I(Y; X_k)}{H(X_k) + H(Y)} \tag{15}$$

is used. This score also reduces the bias towards features with many values and additionally normalizes it to the range [0,1].

Filter *MIM* (Kursa, 2018) ranks all features according to the information they share with the target variable Y

$$J_{\text{MIM}}(X_k) = I(Y; X_k). \tag{16}$$

This is the same score that filter *info.gain* uses as well. However, the filters differ in the way they discretize the features.

The following filter methods calculate the scores of all features iteratively. Thus, the features are selected in a greedy forward manner. Let S denote the set of already chosen features. S is initialized as $S = \{X_k\}$ with $I(Y; X_k) = \max\limits_{j \in \{1,\ldots,p\}} I(Y; X_j)$. In each iteration, the feature that maximizes the respective score is added to S.

Filter *MRMR* (Kursa, 2018) uses the score

$$J_{\text{MRMR}}(X_k) = I(Y; X_k) - \frac{1}{|S|} \sum_{X_j \in S} I(X_k; X_j). \tag{17}$$

The term $I(Y; X_k)$ measures the relevance of the feature by the information this feature has about Y. The term $\frac{1}{|S|} \sum_{X_j \in S} I(X_k; X_j)$ judges its redundancy by assessing the mean information that the feature shares with the features in S. The idea is to find maximally relevant and minimally redundant (MRMR) features.

For filter *JMI* (Kursa, 2018), the score

$$J_{\text{JMI}}(X_k) = \sum_{X_j \in S} I(Y; X_k, X_j) \tag{18}$$

is employed. $I(Y; X_k, X_j)$ is the amount of information about Y that X_k and X_j provide jointly. This quantity can be calculated by using the multivariate variable $X = (X_k, X_j)'$ in the definition of mutual information in Eq. (12). The idea of this score is to include features that are complementary to the already chosen features.

Filter *JMIM* (Kursa, 2018) is a modification of filter *JMI*. The score

$$J_{\text{JMIM}}(X_k) = \min_{X_j \in S} \left\{ I(Y; X_k, X_j) \right\} \tag{19}$$

considers the minimal joint information over all already selected features instead of the sum.

For filter *DISR* (Kursa, 2018), the score

$$J_{\text{DISR}}(X_k) = \sum_{X_j \in S} \frac{I(Y; X_k, X_j)}{H(Y, X_k, X_j)} \tag{20}$$

is used. Like *JMI*, it uses the information about Y provided jointly by X_k and X_j. But additionally, this information is divided by the joint entropy of Y, X_k and X_j. To obtain this entropy, consider the multivariate variable $\widetilde{Y} = (Y, X_k, X_j)'$ and plug it into the above definition of the entropy in Eq. (10). The reason for dividing by the entropy is to avoid selecting features that e.g. attain many different values, see filter *gain.ratio*.

Filter *NJMIM* (Kursa, 2018) is a modification of filter *DISR*. Its score

$$J_{\text{NJMIM}}(X_k) = \min_{X_j \in S} \left\{ \frac{I(Y; X_k, X_j)}{H(Y, X_k, X_j)} \right\} \tag{21}$$

considers the minimal relative joint information over all already selected features instead of the sum.

Filter *CMIM* (Kursa, 2018) has the score

$$J_{\text{CMIM}}(X_k) = \min_{X_j \in S} \left\{ I(Y; X_k | X_j) \right\}. \tag{22}$$

It uses the conditional mutual information

$$I(Y; X_k | X_j) = H(Y | X_j) - H(Y | X_k, X_j) \tag{23}$$

that can be interpreted as the difference in uncertainty about Y before and after X_k is known, while X_j is known anyway. The idea is to select features that provide much information about the class variable, given the information of the already selected features.

2.6. Discretization

Fayyad and Irani (1993) define the minimal description length (MDL) discretization method for continuous features. Their discretization method works recursively: A feature is split at an optimal cut point into two categories. Then, these categories are split recursively until a stopping condition is reached. Let $a_1 < \cdots < a_m$ denote the values of the feature to be discretized. Then the points that are considered as cut points are $\frac{1}{2}(a_1 + a_2), \ldots, \frac{1}{2}(a_{m-1} + a_m)$. The criterion for determining which of these cut points is optimal is the difference in entropy of the class variable before and after the split. The cut point with the greatest decrease in entropy is considered the best cut point. However, this split is only made if the decrease in entropy exceeds a boundary. This boundary is chosen such that the decrease in entropy is greater than the boundary if and only if the costs of performing the split are lower than the costs of not performing it. The costs are assessed by the minimal description length, which is an information theoretic measure. For details and formulas see Fayyad and Irani (1993). Not carrying out a split because of falling below the boundary works as stopping condition. When all recursions have stopped, some cut points $\tilde{a}_1 < \cdots < \tilde{a}_k$ are selected. The feature is then discretized into the categories $(-\infty, \tilde{a}_1], (\tilde{a}_1, \tilde{a}_2], \ldots, (\tilde{a}_k, \infty)$. Note that this method discretizes all values of a feature into one single category if the best cut point does not cause enough decrease in entropy.

A different method to discretize a numeric feature is to simply cut the range of values into q equally spaced intervals and use these intervals as categories. The number of intervals is determined as $q = \max\left\{\min\left\{\frac{n}{3}, 10\right\}, 2\right\}$ where n is the number observations in the data set (Kursa, 2018).

2.7. Overview of all considered filter methods

Table 1 gives an overview of all considered filter methods. If the target variable can be of type multi-class, also binary class variables are allowed. If categorical features are required, numeric features can be used as well. They are automatically discretized by the filter methods. All filter methods are available in the machine learning R package *mlr* (Bischl et al., 2016). If for the implementation in *mlr* other R packages are used, they are indicated.

2.8. Stability of feature selection

The stability of feature selection is defined as the robustness of the set of selected features with respect to different training data sets drawn from the same distribution (Kalousis et al., 2007). To quantify stability, stability measures are used. The stability measure that performs best both in theoretical (Nogueira and Brown, 2016) and in empirical (Bommert et al., 2017) comparisons is the Pearson correlation SC:

Assume that there is a data set containing n observations of the p features $X_1, \ldots, X_p$. Resampling is used to split the data set into m subsets. The feature selection method is then applied to each of the m subsets. Let $V_i \subset \{X_1, \ldots, X_p\}$, $i = 1, \ldots, m$, denote the set of chosen features for the ith subset of the data set. For each set of selected features V_i the vector $z_i \in \{0, 1\}^p$ is defined to indicate which features are chosen. The jth component of z_i is equal to 1 iff V_i contains X_j, i.e. $z_{ij} = \mathbb{I}_{V_i}(X_j)$, $j = 1, \ldots, p$. The resulting stability measure is

$$SC = \frac{2}{m(m-1)} \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} \mathrm{Cor}(z_i, z_j) \tag{24}$$

with $\mathrm{Cor}(z_i, z_j)$ denoting the Pearson correlation between z_i and z_j (Nogueira and Brown, 2016). The Pearson correlation measures the linear association between continuous variables. When applied to binary data like the vectors $z_1, \ldots, z_m$, the Pearson correlation is equivalent to the ϕ-coefficient for the contingency table of each two of these vectors (Rasch et al., 2011, pp. 339 f.). SC takes on values in the interval $[-1, 1]$. A value of 1 means maximal stability, 0 indicates that the feature selection method is as stable as a random feature selection.

2.9. Assessment of predictive performance

To evaluate the predictive performance of a classification method on a binary classification data set, the accuracy is defined as

$$\text{accuracy} = \frac{\text{number of correctly classified instances}}{\text{number of all classified instances}}. \tag{25}$$

The accuracy takes on values in the interval [0, 1]. The higher the accuracy, the better the predictive performance of the classification method.

Table 1
Names in this paper and in *mlr* and general concepts of the filter methods, requirements on the target variable and on the features, and R packages in which the filter methods are implemented. If categorical features are required, numeric features are discretized.

Name	Concept	Target	Features	R package
anova.test (anova.test)	univariate statistical test	multi-class	numeric or binary	*mlr* (Bischl et al., 2016)
limma (available on GitHub)	univariate statistical test	multi-class	numeric or binary	*limma* (Ritchie et al., 2015)
sam (available on GitHub)	univariate statistical test	multi-class	numeric or binary	*samr* (Tibshirani et al., 2011)
kruskal.test (kruskal.test)	univariate statistical test	multi-class	numeric or binary	*mlr* (Bischl et al., 2016)
chi.squared (FSelector_chi.squared)	univariate statistical test	multi-class	categorical	*FSelector* (Romanski and Kotthoff, 2016)
auc (auc)	univariate predictive performance	two-class	numeric	*mlr* (Bischl et al., 2016)
oneR (FSelector_oneR)	univariate predictive performance	multi-class	categorical	*FSelector* (Romanski and Kotthoff, 2016)
univariate.model.score (univariate.model.score)	univariate predictive performance	depends on model, here: multi-class	depends on model, here: numeric or categorical	depends on model, here: *rpart* (Therneau et al., 2017)
variance (variance)	feature variance	arbitrary	numeric	*mlr* (Bischl et al., 2016)
cforest.importance (party_cforest.importance)	random forest importance	multi-class	numeric or categorical	*party* (Strobl et al., 2008)
permutation (ranger_permutation)	random forest importance	multi-class	numeric or categorical	*ranger* (Wright and Ziegler, 2017)
impurity (ranger_impurity)	random forest importance	multi-class	numeric or categorical	*ranger* (Wright and Ziegler, 2017)
info.gain (FSelectorRcpp_ information.gain)	mutual information	multi-class	categorical	*FSelectorRcpp* (Zawadzki and Kosinski, 2017)
gain.ratio (FSelectorRcpp_ gain.ratio)	mutual information	multi-class	categorical	*FSelectorRcpp* (Zawadzki and Kosinski, 2017)
sym.uncert (FSelectorRcpp_ symmetrical.uncertainty)	mutual information	multi-class	categorical	*FSelectorRcpp* (Zawadzki and Kosinski, 2017)
MIM (praznik_MIM)	mutual information	multi-class	categorical	*praznik* (Kursa, 2018)
MRMR (praznik_MRMR)	mutual information	multi-class	categorical	*praznik* (Kursa, 2018)
JMI (praznik_JMI)	mutual information	multi-class	categorical	*praznik* (Kursa, 2018)
JMIM (praznik_JMIM)	mutual information	multi-class	categorical	*praznik* (Kursa, 2018)
DISR (praznik_DISR)	mutual information	multi-class	categorical	*praznik* (Kursa, 2018)
NJMIM (praznik_NJMIM)	mutual information	multi-class	categorical	*praznik* (Kursa, 2018)
CMIM (praznik_CMIM)	mutual information	multi-class	categorical	*praznik* (Kursa, 2018)

3. Experiments and results

In this section, we conduct and evaluate experiments to compare and to benchmark the filter methods described in Section 2. In Section 3.1, all data sets and their properties are given. Section 3.2 provides information about the classification methods used in the experiments. In Sections 3.3 and 3.4, we explain the setup of the experiments and analyze their results in detail. We perform two analyses. The first analysis in Section 3.3 aims to find out which filter

Table 2
Name, number of features, number of instances, relative size of majority class and source of the data sets, sorted by number of features. All data sets are binary classification data sets and contain only numeric features.

Name	p	n	Majority.perc	Source
scene	299	2 407	0.82	OpenML, ID 312
madelon	500	2 600	0.50	OpenML, ID 1485
gina_prior	645	3 468	0.51	OpenML, ID 1042
gina_agnostic	970	3 468	0.51	OpenML, ID 1038
christensen	1 413	198	0.57	datamicroarray
Internet-Advertisements	1 558	3 279	0.86	OpenML, ID 40 978
hiva_agnostic	1 617	4 229	0.96	OpenML, ID 1039
Bioresponse	1 776	3 751	0.54	OpenML, ID 4134
gravier	2 905	168	0.66	datamicroarray
gisette	4 971	7 000	0.50	OpenML, ID 41 026
chiaretti	12 625	111	0.67	datamicroarray
tian	12 625	173	0.79	datamicroarray
yeoh	12 625	143	0.55	datamicroarray
chin	22 215	118	0.64	datamicroarray
burczynski	22 283	101	0.58	datamicroarray
chowdary	22 283	104	0.60	datamicroarray

methods are similar with respect to feature ranking. We also compare the scaling behavior of the filter methods. In the second analysis in Section 3.4, we investigate the performance of the filter methods with respect to classification accuracy and run time.

3.1. Data sets

For our analyses, we use 16 large classification data sets from various domains. Information about these data sets is displayed in Table 2. The selected data sets are from *OpenML* (Vanschoren et al., 2013; Casalicchio et al., 2017) and the R package *datamicroarray* (Ramey, 2016). The data sets from *OpenML* have been selected according to the following criteria: They have exactly two classes, at least 250 features, more instances than features and they have no missing values or nominal features (binary features are converted to numeric features). Also, the smaller class should consist of at least 3% of the instances. For run time reasons, the product of the number of features and the number of instances should not exceed 100 000 000.

The data sets from the R package *datamicroarray* are high-dimensional data sets and contain more features than instances. We use data sets with two classes and create two-class data sets out of the data sets with more classes by selecting only the instances that belong to one of the two largest classes. We only take into account resulting data sets with at least 100 instances and without missing values. For the data sets *burczynski*, *chowdary*, and *tian* we perform a $\log(x+1)$ transformation to normalize the features. The other microarray data sets in Table 2 are already normalized. We omit all data sets where a normalization seems to be necessary, but for which the usual $\log(x+1)$ transformation is not possible due to negative values. Constant features are removed from all data sets.

The data sets contain 299 to 22 283 features and 101 to 7000 instances. The class imbalance reaches from 0.50 (perfect balance) to 0.96 (data set consists almost only of instances of the larger class).

3.2. Classification methods

We employ three classification methods to determine the predictive performance of subsets of features selected by a filter method. We choose specifically these classification methods because they are popular methods that do not perform embedded feature selection. This is important for judging the direct impact of the filter on prediction performance, not in combination with the subsequent classification algorithm that might perform an additional embedded feature selection. There are other state-of-the-art methods for feature selection like Lasso regression, but in this study we focus on filter methods. Also, for very high-dimensional data sets (several hundred thousand features) these methods may need to be applied to a pre-filtered data set in order to achieve acceptable run time or memory consumption. Table 3 displays the names of the methods, the corresponding hyper parameters that must be set and the R package from which the implementation is taken.

k Nearest Neighbors classifies a new instance by a majority vote of the k closest instances (Larose and Larose, 2014, pp. 149 ff.). In order to obtain an ordinary unweighted KNN method, we have to set the parameter *kernel* to *rectangular*.

Table 3
Classification methods with corresponding hyper parameters and R package of which the implementation is used.

Method	R package	Hyper parameters
k Nearest Neighbors (KNN)	kknn (Schliep and Hechenbichler, 2016)	$k \in \{1, 2, \dots, 20\}$
Logistic Ridge Regression (LRR)	glmnet (Simon et al., 2011)	$\lambda \in \{2^x : x \in [-15, 15]\}$
Support Vector Machine (SVM)	kernlab (Karatzoglou et al., 2004)	$C, \sigma \in \{2^x : x \in [-15, 15]\}$

Logistic Ridge Regression is logistic regression combined with a ridge penalty (Izenman, 2013, pp. 150 ff.). The ridge parameter λ balances the goodness of fit (log likelihood) and the size of the regression parameters. For $\lambda = 0$ this is ordinary logistic regression without the ridge penalty. If there are more features than instances in a data set or if there is a hyperplane in feature space which perfectly separates the two classes, $\lambda = 0$ is not feasible. Large values of λ cause all regression coefficients to be shrunken towards 0.

Support Vector Machines use the hyperplane in feature space that is optimal with respect to the maximum margin principle as decision boundary. Kernel functions are used to change the shape of the hyperplane into something non-linear (Izenman, 2013, pp. 369 ff.). We use Support Vector Machines with RBF kernel. There are two hyper parameters: the regularization parameter C and the kernel width parameter σ.

3.3. Similarity of the filter methods

3.3.1. Feature ranking

The goal of this analysis is finding out which filter methods are similar with respect to feature ranking. The question behind this analysis is if the filter methods can be grouped into sets with similar behavior and if these groups contain redundant information such that some members of each group can be neglected. For each data set, we compute the filter scores for all features. For the iterative *praznik* filters, we assess the selection order instead of the filter scores because the scores of different iterations are not comparable.

To assess the similarity of the filter methods, we compare the orders in which they select features. For each data set and each filter method, we determine the selection order of all features. Then we compute the rank correlation between the orders of all pairs of filter methods. This calculation is performed separately for each data set. The results are displayed in Section 1 of the supplementary material. To draw conclusions based on all data sets, we average the results for all data sets with the arithmetic mean. Fig. 1 displays the mean rank correlations between all pairs of filter methods. The higher the rank correlation between two filter methods, the more similar they are.

Fig. 1 shows three groups of similar filter methods. The first group consists of 6 out of the 7 *praznik* filters. The second group is formed by the filters *kruskal.test*, *auc*, *limma*, *anova.test*, and *sam*. The third group contains the filters *info.gain*, *chi.squared*, *sym.uncert*, *oneR*, and *gain.ratio* from the toolboxes *FSelector* and *FSelectorRcpp*. The other filter methods are not similar to any other filter method.

The average similarity value in is 0.5028. The highest mean rank correlations are observed between *anova.test* and *limma* (0.9998), *info.gain* and *chi.squared* (0.9989), *info.gain* and *sym.uncert* (0.9970), *limma* and *sam* (0.9954), *chi.squared* and *sym.uncert* (0.9949), *anova.test* and *sam* (0.9946), *kruskal.test* and *auc* (0.9676), *JMIM* and *JMI* (0.9520), *kruskal.test* and *limma* (0.9198), *kruskal.test* and *anova.test* (0.9195), and *kruskal.test* and *sam* (0.9148).

The similarity of the filters *kruskal.test*, *auc*, *anova.test*, *limma*, and *sam* is easy to understand. The Kruskal–Wallis test can be seen as the non-parametric equivalent of the analysis of variance. The filters *limma* and *sam* perform a moderated version of the F test conducted by filter *anova.test*. Also, there are strong links between the AUC and the Kruskal–Wallis test (Hanley and McNeil, 1982).

Considering the group of similar *praznik* filters, it appears plausible that they select features in a similar order because their scores are all based on mutual information. Within this group, *JMI* and *JMIM* as well as *DISR* and *NJMIM* are especially similar to each other. This makes sense as the scores are modified versions of each other, see Section 2. The filter *MRMR* comes from the same toolbox, but does not appear to be very similar to the other *praznik* filters. In contrast to the other *praznik* filters, *MRMR* also considers mutual information between features, see Section 2.

MIM and *info.gain* use the same score. However, the different discretization methods used by the two toolboxes cause an average rank correlation of only 0.6290 between the two filter methods. The filters from the toolboxes *FSelector* and *FSelectorRcpp* use different filter criteria but all employ the same discretization method. This makes it seem likely that the similarity of these filter methods is due to the same discretization method.

Considering the *ranger* toolbox, the filters *impurity* and *permutation* are not very similar (mean rank correlation 0.4582). Although both of them use a random forest to assess the feature importances, the resulting feature rankings are quite different on average.

All of the previous analyses are based on the aggregated rank correlations. A brief analysis of the plots in Section 1 of the supplementary shows that the similarity structure displayed in Fig. 1 does not represent the similarity structure for all single data sets. Instead, there are mainly two groups of data sets. The first group contains the high-dimensional data sets from the R package *datamicroarray* and some other data sets. The second group is formed by data sets from various

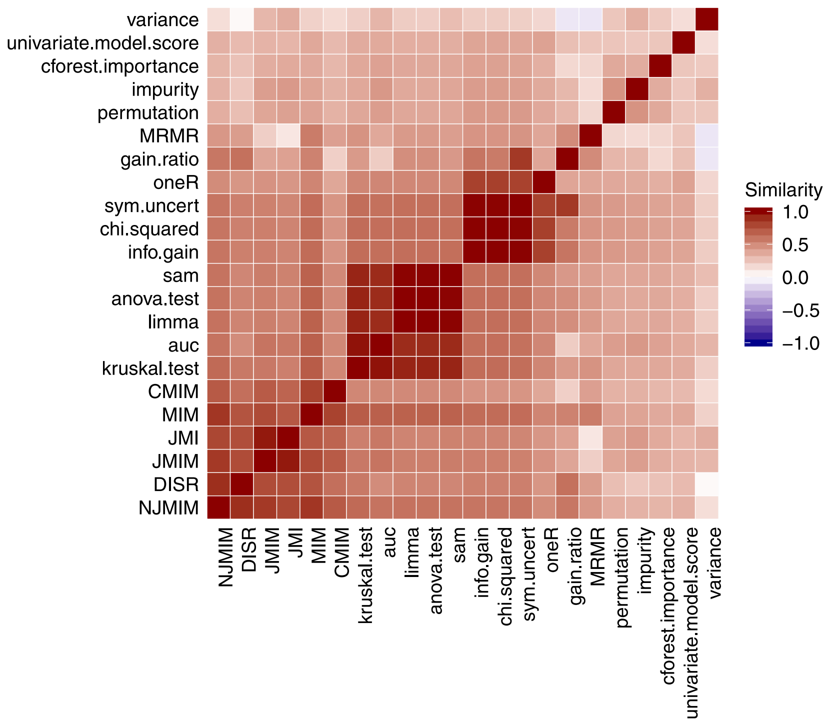

Fig. 1. Rank correlations between the selection order of all pairs of filter methods on all data sets, averaged by the arithmetic mean. The filter methods are ordered by average linkage hierarchical clustering using the mean rank correlation as similarity measure.

domains. In the first group, the similarity structure strongly resembles the one in Fig. 1. In the other group, most filter methods select the features in a similar order. There are only few filter methods that are not similar to the other methods, and these methods differ between the data sets of the second group.

For deciding which of the filter methods in the groups of similar filters can be neglected, we refer to the analysis in Section 3.4 where the filter methods are compared with respect to their performance.

3.3.2. Run times

A similarity analysis of the filter methods with respect to their run times and scaling behaviors can be found in Section 3 of the supplementary material.

3.4. Optimal filter methods

In this analysis, we investigate the performance of the filter methods with respect to predictive performance and run time.

3.4.1. Setup

To determine the predictive performance of the features chosen by the filter methods, we combine each filter method from Section 2 with a selected classification method from Section 3.2. The combined methods first apply the filter, choosing a given percentage of features, and then learn the classification rule using only the remaining features.

To ensure a fair comparison of the filter methods, we compare the performances of the filters with the best classifier and the best hyper parameter settings we found. More precisely, for each filter method and each data set, we tune the classification part and the percentage of features to be chosen simultaneously. We consider the classification method as a hyper parameter as well, which gives us a hierarchical search space. Possible configurations of this search space are for example (12% of features, KNN, $k = 7$) or (94% of features, SVM, $C = 2^{-1.3}$, $\sigma = 2^{13.28}$). We allow percentages of features to be chosen in the range [0%, 100%]. The ranges of the classification hyper parameters are given in Table 3. To find the best configuration, we conduct a random search with budget 100. To obtain unbiased estimates of the performances of the filters, we perform nested cross-validation with 10 outer and 10 inner iterations (Bischl et al., 2012). We make sure that the considered configurations are identical for all filter methods on the same data set in the same outer iteration, in order not to favor any filter method because it happens to be evaluated in combination with better configurations. Also, all filter methods use the same cross-validation splits. The experiments are conducted on a high performance compute cluster in a randomized order using the R package *batchtools* (Lang et al., 2017).

Table 4
Performance criteria and measures for evaluating the filter methods combined with a classification method.

Performance criterion	Performance measure	Aggregation across the 10 outer cross-validation iterations
Predictive Accuracy	Accuracy (see Section 2.9)	Arithmetic mean
Run time	Time for filtering	Median
Run time	Time for training the combined model (filter and classification method)	Median

For each filter method and each data set, we do the following: In each outer iteration, 10% of the data set is used as evaluation data. On the remaining 90% of the data, 10-fold cross-validation is conducted to estimate the performance of 100 randomly drawn configurations. We select the best configuration based on maximal mean accuracy. Ties are resolved by selecting the configuration with the smallest run time (median time for training the combined model). We use the median to aggregate the run times in order to obtain an estimate that is robust against variation caused by the high performance compute cluster. This tuning procedure makes sense because in practice, one is interested in filter methods that allow a good predictive performance, and among these methods, one is interested in methods with short run times. The selected configuration is then evaluated on the evaluation data, calculating the accuracy and the time for training the combined model. Additionally, we measure the time that is needed for filtering only. This way, for each data set and each filter, we obtain 10 evaluations of the best methods from the 10 outer iterations. We aggregate these values by calculating the mean accuracy and the median run times. In the end we have three performance values (mean accuracy, median time for filtering, and median entire run time) per filter method and data set. The performance criteria and measures are summarized in Table 4.

As an important baseline, we compare the results of the filters to results without filtering. Again, we perform a random search with a budget of 100 evaluations and nested cross-validation with 10 outer as well as 10 inner iterations. We calculate the same performance measures and select the best methods with respect to the same criteria. The only difference is that there is no need for tuning a filter percentage and that the run time consists only of the time for training the classification model because no filtering is done.

There are two configurations for which no results can be obtained. As all filters try the same configurations, this means that for all filters the results for the corresponding configuration are missing. One of the configurations is tried on data set *gina_agnostic*, the other one on *madelon*. The failing of both configurations is caused by the selection of less than two features by the filters and the implementation of Logistic Ridge Regression in *glmnet* that requires at least two features. We ignore the two configurations for which no results were obtained for the analyses.

3.4.2. Results

First, we compare the filter methods only with respect to the predictive performance. Fig. 2 shows for two data sets the accuracies of the 10 best configurations corresponding to the 10 outer cross-validation iterations, separately for all filter methods. Remember that the performances are evaluated on data that is not used for tuning. Therefore, the performance values may be interpreted as unbiased estimates of the performances on new data from the data generating process that created the respective data set. The results for the other 14 data sets are displayed in Section 2 of the supplementary material.

The left plot in Fig. 2 shows that for data set *gina_agnostic*, some filters perform considerably better than others. The filters *JMIM*, *permutation*, *impurity*, and *cforest.importance* perform quite well while the filters *DISR*, *MRMR*, and *variance* perform comparably bad. All filters lead on average to a better predictive performance than not filtering at all. The right plot in Fig. 2 demonstrates that for data set *gravier* there are only little differences in the central locations of the accuracies.

In Section 2 of the supplementary material, it can be observed that also for most of the other data sets there are only little differences between the majority of the filter methods with respect to the location of the predictive performance taking into account the spread. For the data sets *scene*, *madelon*, *gina_prior*, *gisette*, *chiaretti*, and *burczynski*, however, some noticeable differences can be observed. The filter methods *permutation*, *impurity*, and *cforest.importance*, lead to comparably high accuracy on several of these data sets. For the filters *JMIM*, *JMI*, *MRMR*, *NJMIM*, *variance*, and no filter a comparably good performance is observed for at least one of them. Not filtering at all, *MRMR*, and *DISR* lead to comparably low predictive accuracy on most of the data sets. The filters *variance*, *oneR*, and *univariate.model.score* perform bad on at least one of them.

Fig. 3 shows the number of times that the filter methods outperform each other. The number displayed in the row of filter A and the column of filter B indicates the number of data sets on which filter A is better than filter B with respect to mean accuracy. Given that two filter methods are equally good but never have exactly the same performance, we expect that each of the filters outperforms the other on approximately 8 of the 16 data sets.

We mark in red when a filter method is better than another filter method more often than 8 times and in blue when this happens less than 8 times. Note that there are several ties with respect to the accuracy of the filter methods. It can

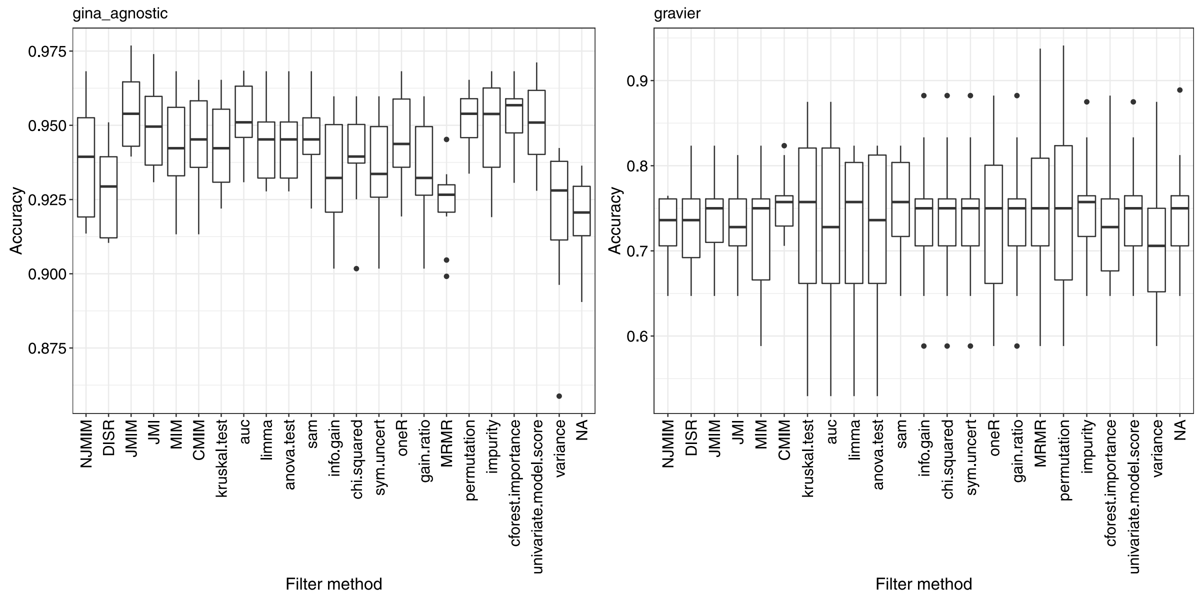

Fig. 2. Boxplots of the accuracies of the best configurations in the 10 outer cross-validation iterations per filter method and data set.

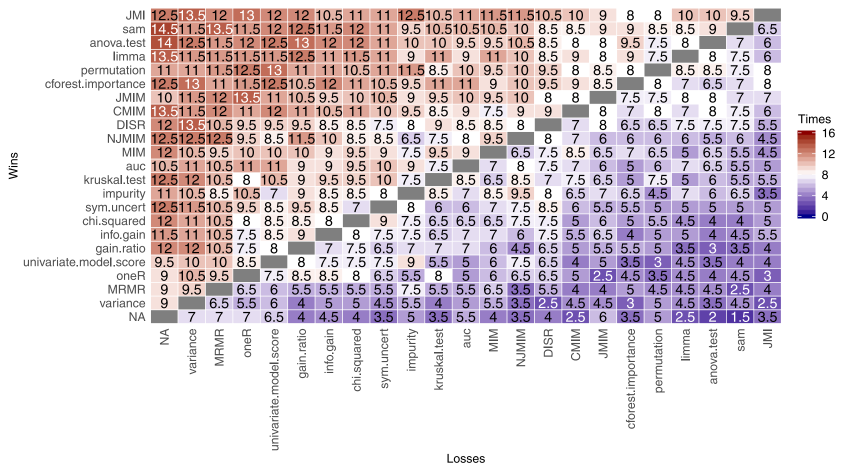

Fig. 3. Number of data sets on which the filter method in the row has a higher mean accuracy than the filter method in the column. Ties are counted as 0.5 for both filters. The filter methods are sorted decreasingly by row sums.

be observed that most filter methods are better than filter *variance* and not filtering at all on most data sets. Also, filter *MRMR* is outperformed by most of the other filters on many data sets. Filters *JMI*, *sam*, *anova.test*, *limma*, *permutation*, and *cforest.importance* are only outperformed by few other filter methods and they outperform many other filter methods themselves on most data sets.

Concluding, there is no filter method that is better than all the other methods on all data sets. However, there are some filters like *permutation* and *cforest.importance* that achieve high accuracy on some data sets and only have a comparably poor performance on few data sets in the analysis.

Next, we compare the filter methods with respect to predictive performance and run time across data sets. For the analysis presented in Fig. 3, we only looked at good or bad predictive performances in comparison to the other filter methods. This analysis also takes into account how much the filter methods differ in performance. We aggregate the performance values of the 10 outer cross-validation iterations into one value per performance measure. Remember that we analyze the accuracy of the best configuration found and the time for fitting the best model. In the following, we first investigate the run time for filtering only and later, we consider the entire run time for filtering and fitting the best classification model.

Fig. 4 displays the mean accuracy and the median run time for filtering of the filter methods separately for all data sets. Fig. 5 aggregates the information of Fig. 4 into one graphic. The right plot focuses on parts of the left plot. To compare the

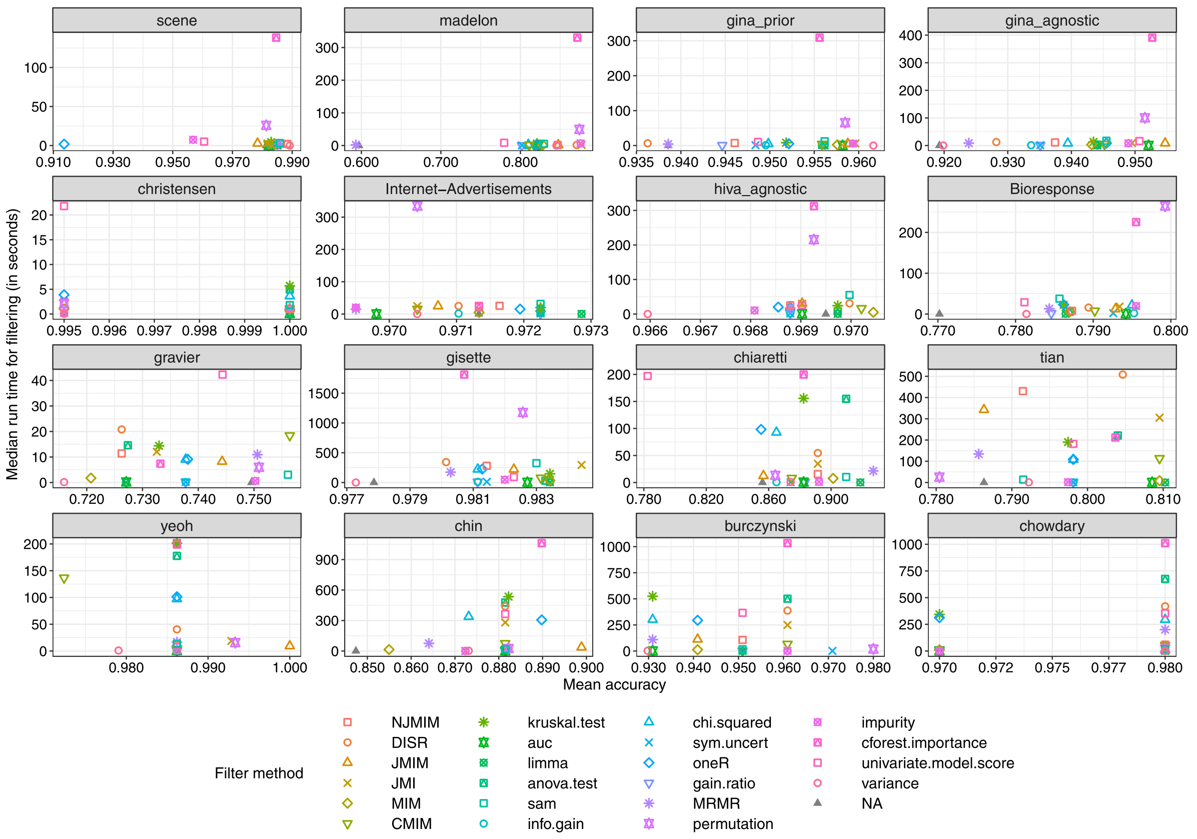

Fig. 4. Mean accuracy and median run time for filtering of filter methods with optimal configurations per data set.

predictive accuracy across data sets, relative accuracies are considered. More precisely, the differences between the mean accuracies of all filter methods to the highest mean accuracy observed on the same data set are considered. Therefore, the smaller the relative accuracy, the better the predictive performance of the filter method when combined with the best configuration. The best filter method per data set always has relative mean accuracy 0. For the time for filtering, the fastest run time (not filtering at all) is already 0 for all data sets. The (transformed) performance criteria are then aggregated over the data sets by computing the median values as well as the upper and lower quartiles. The median value provides information about the central location of the performance measures. The distance of the quartiles shows the variation of the performances across data sets. Fig. 5 displays the median of the performance criteria values for each filter method by a symbol. The quartiles are located at the respective ends of the horizontal and vertical lines. The longer the lines, the greater the variation across data sets. The optimal filter method would be located in $(0, 0)'$.

The left plot in Fig. 5 shows that the relative accuracy of the filter methods and the time for filtering vary across data sets. The interquartile range of the relative mean accuracy across data sets is around 0.02 for most filter methods. As this is a small number, it means that the median value summarizes the central location well. However, the variation is too large to draw conclusions about which filter methods perform best only based on the median. The left plot also shows that filter *cforest.importance* needs much longer for filtering than the other filter methods and that its accuracy is among the best, but not the single best.

The right plot shows the median relative accuracy values and the variation in accuracy for all filter methods except for *cforest.importance*. The variation in time for filtering is omitted because only a small part of the y-axis is investigated. Considering the median performances in both criteria, it can be seen that the filters *JMIM*, *impurity*, *auc*, *limma*, and *variance* as well as not filtering at all are Pareto optimal. Pareto optimality means that there is no other method that performs as good in all criteria and better in at least one criterion. Considering the non Pareto optimal filter methods in Fig. 5, at least one of the Pareto optimal filters performs better in both criteria. Filter *JMIM* is Pareto optimal because it has the best median accuracy among all filter methods. Filter methods *impurity*, *auc*, and *limma* have a good median accuracy and a low median run time for filtering. The *variance* filter and not filtering at all are Pareto optimal because of their fast run time. Applying no filter takes no time, which is always faster than applying any filter and therefore this is always Pareto optimal, independent of its accuracy.

If we now consider the upper quartile of the relative mean accuracy, i.e. the right ends of the horizontal lines, as measure of the predictive accuracy instead of the median value, we obtain a different set of Pareto optimal methods. In

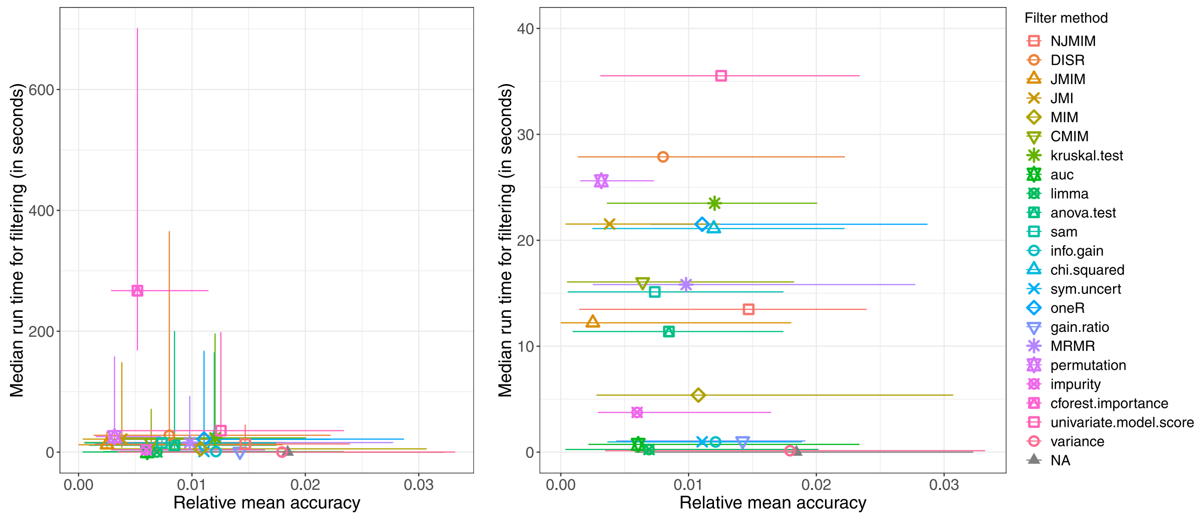

Fig. 5. Relative mean accuracy and median run time for filtering of the filter methods with optimal configurations aggregated over all data sets. The median of both performance measures (relative mean accuracy and median filtering time) across all data sets is displayed by a symbol. The upper and lower quartiles are located at the respective ends of the horizontal and vertical lines. The right plot is a part of the left plot: it focuses on small run times. The optimal filter method would be located in $(0, 0)'$.

this case, *permutation*, *JMI*, *impurity*, *sym.uncert*, *info.gain*, *limma*, and no filter are Pareto optimal. Especially *permutation* but also *JMI* and *impurity* are Pareto optimal because of high accuracy. The others are Pareto optimal because of low run times. Filter *impurity* seems to provide a good compromise of accuracy and run time for filtering. It should be noted that we have selected the configurations based on accuracy. Because the run times of the filters from package *praznik* depend on the number chosen features (see Section 3 of the supplementary material), it might be possible to make them Pareto optimal by selecting fewer features. This would make them faster at the cost of predictive accuracy. As the run times depend on the implementation, any filter method could potentially become Pareto optimal if it were implemented efficiently enough.

Now we analyze the predictive accuracy in combination with the entire run time for filtering and fitting the best model. Considering the entire run time makes sense because it also assesses how long it takes to fit the best classification model with the features selected by the filter. As most filters rank all features and because there are groups of filters with very similar run times (see Section 3 of the supplementary material), it is likely that many filters achieve similar run times even if they choose different numbers of features. However, selecting more features may have a huge impact on the run time for fitting the predictive model of interest.

Because we observe some very long run times in this analysis, we consider the logarithmic run times instead. To be able to apply the logarithm, we have to add a small constant to all run times, as KNN without filtering can be so fast that its run time is measured as 0. For comparing the predictive accuracy and run time across data sets, relative accuracies and relative logarithmic run times are considered. A relative mean accuracy of x means that the mean accuracy of the filter (with the best configuration found) is worse than the mean accuracy of the best filter on the same data set, by an additive factor of x. A relative logarithmic median run time of x means that the median run time of the filter equals the median run time of the best filter on the same data set multiplied with 10^x.

Fig. 6 is an analogous plot to Fig. 5 but shows the logarithmic run time for fitting the best model instead of only the run time for filtering. In comparison to Fig. 5, the relative run time for *permutation* is lower. For the filters *variance* and *anova.test* and not filtering at all, the relative run time is higher than in Fig. 5. When considering only the median values of the performance measures, the filter methods *JMIM*, *impurity*, *auc*, and *limma* are Pareto optimal. With respect to the upper quartile of the relative mean accuracy and the median of the relative logarithmic run time, *permutation*, *impurity*, *sym.uncert*, *info.gain*, and *limma* are Pareto optimal. The sets of Pareto optimal methods here are subsets of the Pareto optimal methods in the analyses based on the run time for filtering only. The filter methods that are not Pareto optimal any more when considering the entire run time are *variance* and no filter, which have a very short time for filtering, as well as *JMI*, which is outperformed by *permutation* with respect to the time for model fitting.

Like in the discussion of Fig. 5, it also has to be mentioned that filter methods could become Pareto optimal by sacrificing accuracy and thus saving run time. Here, this is possible for all filter methods by selecting a configuration that is faster but provides less predictive accuracy. Also, the variation in the performance criteria should be taken into account when interpreting the Pareto optimal methods: there is no subset of filter methods that is clearly better than the others. Filter methods like *permutation* and *impurity* seem favorable because of a comparably good performance and a comparably low variation.

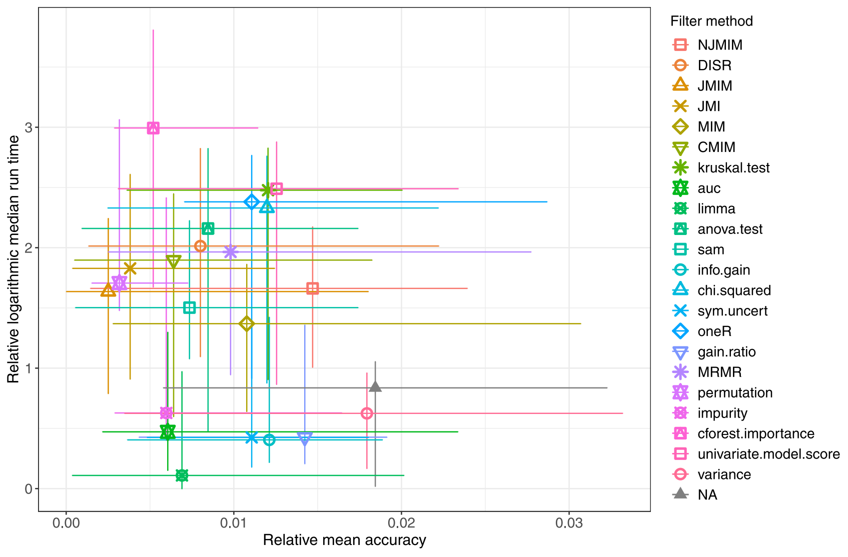

Fig. 6. Relative mean accuracy and relative logarithmic median run time of the filter methods with optimal configurations aggregated over all data sets. The median of both performance measures (relative mean accuracy and relative logarithmic median run time) across all data sets is displayed by a symbol. The upper and lower quartiles are located at the respective ends of the horizontal and vertical lines. The optimal filter method would be located in $(0, 0)'$.

To sum up, there is no subset of filter methods that outperforms all other filter methods. Which filters work best depends on the data set. The results are subject to variation within and across data sets. Nevertheless, some filter methods like *permutation*, *impurity*, *sym.uncert*, *info.gain*, *limma*, *auc*, and *JMIM* seem to work well in many data situations.

Fig. 4 allows a comparison of the two studies in Sections 3.3 and 3.4. It shows for each data set the mean accuracy and the run time for filtering for all filter methods combined with the best configurations. It can be observed that the filter methods that are similar according to our analysis in Section 3.3.1 often have a similar predictive performance when applied on the same data set. However, in some cases even very similar filter methods lead to models with quite different predictive accuracy. This can be explained by the fact that the inclusion or omission of a single relevant feature can already cause big differences in accuracy. Also, filters that needed a similar amount of time for filtering in Section 3 of the supplementary material often have similar run times for filtering on the other data sets as well. Regarding the question whether some of the filter methods can be neglected when searching for a good filter method, it seems reasonable to limit the search space to *permutation*, *impurity*, *sym.uncert*, *limma*, and *JMIM*. With respect to the analyses in Section 3.3, the filters *JMIM*, *limma*, *sym.uncert* can be interpreted as representatives of their groups.

3.4.3. Stability

Another aspect for the comparison of filter methods is their stability. In Section 3.4.2, we have conducted a random search for a good configuration with respect to accuracy for each data set and each filter method. We have performed a nested cross-validation with 10 outer iterations. For each outer iteration, we have chosen one best configuration, which results in 10 configurations per data set and filter method.

Now, we analyze the stability of the feature selection for each filter method on each data set. To do so, we compare the 10 respective sets of selected features and quantify the stability with the stability measure *SC*, see Section 2.8. For comparability across data sets, we transform each stability value x into $x + 1 - m$ where m is the maximal stability value observed for any of the filters on the same data set. This sets the best stability value on each data set to 1. With respect to stability, high values are desirable. When no filter is applied, no feature selection is performed and the stability of feature selection cannot be assessed. *SC* is not defined in this situation.

Fig. 7 shows boxplots of the relative stability values of the filter methods. We observe that all filter methods from the toolbox *FSelectorRcpp* as well as the filters *chi.squared*, *auc*, and *sam* achieve comparably high stability. From our analyses in Section 3.4.2 we know that most of these filter methods are fast in computation. Filters *permutation* and *impurity*, which showed good results in Section 3.4.2 with respect to predictive performance, attain comparably low stability values.

The comparably low stability of the random forest based filter methods and of filter *univariate.model.score* can be explained with the stochasticity of these methods. When a random forest model is fitted to a data set, the output is not deterministic, because the construction of each tree involves a random choice of observations and features. So, when fitting a random forest several times to the same data set, the resulting models are generally a bit different from each other. Now, when fitting random forest models to slightly different data sets, the resulting models are expected to vary even more. With respect to filter *univariate.model.score*, the choice of train and test data is stochastic. When this filter

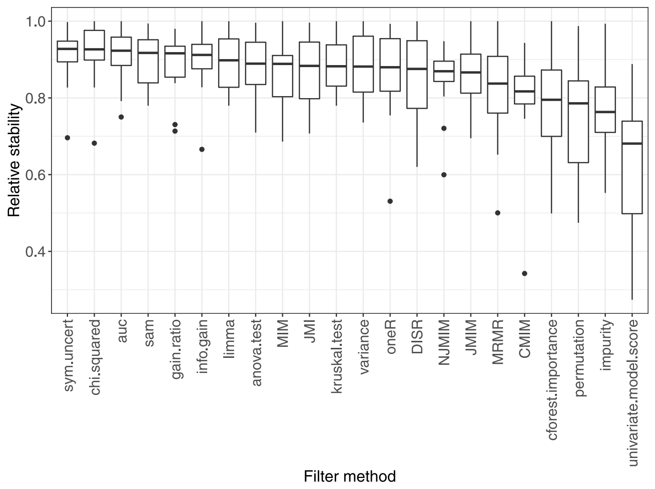

Fig. 7. Relative stability of the filter methods with optimal configurations for all data sets, sorted by median relative stability.

is applied several times to the same data set, the results can be different due to different train–test splits. For slightly different data sets, even larger differences are expected.

4. Summary and conclusions

Feature selection is a key part of data analysis, machine learning and data mining. There are many methods for feature selection, but it is unclear which of these methods perform best. In this analysis we focused on the comparison of filter methods for feature selection. We analyzed 22 filter methods based on 16 high-dimensional classification data sets from various domains.

To find out which of the filter methods are similar with respect to the order in which they rank the features, we computed rank correlations. We observed that for some data sets, especially the data sets containing gene expression data with large number of features, there were three groups of similar filter methods and many filter methods that were not similar to any other method. The filters that were similar to each other mostly came from the same toolboxes. For the other data sets, most filter methods were very similar. Also, we investigated the scaling behavior of the filter methods, identifying groups of filters that behave similarly with respect to run time.

Next, we analyzed the classification accuracy of the features selected by the filter methods and the run time needed for feature selection and for building a good classification model based on the selected features. We found out that there is no subset of filter methods that performs better than the rest of the filter methods on all data sets. Instead, the best filter methods differed between the data sets. Nevertheless, on average all filter methods performed better than not filtering at all. Also, even though there was no clear winner, the random forest importance filter methods *permutation* and *impurity*, the information theoretic filter methods *sym.uncert* and *JMIM*, and the univariate statistical test filter method *limma* performed well on most data sets, which makes them seem advisable. The filters *sym.uncert*, *limma*, and *JMIM* can be seen as representatives of the three groups of similar filter methods. Filters *impurity*, *sym.uncert*, and *limma* achieved very low run times. For filter *permutation* we observed comparably high run times but also very high predictive accuracy.

Choosing the best filter method for a new data set is a matter of available computational resources. Based on our analyses, we come to the following conclusions. If the computational resources only allow trying one filter method, we recommend *permutation*. This filter method allowed fitting classification models with high accuracy on most of the data sets. If some computational resources are available for finding a suitable filter method, we recommend trying *permutation*, *impurity*, *sym.uncert*, *limma*, and *JMIM*. If the resources allow trying all filter methods, we recommend doing this, because only this way the very best filter method for a new data set can be found. Searching for the best filter method can be done by considering it as a tuning parameter, just like we did with respect to the classification method in our experiments.

This paper serves as a reference to people who would like to conduct feature filtering. It can help them choose appropriate filters according to their application scenario, computational resources, and so on. In the future, we will use the results of our analyses to determine the search space for good models. Our analyses could be extended by additionally considering other feature selection methods apart from pure filter methods, which was out of scope for this analysis. Also, in this paper, we performed tuning with respect to predictive performance and then analyzed not only the predictive performance but also the run time and the stability. In future analyses, one could perform multi-criteria tuning with respect to all performance criteria at the drawback of a much more complicated aggregation of the results. Another interesting aspect of research would be discovering data set characteristics that indicate which filter and classification methods perform best. If such characteristics were available, much run time could be saved in the tuning process. Ideally, the characteristics should be cheap to compute in order to obtain a benefit in run time over trying different methods. In the field of optimization, exploratory landscape analysis (Kerschke and Trautmann, 2019) is an approach for defining characteristics of objective functions in order to automatically select the best optimization algorithm. In future research, this approach could be transferred to model selection.

Acknowledgments

This work was supported by Deutsche Forschungsgemeinschaft (DFG), Projects RA 870/7-1 and BI 1902/1-1, and Collaborative Research Center SFB 876, A3.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.csda.2019.106839.

References

Aphinyanaphongs, Y., Fu, L.D., Li, Z., Peskin, E.R., Efstathiadis, E., Aliferis, C.F., Statnikov, A., 2014. A comprehensive empirical comparison of modern supervised classification and feature selection methods for text categorization. J. Assoc. Inf. Sci. Technol. 65 (10), 1964–1987.
Biau, G., Cadre, B., Rouvìère, L., 2019. Accelerated gradient boosting. Mach. Learn. 108 (6), 971–992.
Bischl, B., Lang, M., Kotthoff, L., Schiffner, J., Richter, J., Studerus, E., Casalicchio, G., Jones, Z.M., 2016. mlr: Machine learning in R. J. Mach. Learn. Res. 17 (170), 1–5.
Bischl, B., Mersmann, O., Trautmann, H., Weihs, C., 2012. Resampling methods for meta-model validation with recommendations for evolutionary computation. Evol. Comput. 20 (2), 249–275.
Bolón-Canedo, V., Sánchez-Maroño, N., Alonso-Betanzos, A., 2013. A review of feature selection methods on synthetic data. Knowl. Inf. Syst. 34 (3), 483–519.
Bolón-Canedo, V., Sánchez-Marono, N., Alonso-Betanzos, A., Benítez, J.M., Herrera, F., 2014. A review of microarray datasets and applied feature selection methods. Inform. Sci. 282, 111–135.
Bommert, A., Rahnenführer, J., Lang, M., 2017. A multicriteria approach to find predictive and sparse models with stable feature selection for high-dimensional data. Comput. Math. Methods Med. 2017.
Breiman, L., 2001. Random forests. Mach. Learn. 45 (1), 5–32.
Breiman, L., Friedman, J., Stone, C.J., Olshen, R., 1984. Classification and Regression Trees. CRC Press, Boca Raton, FL, USA.
Brezočnik, L., Fister, I., Podgorelec, V., 2018. Swarm intelligence algorithms for feature selection: A review. Appl. Sci. 8 (9).
Brown, G., Pocock, A., Zhao, M.-J., Luján, M., 2012. Conditional likelihood maximisation: A unifying framework for information theoretic feature selection. J. Mach. Learn. Res. 13, 27–66.
Cai, J., Luo, J., Wang, S., Yang, S., 2018. Feature selection in machine learning: A new perspective. Neurocomputing 300, 70–79.
Casalicchio, G., Bossek, J., Lang, M., Kirchhoff, D., Kerschke, P., Hofner, B., Seibold, H., Vanschoren, J., Bischl, B., 2017. OpenML: An R package to connect to the machine learning platform OpenML. Comput. Stat. 1–15.
Chandrashekar, G., Sahin, F., 2014. A survey on feature selection methods. Comput. Electr. Eng. 40 (1), 16–28.
Darshan, S.S., Jaidhar, C., 2018. Performance evaluation of filter-based feature selection techniques in classifying portable executable files. Procedia Comput. Sci. 125, 346–356.
Dash, M., Liu, H., 1997. Feature selection for classification. Intell. Data Anal. 1, 131–156.
Fayyad, U., Irani, K., 1993. Multi-Interval Discretization of Continuous-Valued Attributes for Classification Learning. Technical report, California Institute of Technology.
Fernández-Delgado, M., Cernadas, E., Barro, S., Amorim, D., 2014. Do we need hundreds of classifiers to solve real world classification problems? J. Mach. Learn. Res. 15, 3133–3181.
Fleuret, F., 2004. Fast binary feature selection with conditional mutual information. J. Mach. Learn. Res. 5, 1531–1555.
Forman, G., 2003. An extensive empirical study of feature selection metrics for text classification. J. Mach. Learn. Res. 3, 1289–1305.
Ghosh, M., Adhikary, S., Ghosh, K.K., Sardar, A., Begum, S., Sarkar, R., 2019. Genetic algorithm based cancerous gene identification from microarray data using ensemble of filter methods. Med. Biol. Eng. Comput. 57 (1), 159–176.
Guyon, I., Elisseeff, A., 2003. An introduction to variable and feature selection. J. Mach. Learn. Res. 3, 1157–1182.
Hall, M.A., 1999. Correlation-Based Feature Selection for Machine Learning (Ph.D. thesis). University of Waikato, Hamilton, New Zealand.
Hanley, J.A., McNeil, B.J., 1982. The meaning and use of the area under a receiver operating characteristic (ROC) curve. Radiology 143 (1), 29–36.
Hira, Z.M., Gillies, D.F., 2015. A review of feature selection and feature extraction methods applied on microarray data. Adv. Bioinform. 2015.
Hoque, N., Singh, M., Bhattacharyya, D.K., 2018. EFS-MI: An ensemble feature selection method for classification. Complex Intell. Syst. 4 (2), 105–118.
Huang, X., Zhang, L., Wang, B., Li, F., Zhang, Z., 2018. Feature clustering based support vector machine recursive feature elimination for gene selection. Appl. Intell. 48 (3), 594–607.
Inza, I., Larrañaga, P., Blanco, R., Cerrolaza, A.J., 2004. Filter versus wrapper gene selection approaches in DNA microarray domains. Artif. Intell. Med. 31 (2), 91–103.
Izenman, A.J., 2013. Modern Multivariate Statistical Techniques: Regression, Classification, and Manifold Learning, second ed. Springer, New York, USA.
Jović, A., Brkić, K., Bogunović, N., 2015. A review of feature selection methods with applications. In: 38th International Convention on Information and Communication Technology, Electronics and Microelectronics, pp. 1200–1205.
Kalousis, A., Prados, J., Hilario, M., 2007. Stability of feature selection algorithms: A study on high-dimensional spaces. Knowl. Inf. Syst. 12 (1), 95–116.
Karatzoglou, A., Smola, A., Hornik, K., Zeileis, A., 2004. kernlab – an S4 package for kernel methods in R. J. Stat. Softw. 11 (9), 1–20.
Ke, W., Wu, C., Wu, Y., Xiong, N.N., 2018. A new filter feature selection based on criteria fusion for gene microarray data. IEEE Access 6, 61065–61076.
Kerschke, P., Trautmann, H., 2019. Automated algorithm selection on continuous black-box problems by combining exploratory landscape analysis and machine learning. Evol. Comput. 27 (1), 99–127.
Kittler, J., 1978. Feature set search algorithms. In: Pattern Recognition and Signal Processing. Sijthoff and Noordhoff, Alphen aan den Rijn, Netherlands, pp. 41–60.
Kohavi, R., John, G.H., 1997. Wrappers for feature subset selection. Artificial Intelligence 97 (1–2), 273–324.
Kruskal, W.H., Wallis, W.A., 1952. Use of ranks in one-criterion variance analysis. J. Amer. Statist. Assoc. 47 (260), 583–621.
Kursa, M.B., 2018. praznik: Collection of information-based feature selection filters. https://CRAN.R-project.org/package=praznik.

Lang, M., Bischl, B., Surmann, D., 2017. batchtools: Tools for R to work on batch systems. J. Open Source Softw. 2 (10).
Larose, D.T., Larose, C.D., 2014. Discovering Knowledge in Data, second ed. John Wiley & Sons, Inc., Hoboken, NJ, USA.
Lazar, C., Taminau, J., Meganck, S., Steenhoff, D., Coletta, A., Molter, C., de Schaetzen, V., Duque, R., Bersini, H., Nowe, A., 2012. A survey on filter techniques for feature selection in gene eexpression microarray analysis. IEEE/ACM Trans. Comput. Biol. Bioinform. 9 (4), 1106–1119.
Li, J., Cheng, K., Wang, S., Morstatter, F., Trevino, R.P., Tang, J., Liu, H., 2018. Feature selection: A data perspective. ACM Comput. Surv. 50 (6).
Liu, Y., 2004. A comparative study on feature selection methods for drug discovery. J. Chem. Inf. Comput. Sci. 44 (5), 1823–1828.
Liu, H., Li, J., Wong, L., 2002. A comparative study on feature selection and classification methods using gene expression profiles and proteomic patterns. Genome Inform. 13, 51–60.
Liu, H., Yu, L., 2005. Toward integrating feature selection algorithms for classification and clustering. IEEE Trans. Knowl. Data Eng. 17 (4), 491–502.
Meyer, P.E., Schretter, C., Bontempi, G., 2008. Information-theoretic feature selection in microarray data using variable complementarity. IEEE J. Sel. Top. Sign. Proces. 2 (3), 261–274.
Mohtashami, M., Eftekhari, M., 2019. A hybrid filter-based feature selection method via hesitant fuzzy and rough sets concepts. Iran. J. Fuzzy Syst. 16 (2), 165–182.
Nogueira, S., Brown, G., 2016. Measuring the stability of feature selection. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases. pp. 442–457.
Peng, H., Long, F., Ding, C., 2005. Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. IEEE Trans. Pattern Anal. Mach. Intell. 27 (8), 1226–1238.
R Core Team, 2017. R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria.
Ramey, J.A., 2016. datamicroarray: Collection of data sets for classification. https://github.com/ramhiser/datamicroarray.
Rasch, D., Kubinger, K.D., Yanagida, T., 2011. Statistics in Psychology using R and SPSS. John Wiley & Sons, Inc., Hoboken, NJ, USA.
Ritchie, M.E., Phipson, B., Wu, D., Hu, Y., Law, C.W., Shi, W., Smyth, G.K., 2015. limma powers differential expression analyses for RNA-sequencing and microarray studies. Nucleic Acids Res. 43 (7), e47.
Romanski, P., Kotthoff, L., 2016. Fselector: Selecting attributes. https://CRAN.R-project.org/package=FSelector.
Saeys, Y., Inza, I., Larrañaga, P., 2007. A review of feature selection techniques in bioinformatics. Bioinformatics 23 (19), 2507–2517.
Sammut, C., Webb, G.I., 2011. Encyclopedia of Machine Learning. Springer, New York, USA.
Sánchez-Maroño, N., Alonso-Betanzos, A., Tombilla-Sanromán, M., 2007. Filter methods for feature selection – A comparative study. In: International Conference on Intelligent Data Engineering and Automated Learning. pp. 178–187.
Schliep, K., Hechenbichler, K., 2016. kknn: Weighted k-nearest neighbors. https://CRAN.R-project.org/package=kknn.
Simon, N., Friedman, J., Hastie, T., Tibshirani, R., 2011. Regularization paths for cox's proportional hazards model via coordinate descent. J. Stat. Softw. 39 (5), 1–13.
Smyth, G.K., 2004. Linear models and empirical Bayes methods for assessing differential expression in microarray experiments. Stat. Appl. Genet. Mol. Biol. 3 (1).
Strobl, C., Boulesteix, A.-L., Kneib, T., Augustin, T., Zeileis, A., 2008. Conditional variable importance for random forests. BMC Bioinformatics 9 (307).
Tang, J., Alelyani, S., Liu, H., 2014. Feature selection for classification: A review. In: Data Classification: Algorithms and Applications. CRC Press, Boca Raton, FL, USA, pp. 37–64.
Therneau, T., Atkinson, B., Ripley, B., 2017. rpart: Recursive partitioning and regression trees. https://CRAN.R-project.org/package=rpart.
Tibshirani, R., 1996. Regression shrinkage and selection via the lasso. J. R. Stat. Soc. Ser. B Stat. Methodol. 58 (1), 267–288.
Tibshirani, R., Chu, G., Narasimhan, B., Li, J., 2011. samr: SAM: Significance analysis of microarrays. https://CRAN.R-project.org/package=samr.
Tusher, V.G., Tibshirani, R., Chu, G., 2001. Significance analysis of microarrays applied to the ionizing radiation response. Proc. Natl. Acad. Sci. USA 98 (9), 5116–5121.
Vanschoren, J., Van Rijn, J.N., Bischl, B., Torgo, L., 2013. OpenML: Networked science in machine learning. ACM SIGKDD Explor. Newsl. 15 (2), 49–60.
Venkatesh, B., Anuradha, J., 2019. A review of feature selection and its methods. Cybern. Inf. Technol. 19 (1), 3–26.
Wah, Y.B., Ibrahim, N., Hamid, H.A., Abdul-Rahman, S., Fong, S., 2018. Feature selection methods: case of filter and wrapper approaches for maximising classification accuracy. Pertanika J. Sci. Technol. 26 (1), 329–340.
Wright, M.N., Ziegler, A., 2017. ranger: A fast implementation of random forests for high dimensional data in C++ and R. J. Stat. Softw. 77 (1), 1–17.
Xue, B., Zhang, M., Browne, W.N., 2015. A comprehensive comparison on evolutionary feature selection approaches to classification. Int. J. Comput. Intell. Appl. 14 (2).
Xue, B., Zhang, M., Browne, W.N., Yao, X., 2016. A survey on evolutionary computation approaches to feature selection. IEEE Trans. Evol. Comput. 20 (4), 606–626.
Yang, J., Honavar, V., 1998. Feature subset selection using a genetic algorithm. In: Feature Extraction, Construction and Selection: A Data Mining Perspective. Springer, New York, USA, pp. 117–136.
Yu, L., Liu, H., 2004. Efficient feature selection via analysis of relevance and redundancy. J. Mach. Learn. Res. 5, 1205–1224.
Zawadzki, Z., Kosinski, M., 2017. FSelectorRcpp: 'Rcpp' implementation of 'FSelector' entropy-based feature selection algorithms with a sparse matrix support. https://CRAN.R-project.org/package=FSelectorRcpp.
Zhu, Z., Ong, Y.-S., Dash, M., 2007. Wrapper-filter feature selection algorithm using a memetic framework. IEEE Trans. Syst. Man Cybern. B 37 (1), 70–76.