Machine Learning (2019) 108:2009–2034
https://doi.org/10.1007/s10994-019-05809-y

A distributed feature selection scheme with partial information sharing

Aida Brankovic[1] · **Luigi Piroddi**[1]

Received: 7 August 2017 / Accepted: 6 May 2019 / Published online: 21 May 2019

Abstract
This paper introduces a novel feature selection and classification method, based on vertical data partitioning and a distributed searching architecture. The features are divided into subsets, each of which is associated to a dedicated processor that performs a local search. When all local selection processes are completed, each processor shares the features of its locally selected model with all other processors, and the local searches are repeated until convergence. Thanks to the vertical partitioning and the distributed selection scheme, the presented method is capable of addressing relatively large scale examples. The procedure is efficient since the local processors perform the selection tasks in parallel and on much smaller search spaces. Another important feature of the proposed method is its tendency to produce simple model structures, which is generally advantageous for the interpretability and robustness of the classifier. The proposed approach is evaluated and compared to other well-known feature selection and classification approaches proposed in the literature on several benchmark datasets. The obtained results demonstrate the effectiveness of the proposed approach, both in terms of classification accuracy and computational time.

Keywords Feature selection · Classification · Model selection · Distributed optimization · Parallel processing

1 Introduction

In the supervised learning framework, the classification task aims at predicting the class label of unseen input instances, based on the knowledge obtained from the perusal of a set of training instances (training set), whose labels are known. This set of available input-output instances is used to train a model in a process called learning, that typically includes two stages, namely

Editor: Karsten Borgwardt.

✉ Aida Brankovic
aida.brankovic@polimi.it

Luigi Piroddi
luigi.piroddi@polimi.it

[1] Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Via Ponzio, 34/5, 20133 Milan, MI, Italy

a feature selection (FS) and a classifier design phase. FS is a combinatorial optimization problem, that aims at reducing the input space, by detecting the smallest possible set of features that allows proper classification. It is a crucial and computationally demanding task for high dimensional problems. Reducing the size of the input space yields a series of benefits, such as a reduced cost associated to data acquisition, a reduced computational demand and thus a shorter training time, a reduced model size and thus a simpler classifier structure. It may ultimately result in an improved classification accuracy, since the classification design process may be adversely affected by the presence of redundant features (Inza et al. 2000).

FS methods can be broadly divided into filter and wrapper methods. The former (see, e.g., Chandrashekar and Sahin 2014) select features based on information contained in the features themselves, independently of the classifier learning phase. In the majority of filter methods, features are considered individually, not taking into account the mutual interaction among different features. This is an important limitation, as features that are individually significant are not necessarily so in combination with others, and viceversa (Xue et al. 2013). Wrapper methods perform the FS task simultaneously with the classifier construction. As a consequence, they are typically more accurate than filter methods, at the price of an increased computational cost. They may also suffer from overfitting problems (Liu and Motoda 2012). Most of the literature on wrapper methods concerns either sequential FS (SFS) algorithms based on incremental model building procedures (Ferri et al. 1994; Sorjamaa et al. 2007), or evolutionary methods, such as genetic algorithms (Smith and Bull 2005; Yang and Honavar 1998), ant colony optimization (Kabir et al. 2012), particle swarm optimization (PSO) (Xue et al. 2013, 2014), harmony search (Diao and Shen 2012).

The main challenge with FS algorithms is to maintain effectiveness when dealing with datasets of increasing size, since the complexity of the combinatorial problem grows exponentially. This adversely affects both the time required to solve the problem and the actual quality of the solution (due, e.g., to local minima issues). To improve the effectiveness and speed up the FS process, several approaches based on a parallelization of the processing have been proposed in the literature (see, e.g., López et al. 2006; de Souza et al. 2006; Guillén et al. 2009). For example, in de Souza et al. (2006) multiple processors perform independent FS tasks on the same data and the final local best results are sent to a master process, which selects the best one in terms of performance. Guillén et al. (2009) propose a parallelized version of the Forward–Backward (FB) SFS. Several feature sets are constructed which differ from the original one by one feature (either added or removed), and an FB run is carried out on each of them. Then, the best of the local solutions is selected.

While parallelization can mitigate the local minima problem by searching for multiple different solutions at the same time, it does not reduce the complexity of the problem if the computational effort is not distributed among the available processors. In this direction, Chu et al. (2007) showed that dividing the dataset either vertically (along the features) or horizontally (along the data samples) can significantly improve the efficiency of the process, resulting in a linear speed-up with the increase of computing resources. In Bolón-Canedo et al. (2014, 2015a) the authors proposed a distributed FS approach with vertical partitioning. Each processor operates on a different subset of features (which configures a much smaller problem than the original one) and the local solutions are eventually merged. A variant of the distributed scheme of Bolón-Canedo et al. (2014) is proposed in Bolón-Canedo et al. (2015b) and Morán-Fernández et al. (2015), where the final feature subset is selected based on a complexity measure (the inverse of the Fisher rate), rather than the classification error. The algorithm is repeated multiple times and employs a voting mechanism governed by the complexity measure, whereby at the end all features with more votes than the computed threshold are eliminated. A distributed algorithm based on a similar reasoning is proposed in

Prasad et al. (2016). A vertical partitioning is applied initially and a feature ranking procedure based on the information gain criterion is applied to each individual subset. The best local results are merged into a final feature set, on which the feature ranking procedure is applied again to obtain the final selection.

All the mentioned distributed approaches are based on a two-stage sequential scheme where a first selection is carried out *separately* on different feature subsets, and a merging phase is finally applied, whereby the local selected features are aggregated and further processed. In this way, however, the local search processes are completely independent and do not share any information. This may lead to poor results if the relevant features are scattered in the different subsets and their actual importance emerges only if they appear combined together (Xue et al. 2014).

Alternative distributed architectures for supervised FS with horizontal and vertical partitioning have been proposed in Zhao et al. (2013) and Banerjee and Chakravarty (2011), where local best solutions are shared among all processors. More in detail, horizontal partitioning is used in Zhao et al. (2013) so that different processors operate on different data samples. At each iteration, the processors independently select one feature to add to the current model (using an SFS method) and then a master processor picks only one of these locally selected features and adds it to the current model. This approach introduces an important element, *i.e.* that the local searches are *repeated* after a common knowledge is established (the current model), that is based on aggregating the local results. However, this method cannot be extended to exploit vertical partitioning as well, and therefore it might be ineffective for vertically large problems. Moreover, it also carries over the defects of the SFS procedure (see, e.g., the discussion in Piroddi and Spinelli (2003)). In Banerjee and Chakravarty (2011) a supervised FS approach based on a filtering method is developed that works for both horizontally and vertically partitioned data, in which local results are shared among all processors before a final (centralized) FS step. A shortcoming of this method is related to the fact that the local selections are not iterated based on common knowledge. Another drawback is typical of filter methods based on univariate feature ranking, which neglect the interaction between features. In particular, features that are considered individually irrelevant are eliminated, although it may well occur that they become relevant in combination with other features (Xue et al. 2014).

This paper presents a novel distributed algorithm for FS, that exploits the benefits associated to parallelization, vertical data partitioning, and information exchange (among the different processors). The features are initially partitioned into subsets, each of which is associated to a dedicated processor that operates a local FS. The algorithm is executed iteratively, and at every round the local solutions (containing the most promising local features) are shared by all the processors to be considered in the subsequent local searches. The iteration of the process allows new useful features to emerge locally, if they combine well with the features shared from the other processors. Notice that each processor operates on a feature subset that includes the best local model found so far. Therefore, it should provide either the same or an improved solution. The iterative procedure terminates when all processors converge on the same selected features or when there is no improvement in terms of performance with respect to the previous round of the algorithm.

Besides the obvious savings in computational cost, the proposed algorithm displays several promising features regarding the performance:

(i) Unlike other distributed algorithms (see e.g. Bolón-Canedo et al. 2014; Prasad et al. 2016; Zhao et al. 2013), where the aggregation of the local solutions is not used to refine the local FS processes, the described iterative process allows combinations of features

to emerge even if their components are originally scattered among the local FS search spaces, and so it results in a "deeper" space search overall.

(ii) Operating separately on smaller feature sets generally leads to a more accurate functioning of the FS algorithm, since the solution space is smaller. It also helps preventing overfitting (especially with heavily unbalanced datasets, such as microarrays) and redundancy (especially when employed with ranking-based filter FS methods).

(iii) The proposed method combines a huge reduction in the problem complexity (achieved through feature distribution), without paying an excessive cost for the corresponding reduction of the search space. Indeed, the information feedback ensures that promising feature combinations are considered in all local processes. Furthermore, a random feature reshuffling at the beginning of each round guarantees that all possible feature combinations can be explored during the full course of the algorithm, even if the search space is reduced for each individual round. This mechanism also allows the algorithm to occasionally escape from local minima.

The presented distributed optimization scheme can in principle be combined with any FS method of choice. In this paper, we employ for this purpose the SFS and the recently introduced RFSC (Randomized Feature Selection and Classifier) algorithm (Brankovic et al. 2018) as representatives of wrapper methods, and the ReliefF algorithm as a representative of filter methods.

The rest of the paper is organized as follows. Section 2 provides the problem formulation and reviews the basics of the SFS, RFSC, and ReliefF methods. Then, Sect. 3 introduces the proposed distributed FS scheme. Several numerical studies on benchmark datasets are discussed in Sect. 4. Finally, some concluding remarks are given in Sect. 5.

2 Preliminaries: Problem definition and FS methods

2.1 The classification problem

We here consider the classification problem in the context of supervised learning. In this framework, a set $\mathcal{D} = \{d_1, \ldots, d_N\}$ of N observations is assumed available, each consisting of an input-output pair $d_k = (\boldsymbol{u}_k, c_k), k = 1, \ldots, N$, where the components $u_p, p = 1, \ldots, N_f$ of vector $\boldsymbol{u}$ are the features and $c \in \{1, \ldots, N_c\}$ is the corresponding class. These observations are used to construct a model that relates the features to the classes and that can be used to estimate the class corresponding to the feature values $\boldsymbol{u}$ associated to a previously unseen sample. This model, referred to as the classifier, has the general form:

$$\hat{c} = f(\boldsymbol{u}), \tag{1}$$

where $\hat{c}$ denotes the class estimate and f is a suitable function of the feature values. The latter is the result of a learning process that requires the set of features $\mathcal{U} = \{u_1, \ldots, u_{N_f}\}$ and the set of data $\mathcal{D}$.

To facilitate the classification process, since first order interactions among the features may be insufficient to derive satisfactory results, the original features are here preprocessed to obtain non-linear expansions (Guyon and Elisseeff 2006) such that a generic extended feature is a monomial of the original features up to a given degree, *i.e.* $u_1^{l_1} \cdot u_2^{l_2} \cdot \ldots \cdot u_{N_f}^{l_{N_f}}$ where $l_1, l_2, \ldots, l_{N_f}$ are non-negative integers such that $l_1 + l_2 + \ldots + l_{N_f} \leq L$. The cardinality of the extended feature set can be obtained as $N_e = \binom{N_f+L}{N_f}$.

Accordingly, in the classifier design we will look for a map from the extended features to the class:

$$\hat{c} = f'(\boldsymbol{\varphi}) = f'(\boldsymbol{\varphi}(\boldsymbol{u})), \tag{2}$$

In the following, we will denote as $\mathcal{R} = \{\varphi_1, \ldots, \varphi_{N_e}\}$ the set of extended features and we will refer to the latter simply as *features*, for brevity sake. Notice that $\mathcal{U} \subseteq \mathcal{R}$ provided that $L \geq 1$.

Classification performance is evaluated as the ratio of the correct classifications over the total number of tested samples

$$J = \frac{1}{N} \sum_{k=1}^{N} \mathbf{1}_{\{\hat{c}_k = c_k\}}, \tag{3}$$

where $\mathbf{1}$ is the indicator function, or equivalently with the percentage error index, $PE = (1 - J)100$. Depending on how the training process is carried out, the classifier performance can be calculated on the total or on a part of the training data (e.g., the validation data). For ease of notation we will denote simply as J the performance index used in the training process (in our case the accuracy ratio on the validation data). The notation J_{te} will be used to indicate the calculation of the same index on the test data (not used during training) for classifier assessment and comparison.

2.2 Feature selection algorithms

FS is typically at the core of the classifier design process, and is employed to reduce the set of features required to perform successfully the classification task. FS addresses the combinatorial problem of finding the smallest subset of features $\mathcal{M} \subseteq \mathcal{R}$ required by the classifier to maximize the classification accuracy, over the set of all possible $2^{N_e} - 1$ subsets of features.

In the following we briefly review the main characteristics of some FS algorithms that will be employed in the subsequent developments, namely the SFS, the ReliefF, and the recently introduced RFSC.

SFS algorithms

Sequential algorithms (Pudil et al. 1994) are deterministic and step-optimal iterative FS algorithms of the wrapper category. The *Forward SFS* algorithm starts from an empty subset and adds one feature at each iteration, selecting the one which combined with the previously selected features maximizes the classifier performance. Features are added over iterations as long as the classifier performance is improving or until the required number of features has been obtained. Conversely, the *Backward SFS* algorithm starts from the full feature set and discards one feature at every iteration, selecting the one whose removal deteriorates the classifier performance the least. Various combinations of the two mentioned strategies are also possible. SFS strategies are step-optimal, and there is no guarantee of reaching the optimal solution due to their incremental nature (Pudil et al. 1994). Indeed, each individual decision regarding the inclusion or elimination of a feature depends on the current model structure, so that early decisions (taken when the structure is still largely inaccurate) will influence the final outcome, and the initialization is critical. In this paper we employed the Forward SFS variant in combination with the k-NN classifier. k-NN is a non-parametric method, that attributes the class label to an input sample based on a majority voting among the k closest training instances.

ReliefF algorithm

The *Relief* algorithm (Kira and Rendell 1992) is a filter method, originally developed for binary classification problems. A weight is attributed to each feature, which evaluates its capability of discriminating between different classes. This weight is progressively updated based on the information that can be extracted from the available samples. More precisely, a sample is extracted from the dataset and for every feature one measures the smallest absolute difference between that sample and those of the same and of the opposite class. These two quantities are referred to as *nearest hit* (nH) and *nearest miss* (nM), respectively. Good features for classification should be such that samples of the same class have similar values, while samples from opposite classes have distant values. In this perspective, a large nH implies that the examined feature is not a good discriminator for the class. On the other hand, a large nM indicates that the examined feature well separates the sample from the members of the opposite class. Accordingly, nH is used to decrease the feature weight, while nM increases it. The procedure is repeated for a sufficient number of samples. Finally, the features with higher weights are returned.

Notice that Relief is a univariate filter method, in that it considers the features only as individuals and does not explore combinations of features. As a consequence it may miss features that are of low significance on their own, but become important in combination with other features. Furthermore, the method does not detect redundant features.

We here employ the *ReliefF* algorithm (Kononenko 1994), that extends the original Relief to deal with multi-class problems, and incomplete or noisy data. It is also reported to be more robust. The classifier design is again carried out with the k-NN method.

RFSC algorithm

The $RFSC$ (Brankovic et al. 2018) is a wrapper algorithm, based on a probabilistic reformulation of the FSC problem. At each iteration, a number of different models (*i.e.*, subsets of features) are extracted from a model distribution and used to assess the importance of each feature. The latter is evaluated as a function of the performance of the classifiers associated to the various models. A feature is considered important if the average performance of the classifiers associated to models that include the feature is greater than the average performance of the remaining classifiers. The rationale is that a "good" feature appears in accurate classifiers more often than not, or, stated otherwise, that it is generally more convenient to include it in the model than to keep it out.

Once the features have been evaluated, the distribution is updated, by reinforcing the probability to extract the more significant features, and the procedure is iterated.

The process terminates when the distribution converges to a limit distribution, corresponding to a specific model.

Differently from most approaches, the evaluation of each feature is not established based on its importance in a specific model, but rather based on the aggregate information that can be extracted on that term from the population of extracted models. In this sense, the feature importance is assessed based on its *global* contribution as opposed to a local evaluation. The randomized nature of the method can also alleviate the problem associated to local minima.

Unlike the k-NN, the RFSC classifier is a parameterized model, which ultimately amounts to a *linear* regression of the type:

$$y = \sum_{j=1}^{N_e} \vartheta_j \varphi_j . \tag{4}$$

The sign of y determines the class associated to the sample. For non-binary classification tasks a vector output is employed.

3 A distributed FS algorithm

3.1 Introduction

The complexity of the combinatorial problem inherent in the FS task increases rapidly with the number of features, and may easily become prohibitive for large-sized problems. This is particularly true for wrapper methods that require to perform the classifier design to evaluate and rank any examined set of features. More importantly, besides the obvious increase in computational complexity, the ability of FS algorithms to reach the optimal feature subset diminishes as the number of features grows, due to the corresponding exponential growth of the search space. This occurs for all FS methods, although with different incidence levels.

To explore the model space more efficiently, we here suggest a *distributed* combinatorial optimization approach, that exploits vertical partitioning and information exchange. The basic idea is to perform separate independent FS tasks on smaller subsets of features, and share the local results among the different optimization processors so that they can improve their selection by combining the locally available features with the most promising ones found elsewhere. As illustrated in Sect. 4, this strategy yields systematically more accurate results, as opposed to non-distributed FS, and significant savings in computational time as well. It is important to notice that any FS method can be employed in the illustrated optimization scheme to perform the local model selection tasks.

The next subsections review in detail the various steps of the proposed DFS (distributed FS) algorithm.

3.2 Generation and updating of the feature bins

The feature set is divided into a desired number of subsets $\mathcal{R}_b$, $b = 1, \ldots, N_b$, denoted *feature bins* in the rest of the paper. A partition of the set of features is first carried out, obtaining $\mathcal{R} = \cup_{b=1}^{N_b} \bar{\mathcal{R}}_b$, where $\bar{\mathcal{R}}_m \cap \bar{\mathcal{R}}_n = \emptyset$ for $m \neq n$. Then, at the beginning of each round of the distributed scheme, the feature bins are obtained as:

$$\mathcal{R}_b = \bar{\mathcal{R}}_b \cup \mathcal{M}, \tag{5}$$

for $b = 1, \ldots, N_b$, where $\mathcal{M}$ is a special subset of features selected at the previous round ($\mathcal{M} = \emptyset$ at the onset of the algorithm).

Given the purpose of the distributed scheme to break the complexity of the problem by reducing the size of the elementary FS task, it is a reasonable rule of thumb to partition the set of features $\mathcal{R}$ in subsets of approximately the same size. Different criteria can be adopted for assigning the features to the bins. In the following a random assignment policy is employed. The number of bins is also a crucial design parameter as discussed later on.

Each feature bin $\mathcal{R}_b$ is assigned to a different processor p_b (as schematically represented in Fig. 1), which is in charge of performing the FS task over $\mathcal{R}_b$. The processors perform an independent optimization over their own feature bins with an FS algorithm of choice (or even with different algorithms), selecting the best feature subset compatible with the available data. In principle, compatibly with the available computing machinery, the N_b FS tasks can be run in parallel. Let $\mathcal{M}_b$ denote the feature subset selected by processor p_b, $b = 1, \ldots, N_b$.

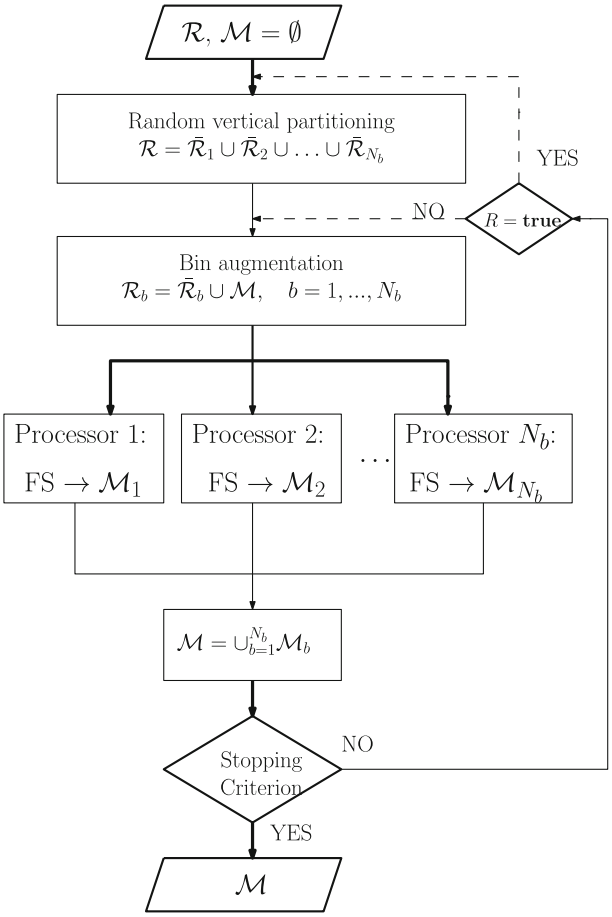

Fig. 1 Flowchart of the proposed distributed FS algorithm

The best of these local models is stored as current best solution. Then, the local models are merged into a unique feature subset $\mathcal{M} = \cup_{b=1}^{N_b} \mathcal{M}_b$, which is used to augment all the feature bins at the next round, and the distributed selection process is iterated. For large N_b values, it is sometimes convenient to limit this merging to the top ranked local models, as explained below.

In this way the most promising features are shared among the processors with the aim of improving the local bests. In principle, at the next round each processor should either retrieve the best of the local models obtained at the previous round (given that its feature bin now contains all the required features) or produce an improving solution. The latter outcome typically occurs when features not previously selected (because judged irrelevant or of smaller importance than others) actually combine well with the newly added features. When no improving solution is found, all processors settle on the same model structure (or possibly on different but equivalent ones).

The convergence of all the local processors on the same solution terminates the process. For practical purposes, other termination conditions are enforced as well. For example, if any of the local best solutions is also a global optimizer (*i.e*, it achieves perfect classification),

the process is terminated. Failure to improve the current best solution over a number (e.g., 3) of subsequent rounds or the exceeding of a prescribed number of rounds (N_r) are also employed as premature termination conditions.

The number of bins N_b is a critical parameter. If the number of bins is chosen too sparingly, the bin size will be large. This may lead to insufficient search space reduction, ultimately defying the very purpose of the distributed scheme, *i.e.* to break the problem complexity. Conversely, if one employs many small bins other problems may occur. Indeed, if the number of bins exceeds the size of the target model, most of the bins will initially *not* contain any feature of that model and presumably return inaccurate results, whereas only the processors associated to bins that contain features of the true model will typically produce meaningful results. As a consequence, the aggregation phase will produce a very large $\mathcal{M}$, with mostly irrelevant features, and propagating it to the various bins will overload them, slowing down convergence significantly and possibly even deteriorating the accuracy of the FS task. This effect can be addressed in various ways. For example, one can apply a further FS stage limited to the features in $\mathcal{M}$, to filter out the irrelevant terms. Alternatively, one can aggregate in $\mathcal{M}$ only the top local best models obtained at each round. The latter solution has been adopted here.

In the discussed version of the algorithm the partition of the feature set is executed at the beginning, thus fixing the most part of the feature bins during the algorithm execution. It is also possible to reshuffle the feature partition at *every* round. This randomization has the important effect of allowing new feature combinations to emerge from round to round, so that the reduction of the search space incurred by applying the distributed scheme does not imply that certain feature combinations are never explored. Besides allowing a richer exploration of the search space, this variation may occasionally allow the algorithm to escape from a local optimum, and for this reason it will be always used in the experimental section. We introduce a flag R (Reshuffling) to distinguish the two versions: if $R = \textbf{true}$, the partition is reshuffled at every round, otherwise it is performed only at the very beginning.

A pseudocode of the proposed DFS scheme is given in Algorithm 1. As the proposed scheme can be combined with the FS method of choice (represented by function FS$(\cdot)$ in the pseudocode, which returns a feature subset and the accuracy performance index of the associated classifier), we synthetically denote with Ψ the corresponding vector of input parameters (which is algorithm-dependent). The main loop goes from line 3 to 20. The vertical partitioning in N_b bins is carried out at line 2 or 4 (depending on the value of the R flag), by means of function DISTRIBUTE$(\cdot)$. Lines 5 to 13 describe the local selection processes, while the aggregation stage is at line 14. Finally, the termination conditions are given from line 15 to 19 (plus line 10).

3.3 Discussion on algorithm convergence

As already explained, at the onset of each DFS round each feature bin is augmented with the features of the best solution found so far, so that—at least in principle—each processor should be capable either to retrieve the same solution or find an even better one by combining the added features with those already present in the bin. Therefore, it is apparent that the convergence properties of the DFS rest on the ability of the FS algorithm operating on the local bins to reach the optimal solution on the subset of features they are in charge of. In turn, the task of retrieving the optimal solutions over the local feature bins is greatly facilitated by the fact that the local processors operate on a relatively small subset of features, so that

Algorithm 1 DFS

Input: $\mathcal{D}$, $\mathcal{R}$, N_b, N_r, Ψ, R.
Output: $\mathcal{M}^\star$, $J^\star$.

$\mathcal{M} = \emptyset$, $J^\star = 0$, $\mathcal{M}^\star = \emptyset$
if $R =$ **false then** $(\bar{\mathcal{R}}_1, \cdots, \bar{\mathcal{R}}_{N_b}) =$ DISTRIBUTE($\mathcal{R}$, N_b) **end if**
for $r = 1$ **to** N_r **do**
  if $R =$ **true then** $(\bar{\mathcal{R}}_1, \cdots, \bar{\mathcal{R}}_{N_b}) =$ DISTRIBUTE($\mathcal{R}$, N_b) **end if**
  for $b = 1$ **to** N_b **do**
    $\mathcal{R}_b = \bar{\mathcal{R}}_b \cup \mathcal{M}$
    $(\mathcal{M}_b, J_b) =$ FS(D, $\mathcal{R}_b$, Ψ)
    if $J_b > J^\star$ **then**
      $\mathcal{M}^\star \leftarrow \mathcal{M}_b$, $J^\star \leftarrow J_b$
      if $J^\star = 1$ **then return end if** ▷ The current best cannot be improved.
    end if
  end for
  $J^\star_{vec}(r) = J^\star$
  $\mathcal{M} = \cup_{b=1}^{N_b} \mathcal{M}_b$
  if $\mathcal{M} = \cap_{b=1}^{N_b} \mathcal{M}_b$ **then return end if** ▷ All local models are equal.
  if $r = N_r$ **then return** ▷ Maximum number of rounds reached.
  else if $r \geq 3$ **then**
    if $(J^\star_{vec}(r-1) = J^\star) \wedge (J^\star_{vec}(r-2) = J^\star)$ **then return end if**
  end if ▷ No appreciable improvement over the last 3 rounds.
end for

the corresponding local search space is easily manageable. Accordingly, we will make the following

Assumption 1 The FS method employed by Algorithm 1 guarantees the optimality of the solutions of the local problems.

A feature subset $\mathcal{M}_b^\star \subseteq \mathcal{R}_b$ is optimal for the bth local FS problem if there does not exist another feature subset $\mathcal{M} \subseteq \mathcal{R}_b$ ($\mathcal{M} \neq \mathcal{M}_b^\star$) with strictly greater value of the performance index. We will denote the corresponding optimal performance as $J_b^\star$. Notice that while there may be multiple optimal solutions due to the discrete nature of the performance index, $J_b^\star$ is uniquely defined. Assumption 1 implies that the FS algorithm will always yield a solution with performance equal to $J_b^\star$.

Lemma 1 *Let $J_b^\star(r-1)$ and $J_b^\star(r)$ be the performance values of the solutions obtained by the bth processor in two consecutive rounds of the DFS. Then, it holds that $J_b^\star(r) \geq J_b^\star(r-1)$.*

Proof Let $J^\star(r-1) = \max_b(J_b^\star(r-1))$ be the maximum performance among the solutions of the N_b local problems at round $r-1$, and let $\mathcal{M}^\star(r-1)$ be the corresponding feature subset. Obviously, $J^\star(r-1) \geq J_b^\star(r-1)$, $b = 1, \ldots, N_b$. At round r, each local problem feature space $\mathcal{R}_b$, $b = 1, \ldots, N_b$ will include $\mathcal{M}^\star(r-1)$, by construction. Therefore, in view of Assumption 1, the optimal local solution will have $J_b^\star(r) \geq J^\star(r-1)$, from which the thesis directly follows. □

Lemma 2 *Let $J^\star(r-1)$ and $J^\star(r)$ be the performance values of the best local solutions obtained among the N_b processors in two consecutive rounds of the DFS. Then, it holds that $J^\star(r) \geq J^\star(r-1)$.*

Proof Indeed, by Lemma 1 it holds that $J^\star(r) = \max_b(J_b^\star(r)) \geq \max_b(J_b^\star(r-1)) = J^\star(r-1)$. □

Theorem 1 *Under Assumption 1, let N be the number of samples on which the performance index is evaluated in the training process. Then, the DFS will converge (in performance) in not more than N rounds.*

Proof Convergence (in performance) is achieved if $J^\star(r) = J^\star(r-1)$. Now, since J can take only $N+1$ different values (J equals the ratio of correctly classified samples), and the sequence $J^\star(r)$ is monotonically non-decreasing (by Lemma 2) and limited from above (it cannot exceed 1), it will converge in a finite number of rounds. Indeed, if there is an improvement, the performance index will increase at least by $\delta J = 1/N$, *i.e.* by a quantum corresponding to a single sample. Since, at most there can be N such consecutive increments, it follows that the algorithm will converge at most in N rounds. □

Notice that, due to the discrete nature of the performance index J, there might be multiple models with equal accuracy. Thanks to the convergence condition $J^\star(r) = J^\star(r-1)$ (the last termination condition of Algorithm 1, which is extended to 3 consecutive rounds for greater robustness), the DFS will converge to one of the equivalent locally optimal solutions. Notice also that the alternative termination conditions (included in Algorithm 1 for practical reasons), are all subsumed by the previously mentioned convergence condition formulated on the performance index. For instance, if all local processors return the same model, at the next round they will operate on the same subset of features and, therefore, they will not be able to improve the local solutions.

A final remark is due regarding the version of the algorithm with $R =$ **true**, which involves a reshuffling of the feature partition at the beginning of each round. When the condition $J^\star(r) = J^\star(r-1)$ is reached with the basic version of the algorithm, it is no use protracting the algorithm for further rounds, since the feature bins will not be modified anymore. On the other hand, applying reshuffling allows the algorithm to explore new feature combinations, and possibly escape from the previous local optimum. As such it greatly enhances the probability of finding the actual global optimum.

4 Experimental study

4.1 Experiment design

This section reports the results of various tests carried out to assess the performance of the proposed distributed FS architecture. Twelve numerical datasets, collected from the UCI machine learning repository (Newman et al. 1998), are used in the experiments. In the following tests, the original datasets are preprocessed as follows.

All original features are first normalized in the [0, 1] range, according to the following expression:

$$u_p(k) = \frac{u_{p,orig}(k) - u_{p_{min}}}{u_{p_{max}} - u_{p_{min}}},$$

for $k = 1, \ldots, N$, where $u_{p,orig}(k)$ is the original numeric value of the kth observation of a feature p in a given dataset, and $u_{p_{max}}$ and $u_{p_{min}}$ are the maximum and minimum value of the pth attribute in the dataset, respectively.

Then, as already mentioned, the original features are polynomially expanded, so that the final search space ranges from 80 to 14000 features. A different maximum nonlinearity degree is adopted depending on the size of the considered problem. More precisely, we used $L = 3$ for small datasets (Bupa and Iris) and $L = 2$ for medium- and big-sized datasets, with the

Table 1 Main characteristics of the considered datasets

Dataset	N_c	N	Sample distribution	N_f	N_e
Bupa	2	345	145/200	6	84
Colon	2	62	22/40	2000	2000
HillValley	2	606	301/305	100	5151
Ionosphere	2	351	225/126	33	595
Iris	3	150	50/50/50	4	70
Madelon	2	4400	2200/2200	500	500
Musk1	2	476	207/269	166	14028
Ovarian	2	253	162/91	15154	15154
Sonar	2	208	111/97	60	1891
Vehicle	4	846	199/217/218/212	18	190
WDBC	2	569	212/357	30	496
Wine	3	178	59/71/48	13	105

only exception of the Madelon dataset. Finally, for the largest datasets (Colon and Ovarian) we set $L = 1$. The FS task was carried out on the resulting extended feature set $\mathcal{R}$. The main characteristics of these datasets are presented in Table 1, where N_f, N_e and N_c denote the size of the original feature set, the number of extended features and the number of classes, respectively.

To evaluate the performance of the proposed algorithm and provide a fair comparison with the literature, two validation methods are used, namely 10-fold cross validation (10-FCV) and random 70–30 horizontal data partitioning (70% of the samples used for training and 30% for testing). For 10-FCV, the data are randomly split into 10 disjoint and non-overlapping sets, called folds. The selection procedure is repeated 10 times, each time using a different fold for testing and the remaining ones for training. The algorithm performance for UCI datasets is computed averaging the performance on the testing set over the 10 runs.

For the 70–30 validation, 10-FCV was performed as an inner loop on the training data. To account for the non-determinism of the algorithm the procedure is further repeated 10 times and the performance averaged. To allow a fair comparison with Bolón-Canedo et al. (2014), the algorithm is performed 5 times on microarray data.

Besides evaluating the performance in terms of classification accuracy, we also considered the Cohen's Kappa rate (Ben-David 2008), which is particularly indicated when dealing with imbalanced data and random hits (Cano et al. 2013). Such index is based on the concept of *confusion matrix*, which is a square matrix C of size $N_c \times N_c$, N_c being the number of classes, where C_{ij} is the number of samples of class i attributed to class j. The diagonal elements of C represent the number of correct classifications, while the non-diagonal terms account for misclassified samples. The Kappa rate is then defined as:

$$K = \frac{N \sum_{i=1}^{N_c} C_{ii} - \sum_{i=1}^{N_c} C_{i\cdot} C_{\cdot i}}{N^2 - \sum_{i=1}^{N_c} C_{i\cdot} C_{\cdot i}},$$

where C_{ii} is the total number of correct classifications for the class i, $C_{i\cdot} = \sum_{j \neq i} C_{ij}$ is the number of samples of class i that have been misclassified (false negatives), and $C_{\cdot j} = \sum_{i \neq j} C_{ij}$ number of samples that have been wrongly attributed to class i (false positives). As a result, K ranges from -1 (total disagreement) to 0 (random classification) to 1 (total agreement).

4.2 Algorithm settings

In the following we study the DFS scheme in combination with different FS algorithms, namely the SFS, the RFSC and the ReliefF. Accordingly, we will refer to the corresponding DFS schemes as dSFS, dRFSC and dReliefF, respectively. We next list the main settings for the SFS, the RFSC and the ReliefF algorithms, followed by some remarks on the setup of the distributed scheme.

At each iteration, the SFS operates by testing each available feature for inclusion in the selected subset and actually adding only the most improving one. The iterative procedure is repeated as long as improvements are obtained by adding further features. Since in our framework improvements are discrete, no parameters are required to operate the algorithm.

The RFSC requires a careful setting of various parameters. Using the notation reported in Brankovic et al. (2018), the initial model distribution was defined so that the probability of selecting any given feature is equal to $\mu_0 = 1/N_e$. The number of models generated at each iteration was set to $N_p = 10$. The significance confidence interval for rejecting redundant features was set to $\alpha = 0.998$. This parameter influences the model size, in that the closer it is to 1, the more terms are rejected by the statistical test. To constrain the computational time of the processors in case of slow convergence, the maximum number of iterations was set to $N_i = 100$. The probability threshold for extracting the selected model structure from the feature distribution was set to $\bar{\mu} = 0.7$. We redirect the interested reader to Brankovic et al. (2018) for a full explanation of the role and settings of the RFSC parameters.

The only parameter to design for the ReliefF algorithm is the number of features to be retained after ranking. This parameter is set to $\frac{N_e}{N_b}\frac{r_{perc}}{100}$, where r_{perc} denotes the percentage of retained features. Parameter r_{perc} was set to 35 for all datasets, *i.e.* the 35% top ranked features are extracted from each bin, at every iteration, to generate the local model $\mathcal{M}_b$, $b = 1, \ldots, N_b$.

Another important parameter for the DFS is the number of bins N_b (or equivalently the size of the bins). This parameter is influenced by many factors (such as the adopted FS algorithm, N_e, and N), so that there is no straightforward rule that can be invoked for its setting, and some degree of trial-and-error is necessary for its correct dimensioning. Some rules of thumb are reported below.

The SFS algorithm, due to its greedy and exhaustive searching nature, works better with small sets of features. Accordingly, we used 10 bins for datasets with $N_e \leq 1000$, in order to get less than 100 features per bin. Larger N_b values were used for the other datasets. Notice also, that the number of features in a bin should be less or equal to the number of samples to avoid numerical issues such as overfitting. This implies that $N_b \geq N_e/N$.

ReliefF is relatively insensitive to the bin size (the computational complexity grows linearly with the number of features). However, splitting the feature set will favor the emergence of other features, besides those that are individually ranked as the best. In this way, features that are not individually significant but that are crucial for good classification when suitably combined with others may be detected. With ReliefF we employed 5, 10, and 15 bins, for small, medium, and large datasets, respectively.

As for the RFSC, since a very small number of models is extracted at each iteration ($N_p = 10$) and the probability of selecting any given feature is also set to a small value ($\mu_0 = 1/N_e$), a relatively small bin size is indicated in order to ensure an adequate representativeness of all the features in the population of extracted models. If one wants to operate with larger bins, it is necessary to increase N_p (or μ_0) accordingly, for the same reason. On the other hand, if the bin size is too small, the RFSC may fail to converge in a reasonable time. With respect to the considered datasets the number of bins was set so as to generate bins of approximately

Table 2 Number of bins (N_b) employed for each dataset, as a function of the FS algorithm

Dataset	SFS	RFSC	ReliefF
Bupa	10	4	5
Colon	66	40	47
HillValley	50	51	15
Ionosphere	10	12	10
Iris	10	3	5
Madelon	10	10	10
Musk1	100	56	15
Ovarian	80	303	86
Sonar	20	18	15
Vehicle	10	4	5
WDBC	10	10	10
Wine	10	2	5

size 25 and 50 for small and medium cases, respectively. Larger bins are used for the other datasets.

Table 2 reports the settings for N_b adopted in this study.

In the cases where a large number of bins is adopted, bin overloading typically occurs at the first iterations of the distributed algorithm, since in the aggregation phase a lot of regressors are added to each bin. To avoid this, information sharing is limited to the five top ranked local models.

The proposed DFS algorithm was implemented in Matlab (version 2016a) and executed on an Intel(R) Core i7-3630QM machine, with 4.3GHz CPU, 32GB of RAM, and a 64-bit Operating System.

4.3 Algorithm sensitivity tests

4.3.1 Effects of the randomized nature of the DFS scheme

The DFS scheme involves a random shuffling and distribution of the features in the N_b bins at every iteration. To get a better insight on the effects related to the randomized nature of the algorithm, we analyzed the variability of the classification performance results by means of a Monte Carlo test on the same problem. We considered the WDBC dataset employing the same training-test partitioning of the data for this purpose, and tested both the centralized and distributed architectures for all 3 FS methods. In the latter case 10 feature bins are employed. The total feature set amounts to $N_e = 496$ terms (30 original features, polynomial expansion of degree $N_d = 2$). Overall, 100 Monte Carlo simulations have been performed for each method.

Regarding in particular the dRFSC, it is important to note that an additional source of randomization is present besides that related to the distribution of the features in the feature bins, due to the randomized nature of the RFSC algorithm. It is therefore important to assess the robustness of the selection results especially with reference to large search spaces. Furthermore, notice that in the distributed scheme the RFSC operates on much smaller feature subsets, and can therefore tolerate a smaller number of extracted models at each iteration. In

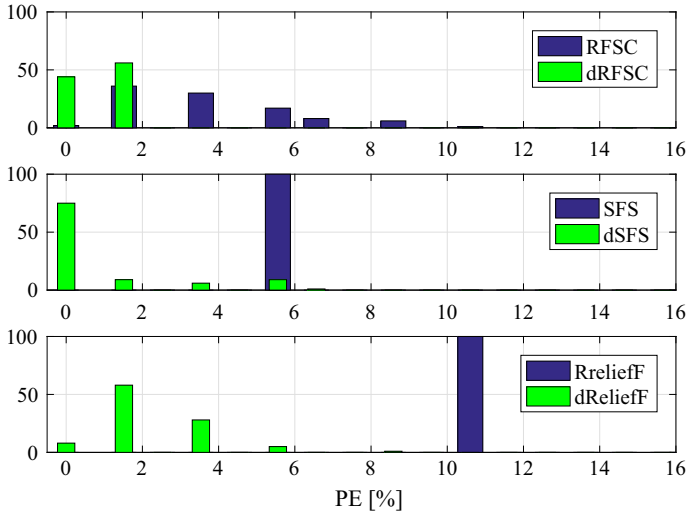

Fig. 2 Test error performance of the RFSC, dRFSC, SFS, dSFS, ReliefF and dReliefF over 100 Monte Carlo runs performed on the same training/test data split of the WDBC dataset

view of this, we used $N_p = 100$ for the centralized scheme and $N_p = 10$ for the distributed one. The maximum number of RFSC iterations was set to $N_i = 100$ in both cases.

Figure 2 shows the distribution of the classification error on the test set with the centralized (RFSC, SFS, ReliefF) and the distributed (dRFSC, dSFS, dReliefF) approaches. A strong dominance of the distributed approach over the centralized one is apparent in all three cases. Nearly half of the times (46%) the dRFSC algorithm picked solutions with 0 classification error on the test data, while this figure falls to just 2% for the centralized RFSC. In general, the RFSC displays a higher variance of the error, which is probably related to the complexity of the search space, that grows exponentially with size (the centralized RFSC operates on a search space of 2^{N_e} possible model structures, while the local RFSC instances used in the dRFSC approach work on sets which are several orders of magnitude smaller). Even if the centralized RFSC employs a much higher N_p value, this is not enough to explore the huge search space efficiently.

The dSFS algorithm outperformed the SFS 90% of the times, while the dReliefF algorithm had 100% dominance over ReliefF. Among the distributed algorithms, dRFSC displays the smallest variance of the error.

4.3.2 Effects of the training/test data partitioning

The classification accuracy depends on how the partitioning of the dataset in the training and testing sets is performed. To analyze this effect we carried out 100 Monte Carlo simulations on the WDBC dataset, regenerating every time the training-testing partition. In this analysis, we compared the results obtained with the distributed RFSC, SFS and ReliefF architectures. To provide a fair comparison, all 3 algorithms were trained and tested on the same data for each Monte Carlo simulation. Figure 3 reports the distribution of the classification error for the dRFSC, dSFS and dReliefF on the test data. Once again, the error distribution of the dRFSC is mainly concentrated near zero, and 99% of the times it is below 4%, which shows the robustness of the proposed approach with respect to the data division issue.

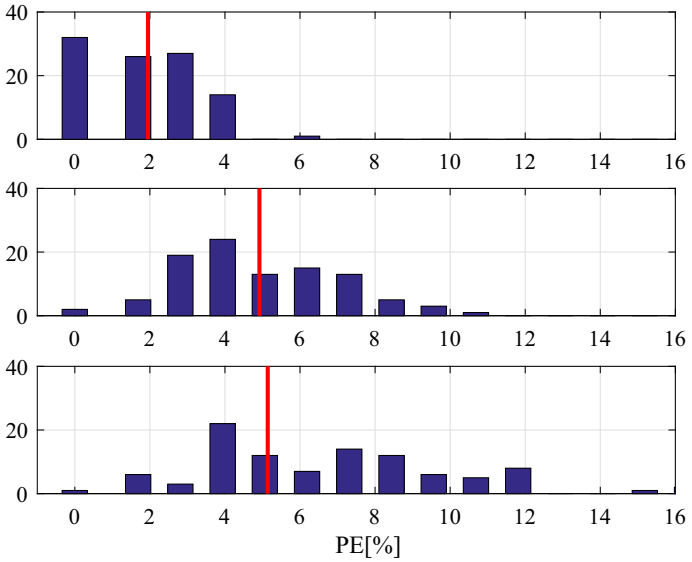

Fig. 3 Monte Carlo evaluation of the test error with dRFSC (top), dSFS (middle) and dReliefF (bottom) for 100 different training/test data partitions

Both the dSFS and dReliefF present a much larger average error and also a much wider spread of the results, with occasionally very bad classification performance.

4.3.3 Performance dependence on the number of bins

In the DFS scheme, the number of bins has a strong impact on the classification performance. To explore the variability of the classification error performance with respect to the number of generated bins, we tested the dSFS algorithm on the WDBC dataset using a fixed training-test data split, but different bin settings, $N_b = 10, 20, \ldots, 80$).

Twenty simulations were carried out for each case, assuming a fixed maximum number of rounds. The results are reported in Fig. 4, emphasizing for each case the range of performances and the median, which indicates the "typical" result for a given number of bins. Apparently, the typical performance is best for an intermediate value of N_b. In other words, one generally obtains worse performance when there are either too few bins (because their size is too large) or too many (because many unnecessary features are shared at the end of each DFS round).

4.3.4 Analysis of algorithm's runtime

In order to characterize the algorithm's runtime scaling as a function of the number of available processors (N_b) and the number of features (N_f) we performed an exhaustive test on the Musk1 and Ovarian datasets. The results are shown in Fig. 5. The runtime is measured as the total time required by the DFS algorithm to converge. In our experiments we used the 8 cores of the machine to run 8 FS local problems at a time (using Matlab parfor loops). Obviously, additional time savings can be obtained with more powerful computing machinery that can fully leverage the parallelism of the proposed scheme.

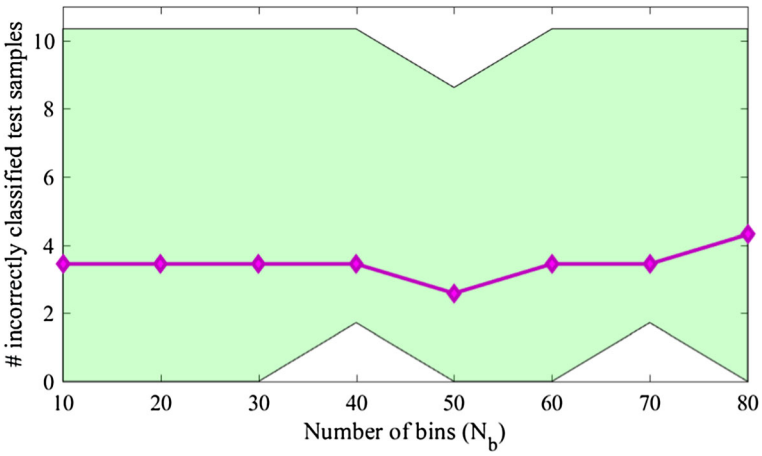

Fig. 4 Test error dependence on N_b: range and median

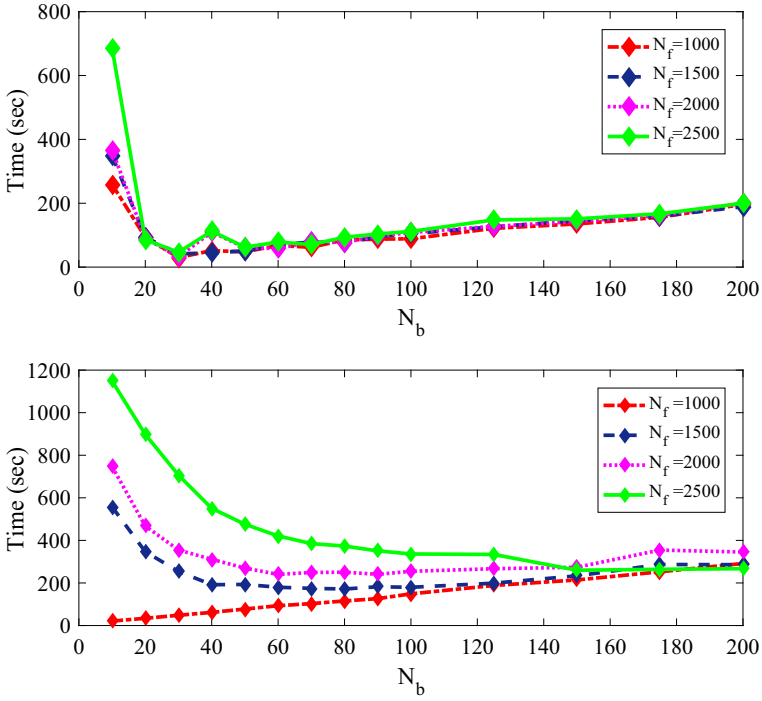

Fig. 5 DFS algorithm runtime as a function of the number of available processors (N_b) and the number of features (N_f) for the Musk1 (top) and Ovarian dataset (bottom)

As shown in Fig. 5, the runtime decreases exponentially with N_b down to an inflection point where its starts increasing again. This confirms the importance of a correct sizing of the N_b parameter (see Sect. 3.2). If too small a value is applied, the bin size will be large and hence the advantages of the distributed scheme will not be fully exploited. In particular, in the low value range adding further processors greatly affects the runtime. After a certain value it doesn't pay off to increase N_b, since what is gained by reducing the size of the local FS problems is lost due to the corresponding increase of the number of rounds of the DFS algorithms that are required to reach convergence.

A slightly different pattern is observed with the Ovarian dataset for $N_f = 1000$. Apparently, for this problem size the distributed approach is already effective for a very small number of processors ($N_b = 10$), and adding further ones does not reduce the runtime (the minimum of the curve is not shown).

4.4 Comparative analysis

4.4.1 Centralized versus distributed FS scheme

Table 3 reports extensive simulation results obtained on the twelve considered datasets using all the mentioned centralized and distributed FS algorithms. For each case we report the number of selected original (N_{fs}) and extended (N_{es}) features (clearly, $N_{fs} \leq N_f$ and $N_{es} \leq N_e$), the accuracy performance index $\mathcal{J}$, the Kappa rate K, and the elapsed time ET. Apparently, there is a systematic gain in using the distributed scheme as opposed to the centralized one, independently of the adopted FS algorithm. This is generally due to the fact that the complexity of the FS task increases exponentially with the number of considered features, and all wrapper methods tend to perform better when the search space is smaller. The distributed approach is ultimately beneficial also for a filter method like ReliefF, since in the centralized case it just picks the individually top ranked features, which, as mentioned before, do not necessarily coincide with the best features to combine in the classifier design.

The inspection of Table 3 reveals that dSFS and dRFSC generally provide smaller models than their centralized counterparts. Besides having less extended features, these models also contain less original features. As for the dReliefF method the final model size depends on the r_{perc} parameter which is set in the same way for the centralized and distributed schemes. dSFS typically returns the smaller models, whereas dRFSC and dReliefF provide more often the most accurate results.

Finally, there appears to be a significant gain (sometimes even an order of magnitude) in computational time inherent in the adoption of the DFS scheme, implying that the searching mechanism is much more efficient than in the centralized case.

4.4.2 Proposed distributed FS scheme versus off-the-shelf methods

Table 4 below reports the results of a comparative analysis of some off-the-shelf nonlinear classifiers (kernelized SVM (kSVM), random forests (RF), and gradient boosting trees (xgBoost, denoted GBT)), and sparse linear models (L1-penalized models (L1), Elastic Net (EN)) against the proposed DFS algorithm (in the three versions dRFSC, dSFS, dReliefF). The mentioned off-the-shelf algorithms are executed using a standard nested kFCV scheme for algorithm evaluation. More precisely, for each training-test data partition resulting from the outer 10-FCV loop, a further (inner) 5-FCV loop is carried out on the training part to find the optimal hyperparametrization (for that particular partition). Then, the model is re-

Table 3 Comparative analysis of the centralized and distributed schemes for the RFSC, SFS and ReliefF algorithms (J_{te} and K_{te})

Method	Dataset	N_{fs}	N_{es}	J_{te}	K_{te}	ET [s]	Dataset	N_{fs}	N_{es}	J_{te}	K_{te}	ET [s]
RFSC	Bupa	5.8	7.4	0.7884	0.4950	14.6	Musk1	46.2	23.2	0.8132	0.6201	51.6
dRFSC		5.8	4.3	0.7945	0.5701	1.9		48	22.5	0.8216	0.6372	40.7
SFS + 5NN		3.9	4.2	0.6228	0.2104	24.2		20	11	0.8531	0.7049	11712.2
dSFS + 5NN		3.4	2.7	0.6576	0.3424	12.6		15	7.5	0.8671	0.7360	1328.9
ReliefF + 5NN		3	6	0.5704	0.1154	0.4		84	327	0.8461	0.6937	41.9
dReliefF + 5NN		5	6	0.6724	0.4348	0.2		84	327	0.8559	0.7143	4.2
RFSC	Colon	–	–	–	–	–	Ovarian	–	–	–	–	–
dRFSC		2.3	2.3	0.8421	0.6209	0.3		5	5	0.9653	0.9251	2.23
SFS + 5NN		3	3	0.7579	0.2963	605		2	42	0.9868	0.9710	3344
dSFS + 5NN		1.25	1.25	0.8026	0.4673	15.8		2.25	2.25	0.9868	0.9710	130
ReliefF + 5NN		21	21	0.5529	0.5529	0.37		62	62	1	1	16.4
dReliefF + 5NN		21	21	0.8526	0.6211	0.01		62	62	1	1	14.4
RFSC	HillValley	8.3	3.7	0.9277	0.8552	18.6	WDBC	11.5	10.3	0.9827	0.9621	66.0
dRFSC		5.6	2.8	0.9859	0.9846	7.7		8.1	5.2	0.9860	0.9674	8.5
SFS + 5NN		5.6	4.5	0.5549	0.1095	658.7		6.1	3.5	0.9507	0.8942	129.9
dSFS + 5NN		4.3	4.5	0.5604	0.1214	250.4		4.4	2.5	0.9597	0.9129	32.2
ReliefF + 5NN		39	120	0.4890	-0.0207	30.4		10	17	0.9596	0.9116	4.1
dReliefF + 5NN		39	120	0.5274	0.0616	2.8		21	17	0.9825	0.9621	0.8
RFSC	Ionosphere	16.4	14.7	0.9330	0.8541	57.0	Sonar	25.8	18.7	0.8806	0.8101	72.0
dRFSC		13.5	11.8	0.9487	0.8861	2.4		9.6	5.1	0.9090	0.8164	1.21
SFS + 5NN		4.7	2.8	0.9028	0.7818	129.8		8.5	4.8	0.7167	0.5346	624.6
dSFS + 5NN		3.9	2.1	0.9198	0.8215	21.6		6.9	3.7	0.8073	0.6093	131.1
ReliefF + 5NN		10	20	0.9004	0.7699	2.3		19	44	0.7753	0.5460	3.1
dReliefF + 5NN		18	20	0.9403	0.8653	0.5		42	44	0.8075	0.7640	1.4
RFSC	Iris	3.2	6.1	0.9666	0.9500	10.0	Vehicle	10.9	16.6	0.7888	0.7186	162.4
dRFSC		2.8	4.5	0.9902	0.9900	1.3		8.1	5.2	0.8003	0.7339	25.3
SFS + 5NN		2.6	2.2	0.9600	0.9480	6.8		10.3	8.5	0.7010	0.6558	130.2
dSFS + 5NN		1.7	1.7	0.9800	0.9745	2.10		9.6	7.8	0.7292	0.6886	57.6
ReliefF + 5NN		2	5	0.9600	0.9457	0.08		9	13	0.6913	0.6436	3.3
dReliefF + 5NN		4	5	0.9800	0.9740	0.06		10	13	0.7671	0.7314	1.2
RFSC	Madelon	3.5	3.5	0.6160	0.2212	370.3	Wine	7.3	7.5	0.9944	0.9916	12.0
dRFSC		3.5	3.5	0.6347	0.2692	34.5		3.6	2.2	0.9944	0.9916	1.16
SFS + 5NN		7.5	7.5	0.8966	0.7933	326.1		6.7	4.6	0.9493	0.9306	40.1
dSFS + 5NN		7.4	7.4	0.9021	0.8043	60.3		5.6	3.6	0.9777	0.9698	11.2
ReliefF + 5NN		18	18	0.8910	0.7820	42.2		6	7	0.9323	0.9080	0.2
dReliefF + 5NN		18	18	0.9083	0.8166	4.7		8	7	0.9944	0.9916	0.1

Table 4 Comparison of the DFS with various off-the-shelf FS methods on nine UCI benchmarks

Dataset	Bupa		Iris		Vehicle	
Method	J_{te}	K_{te}	J_{te}	K_{te}	J_{te}	K_{te}
SVM	0.7137	0.3976	0.9533	0.9300	**0.8465**	**0.7952**
RF	0.7361	0.4473	0.9533	0.9300	0.7566	0.6755
GBT	0.7101	0.3967	0.9400	0.9100	0.7644	0.6859
L1	0.5975	0.1455	0.8733	0.8100	0.7057	0.6074
EN	0.5945	0.1359	0.8933	0.8400	0.7212	0.6277
dRFSC	**0.7945**	**0.5701**	**0.9902**	**0.9900**	0.8003	0.7339
dSFS	0.6576	0.3424	0.9800	0.9745	0.7292	0.6886
dReliefF	0.6724	0.4348	0.9800	0.9740	0.7671	0.7314
Dataset	HillValley		Musk1		WBCD	
Method	J_{te}	K_{te}	J_{te}	K_{te}	J_{te}	K_{te}
SVM	0.6766	0.3507	**0.9348**	**0.8675**	0.9738	0.9428
RF	0.5460	0.0911	0.8889	0.7731	0.9615	0.9169
GBT	0.5576	0.1148	0.8932	0.7817	0.9596	0.9129
L1	0.5033	0.0000	0.7462	0.4863	0.9581	0.9082
EN	0.5066	0.0087	0.7310	0.4406	0.9649	0.9245
dRFSC	**0.9859**	**0.9864**	0.8216	0.6372	**0.9860**	**0.9674**
dSFS	0.5604	0.1214	0.8671	0.7360	0.9597	0.9129
dReliefF	0.5274	0.0616	0.8559	0.7143	0.9825	0.9621
Dataset	Ionosphere		Sonar		Wine	
Method	J_{te}	K_{te}	J_{te}	K_{te}	J_{te}	K_{te}
SVM	0.9402	0.8680	0.8187	0.6360	0.9667	0.9667
RF	0.9289	0.8438	0.7982	0.5933	0.9719	0.9576
GBT	0.9289	0.8420	0.8712	0.7412	0.9660	0.9484
L1	0.7892	0.5267	0.7404	0.4759	0.9719	0.9571
EN	0.8291	0.5970	0.6841	0.3536	0.9719	0.9573
dRFSC	**0.9487**	**0.8861**	**0.9090**	**0.8164**	**0.9944**	**0.9916**
dSFS	0.9198	0.8215	0.8073	0.6093	0.9777	0.9698
dReliefF	0.9403	0.8653	0.8075	0.7640	**0.9944**	**0.9916**

estimated with that value of the hyperparameters on the training data and evaluated on the test data for that data partition, and the whole procedure is repeated for all the other folds of the outer 10-FCV loop. The hyperparameter optimization is carried out with a gridding approach. For this purpose, the `GridSearchCV` routine of the scikit-learn Python environment is employed on every fold of the outer loop, with the `refit` option.

All algorithms are tested on nine UCI benchmarks using 10-FCV,[1] in terms of performance (J_{te}) and Kappa rate (K_{te}) on the test data, using the same training-test data partitions used for the DFS.

[1] For ease of comparison, we considered for this analysis only the datasets for which 10-FCV results are available also for other methods in the literature.

With the exception of two datasets (out of nine) the DFS algorithm in the dRFSC version provides the best results both in terms of J_{te} and K_{te}. Notice that in those cases the DFS outperformed also ensemble approaches such as RF and xgBoost. This demonstrates the possible advantages of the distributed scheme in general, and also the effectiveness of the randomized scheme of the dRFSC in particular, which has an increased chance to escape from local minima.

4.4.3 Proposed distributed FS scheme versus literature methods

This section compares the results obtained with the proposed DFS scheme (in the 3 versions, namely dRFSC, dSFS and dReliefF) to those reported in the literature (see, in particular, Xue et al. 2013, 2014; Sreeja and Sankar 2015; Cano et al. 2013; Lin and Chen 2009; Bolón-Canedo et al. 2015a). Tables 5 and 6 report respectively the performance (both in terms of classification accuracy and Kappa rate) and the model size (in terms of the number of selected features N_{es}) of the obtained classifiers for 10 UCI datasets (the larger Colon and Ovarian datasets are considered separately). Table 9 in "Appendix A" provides the references for all the methods considered in this comparative analysis.

The best results in terms of classifier performance are typically associated to one of the 3 distributed schemes. A similar pattern is observed also regarding the classifier size. The results (see Table 6) also support the choice of extending the original set of features with a polynomial expansion, in that in general the obtained models outperform the literature while using a very limited number of the original features.

To appreciate the robustness of the proposed distributed scheme, we report in Table 7 the average ($\bar{J}_{te}$) and standard deviation (std(J_{te})) of the classification performance obtained by the dRFSC on the test data over 10 independent runs. In most cases the standard deviation is as low as 1%, indicating the good robustness properties of the algorithm. The highest standard deviation value is obtained for the Musk1 dataset. In terms of the $\bar{J}_{te} \pm \text{std}(J_{te})$ range, the presented algorithm still outperforms most of the methods from the literature presented in Table 5, and most markedly the Bupa, HillValley, Ionosphere and Sonar datasets.

Finally, Table 8 reports the results (in terms of model size and classification performance) obtained for the two large microarray datasets (Colon and Ovarian). Besides the good performance in terms of accuracy (see the dReliefF algorithm in particular), the obtained results also demonstrate the improvements that can be gained with the proposed methods in terms of model size with respect to the distributed FS approaches discussed in Bolón-Canedo et al. (2015a).

5 Conclusions

We proposed a novel distributed scheme for feature selection and classification problems, applicable in combination with any FS algorithm of choice, that exploits the benefits of parallel processing, vertical data partitioning, and information exchange. To the authors' knowledge, this is the first time that all these features are included into a unique algorithm.

The distributed scheme does not employ any *a priori* filtering, so that all features are considered equally important at the beginning. The features are distributed among different processors which perform separate independent FS tasks limited to the portion of features they have access to. The results of these selection processes are aggregated and the selected features fed back to the processors to be merged with the local features before a new round

Table 5 Comparative analysis: performance (J_{te} and K_{te})

FS Method + Classifier	Bupa		HillValley		Ionosphere		Iris		Madelon	
	J_{te}	K_{te}	J_{te}	K_{te}	J_{te}	K_{te}	J_{te}	K_{te}	J_{te}	K_{te}
ACO + PMC	0.6725	0.3259	–	–	0.9373	0.8604	0.9600	0.9400	–	–
Att.-Cls.WM + DGC+	0.6744	0.3076	–	–	0.9311	0.8487	0.9533	0.9300	–	–
Att. WV + DGC	0.6525	0.2220	–	–	0.6724	0.1142	0.9533	0.9300	–	–
− + KNN	0.6066	0.1944	–	–	0.8518	0.6494	0.9400	0.9100	–	–
− + KNN-A	0.6257	0.2021	–	–	0.9372	0.8595	0.9533	0.9300	–	–
− + DW-KNN	0.6376	0.2645	–	–	0.8747	0.7083	0.9400	0.9100	–	–
− + Cam-NN	0.5962	0.1024	–	–	0.7379	0.5145	0.9467	0.9200	–	–
− + CNN	0.6316	0.2571	–	–	0.8917	0.7526	0.9267	0.8900	–	–
SSMA + SFLDS	0.6426	0.2731	–	–	0.9088	0.7986	0.9533	0.9300	–	–
forward FS + LDA	0.6110	–	–	–	0.8530	–	0.9630	–	–	–
backward FS + LDA	0.6430	–	–	–	0.9090	–	0.9370	–	–	–
PSO + LDA	0.6520	–	–	–	0.9220	–	0.9700	–	–	–
PSO(4-2) + 5NN	–	–	0.5777	–	0.8727	–	–	–	0.7886	–
PSOMulti + 5NN	–	–	0.5757	–	0.9050	–	–	–	0.7652	–
(DCF) + RFSC	0.7884	0.4950	–	–	0.9330	0.8541	0.9666	0.9500	–	–
dRFSC	**0.7945**	**0.5701**	**0.9859**	**0.9864**	**0.9487**	**0.8861**	**0.9902**	**0.9900**	0.6347	0.2692
dSFS + 5NN	0.6576	0.3424	0.5604	0.1214	0.9198	0.8215	0.9800	0.9745	0.9021	0.8043
dReliefF + 5NN	0.6724	0.4348	0.5274	0.0616	0.9403	0.8653	0.9800	0.9740	**0.9083**	**0.8166**

FS Method + Classifier	Musk1		Sonar		Vehicle		WDBC		Wine	
	J_{te}	K_{te}	J_{te}	K_{te}	J_{te}	K_{te}	J_{te}	K_{te}	J_{te}	K_{te}
ACO + PMC	–	–	0.9087	**0.8164**	–	–	–	–	0.9755	0.9659
Att.-Cls. WM + DGC +	–	–	0.8487	0.6943	0.7116	0.6152	–	–	0.9731	0.9590
Att. WV + DGC	–	–	0.7694	0.5187	0.6572	0.5437	0.9619	–	0.9706	0.9552
− + KNN	–	–	0.8307	0.6554	0.7175	0.6233	–	–	0.9549	0.9318
− + KNN-A	–	–	0.8798	0.7549	0.6879	0.5844	–	–	0.9663	0.9491
− + DW-KNN	–	–	0.8648	0.7248	0.7258	0.6342	-	–	0.9438	0.9152
− + Cam-NN	–	–	0.7743	0.5364	0.6170	0.4905	–	–	0.9497	0.9228
− + CNN	–	–	0.8940	0.7861	0.7423	0.6563	–	–	0.9663	0.9491
SSMA + SFLDS	–	–	0.8079	0.6100	0.9145	0.5273	–	–	0.9438	0.9145
forward FS + LDA	–	–	0.7610	–	0.7500	–	–	–	0.9660	–
backward FS + LDA	-	–	0.8550	–	0.7900	–	–	–	0.9990	–
PSO + LDA	–	–	0.9050	–	0.7940	–	–	–	**1.0000**	–
PSO(4-2) + 5NN	0.8494	–	0.7816	–	–	–	0.9398	–	–	–
PSOMulti + 5NN	0.8454	–	–	–	–	–	–	–	–	–
(DCF) + RFSC	–	–	0.8806	0.8101	–	–	0.9827	0.9621	0.9944	**0.9916**
dRFSC	0.8216	0.6372	**0.9090**	**0.8164**	**0.8003**	**0.7339**	**0.9860**	**0.9674**	0.9944	**0.9916**
dSFS + 5NN	**0.8671**	**0.7360**	0.8073	0.6093	0.7292	0.6886	0.9597	0.9129	0.9777	0.9698
dReliefF + 5NN	0.8559	0.7143	0.8075	0.7640	0.7671	0.7314	0.9825	0.9621	0.9944	**0.9916**

Table 6 Comparative analysis: model size (average number of selected features N_{fs})

FS Method + Classifier	Bupa	Ionosphere	Iris	HillValley	Madelon	Musk1	Sonar	Vehicle	WDBC	Wine
FW FS + LDA	3.6	4.8	2.3	–	–	–	10.7	11.5	–	7.1
BW FS + LDA	4.7	30.4	3.9	–	–	–	56.4	16.5	–	12.8
PSO + LDA	4.6	21.7	3.6	–	–	–	38.1	15.5	–	12.3
PSO(4-2) + 5NN	–	3.26	–	12.22	203.32	76.54	11.24	10.16	**3.46**	6.84
dRFSC	5.8	13.5	2.8	5.6	**3.5**	48	9.6	**8.1**	8.1	**3.6**
dSFS + 5NN	**3.4**	**3.9**	**1.7**	**4.3**	7.4	**15**	**6.9**	9.6	4.4	5.6
dRelief + 5NN	5	18	4	39	18	84	42	10	21	8

Table 7 Average and standard deviation of the performance for the dRFSC algorithm

FS Method + Classifier	Bupa		HillValley		Ionosphere		Iris		Madelon	
	$\bar{J}_{te}$	std(J_{te})	$\bar{J}_{te}$	std(J_{te})	$\bar{J}_{te}$	std(J_{te})	$\bar{J}_{te}$	std(J_{te})	$\bar{J}_{te}$	std(J_{te})
dRFSC	0.7672	0.0105	0.9804	0.0167	0.9332	0.0062	0.9700	0.0080	0.5894	0.0245
FS Method + Classifier	Musk1		Sonar		Vehicle		WDBC		Wine	
	$\bar{J}_{te}$	std(J_{te})	$\bar{J}_{te}$	std(J_{te})	$\bar{J}_{te}$	std(J_{te})	$\bar{J}_{te}$	std(J_{te})	$\bar{J}_{te}$	std(J_{te})
dRFSC	0.8153	0.0377	0.9033	0.0139	0.7721	0.0213	0.9816	0.0023	0.9792	0.0090

Table 8 Comparative analysis: performance (N_{fs} and J_{te})

FS Method + Classifier	Colon		Ovarian	
	N_{fs}	J_{te}	N_{fs}	J_{te}
INT DF + 1NN	16	0.7700	27	0.9786
INT DRF + 1NN	16	0.7000	27	**1.000**
INT DRF0 + 1NN	16	0.8500	27	**1.000**
IG10 DF + 1NN	200	0.8000	1516	0.9905
IG10 DRF + 1NN	200	0.7000	1516	**1.000**
IG10 DRF0 + 1NN	200	0.8000	1516	**1.000**
Rel10 DF + 1NN	200	0.8300	1516	0.9810
Rel10 DRF + 1NN	200	0.7000	1516	**1.000**
Rel10 DRF0 + 1NN	200	0.8000	1516	**1.000**
dRFSC (avrg)	2.3	0.8421	5	0.9653
dRFSC (best)	3	0.8947	7	1.000
dSFS + 1NN (avrg)	**1.25**	0.8026	**2.25**	0.9868
dSFS + 1NN (best)	1	0.8421	3	1.000
dReliefF + 1NN (avrg)	21	**0.8526**	62	**1.000**
dReliefF + 1NN (best)	21	0.8947	62	1.000

of FS. For increased robustness, the local features are reshuffled among the processors at the beginning of every round, while all the selected features are transmitted to all the processors. This scheme allows to perform the FS task on datasets with a large number of features, by breaking the complexity of the problem and distributing over the processors. The information exchange, the feature re-shuffling and the process iteration ensure that the whole search space is adequately explored.

An extensive analysis on various public datasets of various dimensions revealed the advantages of the proposed distributed scheme over the corresponding centralized approaches, independently of the employed FS algorithms. Indeed, three very different FS algorithms have been tested, namely SFS, RFSC, and ReliefF, and all resulted in improved performance when used according to the distributed approach. Significant gains are observed also in terms of computational time, which is an indirect confirmation of the increased efficiency of the searching process. Another interesting feature of the DFS scheme is that, when used in combination with wrapper methods as SFS and RFSC, it yields more compact classifier models compared to the non-distributed methods.

The proposed DFS scheme has also been evaluated against some of the most effective recent algorithms from the literature with quite promising results. Indeed, for each of the studied datasets the best performing method is generally one of the DFS schemes.

A. References for FS methods mentioned in the paper

See Table 9.

Table 9 FS methods and corresponding references

FS method	References
ACO + PMC	Sreeja and Sankar (2015)
Att.-Cls. WM + DGC +	Cano et al. (2013)
Att. WV + DGC	Cano et al. (2013)
− + KNN	Cano et al. (2013)
− + KNN-A	Cano et al. (2013)
− + DW-KNN	Cano et al. (2013)
− + Cam-NN	Cano et al. (2013)
− + CNN	Cano et al. (2013)
SSMA + SFLDS	Cano et al. (2013)
forward FS + LDA	Lin and Chen (2009)
backward FS + LDA	Lin and Chen (2009)
PSO + LDA	Lin and Chen (2009)
PSO(4-2)	Xue et al. (2014)
PSOMulti	Xue et al. (2013)
INT DF	Bolón-Canedo et al. (2015a)
INT DRF	Bolón-Canedo et al. (2015a)
INT DRF0	Bolón-Canedo et al. (2015a)
IG10 DF	Bolón-Canedo et al. (2015a)
IG10 DRF	Bolón-Canedo et al. (2015a)
IG10 DRF0	Bolón-Canedo et al. (2015a)

Table 9 continued

FS method	References
Rel10 DF	Bolón-Canedo et al. (2015a)
Rel10 DRF	Bolón-Canedo et al. (2015a)
Rel10 DRF0	Bolón-Canedo et al. (2015a)

References

Banerjee, M., & Chakravarty, S. (2011). Privacy preserving feature selection for distributed data using virtual dimension. In *Proceedings of the 20th ACM international conference on Information and knowledge management* (pp. 2281–2284).

Ben-David, A. (2008). Comparison of classification accuracy using Cohen's weighted kappa. *Expert Systems with Applications*, *34*(2), 825–832.

Bolón-Canedo, V., Sánchez-Maroño, N., & Alonso-Betanzos, A. (2015b). A distributed feature selection approach based on a complexity measure. In *International work-conference on artificial neural networks* (pp. 15–128). Spain: Palma de Mallorca.

Bolón-Canedo, V., Sánchez-Marono, N., & Cerviño-Rabuñal, J. (2014). Toward parallel feature selection from vertically partitioned data. In *ESANN*

Bolón-Canedo, V., Sánchez-Maroño, N., & Alonso-Betanzos, A. (2015a). Distributed feature selection: An application to microarray data classification. *Applied Soft Computing*, *30*, 136–150.

Brankovic, A., Falsone, A., Prandini, M., & Piroddi, L. (2018). A feature selection and classification algorithm based on randomized extraction of model populations. *IEEE Transactions on Cybernetics*, *48*(4), 1151–1162.

Cano, A., Zafra, A., & Ventura, S. (2013). Weighted data gravitation classification for standard and imbalanced data. *IEEE Transactions on Cybernetics*, *43*(6), 1672–1687.

Chandrashekar, G., & Sahin, F. (2014). A survey on feature selection methods. *Computers & Electrical Engineering*, *40*(1), 16–28.

Chu, C., Kim, S. K., Lin, Y. A., Yu, Y., Bradski, G., Ng, A. Y., et al. (2007). Map-reduce for machine learning on multicore. *Advances in neural information processing systems*, *19*, 281.

de Souza, J. T., Matwin, S., & Japkowicz, N. (2006). Parallelizing feature selection. *Algorithmica*, *45*(3), 433–456.

Diao, R., & Shen, Q. (2012). Feature selection with harmony search. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, *42*(6), 1509–1523.

Ferri, F., Pudil, P., Hatef, M., & Kittler, J. (1994). Comparative study of techniques for large-scale feature selection. *Pattern Recognition in Practice*, *IV*, 403–413.

Guillén, A., Sorjamaa, A., Miche, Y., Lendasse, A., & Rojas, I. (2009). Efficient parallel feature selection for steganography problems. In *Bio-inspired systems: Computational and ambient intelligence* (pp. 1224–1231).

Guyon, I., & Elisseeff, A. (2006). An introduction to feature extraction. In I. Guyon, M. Nikravesh, S. Gunn, & L. A. Zadeh (Eds.), *Feature extraction. Studies in fuzziness and soft computing,* (Vol. 207). Berlin, Heidelberg: Springer. https://doi.org/10.1007/978-3-540-35488-81.

Inza, I., Larrañaga, P., Etxeberria, R., & Sierra, B. (2000). Feature subset selection by bayesian network-based optimization. *Artificial Intelligence*, *123*(1), 157–184.

Kabir, M. M., Shahjahan, M., & Murase, K. (2012). A new hybrid ant colony optimization algorithm for feature selection. *Expert Systems with Applications*, *39*(3), 3747–3763.

Kira, K., & Rendell, L. A. (1992). A practical approach to feature selection. In *Proceedings of the ninth international workshop on machine learning* (pp. 249–256).

Kononenko, I. (1994). Estimating attributes: Analysis and extensions of relief. In *European conference on machine learning* (pp. 171–182). Springer.

Lin, S. W., & Chen, S. C. (2009). PSOLDA: A particle swarm optimization approach for enhancing classification accuracy rate of linear discriminant analysis. *Applied Soft Computing*, *9*(3), 1008–1015.

Liu, H., & Motoda, H. (2012). Feature selection for knowledge discovery and data mining (Vol. 454). Springer Science & Business Media.

López, F. G., Torres, M. G., Batista, B. M., Pérez, J. A. M., & Moreno-Vega, J. M. (2006). Solving feature subset selection problem by a parallel scatter search. *European Journal of Operational Research*, *169*(2), 477–489.

Morán-Fernández, L., Bolón-Canedo, V., & Alonso-Betanzos, A. (2015). A time efficient approach for distributed feature selection partitioning by features. In *Conference of the Spanish association for artificial intelligence* (pp. 245–254). Springer.

Newman, D., Hettich, S., Blake, C., & Merz, C. (1998). UCI repository of machine learning databases. Retrieved June 28, 2016 from http://www.ics.uci.edu/~mlearn/MLRepository.html.

Piroddi, L., & Spinelli, W. (2003). An identification algorithm for polynomial NARX models based on simulation error minimization. *International Journal of Control*, *76*(17), 1767–1781.

Prasad, B. R., Bendale, U. K., & Agarwal, S. (2016). Distributed feature selection using vertical partitioning for high dimensional data. In *International conference on advances in computing, communications and informatics (ICACCI)* (pp. 807–813). IEEE

Pudil, P., Ferri, F., Novovicova, J., & Kittler, J. (1994). Floating search methods for feature selection with nonmonotonic criterion functions. In: *12th International conference on pattern recognition* (Vol. 2, pp. 279–283).

Smith, M. G., & Bull, L. (2005). Genetic programming with a genetic algorithm for feature construction and selection. *Genetic Programming and Evolvable Machines*, *6*(3), 265–281.

Sorjamaa, A., Hao, J., Reyhani, N., Ji, Y., & Lendasse, A. (2007). Methodology for long-term prediction of time series. *Neurocomputing*, *70*(16), 2861–2869.

Sreeja, N., & Sankar, A. (2015). Pattern matching based classification using ant colony optimization based feature selection. *Applied Soft Computing*, *31*, 91–102.

Xue, B., Zhang, M., & Browne, W. N. (2013). Particle swarm optimization for feature selection in classification: A multi-objective approach. *IEEE Transactions on Cybernetics*, *43*(6), 1656–1671.

Xue, B., Zhang, M., & Browne, W. N. (2014). Particle swarm optimisation for feature selection in classification: Novel initialisation and updating mechanisms. *Applied Soft Computing*, *18*, 261–276.

Yang, J., & Honavar, V. (1998). Feature subset selection using a genetic algorithm. In *Feature extraction, construction and selection* (pp. 117–136). Boston, MA: Springer.

Zhao, Z., Zhang, R., Cox, J., Duling, D., & Sarle, W. (2013). Massively parallel feature selection: An approach based on variance preservation. *Machine Learning*, *92*(1), 195–220.