POLITECNICO
MILANO 1863

POLITECNICO DI MILANO
DEPARTMENT OF ELECTRONICS, INFORMATION AND BIOENGINEERING
DOCTORAL PROGRAMME IN SYSTEMS AND CONTROL

DISTRIBUTED RANDOMIZED MODEL SELECTION FOR NONLINEAR IDENTIFICATION AND SUPERVISED MACHINE LEARNING

Doctoral Dissertation of:
Aida Brankovic

Supervisor:
Prof. Luigi Piroddi

Tutor:
Prof. Paolo Bolzern

The Chair of the Doctoral Program:
Prof. Andrea Bonarini

2017 – XXIX

$$\mathcal{E}^* = \{\ 40,10,8,200,30,1,50,40,8,200,30,1,5,30,30,1,40,60,2\ \}$$

Acknowledgement

This life chapter I close with the three main lessons I have learned in this four years long journey: i) "Success is journey, not a destination. The doing is more often more important than the outcome." ii) Word 'Impossible' means 'I m possible'. iii) "It's your road, and yours alone. Others may walk it with you, but no one can walk it for you." Thanks to thee who were walking with me...part of it, whole it, who gave a contribution to it.

Still, some of them deserve to be extra mentioned. Thanks to eversince my biggest support - my family (hvala vam Bucko, Noni, Edincicu, Vehbicu, neno i dedo Dervisu). Thanks to Secerica, Fatma and to my teachers. Thanks to AdnanT for his advice and suggestions while looking for a PhD position. Thanks to Marco, the biggest support besides my family within whole this journey. Thanks to my great friends Yang Yang, Hafsa and Cigdem, carrissimi e bravissimi collegi Steffanel (StefanoR), Rixy (RiccardoV), Bashy (BasakS), Danchy (DanieleI) e grandissimo Alchy (AlessandroF), and to Matteo Avelina and Marjan Hosseini, Master students, with whom I collaborated on the certain topics covered by this thesis.
Eventually... The main credits go to my, I dare to say, Scientific Father - Prof. Luigi Piroddi, for his limitless support, encouragement and unselfish knowledge sharing.

Abstract

TODAY thousands of variables or features are often used in classification problems. It is therefore crucial to select the most relevant ones in order to obtain robust, reliable, and easily interpretable models, not to mention storage space and classification time issues. Feature Selection (FS) aims precisely at selecting features that allow a good discrimination among samples of different classes. Suitable criteria are required to remove irrelevant and redundant features. Similar issues are encountered in nonlinear identification. For example when identifying polynomial NAR[MA]X models from data one is faced with the task of selecting the most appropriate model structure to represent the underlying system. This task, denoted Model Structure Selection (MSS), is akin to FS.

Both mentioned tasks configure combinatorial optimization problems aiming at selecting the combination of features or model terms that result in the most accurate classifier or model. The objective of this thesis is to investigate the possibility of employing some recent randomized techniques, originally developed in the nonlinear identification area, to FS problems, and to extend those techniques in both contexts, in order to deal with large-size problems. Indeed, the difficulty of these subset selection techniques increases rapidly with the size, given the exponential complexity of the underlying combinatorial problems.

The first outcome of the research is a novel classification approach (denoted RFSC, for Randomized Feature Selection and Classification), adapted from the nonlinear model identification framework, which jointly addresses the feature selection and classifier design tasks. The classifier is constructed

as a polynomial expansion of the original features and a selection process is applied to find the relevant model terms. The selection method progressively refines a probability distribution defined on the model structure space, by extracting sample models from the current distribution and using the aggregate information obtained from the evaluation of the population of models to reinforce the probability of extracting the most important terms. The performance of the RFSC was found to be quite satisfactory on small-/medium size problems.

To address large size problems, a distributed scheme is here proposed, which employs a vertical partitioning on the features and operates the selection in parallel on different feature subsets. The method alternates the parallel selection phase with a partial information exchange among the different processors, which reinforces the probability promising terms to be selected. The proposed scheme is applicable to both nonlinear identification and FS problems and in both frameworks it resulted in significant improvements in performance and efficiency. Moreover, the method has a tendency to produce small models, easily amenable to interpretation. While capable of addressing much larger problems than the non-distributed approach developed previously, the distributed scheme was found to be ineffective when dealing with extra-large search spaces (as are encountered, e.g., with micro-arrays), due to computational issues associated with parameter estimation and classifier design. An alternative version of the distributed scheme was then developed to target micro-arrays in particular, which employs a non-parametric multivariate filter algorithm and population extraction using the distance correlation index (dCor) as a criterion.
Finally, while analyzing the behavior of the RFSC, it was noticed that structurally different classifiers may result in equivalent performance due to the discrete nature of the $0-1$ loss function in classification problems. The randomized characteristic of the RFSC was then exploited to generate ensembles of classifiers. In most cases the results demonstrate an improved accuracy when ensembles of classifiers are employed with respect to the 'single classifier' case.

All proposed methods have been evaluated and compared to other well-known FS and MSS methods on standard benchmarks for classification/nonlinear identification problems. The results show the effectiveness of the proposed methods with respect to competitor methods both in terms of prediction accuracy and model complexity.

Contents

List of Abbreviations and Symbols

List of abbreviations

AMS	*Average Model Size*
CV	*Cross-validation*
$dCor$	*Distance Correaltion*
DCORFS	*dCor-based Feature Selection*
D^2CORFS	*Distributed dCor-based Feature Selection*
DFS	*Distributed Feature Selection*
dMSS	*Distributed Model Structure Selection*
dRaMSS	*Distributed Randomized Model Structure Selection*
E-RFSC	*Ensemble-based Randomized Feature Selection and Classification*
FS	*Feature Selection*
FSC	*Feature Selection and Classification*
LOOCV	*Leave-One-Out Cross-Validation*
LS	*Least Squares*
ML	*Machine Learning*
MS	*Model Selection*
MSE	*Mean Squared Error*
MSPE	*Mean Squared Prediction Error*
MSS	*Model Structure Selection*
dMSS	*Distributed Model Structure Selection*
MSSE	*Mean Squared Simulation Error*
NAR(MA)X	*Non-Linear Auto-Regressive Moving Average with Exogeneous Input*
NB	*Naive Bayes*
NL	*Non-Linear*
NRMSE	*Normalized Root Mean Squared Error*
k-NN	*k-Nearest Neighbours*
PE	*Classification error rate*
PEM	*Prediction Error Minimization*

RaMSS	*Randomized Model Structure Selection*
RFSC	*Randomized Feature Selection and Classification*
SFS	*Sequential Feature Selection*
SVM	*Space Vector Machine*
TIP	*Term Inclusion Probability*
TN	*True Negative*
TP	*True Positive*
TNR	*True Negative Rate*
TPR	*True Positive Rate*

Mathematical notation

$\emptyset$	*Empty set*
$\cup$	*Set union operator*
$\cap$	*Set intersection operator*
$\setminus$	*Set difference operator*
$\mathbb{N}$	*Set of positive integers*
$\mathbb{R}$	*Set of real numbers*
$\mathbb{R}^+$	*Set of non-negative real numbers*
$\|\cdot\|$	*Euclidean norm*
$\|s\|$	*Cardinality of a set s*
∇f	*Gradient of the funtion* $f(\cdot)$
$\mathcal{B}$	*Surjective function, i.e. distribution map*
$\mathcal{E}[x]$	*Expectation*

Values of features $x_1, ..., x_{N_f}$ *for a given data sample* k *are given in a vector form as* $\mathbf{x(k)} = [\mathbf{x_1(k)}, ..., \mathbf{x_{N_F}(k)}]$.
Observations are also presented as a set of observations $\mathcal{D} = \{\mathbf{x^{(k)}}, \mathbf{k} = \mathbf{1}, \cdots, \mathbf{N}\}$ *where* $\mathbf{x^{(k)}}$ *stands for a input-output values corresponding to a data sample* k.

CHAPTER 1

Introduction

TODAY thousands of variables or features are used in various Machine Learning (ML) and pattern recognition problems. Therefore, Feature selection (FS) is crucial task that aims at selecting a minimum subset of features that are capable of discriminating samples that belong to different classes. FS is important for reducing storage space and classification time, performing semantic analysis and aiding data interpretation and improving prediction accuracy. It is a difficult task because it amounts to solving a combinatorial problem whose complexity is exponential in the number of features (curse of dimensionality). FS is also important in the nonlinear identification context, where it is employed for model structure selection (MSS) purposes.
Suitable selection criteria are required to remove irrelevant and redundant features. Several techniques and approaches have been devised to address the FS problem. In general, search mechanisms can be divided into:

- incremental
- randomized
- regularization based

Incremental search mechanisms

Due to their simplicity, FS algorithms based on incremental model building techniques are extremely popular in the field of nonlinear system identification, as well as ML. These algorithms, often referred to as (forward/backward) subset selection methods, are greedy procedures that build the model structure by progressively selecting the terms to include or remove [57]. One of the most used approaches of this family in NL identification is probably the Forward Regression Orthogonal Estimator (FROE) [15, 28, 71], which exploits Orthogonal Least Squares (OLS) to decouple the estimation of the various regressors. The FROE is a forward selection method that adds to the current model one regressor per iteration based on an importance index called Error Reduction Ratio (ERR), which evaluates the marginal contribution of each term to the model accuracy. At each iteration, the ERR index is computed for each of the regressors not yet included in the model and the one with the highest value is selected. The procedure is repeated until a given number of regressors has been selected or if the required prediction accuracy is achieved. Several variants have been introduced in the literature using both forward and backward regression schemes (*e.g.* [41, 50, 53, 62, 80, 86, 98, 120]).
The same rationale is applied also in regression problems, in the so called *stepwise regression* subset selection methods. Other techniques based on stepwise regression such as backward stepwise regression, stagewise and leaps-and-bounds regression [87] are also considered classical methods for regression problems.

In classification problems this family of algorithms is known as Sequential Feature Selection (SFS). 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 [100]. 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

incomplete) will influence the final outcome, and the initialization is critical. Several variants of the sequential search method have been discussed in the literature, such as the PTA(l, r) (Plus l and Take Away r), the SFFS (Sequential Forward Floating Selection algorithm) and the SBFS (Sequential Backward Floating Selection algorithm) [43].

Randomized search mechanisms

To overcome the limitations of classical subset selection algorithms, which are mainly related to their inherently local search mechanism, new approaches based on randomized algorithms have been recently introduced. In the nonlinear (NL) identification field, some of them are based on evolutionary paradigms. For example, a MSS algorithm that exploits randomness in choosing potential regressors has been proposed in [103]. More recent randomized approaches, not based on evolutionary paradigms but on the Expectation Maximization (EM) approach have been presented in [9]. A computational Bayesian approach to determine via numerical sampling methods the posterior probability distributions of model structure and parameter estimates is employed in [10]. The algorithm is based on the Reversible Jump Markov Chain Monte Carlo (RJMCMC) procedure [47], which is a generalization of the Metropolis-Hastings (MH) algorithm [58] used in MCMC sampling. The RJMCMC procedure allows *jumps* between parameter spaces of different dimensions, a feature that makes the algorithm particularly suitable for the MSS task. In fact, thanks to its transdimensional nature, it can be used to take into account the update of the model structure as part of the posterior target distribution. Briefly, the method operates by randomly deciding at each step either to add/remove a regressor or to update the parameters. It jointly performs the identification of both the structure and the parameters, which though appealing in principle, turns out to be very problematic, since the parameter distribution over different model structures is usually quite complex and so their values do not vary continuously across structure variations [4]. Another randomized approach with quite promising features and that generally performs better than the RJMCMC, is the RaMSS algorithm [4], described in more detail in Section 3.2. Other randomized approaches developed for application in ML classification tasks, not based on evolutionary paradigms, have been proposed in a few recent works. For example, the approach described in [105] involves an initial random selection of the feature subset, which is subsequently updated according to a randomized scheme that may substitute or remove a single feature with a given probability. Though the

update process is randomized, the feature selection process is still incremental, with all the pros and cons of local search methods. Furthermore, the method only considers the performance of the current model as a whole, without distinguishing the relevance of the individual features within the model, thus making the update process blind with respect to the features. The methods discussed in [89] aim at the reduction of the original search space by random projections of the feature space on a smaller space and random extractions of feature subsets on which to perform the classification task. Again, the strategy appears to be completely blind and does not exploit the information resulting from each single subset that has been extracted and processed.

Regularization search mechanisms

Regularization methods have often been used for nonlinear MSS purposes. These methods amount to solving linear regression problems with additional constraints (typically related to the model size) to avoid ill-conditioning and/or to prevent overfitting. Ridge regression, the non-negative garrote and the least-absolute shrinkage and selection operator (LASSO), fall into this category [57]. While these methods are effective in obtaining parsimonious models, they are not necessarily equal to the much more challenging task of selecting the correct model structure (see, *e.g.*, the discussion in [19] relative to the LASSO). In ML regression problems these methods are referred as *srinkage methods* [127]. Regularization methods with application to classification problems are discussed in [92], where it was shown that for logistic regression with L1 regularization the sample complexity grows only logarithmically in the number of irrelevant features. In view of this, it is an appealing algorithm since it requires solving only a convex optimization problem. On the other hand, the sample complexity for the logistic regression method with L2 regularization grows at least linearly in the number of irrelevant features and thus may be ineffective in settings where only a few features are relevant, and the number of training examples is significantly smaller than the input dimension.

In the supervised learning framework for classification there are three main categories of FS methods, classified according to how the feature selection and classifier construction processes interact.

Filter algorithms select features based only on data-related properties, *i.e.* independently of the classifier design. For example, univariate filter methods are based on an individual feature assessment, which allows

a ranking of the features. Then, the top features in the ranking are selected. As interactions among features are not taken into account, it is not infrequent that redundant terms might be selected in this way [76]. Furthermore, features that are individually not significant are discarded, although they may actually reveal a strong discriminatory power in combination with others [72]. Multivariate filter methods overcome this problem by evaluating subsets of features according to some scoring function. This typically comes at a high cost in terms of computational load, especially when dealing with high dimensional data, given the exponential complexity of the combinatorial problem. Several multivariate filter methods have been proposed in the literature to overcome this issue such as, the Minimal-Redundancy Maximal-Relevance (MRMR) algorithm [38], Correlation Based Filtering (CFS) [54], Markov Blanket Filtering (MBF) [70]. All three methods are based on an incremental strategy for the selection of feature subset candidates. This has several drawbacks, both conceptual and computational, the most important of which being that the decision on which feature to add or remove at a given step of the selection process depends locally on the currently selected feature subset. In other words, the relevance of a specific feature is not evaluated as a global property, but rather as a local one. This may stray the selection process from the path leading to the optimal structure. Also, what is optimized at every step is only the local improvement of the current feature subset with an elementary feature variation. In this way selection errors are propagated throughout the process. Finally, the incremental strategy depends critically on the threshold adopted as a stopping criterion. For these reasons, all three methods are subject to redundancy and overfitting problems, especially if applied to datasets with extremely unbalanced dimensions such as microarrays [72]. Notice finally that MRMR and CFS employ the Pearson correlation coefficient, which is applicable only to two-dimensional feature vectors, and under the assumption of normal data distribution.

Wrapper algorithms are typically more accurate than filter-based ones, as they perform FS simultaneously with the classifier design, so that feature subsets are evaluated according to the performance the associated classifiers can achieve. The space of feature subsets is explored with heuristic rules (exhaustive search is typically not feasible) by employing mechnisms previously discussed. For example, the sequential FS (SFS) approach incrementally builds the model, by adding at each step the feature that most improves the classifier performance. Different wrapper algorithms based on the SFS search mechanism are proposed in the literature, such as the PTA(l, r) (plus l and take away r) [113], the SFFS (Sequential Forward Floating Selection)

algorithm and the SBFS (Sequential Backward Floating Selection) algorithm [100]. The same greedy policy is sometimes used in multivariate filter methods (see, *e.g.*, [38], [54], [107]), where it is used as a mechanism for generating model candidates, which are then evaluated according to the metric of choice. Greedy policies such as the SFS algorithms are subject to the mentioned redundancy and overfitting issues as the incremental methods discussed above. In addition, the classifier bias can negatively affect the FS process [73]. Besides sequential approaches, a significant amount of work has been devoted to evolutionary methods such as Genetic Algorithms (GA) [90], [109], [126], Genetic Programming (GP) [90], [88], [117], Particle Swarm Optimization (PSO) [125], [129], [64], [124], Ant Colony Optimization (ACO) [82], [69], hybrid GAs [94], [63], Harmony Search (HS) [102], [35].
Embedded algorithms combine the benefits of both explained approaches. A feature screening is initially performed using a filter-based approach, followed by the application of a wrapper method to refine the final solution. They appear to be convenient in large scale problems such as microarrays ($e.g.$ [52], [51]).

From the above discussion it is apparent that, regardless of the specific category, all FS methods may suffer from redundancy and overfitting issues especially with datasets having large or disproportionate dimensions. Another significant aspect is the computational cost involved in the combinatorial search of the space of feature subsets, which is further increased in wrapper methods which add the classifier design on top of each feature subset evaluation. As an attempt to overcome some of the discussed drawbacks of the existing methods, the first outcome of the research was the development of a novel classification approach denoted RFSC (Randomized FSC) algorithm [20], which jointly addresses the FS and classifier design tasks. The classifier is constructed as a polynomial expansion of the original attributes and a selection process is applied to find the relevant terms of the model. The selection method progressively refines a probability distribution defined on the model structure space, by extracting sample models from the current distribution and using the aggregate information obtained from the evaluation of the population of models to reinforce the probability of extracting the most important terms. In order to reduce the initial search space, a preprocessing technique called distance correlation filtering was also applied. Though the obtained results were satisfactory with small/medium size problems, it was observed that the performance of the algorithm deteriorates when dealing with larger data-sets, as the com-

plexity of the combinatorial problem grows exponentially. More precisely, both the time required to solve the problem and the actual quality of the solution (due, e.g., to local minima issues) are affected.

In NL identification no attempt has been made to parallelize and/or distribute the selection process. On the other hand, several approaches based on a parallelization of the processing have been proposed in the ML literature (see, *e.g*, [85], [33], [48]) to improve the effectiveness and speed up the selection process. For example, in [33] 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. Guillen et al. [48] 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. [29] showed that dividing the data-set 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 [18], [17] 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. 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 [125].

Alternative distributed architectures for supervised FS with horizontal and vertical partitioning have been proposed in [131] and [11], where local best solutions are shared among all processors. More in detail, horizontal partitioning is used in [131] 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, the

method of [131] 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 [**?**]). In [11], 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 [125].
To overcome the problem, an algorithm based on vertical data partitioning and a distributed searching architecture was proposed. It was tested both for model structure selection problems in the context of nonlinear identification (dRaMSS [8]) and FSC (DFS [22]) problems. In both cases, significant improvements have been obtained for large problems in the execution time and the classification performance.

The basic idea underlying the proposed distributed method is as follows. The regressors/features are divided into subsets (vertical data-set partitioning), 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 regressors/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, this architecture is capable of addressing relatively large scale examples. The procedure is also efficient since the local processors perform the selection tasks in parallel and on much smaller search spaces. An 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 model. Though effective on medium/large problems, the proposed approach still appears ineffective when dealing with extra-large searching spaces (e.g., micro-arrays) due to computational issues. As an attempt to solve this problem, a multivariate filter algorithm (D^2CORFS [21]) exploiting the benefits of vertical partitioning and of the described distributed architecture is proposed. The method is based on a distance correlation index that measures the output dependence on a given subset of features. It was shown to be quite effective in problems such as micro-arrays.

Structure of the thesis

The thesis is organized as follows:

- Chapter 2 provides the reader with the basic theoretical frameworks and concepts used for developing the proposed concepts.
- Chapter 3 presents a detailed explanation of the randomized model search idea, which is based on the randomized extraction of populations of models. A novel wrapper algorithm (RFSC [20]) based on this concept is discussed in details.
- In Chapter 4 a distributed scheme with partial information sharing is introduced, that is applicable to both the NL [8] and the ML [22] frameworks. It exploits the benefits associated to parallelization, vertical data partitioning, and information exchange (among the different processors). The presented distributed optimization scheme can in principle be combined with any FS method of choice. The proposed distributed scheme is exploited to develop a novel multivariate filter FS algorithm denoted as D2CORF [21]) which employs the distance correlation (dCor) index as a criterion function to reinforce the probability of extracting the most important terms based on concept discussed in Chapter 3.
- The results of a new ensemble method which exploits the random nature of the RFSC algorithm together with sample manipulation as a mechanism to generate ensembles of classifiers are presented in Chapter 5.
- Some concluding remarks are drawn in Chapter 6.

CHAPTER 2

Preliminaries

This chapter provides the basic theoretical framework and concepts used in this thesis. First, it is discussed why FS is a prerequisite for obtaining good results in classification problems. In Sections 2.2 and 2.3, a general introduction to classification and regression in the context of supervised machine learning, is provided, as well as the principles on which well-established classification algorithms are based. Then, the nonlinear identification problem is introduced, which for certain classes of nonlinear dynamical models (*e.g.*, the NARX/NARMAX), amounts to a linear regression where the main task is the selection of appropriate regressors that best represent the underlying process. This model structure selection (MSS) turns out to be a task akin to FS. Technical issues related to nonlinear model identification, such as parameter estimation and model evaluation, are discussed in Sec. 2.6. Finally, Sec. 2.7 provides metrics which are used in both discussed frameworks, and introduces the distance correlation (dCor) index as a dependence measure for random variables.

2.1 Supervised learning and the feature selection problem

Supervised learning is the process of learning a map between a set of input variables and an output variable, with the aim of using this map to predict the outputs for unseen data [31]. A dataset generally consists of a set of N observations $\{\boldsymbol{u}(k), y(k), k = 1, \ldots, N\}$, where $\boldsymbol{u} = [u_1, \ldots, u_{N_f}]$ is the input vector and y is the output. The input-output pairs used in the learning process constitute the *training set*, while those used for model evaluation belong to the *test set*. Input variables are often denoted *features* (or attributes), especially in connection with classification problems. A feature can be defined as an individual measurable property of the process being observed [27]. Outputs can be discrete (usually denoted *classes*) or continuous, and the learning problem is called a classification or a regression problem, respectively. Accordingly, the inferred function is called a *classifier* if the output takes discrete values and a *regression function* if the output is continuous [61].

It is often the case that the number of available features is large. Indeed, improvements in data collection technologies resulted in a drastic data expansion, from tens to thousands of variables, and more. However, this information explosion has also its drawbacks. Indeed, most of the available features are redundant or irrelevant, and as such they may have a negative effect on the learning process and on the accuracy of the resulting model, as they can add more noise than useful information [72]. Reducing the set of features to the really essential ones has various advantages: it reduces the computational cost of the learning process and simplifies the model structure, thus facilitating its interpretation and data understanding, and ultimately improves both the model accuracy and robustness [32].

Hence, a dimensionality reduction (in terms of the input variables) has become a crucial task in different fields such as bioinformatics, machine learning (ML), pattern recognition (PR), etc. In general, it is performed either by *feature extraction* or by *feature selection* (FS) [101], or combinations thereof. The first approach transforms the initial feature space into a lower dimensional space. On the other hand, FS reduces the problem dimensionality by selecting the best possible subset of the complete input feature set. In this work we focus on the latter approach, which can be formalized as follows.

Let $\mathcal{F} = \{u_1, \ldots, u_{N_f}\}$ be a set of N_f possible input variables. Then,

$\mathcal{S} = 2^{\mathcal{F}}$ denotes the set of all possible feature subsets. Given $s \in \mathcal{S}$, $|s|$ denotes its size (the number of input variables it contains). Let $\mathcal{J}$ be some criterion function (*e.g.*, a performance index or some statistical metric), used to evaluate feature subsets. Without any loss of generality, let us consider a higher value of $\mathcal{J}$ to indicate a better feature subset. Then the FS problem can be formulated as follows: find the subset $s^\star$ for which

$$\mathcal{J}(s^\star) = \max_{s \in \mathcal{S}} \mathcal{J}(s). \tag{2.1}$$

In other words, FS can be envisaged as a combinatorial problem aiming at selecting a feature subset that maximizes the chosen criterion function.

A model $\hat{y} = f(s)$ is a function of (a subset of) features that provides an estimate of the output corresponding to its arguments. Once the modeling procedure is defined (*e.g.*, a linear regression), a model is fully determined by the subset of features constituting its arguments. In other words, there is a bijective mapping from subsets of features to models.

2.2 Supervised learning: The classification problem

In classification problems one is interested in constructing a model that captures the relationship between features (inputs) and classes (outputs) through a learning process operating on the available observations (input-output pairs).

Assume that a set of N observations is available, each consisting of a pair $(\boldsymbol{u}(k), c(k))$, $k = 1, \ldots, N$, where the components u_p, $p = 1, \ldots, N_f$ of vector $\boldsymbol{u}$ are the original features and $c \in \{1, \ldots, N_c\}$ is the corresponding class (assumed known, according to the supervised learning framework).

The objective is to construct a model that is capable of predicting correctly the class for observations unseen in the learning phase, of the following type:

$$\hat{c}(k) = f(\boldsymbol{u}(k)), \tag{2.2}$$

where $\hat{c}$ denotes the class estimate and f is a function. The simplest case is binary classification, which can be addressed *e.g.* by assigning the estimated class value depending on the sign of $f(\boldsymbol{u}(k))$. Multiclass problems are generally addressed by a generalization of binary classifiers, according to two main strategies, namely *one-vs.-all* and *one-vs.-one*. In the former

case, a binary classifier is trained for each class (the output value representing the membership of the input sample to the given class). The samples of that class are given as positive samples, and all other samples as negatives. In the one-vs.-one strategy, a binary classifier is trained for each pair of classes, and must discriminate these two only. In prediction, a majority voting scheme is applied to decide the actual estimated class. In this work we adopt a one-vs.-all approach, and accordingly recode the output as an N_c-dimensional vector $\boldsymbol{y}$, with binary components, defined as:

$$y_i(k) = \begin{cases} 1, & c(k) = i \\ -1, & \text{otherwise} \end{cases} \tag{2.3}$$

where $i = 1, \ldots, N_c$. Accordingly, the multiclass problem is recast as N_c binary classification problems, resulting in the following N_c models:

$$\hat{y}_i(k) = f_i(\boldsymbol{u}(k)),\ i = 1, \ldots, N_c. \tag{2.4}$$

Notice that if $N_c = 2$, a single output is sufficient to discriminate between the two classes, the -1 value of y_1 being directly associated to class 2.

To avoid ambiguities in the class estimation, the actual class estimate is conventionally assumed as the label corresponding to the individual classifier returning the largest value:

$$\hat{c}(k) = \arg \max_{i=1,\ldots,N_c} \hat{y}_i(k). \tag{2.5}$$

In summary, the multiclass problem can be addressed by training one binary classifier for each class, that discriminates if a sample belongs to that class or not. Accordingly, in the following we shall focus on the training and evaluation of the binary classifiers $\hat{y}_i(k)$, $i = 1, \ldots, N_c$.

In the literature, different algorithms have been proposed regarding the classifier design problem, based on Artificial Neural Networks (ANN) [95], Support Vector Machines (SVM), often in combination with Radial Basis Functions (RBF) as kernel functions [49], Twin Support Vector Machines (TSVM) [123], instance based learning methods such as Nearest Neighbor (NN) and Data Gravitational Classification (DGC) [2], evolutionary methods as genetic programming (GP) [40] and PSO [84], [108]. Instance based algorithms are particularly appealing due to their simple classification logic and generally satisfactory performance. In the NN (or 1-NN) algorithm, a new sample is simply assigned to the class of the nearest previously available sample. This is one of the most used and well known classification

algorithms due to its simplicity, though it is reported to suffer from various drawbacks such as high dimensionality, low efficiency, high storage requirements and sensitivity to noise [116]. Cases where the classes are non-separable or overlapping appear to be particularly critical [78]. The k-NN extension classifies a new instance based on the majority of the k nearest neighbors. Several variants of the k-NN method have been proposed in the literature, that typically introduce weighted distances or similar concepts to improve the performance, such as the Adaptive k-NN (KNN-A) [118], the Distance Weighted k-NN (DW-KNN) [39], the Center NN (CNN) [45], the Cam weighted distance NN (CamNN) [132]. DGC algorithms [23] have also been put forward as an attempt to overcome the mentioned problems of the NN algorithm. In this respect, it is worth mentioning the work of Peng *et al.* [96], which employs feature weighting and a tentative random feature selection algorithm to compute the feature weights. An enhancement of the DGC algorithm, denoted DGC+ is proposed in [23] to deal with imbalanced data. Peng [97] also proposed a fast feature weighting algorithm for DGC that evaluates features by using discrimination and redundancy.

In the following we provide the basic principles of the classification methods that are used in this thesis.

2.2.1 Linear regression classifier

A convenient way to represent the unknown functions $f_i(\cdot)$, $i = 1, \ldots, N_c$, is to employ parametric functional expansions, so that the class estimate $\hat{y}_i$:

$$\hat{y}_i(k) = \left(\sum_{j=1}^{N_F} \vartheta_j^{(i)} \varphi_j(k) \right) = \boldsymbol{\varphi}^T(k) \boldsymbol{\vartheta}^{(i)}, \tag{2.6}$$

$i = 1, \ldots, N_c$, where $\boldsymbol{\vartheta}^{(i)}$ is a vector of unknown parameters (associated to the ith output), and $\boldsymbol{\varphi}(k) = [\varphi_1(k), \ \ldots, \ \varphi_{N_F}(k)]^T$, where $\varphi_j(k) = \varphi_j(\boldsymbol{u}(k))$, $j = 1, \ldots, N_F$, is a given (nonlinear) function of the features. It predicts class 1 if $\hat{y}_i(k) > 0$ otherwise class -1.

In view of the fact that equation (2.6) actually configures a linear regression, these (extended) features are also called regressors.

Various types of basis functions can be used to construct the functional expansions, such as Fourier series, piecewise linear models, polynomial models, radial basis functions, and sigmoidal neural networks, all having

the universal approximation property. In this work, we will consider polynomial expansions, so that a generic term $\varphi_j(k)$ takes the form:

$$\varphi_j(k) = \begin{cases} u_{p_1}(k) \cdot u_{p_2}(k) \cdot \ldots \cdot u_{p_l}(k), & l > 0 \\ 1, & l = 0 \end{cases} \tag{2.7}$$

where $p_s \in \{1, \ldots, N_f\}$, $s = 1, \ldots, l$, with $0 \leq l \leq L$, L being the maximum allowed degree of the polynomial expansion for classification problems.

The polynomial expansion is a natural generalization of a purely linear model, that is useful when the first order interactions among the features are insufficient to obtain satisfactory results [51]. Most conveniently, this formulation still provides a linear-in-the-parameters model, which greatly facilitates parameter estimation. On the down side, the number of terms in a polynomial expansion increases rapidly with the maximum degree and the number of arguments (curse of dimensionality). Conventional practice has it that relatively small models of this category are suitable for various applications. It is also well-known that the smaller the size of the model, the more robust it will be and the more capable of generalizing to new observations. Therefore, a crucial problem consists in selecting the best terms of type (2.7) for the model.

2.2.2 Linear Support Vector Machine (SVM) classifier

SVM is a supervised ML tool applicable to both the mentioned supervised learning tasks, namely the regression and classification problems. Here we discuss its application to the latter case.

Fig. 2.1 illustrates a simple two dimensional classification problem for a set of linearly separable data (samples of the two classes are shown in different colors). The coordinates of the individual instances are called *support vectors*. A linear classifier has a form given by:

$$y(\boldsymbol{\varphi}) = \boldsymbol{w}_{SVM}^T \boldsymbol{u} + b_{SVM}, \tag{2.8}$$

where vector $\boldsymbol{w}_{SVM}$ is a weight vector and b_{SVM} is a bias. In two dimensions, as in the given 2-dimensional example, it represents a line splitting the plane into two parts, one for the positive class and one for the negative one. For an n-dimensional space (2.8) is a hyperplane. Learning the linear SVM classifier amounts to finding the hyperplane which provides optimal separation among the two classes, by maximizing the distance of

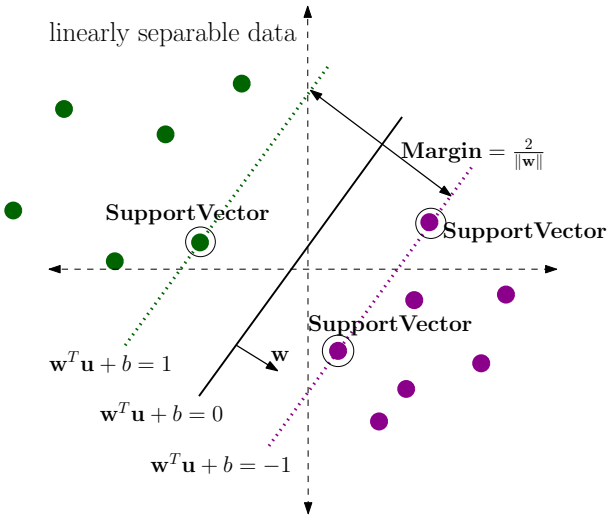

Figure 2.1: *Two-dimensional classification problem with SVM classifier.*

the nearest data points from the hyperplane. Hence, learning a SVM classifier can be formulated as a quadratic optimization problem subject to linear constraints, as follows:

$$\arg\min_{\boldsymbol{w}_{SVM}} \frac{2}{\|\boldsymbol{w}_{SVM}\|} \tag{2.9}$$

$$\text{subject to} \tag{2.10}$$

$$y_k(\boldsymbol{w}_{SVM}^T \boldsymbol{u}_k + b_{SVM}) \geq 1 \quad k = 1, \ldots, N. \tag{2.11}$$

2.2.3 k-Nearest Neighbors (k-NN) classifier

Given a new datum $\boldsymbol{u}_{N+1}$, the k-NN algorithm identifies among the training input vectors $\boldsymbol{u}_k$, $k = 1, \ldots, N$ the k nearest neighbors to it, regardless of their class labels. The new data instance is assigned to the class corresponding to the majority of the k nearest neighbors. The distance between input samples can be calculated using different metrics, such as the Euclidean, the Minkowski or the Mahalanobis distances. To avoid ambiguity, k has to be an odd number in biclass problems, while in the multiclass case k must not be a multiple of the total number of classes. The k-NN method works on a very simple principle, and is very easy to apply. However, it is known to be sensitive to noise [23]. Moreover, all attributes are assumed to have the same impact on the computation of the distance, which can be a drawback in the presence of redundant terms, possibly leading to a misclassification.

2.2.4 Naive Bayes (NB) classifier

The NB is a statistical classifier based on Bayes Theorem, which predicts the class membership probability and assigns an unseen data sample to the class with the highest membership probability. It is based on *conditional independence*, that is the assumption that the feature values are conditionally independent of one another given the class y_i of the sample.

According to Bayes Theorem, a *posteriori* probability that the a sample $\boldsymbol{u} = [u_1, \cdots, u_{N_f}]$ [1], belongs to class y_i, $i = 1, \ldots, N_c$ is given by:

$$P(y_i|\boldsymbol{u}) = \frac{P(\boldsymbol{u}|y_i)P(y_i)}{P(\boldsymbol{u})}, \tag{2.12}$$

where $P(\boldsymbol{u}|y_i)$ is the a posteriori probability of $\boldsymbol{u}$ conditioned on class y_i, $P(\boldsymbol{u})$ is the a priori probability of $\boldsymbol{u}$ and $P(y_i)$ is the a priori probability of y_i.
Applying the naive assumption of conditional independence, $P(\boldsymbol{u}|y_i)$ in 2.12 can be expressed as:

$$P(\boldsymbol{u}|y_i) \approx \prod_{j=1}^{N_f} P(u_j|y_i). \tag{2.13}$$

Probabilities $P(u_j|y_i)$, $j = 1, \cdots, N_f$ can be estimated from the training set. In case u_j is a categorical variable, $P(u_j|y_i)$ is actually the number of samples of class y_i in the training set having the feature value u_j, divided by the number of samples of class y_i in N. Otherwise, if the feature values are continuous it is assumed that they have a Gaussian distribution g with a mean μ and standard deviation (STD) σ so that:

$$P(u_j|y_i) = g(u_k, \mu_{y_i}, \sigma_{y_i}) \tag{2.14}$$

where μ_{y_i} and σ_{y_i} are the mean and STD of an attribute u_k of class C on the training data.
Since the denominator in 2.12 does not depend on the class and the feature values u_j are given, only the numerator matters when computing $P(y_i|\boldsymbol{u})$. In the classification phase, a new data sample is attributed to the class y_i for which $P(\boldsymbol{u}|y_i)P(y_i))$ is maximized.

[1] Here $\boldsymbol{u} = [u_1, \cdots, u_{N_f}]$ denotes vector of N_f feature values for a given data sample.

2.3 Supervised learning: The linear regression problem

In regression problems the output variable is not categorical, but continuous. Input variables are called *independent variables*, and the output(s) the *dependent variable(s)*. The output can be either a vector $\boldsymbol{y} = [y_1, \ldots, y_M]$, M denoting the number of outputs (*multiple regression problem*) or a scalar. For simplicity, we will focus on the scalar case.

The output y is a function of $\boldsymbol{u}$, through the parameter vector $\boldsymbol{\vartheta}$. The regression is linear if $y(k)$ is linear in $\boldsymbol{\vartheta}$. In this work we focus on linear regression problems. Given a set of N observations, each consisting of a pair $(\boldsymbol{u}(k), y(k))$, $k = 1, \ldots, N$, where $\boldsymbol{u}(k)$ is vector of input values and $y(k)$ is the corresponding (scalar) output. The aim is to find a model of the form

$$\hat{y}(k) = \sum_{j=1}^{N_F} \vartheta_j \varphi_j(k) = \boldsymbol{\varphi}^T(k)\boldsymbol{\vartheta}, \tag{2.15}$$

where $\boldsymbol{\varphi}(k) = [\varphi_1(k),\ \ldots,\ \varphi_{N_F}(k)]^T$, $\varphi_j(k)$, $j = 1, \ldots, N_F$ being (possibly nonlinear) functions of the original input variables, of the form (2.7), and $\boldsymbol{\vartheta} = [\vartheta_1,\ \ldots,\ \vartheta_{N_F}]^T$ is the vector of unknown parameters.
This formulation has the advantage that the model is linear-in-the-parameters so simple parameter estimation methods such as Least Squares (LS) regression can be applied, what greatly facilitates parameter estimation. Another benefit is that it has provable convergence conditions.

2.4 Validation procedures in classification problems

The aim of the classifier design is to learn a map between a set of input variables and an output variable, capable to predict the outputs for unseen data. A good model has good predictive capabilities. In order to assess the accuracy and reliability of the learned models data are split into two parts: training set which is used in the learning process, and test set which is used for model evaluation.
The validation method based on this principle is called *hold out*. Data samples are randomly divided into two subsets, one of which is employed for training and the other one for testing. It involves just a single execution of an algorithm. In the work presented here, 70% samples are employed for training and 30% for testing.
To reduce the dependence of the results on the choice of training and test data, the classification performance is usually evaluated using the k-fold

cross validation (k-FCV) approach. Briefly, the dataset is split into k (equal and non-overlapping) subsets (folds), possibly uniformly representative of all classes. Then, $k-1$ folds are used for training and the remaining ones for testing, the procedure being repeated k times so that all folds are tested once. The algorithm performance is finally computed as the average over the ten independent runs. Leave-one-out-cross-validation (LOOCV) is a particular case of k-FCV, with $k = N$.

2.5 Nonlinear system identification

System identification refers to the procedure of building a mathematical description of the dynamic behavior of a system or process from measured data [110]. The black-box identification of nonlinear dynamical models is a challenging problem that has received much attention in the last decades.

A finite-order, nonlinear, discrete-time, dynamical model can be represented as a recursive expression that relates the current output value $y(k)$ to the past values of the input ($u(\cdot)$) and output signals:

$$y(k) = f(\boldsymbol{x}(k)) + e(k) \tag{2.16}$$

where $f(\cdot)$ is a nonlinear function, $e(\cdot)$ is a noise term assumed to be drawn from an independent and identically distributed (i.i.d.) sequence, and $\boldsymbol{x}(k) = [y(k-1), \ldots, y(k-n_y), u(k-1), \ldots, u(k-n_u)]$, n_u and n_y being the input and output maximum lags, respectively [62]. Equation (2.16) is referred to as NARX (Nonlinear AutoRegressive with eXogenous variables) model representation, a subclass of the more general NARMAX (Nonlinear AutoRegressive Moving Average with eXogenous variables) class introduced in [74, 75], which also allows for past noise terms in $\boldsymbol{x}(k)$, therefore allowing for a more flexible representation of the disturbance model. The NAR[MA]X representation provides a unified representation for a wide class of nonlinear systems [14]. NARX/NARMAX models have been successfully employed in many different application fields, because of their flexibility and representative capabilities.

The nonlinear system identification problem can be described as the problem of finding an estimator $\hat{f}(\cdot)$ for function $f(\cdot)$ by processing a set of input-output observations $\{(u(k), y(k)), k = 1, \ldots N\}$. Assuming that f is continuous, one may resort to a universal approximator, *i.e.* a functional expansion that can approximate any continuous function to an arbitrary level of accuracy, provided that it is endowed with sufficient degrees of freedom. In the literature different approximation functions for nonlinear

system identification are introduced, such as piecewise linear models, polynomial expansions, multivariate adaptive regression splines [44], Gaussian processes, wavelets, multilayer perceptron neural networks, radial basis functions. Still, functional expansions that provide a linear combination of nonlinear basis functions are the most common choice:

$$\hat{y}(k) = \hat{f}(\boldsymbol{x}(k), \boldsymbol{\vartheta}) = \sum_{j=1}^{N_F} \vartheta_j \, \varphi_j(k), \tag{2.17}$$

where $\hat{y}(k)$ denotes the estimate of $y(k)$, $\varphi_j(k) = \varphi_j(\boldsymbol{x}(k))$, $j = 1, \ldots, N_F$, are known nonlinear basis functions and ϑ_j, $j = 1, \ldots, N_F$, are parameters. Expression (2.17) is a linear regression, where the nonlinear basis functions are referred as regressors and as such it carries over the properties discussed in Section 2.3.

Equation (2.17) can also be expressed in vector form:

$$\hat{y}(k) = \boldsymbol{\varphi}(k)^T \boldsymbol{\vartheta} \tag{2.18}$$

$\boldsymbol{\varphi}(k) = [\varphi_1(k), \ \ldots, \ \varphi_{N_F}(k)]^T$ denoting the regressor vector and $\boldsymbol{\vartheta} = [\vartheta_1, \ \ldots, \ \vartheta_{N_F}]^T$ the parameter vector. The advantages of linear regressions such as (2.18) have already been discussed in Section 2.3.

Among all possible linear-in-the-parameters function expansions, a popular choice is the polynomial one. In the polynomial NARX model $f(\boldsymbol{x}(k))$ is a linear combination of all possible monomials in the components of $\boldsymbol{x}(k)$ up to a given degree n_d, so that a generic term $\varphi_j(k)$ takes the form:

$$\varphi_j(k) = \begin{cases} y(k-d_{j1}) \ldots y(k-d_{jr}) u(k-d_{j,r+1}) \ldots u(k-d_{jl}), & l > 0 \\ 1, & l = 0 \end{cases} \tag{2.19}$$

with $d_{j1}, \cdots, d_{jl} \in \mathbb{N}$, L being the maximum degree of the polynomial expansion. Applying (2.19) to (3.8) results in polynomial NARX models, which can be considered as a direct extension of their linear counterparts since the nonlinearities are explicitly modeled as products of the linear variables.
This allows an easier interpretation of the model, because terms can be more directly related to the physical aspects of the system under consideration. Additionally, each element of the parameter vector ϑ can be interpreted as a measure of the importance of the associated physical phenomena and the existence of cross terms can also reveal the existence of

specific nonlinear dependencies among them as well. Other models, such as artificial neural network or wavelets, are usually capable of providing more compact representations, but they are not easily interpretable as there is not a direct relationship between model terms and physical variables.

The similarity of expressions (2.6), (2.15) and (2.18), which represent the model form in the classification, the linear regression and the nonlinear identification problems, respectively, is apparent. Notice, however, that the input-output relationship in classification and regression problems is typically non-dynamic, *i.e.* the output does not depend on the previous input or output values, whereas it is dynamic in the nonlinear identification case. Also, in the classification problem the outputs (and sometimes the inputs as well) take values in a discrete set, whereas in regression and identification problems the input and output signals take continuous values.

2.6 Parameter estimation in linear regression problems

The linear-in-the-parameters structure of the NARX and linear regression models given by (2.18) and (2.15), allow the application of well established, computationally convenient techniques for parameter estimation of the Least Squares (LS) family. The loss function is defined as the sum (or mean) of the squares of the output estimation errors $e(k) = y(k) - \hat{y}(k)$, $k = 1, \ldots, N$. In the context of dynamical NARX models $e(k)$ coincides with the one-step ahead prediction error, so that the model minimizing the loss function is that with the best one-step ahead prediction capabilities, according to the Prediction Error Minimization (PEM) paradigm. For classification problems, on the other hand, due to the discrete nature of the output, the loss function has a different form and accounts for the rate of misclassified observations (with respect to class i), the so called $0-1$ loss. Due to the hard nonlinearity enforced by this loss function, it is customary to approximate it with smoother functions, in order to facilitate the optimization process.

2.6.1 Least Squares

In linear regression problems, the cost function is given by the Mean Squared Error (MSE) :

$$J_p(\boldsymbol{\vartheta}) = \frac{1}{N}\sum_{k=1}^{N}(y(k) - \hat{y}(k))^2, \tag{2.20}$$

where $\hat{y}(k)$ is the estimate of the output at time k. For dynamical NARX models it holds that $\hat{y}(k) = \hat{y}(k|k-1)$, which is the optimal one-step-ahead prediction of $y(k)$ (it is based on information up to time $k-1$). Accordingly, in that context, (2.20) is referred to as the Mean Squared Prediction Error (MSPE).

The optimal model is such that

$$\hat{\boldsymbol{\vartheta}} = \arg\min_{\boldsymbol{\vartheta}} J_p(\boldsymbol{\vartheta}), \tag{2.21}$$

and can be calculated in closed form with the LS formula. More in detail, expression (2.20) can be conveniently rewritten in matrix form:

$$J_p(\boldsymbol{\vartheta}) = \frac{1}{N} \|\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{\vartheta}\|^2, \tag{2.22}$$

where $\boldsymbol{y} = [y(1), \ldots, y(N)]^T$, and $\boldsymbol{\Phi}$ is the $N \times m$ regression matrix

$$\boldsymbol{\Phi} = \begin{bmatrix} \varphi_1(1) & \varphi_2(1) & \cdots & \varphi_m(1) \\ \varphi_1(2) & \varphi_2(2) & \cdots & \varphi_m(2) \\ \vdots & \vdots & \ddots & \vdots \\ \varphi_1(N) & \varphi_2(N) & \cdots & \varphi_m(N) \end{bmatrix}. \tag{2.23}$$

The LS solution of (2.21) is given by

$$\hat{\boldsymbol{\vartheta}} = (\boldsymbol{\Phi}^T\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^T\boldsymbol{y}, \tag{2.24}$$

provided that matrix $\boldsymbol{\Phi}^T\boldsymbol{\Phi}$ is positive definite, a condition that is fulfilled if $\boldsymbol{\Phi}$ has full rank.

2.6.2 Logistic regression

As already mentioned, the modeling task for classifiers taking the form of a linear regression is addressed separately for each class. In the following, we shall focus on the modeling of classifier $\hat{y}_i$ associated to class i. For ease of notation we will drop the indexing on class i.

For a given structure, the parameter estimation for a model of type (2.6) is typically formulated as a minimization problem with reference to a loss function defined as $\mathcal{L} : \{-1, +1\} \times \mathbb{R} \to \mathbb{R}_+$. A standard loss function evaluates the model performance as the ratio of misclassified observations (with respect to class i) over the total number of tested samples. The resulting optimization problem is given by

$$\min_{\vartheta} \ \frac{1}{N} \sum_{k=1}^{N} \mathcal{L}_{0-1}(y(k), \hat{y}(k)), \tag{2.25}$$

where $\mathcal{L}_{0-1}$ is the *0-1 loss function*, defined as $\mathcal{L}_{0-1}(q_1, q_2) = \mathbf{1}_{\{q_1 q_2 < 0\}}(q_1, q_2)$.

Due to the hard nonlinearity enforced by this loss function, the latter is usually approximated with functions with nicer properties regarding optimization (*e.g.*, hinge, squared hinge, logistic, exponential). In the following, the *logistic loss* will be employed for this purpose, resulting in the following reformulation of the optimization problem:

$$\min_{\vartheta} \frac{1}{N} \sum_{k=1}^{N} \log(1 + e^{-y(k)\hat{y}(k)}). \tag{2.26}$$

The reader should note that $\log(1 + e^{(\cdot)})$ is a strictly convex function, and $\hat{y}(k)$ is linear in ϑ. Therefore, the resulting cost function is strictly convex in ϑ, and the minimizer of (2.26) is unique.

Although a closed-form solution to the above optimization problem does not exist, the logistic loss is a continuous differentiable function, which allows to apply gradient descent methods in the optimization process. In this work, a standard Newton's iterative optimization scheme is applied to solve problem (2.26).

2.6.3 Statistical test for regressor significance

The rejection of redundant terms is a crucial step in the identification procedure. It can be shown that the covariance matrix of the LS estimator $\hat{\boldsymbol{\vartheta}}$ is equal to

$$\text{cov}(\hat{\boldsymbol{\vartheta}}) = (\boldsymbol{\Phi}^T \boldsymbol{\Phi})^{-1} \sigma_e^2 \tag{2.27}$$

where σ_e^2 is the variance of the noise signal $e(\cdot)$ which can be estimated as

$$\hat{\sigma}_e^2 = \frac{1}{N-m} \|\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{\vartheta}\|^2 = \frac{N}{N-m} J_p. \tag{2.28}$$

Using $\hat{\sigma}_e^2$ in (2.27) we can estimate the variance of the jth regressor as:

$$\hat{\sigma}_j^2 = \hat{\sigma}_e^2 V_{jj}, \tag{2.29}$$

where V_{jj} is the jth diagonal element of $V = (\boldsymbol{\Phi}^T \boldsymbol{\Phi})^{-1}$.

The variances $\hat{\sigma}_j^2$ can be used to test the statistical relevance of each regressor using a Student's *t*-test. More precisely, the jth regressor is considered to be statistically irrelevant (and therefore rejected) if the interval

$$[\hat{\vartheta}_j - \hat{\sigma}_j t_{\alpha, N-\tau}, \hat{\vartheta}_j + \hat{\sigma}_j t_{\alpha, N-\tau}] \tag{2.30}$$

contains 0, $t_{\alpha,N-\tau}$ being the $100(1-\alpha)$ percentile of the Student's t distribution with $N-\tau$ degrees of freedom and significance confidence interval α, where τ is the number of components of ϑ.

A slightly different reasoning applies to the logistic regression problem, for which the Newton method has been applied to solve the optimization problem (2.26), and not the LS formula. According to [16], the update equation of the Newton method is structurally equivalent to an Iteratively Reweighted Least Squares algorithm, so that upon convergence one can evaluate the parameter covariance as in standard Weighted Least Squares. Therefore, the variance $\hat{\sigma}_j^2$ of the estimated parameters is given by:

$$\hat{\sigma}_j^2 \approx \hat{\sigma}_e^2 G_{jj}^{-1}, \tag{2.31}$$

where $\hat{\sigma}_e^2$ is variance of the residuals and G_{jj} is the jth diagonal element of the Hessian $\boldsymbol{G} = \boldsymbol{\Phi}^T \boldsymbol{R} \boldsymbol{\Phi}$ upon convergence, $\boldsymbol{\Phi}$ being the regression matrix and $\boldsymbol{R}$ a diagonal $N \times N$ matrix with diagonal elements given by:

$$R_{kk} = \tilde{y}(k)(1-\tilde{y}(k)), \tag{2.32}$$

$k = 1, \ldots, N$, where $\tilde{y}(k) = 1/(1+e^{y(k)\hat{y}(k)})$.

Thus defined, the variance (2.31) can be employed in a Student's t-test as well, to determine the statistical relevance of each regressor.

2.7 Model evaluation metrics

2.7.1 Model evaluation metrics in nonlinear identification

As already mentioned, in the PEM framework it is natural to evaluate models according to their one-step-ahead predictive capabilities, using the MSPE index. However, to improve the robustness of the MSS task it is sometimes useful to compare models also in terms of their long range dynamics, using the Mean Square Simulation Error (MSSE) besides the MSPE [98]:

$$MSSE = J_s(\boldsymbol{\vartheta}) = \frac{1}{N}\sum_{k=1}^{N}(y(k)-\hat{y}_s(k))^2, \tag{2.33}$$

where $\hat{y}_s$ denotes the simulated output of the model (sometimes referred to as free-run prediction). In this framework the model output at a given time instant k is calculated as a function of the previous model output samples $\hat{y}_s(k-1), \ldots, \hat{y}_s(k-n_y)$ as opposed to the measured output samples

$y(k-1), \ldots, y(k-n_y)$. For a more detailed comparison of prediction- and simulation-based identification approaches the reader is referred to [98], [30], [1]. For computational reasons, we here limit the use of J_s only to the evaluation of models for the MSS task, whereas parameter estimation is still carried out by minimization of the J_p index.

Furthermore, for MSS purposes an exponential version of these cost functions is used, which remaps them in the $[0, 1]$ range (high-performance models being close to 1), [111]. More in detail, let $\mathcal{J}_p = e^{-KJ_p}$, and $\mathcal{J}_s = e^{-KJ_s}$, where $K > 0 \in \mathbb{R}$ is a tuning parameter (K is sometimes denoted "risk sensitivity" parameter, see *e.g.* [77]). In this way, the difference between models with similar performance is amplified, which facilitates the discrimination process operated by the MSS procedure. This is especially important in prediction error minimization approaches where many models of comparable high performance may sometimes be obtained: with the exponential index one can detect even small improvements in the performance.

Accordingly, for MSS purposes models are evaluated in terms of the following composite index [1], [4]:

$$\mathcal{J} = \alpha_J \mathcal{J}_s + (1 - \alpha_J)\mathcal{J}_p, \tag{2.34}$$

where $\alpha_J \in [0, 1]$ is a user defined parameter that determines the balance between the simulation and prediction indices.

2.7.2 Model evaluation metrics in classification problems

Classifiers are commonly evaluated with several metrics so that different aspects of the obtained model can be assessed. The basic evaluation metric is the correct classification rate, defined as the ratio of correctly classified samples over the total number of tested samples:

$$\mathcal{J} = 1 - \frac{1}{N} \sum_{k=1}^{N} \mathcal{L}_{0-1}(y(k), \hat{y}(k)) \tag{2.35}$$

The performance index (2.35) can be reformulated as:

$$\mathcal{J} = \frac{TP + TN}{N}, \tag{2.36}$$

where TP and TN denote the number of correctly classified samples of the positive and negative classes, respectively. The total number of samples equals the sum of misclassified and correctly classified samples of

both classes, *i.e.* $N = TP + TN + FP + FN$, where FP and FN are the misclassified samples of the positive and negative classes, respectively. The sensitivity of the classifier is measured by the true positive rate $TPR = \frac{TP}{TP+FN}$, *i.e.* the ratio of correctly classified positive samples over the total number of positive samples. Conversely, the specificity is captured by the true negative rate $TNR = \frac{TN}{TN+FP}$, *i.e.* the ratio between correctly classified negative samples over the total number of negative samples. The Gmean and Fscore indices combine both criteria:

$$Gmean = \sqrt{TPR \cdot TNR} \tag{2.37}$$

$$Fscore = 2\frac{TPR \cdot TNR}{TPR + TNR}. \tag{2.38}$$

An alternative accuracy measure for classifiers is the Cohen's Kappa rate [12], which is capable of dealing more effectively with imbalanced data. The Kappa statistic was originally designed to compare two different classifiers to measure the degree of (dis)agreement, compensating for chance (dis)agreements, but can be used to evaluate the merit of a specific classifier by comparing it to an "ideal" classifier producing the exact classifications. Let the *confusion matrix* be an $N_c \times N_c$ matrix C such that C_{ij} equals the number of samples that are classified in class i by classifier 1 and j by classifier 2, and denote by $C_{i\cdot} = \sum_{k=1}^{N_c} C_{ik}$ and $C_{\cdot j} = \sum_{k=1}^{N_c} C_{kj}$ the row and column counts (that represent the individual classification counts). Then, the Kappa rate is defined as follows:

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

and ranges from -1 (total disagreement) to 0 (random classification) to 1 (total agreement). The Kappa statistic is very useful for multi-class problems, in that it measures the classifier accuracy while compensating for random successes [23].

2.8 Measuring the dependence of random variables

Various statistical tests have been developed in the literature to test the dependence of random vectors. The dCor criterion, proposed in [115] and refined in [114], is a generalization of the correlation concept that provides a reliable dependence measure between random vectors of arbitrary dimension. It is applicable to both discrete and continuous random variables, and

does not require any *a priori* assumption on their distribution. The higher the dependence between vectors, the higher the dCor index (the maximum value is 1 for linearly related variables).

In view of its inherent robustness to redundancy and overfitting issues, the dCor has been studied for variable selection in regression problems [127], where it was employed in combination with incremental model building. When compared to the stepwise Akaike information criterion (AIC) or the Lasso method, it displays comparable or better performance even in the presence of nonlinear dependencies [127]. It has also been applied for feature screening purposes in ultrahigh-dimensional data [81], where it proved more effective than a classical screening procedure based on the classical Pearson's correlation coefficient.

In the rest of this Section, we provide a brief assessment of the dCor criterion, as of [115].

Let $\boldsymbol{x} = [x_1, \ldots, x_p]^T$ and $\boldsymbol{z} = [z_1, \ldots, z_q]^T$ be two random vectors, such that $\mathbb{E}(\|\boldsymbol{x}\| + \|\boldsymbol{z}\|) < \infty$, where $\|\cdot\|$ denotes the Euclidean norm. Let also $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(N)}$ be N i.i.d. realizations of $\boldsymbol{x}$, and $\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(N)}$ the corresponding i.i.d. realizations of $\boldsymbol{z}$. Now, the empirical distance covariance (briefly, dCov) is defined as

$$\nu_N^2(\boldsymbol{x}, \boldsymbol{z}) = \frac{1}{N^2} \sum_{k,\, l=1}^{N} A_{kl} B_{kl}, \tag{2.39}$$

where

$$\begin{aligned} A_{kl} &= a_{kl} - \bar{a}_{k\cdot} - \bar{a}_{\cdot l} + \bar{a}_{\cdot\cdot}, \\ B_{kl} &= b_{kl} - \bar{b}_{k\cdot} - \bar{b}_{\cdot l} + \bar{b}_{\cdot\cdot}, \end{aligned}$$

with

$$a_{kl} = \|\boldsymbol{x}^{(k)} - \boldsymbol{x}^{(l)}\|, \quad b_{kl} = \|\boldsymbol{z}^{(k)} - \boldsymbol{z}^{(l)}\|,$$

and

$$\begin{aligned} \bar{a}_{k\cdot} &= \frac{1}{N} \sum_{l=1}^{N} a_{kl}, \quad \bar{a}_{\cdot l} = \frac{1}{N} \sum_{k=1}^{N} a_{kl}, \quad \bar{a}_{\cdot\cdot} = \frac{1}{N^2} \sum_{k,\, l=1}^{N} a_{kl}, \\ \bar{b}_{k\cdot} &= \frac{1}{N} \sum_{l=1}^{N} b_{kl}, \quad \bar{b}_{\cdot l} = \frac{1}{N} \sum_{k=1}^{N} b_{kl}, \quad \bar{b}_{\cdot\cdot} = \frac{1}{N^2} \sum_{k,\, l=1}^{N} b_{kl}. \end{aligned}$$

Then, the empirical dCor is the square root of

$$\mathcal{R}_N^2(\boldsymbol{x},\boldsymbol{z}) = \begin{cases} \frac{\nu_N^2(\boldsymbol{x},\boldsymbol{z})}{\sqrt{\nu_N^2(\boldsymbol{x})\nu_N^2(\boldsymbol{z})}}, & \nu_N^2(\boldsymbol{x})\nu_N^2(\boldsymbol{z}) > 0 \\ 0, & \nu_N^2(\boldsymbol{x})\nu_N^2(\boldsymbol{z}) = 0 \end{cases} \tag{2.40}$$

In the assumption that $\mathbb{E}(\|\boldsymbol{x}\|+\|\boldsymbol{z}\|) < \infty$, it holds that the sampled version of the dCor tends to the corresponding probabilistic quantity, denoted $\mathcal{R}$:

$$\lim_{N\to\infty} \mathcal{R}_N^2(\boldsymbol{x},\boldsymbol{z}) = \mathcal{R}^2(\boldsymbol{x},\boldsymbol{z}). \tag{2.41}$$

It also holds that $0 \leq \mathcal{R}(\boldsymbol{x},\boldsymbol{z}) \leq 1$, and $\mathcal{R}(\boldsymbol{x},\boldsymbol{z}) = 0$ iff $\boldsymbol{x}$ and $\boldsymbol{z}$ are independent. Similarly, $0 \leq \mathcal{R}_N(\boldsymbol{x},\boldsymbol{z}) \leq 1$, and if $\mathcal{R}_N(\boldsymbol{x},\boldsymbol{z}) = 1$, then there exists a vector ζ, a nonzero real number τ and an orthogonal matrix $\mathbf{P}$ such that $\boldsymbol{z} = \zeta + \tau\boldsymbol{x}\mathbf{P}$.

In view of the last property, $\mathcal{R}_N(\boldsymbol{x},\boldsymbol{z})$ can be indeed used as a measure of linear dependence between random vectors. Fortunately, it can be verified that the proposed index is also sensitive to nonlinear input-output relationships.

2.8.1 The unbiased dCor index

It is worth mentioning that the bias of the dCor index increases with the dimension of the random vectors dimensions. As discussed in [114], for fixed number of samples N the dCor tends to 1 as $p, q \to \infty$. Thus, it might be hard to interpret the obtained index in high dimensional cases. This problem is investigated in [114] where an unbiased version of the dCor index is introduced, which is amenable for high dimensional problems. Here, the following quantities A_{kl}^* and B_{kl}^* are used instead of A_{kl} and B_{kl}:

$$A_{kl}^* = \begin{cases} \frac{N}{N-1}(A_{kl} - \frac{a_{kl}}{N}), k \neq l \\ \frac{N}{N-1}(\bar{a}_{k\cdot} - \bar{a}_{\cdot\cdot}), k = l \end{cases} \tag{2.42}$$

$$B_{kl}^* = \begin{cases} \frac{N}{N-1}(B_{kl} - \frac{b_{kl}}{N}), k \neq l \\ \frac{N}{N-1}(\bar{b}_{k\cdot} - \bar{b}_{\cdot\cdot}), k = l \end{cases} \tag{2.43}$$

Let

$$\mathcal{U}_N^*(\boldsymbol{x},\boldsymbol{z}) = \sum_{k\neq l}^{N} A_{kl}^* B_{kl}^* - \frac{2}{N-2}\sum_{k=1}^{N} A_{kk}^* B_{kk}^*. \tag{2.44}$$

The modified dCov and dCor indices are given respectively by:

$$\nu_N^*(\boldsymbol{x}, \boldsymbol{z}) = \frac{\mathcal{U}_N^*(\boldsymbol{x}, \boldsymbol{z})}{N(N-3)}, \tag{2.45}$$

$$\mathcal{R}_N^*(\boldsymbol{x}, \boldsymbol{z}) = \frac{\nu_N^*(\boldsymbol{x}, \boldsymbol{z})}{\sqrt{\nu_N^*(\boldsymbol{x})\nu_N^*(\boldsymbol{z})}}. \tag{2.46}$$

For simplicity, in the rest of this thesis we will drop the asterisk symbol and use the notation $\mathcal{R}_N$ to denote the unbiased dCor index.

2.8.2 dCor dependence test

The dCor index can be used to design a statistical test for assessing the dependence between two random vectors.
Let x and z be two random variables such that $\mathbb{E} = \|x\| + \|z\| < \infty$, where $\|\cdot\|$ denotes the absolute value. Let $H_0 : x$ *and* z *independent* be the null hypothesis.

Then, the statistical test proposed in [115] rejects H_0 if

$$\frac{N\,\nu_N^2(x, z)}{S} > \mathcal{N}^{-1}\Big(1 - \frac{\alpha_d}{2}\Big)^2, \tag{2.47}$$

where $\mathcal{N}(\cdot)$ denotes the normal cumulative distribution function, α_d is the significance level of the test, and

$$S = \bar{a}_{..}\bar{b}_{..}. \tag{2.48}$$

This test can be employed fruitfully in FS, by retaining only features for which there is enough statistical evidence to reject the independence hypothesis.

2.8.3 Sensitivity of the dCor to redundant terms

We next present some illustrative simulations that emphasize the robustness of the dCor index in the presence of redundant terms. Let $\boldsymbol{x} = [x_1, \ldots, x_6]^T$ be a random vector and $z = 3x_1$, and assume that N i.i.d. realizations of both $\boldsymbol{x}$ and z are available. All elements of the $\boldsymbol{x}$ vector are independently drawn from the same distribution. Table 2.1 reports the dCor value calculated for different subsets of inputs on average over 1000 Monte Carlo tests performed for data generated with different distributions (normal, Poisson and lognormal). The evaluated input subsets are $\{x_1, \ldots, x_{1+k_r}\}$, for $k_r = 0, \ldots, 5$, corresponding to the exact model and 5 redundant models with increasing number of redundant terms. While the dCor equals 1 for the

Table 2.1: *Average dCor measure over 1000 Monte Carlo tests for increasingly redundant models (true model: $z = 3x_1$).*

Number of redundant terms	Data distribution		
	Normal	Poisson	Lognormal
0	1.0000	1.0000	1.0000
1	0.9873	0.9835	0.9765
2	0.9778	0.9729	0.9573
3	0.9697	0.9640	0.9406
4	0.9623	0.9560	0.9262
5	0.9555	0.9488	0.9130

Table 2.2: *Average dCor measure over 1000 Monte Carlo tests for increasingly redundant models (true model: $z = 3x_1^2$).*

Number of redundant features	Data distribution		
	Normal	Poisson	Lognormal
0	0.5731	0.9682	0.9221
1	0.5447	0.9570	0.9106
2	0.5261	0.9490	0.8997
3	0.5120	0.9419	0.8897
4	0.5008	0.9352	0.8808
5	0.4916	0.9289	0.8716

true model (including only x_1), its value decreases as we introduce further terms, regardless of the distribution of the data.

A similar result holds even if the input-output relationship is nonlinear, *e.g.* $z = 3x_1^2$, although this time the dCor associated to the model containing only x_1 is less than 1: any further term added to the model decreases the dCor. The results are reported in Table 2.2.

Inspecting the results presented in Tables 2.1-2.2 leads to the conclusion that the dCor index is highly sensitive to the presence of redundant terms, and is maximal in the absence of redundant terms. This property proves to be crucial for the detection of redundant terms in the FS task.

CHAPTER 3

Randomized Selection Strategy

FS algorithms based on incremental model building techniques are extremely popular due to their simplicity. The main problem with stepwise subset selection algorithms is that the feature inclusion policy is based on local estimates of the term importance. Indeed, the objective function of choice (*e.g.*, an error index or a statistical index) measures the added benefit that a new term can provide if included in the current model. Therefore, the inclusion of a term in the model depends on the terms that have already been selected during previous iterations. As a consequence, it may happen that some of the terms selected during early iterations turn out to be irrelevant once the algorithm has identified the complete model structure [42]. Although there are ways to mitigate this issue, *e.g.* by introducing a pruning mechanism to eliminate redundant terms [99], or by iterating the method with different initializations (see the iOFR method discussed in [50]), they are not sufficient to prevent the algorithm from occasionally taking wrong search paths.

In the following we introduce a novel FS scheme, which overcomes this issue by determining the importance of terms based on a population of models rather than a single one. This aggregate information provides

a more robust estimation of the relevance of each term and guides the selection process more effectively. The method is based on a probabilistic reformulation of the selection problem, where a probability distribution is defined in the model space, and two operations are alternated to progressively refine it until convergence to a limit distribution representing a single model. The distribution is used first to generate sample models, which are used to evaluate the importance of model terms, and then updated based on the latter information, to reinforce the probability of extracting the most promising terms. The proposed randomized FS scheme is applicable to both frameworks discussed in this thesis.

A first application of this concept resulted in the RaMSS algorithm for the identification of NARX models [4], [13]. In this thesis, two novel classification algorithms (namely, the RFSC and the DCORF) have been developed based on the same selection scheme. The RFSC is a wrapper algorithm, where the classifier takes the form of a linear regression, and features are selected according to the mentioned randomized scheme based on the classifier performances. The DCORF is a multivariate filter method which will be discussed in Chapter 4.

3.1 Probabilistic formulation of the model structure selection problem

Let $\mathcal{R} = \{\varphi_1, \cdots, \varphi_{N_F}\}$ be the set of N_F possible model terms. As discussed in the previous chapter, these terms are polynomial functions of the features in a classification problem, or of the past input and output samples in an identification framework. In the simplest case (polynomial expansion of degree 1), they coincide with the original features or the past samples themselves. In this framework, with a slight abuse of notation, a model denotes a subset of $\mathcal{R}$. Accordingly, $\mathcal{S} = 2^{\mathcal{R}}$ is the set of all possible model structures. We will denote as $\mathcal{J}$ the criterion used to evaluate models.

Model structure selection is the problem of selecting the optimal subset of $\mathcal{R}$ according to the criterion $\mathcal{J}$. As such it is a combinatorial problem, in that the space of solutions contains $2^{N_F} - 1$ possible models (each of the N_F terms can either belong to a given model or not, and the empty model is neglected). The exponential complexity of this combinatorial problem makes it impossible to adopt exhaustive search methods to find the target model. A convenient solution approach involves its reformulation as an optimization problem over the probability distribution of model structures [4].

Let ϕ denote a discrete variable taking values in $\mathcal{S}$ according to a probability distribution $\mathcal{P}_\phi$. The performance of ϕ is also a random variable, and its expectation is given by

$$\mathbb{E}[\mathcal{J}(\phi)] = \sum_{s \in \mathcal{S}} \mathcal{J}(s)\mathcal{P}_\phi(s). \tag{3.1}$$

Index (3.1) is maximized when the probability mass concentrates on the model structure associated to the highest value of $\mathcal{J}$ (or one of the possible best model structures, if the minimum is not unique).

Therefore, the problem of finding the best $s \in \mathcal{S}$ can be formulated as follows

$$\mathcal{P}_\phi^\star = \arg\max_{\mathcal{P}_\phi} \mathbb{E}[\mathcal{J}(\phi)], \tag{3.2}$$

where $\mathcal{P}_\phi^\star$ is such that $\mathcal{P}_\phi^\star(s^\star) = 1$.

A convenient parametrization for $\mathcal{P}_\phi$ is obtained by associating a Bernoulli random variable ρ_j to each term φ_j, that models the probability that φ_j belongs to the target model:

$$\rho_j \sim Be(\mu_j), \tag{3.3}$$

$j = 1, \ldots, N_F$, where $\mu_j \in [0, 1]$. According to this representation, a model extraction from $\mathcal{P}_\phi$ implies testing each term for inclusion, by extracting a value from the respective Bernoullian distribution. Term φ_j is included if the outcome of the jth extraction is 1, and omitted in case of a 0. The former event has probability μ_j, whereas the probability of getting a 0 is given by $1 - \mu_j$. Accordingly, in the rest of the thesis we will denote μ_j as the *Term Inclusion Probability* (TIP) of the jth term, and define $\boldsymbol{\mu} = [\mu_1 \cdots \mu_{N_F}]^T$ as the vector of TIPs. For simplicity, we assume that all random variables ρ_j, $j = 1, \ldots, N_F$ are independent. In summary, the probability distribution $\mathcal{P}_\phi$ over the models in $\mathcal{S}$ can be written as:

$$\mathcal{P}_\phi(s) = \prod_{j:\varphi_j \in s} \mu_j \prod_{j:\varphi_j \notin s} (1 - \mu_j) \tag{3.4}$$

for any $s \in \mathcal{S}$. If all TIPs have values in $\{0, 1\}$ only, a limit distribution is obtained with all probability mass concentrated on a specific model $\tilde{s}$ (containing all the terms whose TIPs equal 1). In that case, it follows that $\mathcal{P}_\phi(\tilde{s}) = 1$. The objective of the model selection procedure will therefore be that of adapting the TIPs until convergence to the target limit distribution associated to an optimal model $s^\star$.

To evaluate the importance of a given term we consider an aggregate indicator $\mathcal{I}_j$ that compares the average performance of the models including

the jth term with that of the remaining ones:

$$\mathcal{I}_j = \mathbb{E}[\mathcal{J}(\phi)|\varphi_j \in \phi] - \mathbb{E}[\mathcal{J}(\phi)|\varphi_j \notin \phi], \tag{3.5}$$

where $j = 1, \ldots, N_F$. Thanks to the averaging over all models, indicator $\mathcal{I}_j$ can be interpreted as a global measure of the term importance. Additionally, an important property holds: if $\boldsymbol{\mu}$ is not distant from the target limit distribution corresponding to $s^\star$, it can be shown (see Theorem 1), that $\mathcal{I}_j > 0$ if and only if $\varphi \in s^\star$.

Theorem 1 ([4, Theorem 1])**.** *Let $\mathcal{P}_\phi$ be the probability distribution induced by $\boldsymbol{\mu}$, as indicated by* (3.4)*. Then there exists $\delta \in (0, 1)$ such that if $\mathcal{P}_{\tilde{\phi}}(s^\star) \geq \delta$ it holds that $\mathcal{I}_j > 0$ if $\varphi_j \in s^\star$ and $\mathcal{I}_j < 0$ if $\varphi_j \notin s^\star$, for all $j = 1, 2, \ldots, N_F$.*

3.2 Randomized model structure selection

Based on the probabilistic reformulation of the structure selection problem discussed in the previous section, we here describe an iterative optimization approach that operates on the model distribution $\mathcal{P}_\phi(s)$ with the aim of maximizing the average performance given by (3.1). MSS is performed by progressively refining the model distribution $\mathcal{P}_\phi$, until it converges to the target limit distribution. This is achieved by means of an iterative learning process, which alternates sampling and updating operations on the distribution. Sampling is used to gather information on the importance of individual model terms. The update step reflects the acquired knowledge by reinforcing the probability associated to important terms.

More in detail, at the beginning of each iteration, a set of models is extracted from the space of all possible model structures using the current Bernoullian distributions associated to terms in the pool of candidate. These models are individually estimated and evaluated. Then, the importance of each term is assessed with index $\mathcal{I}_j$, $j = 1, \ldots, N_F$, or rather a sampled version of it, denoted $\tilde{\mathcal{I}}_j$, based on the extracted models. Indeed, an exact evaluation of $\mathcal{I}_j$ is not practically feasible, since it would require an exhaustive approach on the model space. The robustness of this approximation, which is computed over a subset of the model space, clearly depends on the number of models N_p extracted at each iteration. On the other hand, one wants to limit the value N_p, for obvious computational reasons. Therefore, in order to mitigate the effects related to the uncertainty of this estimate, the new information is suitably averaged with the already available knowledge on the term importance gathered up to the current iteration,

using the following TIP update equation:

$$\mu_j(t+1) = \text{sat}(\mu_j(t) + \gamma\tilde{\mathcal{I}}_j) \tag{3.6}$$

for $j = 1, \ldots, N_F$, where $\text{sat}(x) = \min(\max(x, 0), 1)$ is a function that ensures that the calculated μ_j values will not exceed the interval $[0, 1]$, and γ is a step-size parameter. The value of the latter parameter is adapted at each iteration as well:

$$\gamma = \frac{1}{10(\mathcal{J}_{max} - \bar{\mathcal{J}}) + 0.1}, \tag{3.7}$$

where $\mathcal{J}_{max}$ and $\bar{\mathcal{J}}$ are the maximum and average of the objective function values among the models extracted at the current iteration, respectively. In practice, the larger the variance of the model performances, the smaller the step-size, indicating a lower level of reliability of the computed global measure of the term importance $\mathcal{I}_j$.

The procedure is iterated as long as the TIPs continue evolving and stops upon reaching a limit distribution.

A sketch of the basic loop of the proposed randomized model selection procedure is presented below as Algorithm 1.

Algorithm 1 Randomized Model Selection

Input: $\mathcal{R}$, N_p, α, $\boldsymbol{\mu}_{init}$, ϵ, N_i
Output: $\boldsymbol{\mu}$

```
μ(1) ← μ_init
for t = 1 to N_i do
    for n_p = 1 to N_p do
        Generate random model structure s ∈ S
        Evaluate performance J of s
    end for
    for j = 1 to N_F do
        Evaluate importance of jth term with Ĩ_j
        Update jth TIP according to (3.6)
    end for
    if max_{j=1,...,N_F} |μ_j(t+1) − μ_j(t)| ≤ ε then
        Break
    end if
end for
```

3.3 The RaMSS algorithm

A first application of the randomized model structure selection procedure described in the previous sections has been targeted at the identification of

NARX models [4], and resulted in the RaMSS algorithm. Here the nonlinear dynamic model is expressed as a linear regression of nonlinear monomials of the past input and output samples:

$$y(k) = \boldsymbol{\varphi}(k)^T \boldsymbol{\vartheta}. \tag{3.8}$$

The model terms (usually referred to as regressors) are the elements of vector $\boldsymbol{\varphi}(k) = [\varphi_1(k),\ \ldots,\ \varphi_{N_F}(k)]^T$, while $\boldsymbol{\vartheta} = [\vartheta_1,\ \ldots,\ \vartheta_{N_F}]^T$ is the parameter vector. As already discussed, the regressors are monomials in $y(k-1), \ldots, y(k-n_y)$, $u(k-1), \ldots, u(k-n_u)$.

After the extraction of a model structure from the Bernoullian distributions, the model parameters are estimated (with LS), and the model performance is rated according to performance index (2.34). To avoid distorting the selection process due to over-parametrization issues, a statistical test (see Section 2.6.3) is used to remove redundant terms after the parameter estimation phase. The rejection of redundant terms is a crucial step in the identification procedure as it eliminates statistically non-significant regressors. For this purpose, the statistical significance of each estimated parameter is calculated with a Students t-test so that the parameters statistically indistinguishable from 0 are ruled out. If any model terms are removed at this stage, the model parameters are re-estimated prior to model evaluation.

The interested reader is addressed to [4] for further details on the algorithm and the setting of its parameters.

A variant of the RaMSS algorithm, denoted C-RaMSS (for *Correlated*-RaMSS), was proposed in [13]. Unlike the original RaMSS which is based on an independence assumption between regressors, the C-RaMSS includes second-order information, *i.e.* it accounts for the correlation between regressors. Extracting samples from a real multivariate Bernoulli distribution (in the standard RaMSS multiple independent univariate Bernoulli distributions are used instead) is by no means an easy task, and special tools are required. Also, the method requires an additional update equation for the covariance (or correlation) matrix of the model terms. Using the covariance matrix allows the algorithm to take into account well-matched pairs of terms as well, and not only promising individual terms. Overall, it typically boosts the convergence process, and sometimes can even improve its robustness and reliability. However, this comes at some cost in terms of computation effort and for this reason has not been explored further in this work. The interested reader is referred to [13] for additional details.

3.4 The RFSC algorithm

The explained randomized model selection scheme was also used to develop a feature selection method for classification problems (RFSC [20]). The RFSC (for Randomized FS and Classification) is a wrapper method where each feature subset is evaluated in terms of the corresponding classifier performance, through index (2.35), *i.e.* the correct classification rate. The classifier is defined as a linear regression with respect to the augmented features (*i.e.*, monomials of the original features), according to (2.6). Its parameters are estimated using the logistic regression approach (see Section 2.6.2), and similarly to what done for NARX models, a statistical test is employed to remove possible redundant terms, as explained in Section 2.6.3 (if any redundant terms are detected, they are eliminated and the parameters re-estimated prior to evaluation).

A sketch of the basic loop of the proposed RFSC procedure is presented below as Algorithm 2.

Algorithm 2 RFSC

Input: $\{\boldsymbol{u}(k), y(k)\}$, $\mathcal{R}$, N_p, α, $\boldsymbol{\mu}_{init}$, ϵ, N_i
Output: $\boldsymbol{\mu}$

```
μ(1) ← μ_init
for t = 1 to N_i do
    for n_p = 1 to N_p do
        Generate random model structure s ∈ S
        Estimate parameter vector ϑ by solving (2.26)
        Compute σ̂_j^2 according to (2.31)
        Apply the statistical test according to (2.30)
        Remove redundant terms
        Re-estimate parameter vector ϑ
        Evaluate model performance according to (2.35)
    end for
    for j = 1 to N_F do
        Evaluate importance of jth term with Ĩ_j
        Update jth TIP according to (3.6)
    end for
    if max_{j=1,...,N_F} |μ_j(t+1) − μ_j(t)| ≤ ε then
        Break
    end if
end for
```

A high-dimensional feature space can hamper FS algorithms by slowing down the search process and by increasing the chances of getting stuck in local minima. To tackle this issue a common approach is to perform a pre-

filtering of the feature space. Specifically, it would be desirable to identify those features that are relevant in describing the output, and those which are not. We address this problem as an optional procedure executed before Algorithm 2 which analyzes the dependence of the output on each feature. The rationale is as follows: if a feature u_j is not important in the description of the output y_i, then we expect y_i and u_j to be independent. The reader should note that at this point we are just interested in characterizing the dependence/independence of the output from a specific feature, not the strength nor the "shape" of such dependence, tasks that are performed during the FS process.

Algorithm 3 Feature set preprocessing for a class i.

Input: $\{\mathbf{u}(k), y_i(k)\}$, $\mathcal{F} = \{u_1, \dots, u_{N_f}\}$, α_d
Output: $\tilde{\mathcal{F}}^i$

$\tilde{\mathcal{F}}^i \leftarrow \mathcal{F}$
for $j = 1$ **to** N_f **do**
  $H_0^j \leftarrow \mathbf{true}$
  $\mathbf{x} \leftarrow [u_j(1) \cdots u_j(N)]^T$
  $\mathbf{z} \leftarrow [y_i(1) \cdots y_i(N)]^T$
  Compute $\nu_N^2(\mathbf{x}, \mathbf{z})$ as in (2.39)
  Compute S as in (2.48)
  if $N\nu_N^2(\mathbf{x}, \mathbf{z})/S > \mathcal{N}^{-1}(1 - \alpha_d/2)^2$ **then**
    $H_0^j \leftarrow \mathbf{false}$
  end if
  if H_0^j **then**
    $\tilde{\mathcal{F}}^i \leftarrow \tilde{\mathcal{F}}^i \setminus \{u_j\}$
  end if
end for

There exist various statistical tests designed to assess the dependence between two random vectors. We here employ the one described in Section 2.8.2 based on the dCor which is very flexible and can handle both discrete and continuous random vectors, without any assumption on their distributions, making it particularly amenable for classification purposes. More in detail, let x and z be two random variables such that $\mathbb{E}[|x|+|z|] < \infty$, where $|\cdot|$ denotes the absolute value. In our case we have $x = u_j$ and $z = y_i$ for any i and j. We want to test the null hypothesis $H_0 : x$ *and* z *independent*. Let $\mathbf{x} = [u_j(1) \cdots u_j(N)]^T$ be a vector of i.i.d. realizations of x, and $\mathbf{z} = [y_i(1) \cdots y_i(N)]^T$ the corresponding realizations of z. The statistical test rejects H_0 if inequality (2.47) holds. For each class i, inequality (2.47) is tested for all j, and only those features u_j for which there is enough statistical evidence to reject the independence hypothesis are considered in the

FS process for determining the classifier $\hat{y}_i$. The prefiltering procedure, is summarized in Algorithm 3.

3.5 Analysis of the RSFC algorithm

This section illustrates various experiments carried out to assess the performance of the proposed RFSC algorithm. Eight numerical datasets from the UCI machine learning repository [91] have been analyzed. The main features of the selected datasets are given in Table 3.1.

All the input data in the original feature sets have been normalized in the $[0, 1]$ range according to:

$$u_p(k) = \frac{u_{p,raw}(k) - u_{p_{min}}}{u_{p_{max}} - u_{p_{min}}}, \tag{3.9}$$

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

Table 3.1: *Main characteristics of the used datasets, [91].*

Dataset name	No. of samples	No. of features	Feature types Real	Integer	No. of classes
Bupa	345	6	1	5	2
HillValley	606	0	100	0	2
Ionosphere	351	34	32	1	2
Iris	150	4	4	0	3
Musk1	476	166	0	166	2
Sonar	208	60	60	0	2
WDBC	569	30	13	0	2
Wine	178	13	13	0	3

The classification performance has been evaluated using the 10-fold cross validation (10-FCV) approach as explained in Section 2.4. The algorithm performance is computed as the average over ten independent runs. Given the randomized nature of the RFSC, different results may be obtained on each run, especially on datasets with large feature sets, for which full exploration may be too costly. For this reason, the application of the RFSC on each fold is repeated 10 times and the best model retained.
To obtain results more directly comparable to the literature, the performance on the HillValley and Musk1 datasets is evaluated with the hold out method. The algorithm performance is computed as the average of 10

independent runs with a random data division of the training-testing pairs.

Regarding the initial parameter setup for the RFSC, the number of iterations was set to $N_i = 300$, the maximum nonlinearity degree to $L = 2$, the number of generated models to $N_p = 100$, the significance confidence interval to $\alpha = 0.99$ and all initial TIPs to $1/N_F$. Parameter α also influences the average model size, by acting on the threshold for the rejection of redundant terms. The closer α is to 1, the more terms are rejected (and greater is the confidence that only meaningful terms are retained), and the smaller is the average model size. Parameter ϵ in the termination condition has been set to $\epsilon = 0.002$. Finally, the significance level α_d for the statistical test based on the dCor in the prefiltering phase was set to 0.99 for the HillValley, Ionosphere, and Musk1 databases, to 0.87 for the Sonar database, and to 0.9999 for the WBCD database. The proposed algorithm was implemented in Matlab (version 2012b) and executed on an Intel(R) Core i7-3630QM machine, with 2.4GHz CPU, 8GB of RAM, and a 64-bit Operating System.

3.5.1 An illustrative example

To get a greater insight in the mechanisms of the selection process, we here illustrate the RFSC behavior with reference to the WDBC dataset, which has 30 attributes and 2 class labels. Assuming a maximum nonlinearity degree of $L = 2$, the total number of extended features is $N_F = 496$. We will focus on two independent runs of the RFSC algorithm. Both runs returned a 7-terms model (denoted Model 1 and Model 2) with no common model terms and only one common feature. We refer to the terms of the returned models as "final" terms. It is worth mentioning that despite their different structure, Model 1 and Model 2 both exhibit 0 classification errors on the validation dataset.

Figures 3.1-3.2 (top) show the evolution of the TIP values for both runs. In both cases various terms are initially considered promising and their TIPs increased. In the first run (Fig. 3.1, top) the TIPs of the final terms keep increasing from the very first iterations, whereas the other terms are progressively discarded as the algorithm progresses. On the other hand, in the second run (Fig. 3.2, top) most terms are selected or discarded in the first 25 iterations, but the last term is selected at a later stage (around iteration 40), essentially after two other terms have been rejected. Before final convergence, other terms are tested but ultimately discarded.

It is interesting to note that in both cases some terms are initially se-

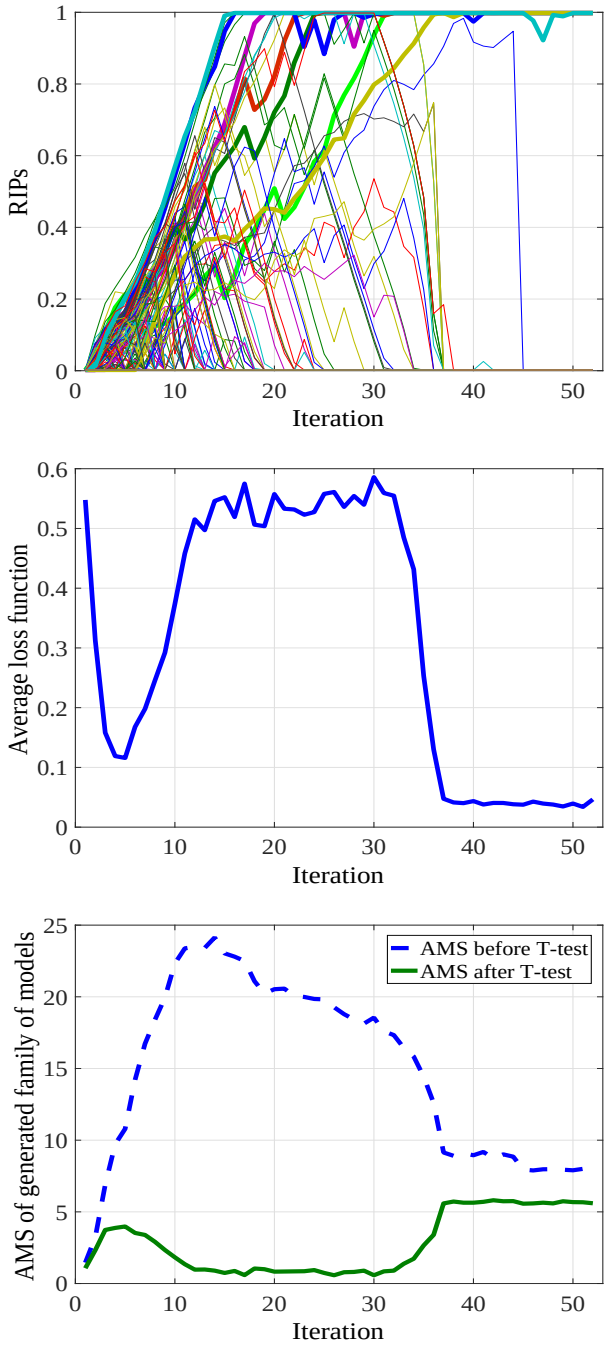

Figure 3.1: *Model 1: Evolution of the TIPs during the selection process (top, thicker lines indicate the terms contained in the final model), average loss function (middle), average model size (bottom) before (dashed) and after (solid) the t-test.*

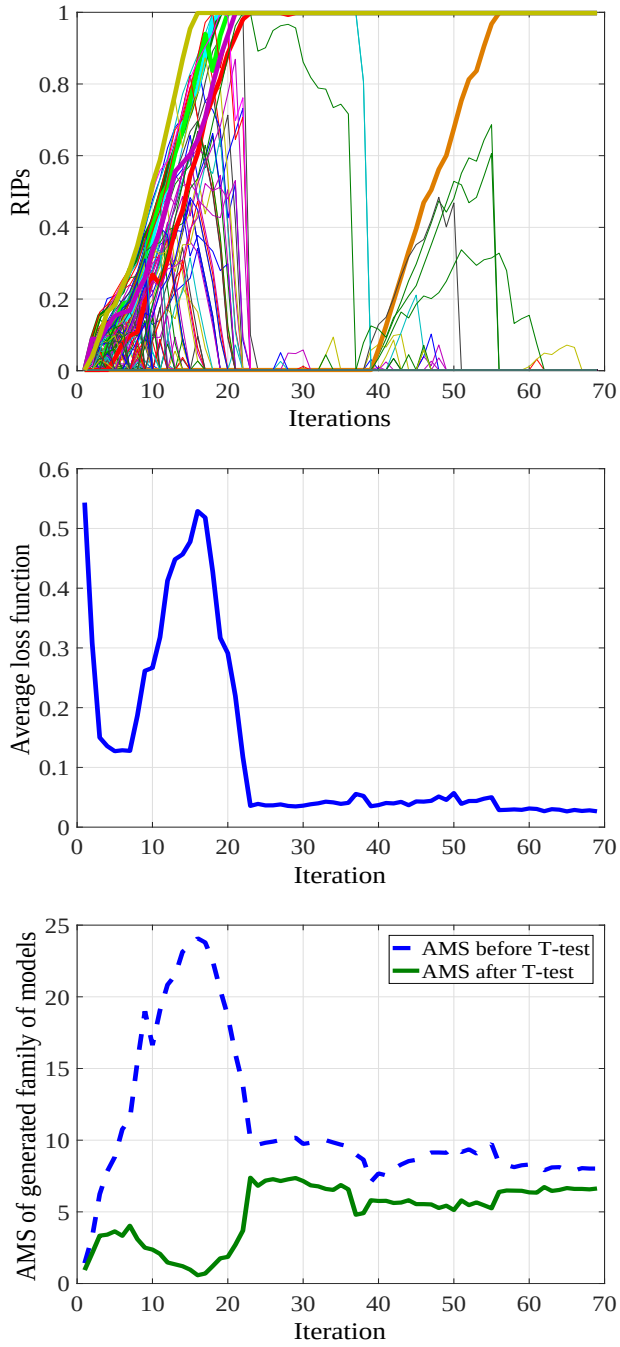

Figure 3.2: *Model 2: Evolution of the TIPs during the selection process (top, thicker lines indicate the terms contained in the final model), average loss function (middle), average model size (bottom) before (dashed) and after (solid) the t-test.*

lected, to the point that their TIPs rise to 1, but are subsequently rejected in favor of other terms. If we compare (column-wise) this behavior of the TIPs with the evolution of the average loss function (average value of the loss function of the N_p extracted models at a given iteration) in Figures 3.1-3.2 (middle), it is clear that the algorithm is exploring model structures with a higher average loss function in order to ultimately escape from structures that represent only local minima.

Figures 3.1-3.2 (bottom) show the average model size (AMS) at each iteration for both runs. For Model 1, the AMS of the generated models (measured before the application of the statistical test) grows rapidly in the beginning and starts decreasing significantly only after iteration 10. Later on, after iteration 38, the model size does not change significantly. On the other hand, the AMS measured after the statistical test is very low from iterations 10 to 30, indicating that the algorithm is systematically rejecting tentative terms as redundant. It is only between iterations 30 to 40 (*i.e.*, when the final two terms have been added), that the model size converges to its final value. Similarly, for Model 2 the AMS before the t-test increases at the beginning, reaching a peak around iteration 15, and then stabilizes after iteration 20.

Notice that in both runs the AMS value is always reduced after the test, indicating the effectiveness of the latter in detecting redundant terms.

3.5.2 Interpretation of the results

As previously stated, all input data points $u_p(k)$, with $p = 1, \ldots, N_f$ and $k = 1, \ldots, N$, have been normalized to be in the $[0, 1]$ interval. Since each term $\varphi_j(k)$ is constructed as a product of features, $\varphi_j(k)$ takes values in $[0, 1]$ as well, for all $j = 1, \ldots, N_F$ and $k = 1, \ldots, N$.

Now, the estimated model is of the form (2.6), where only the selected terms are associated to non-zero parameters. The predicted class for the kth observation is given only by the sign of $\hat{y}_i$, while the absolute value of $\hat{y}_i$ is related to the reliability of the prediction. Since $\varphi_j(k)$ is non-negative, the information about the sign is carried by the coefficients $\vartheta^{(i)}$ of the linear combination in (2.6). Therefore, the model can be decomposed in two additive components based simply on the sign of the parameters:

$$\hat{y}_i(k) = \hat{y}_i^+(k) - \hat{y}_i^-(k) = \boldsymbol{\varphi}(k)_+^T \vartheta_+^{(i)} - \boldsymbol{\varphi}(k)_-^T(-\vartheta_-^{(i)}), \tag{3.10}$$

where the first component $\hat{y}_i^+(k) = \boldsymbol{\varphi}(k)_+^T \vartheta_+^{(i)}$ is associated to terms with positive coefficients and the second one $\hat{y}_i^-(k) = \boldsymbol{\varphi}(k)_-^T(-\vartheta_-^{(i)})$ to terms with negative coefficients. This decomposition has the following very nice

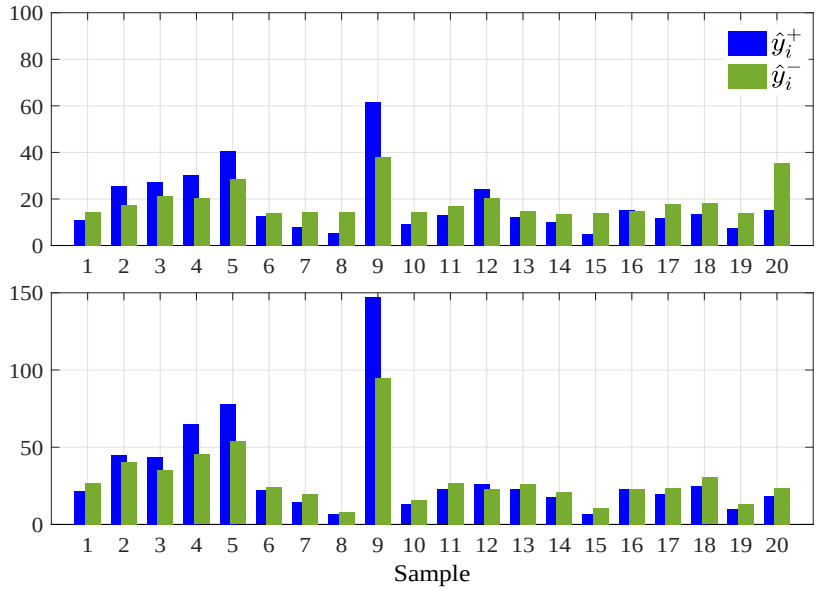

Figure 3.3: *Values of $\hat{y}_i^+(k)$ and $\hat{y}_i^-(k)$ for 20 validation samples: Model 1 (top) and Model 2 (bottom).*

and clear interpretation: features which appear in model terms inside $\hat{y}_i^+(k)$ are representative for class i, whereas features appearing in $\hat{y}_i^-(k)$ are against class i. The "strongest" group of (extended) features in the ith model determines the sign of $\hat{y}_i$, and therefore if the predicted class should be class i or not. If multiple classes exhibit a positive $\hat{y}_i$, then the class is determined by the most "confident" classifier, *i.e* the one with the largest difference between $\hat{y}_i^+(k)$ and $\hat{y}_i^-(k)$.

In Figure 3.3, we report the values of the two quantities $\hat{y}_i^+(k)$ and $\hat{y}_i^-(k)$ for 20 validation data points. The two plots in Figure 3.3 (top and bottom) refer to the final models of the two runs of the RFSC algorithm analyzed in the previous section. Both models exhibit 0 classification errors on the test set (56 samples).

From Figure 3.3, it is also apparent that despite the fact that both models exhibit 0 classification errors, they are not equivalent in terms of reliability. In particular, the value of $\delta_i(k) = (\hat{y}_i^+(k) - \hat{y}_i^-(k))/\max(\hat{y}_i^+(k), \hat{y}_i^-(k))$ can be interpreted as the "confidence" the model has in attributing class i to the kth sample. Apparently, Model 1 has generally greater values of δ_i. This difference is not currently captured by the performance index (2.35), and

therefore the two models are considered equivalent for the RFSC algorithm.

To conclude the analysis of the results, we report in Table 3.2 the average size of the final model structures obtained by the 10-FCV procedure. Specifically, Table 3.2 displays the number of original features N_f, the number of features after the $dCor$-based prefiltering $N_f^\star$, the average number of features n_f used by the classifier over the 10 folds, the number of terms N_F generated based on the original features, the number of terms $N_F^\star$ generated based on the pre-filtered features, the average number of terms n_F used by the classifier over the 10 folds[1]. In the non-binary classification problems (Iris and Wine datasets), a separate modeling is carried out for each class. In those cases, the classifier size (in terms of the number of original and extended features used) is calculated by performing the union over the individual class models $\hat{y}_i$, $i = 1, \ldots, N_c$.

By inspecting Table 3.2, it is noticeable that while the RFSC algorithm processes a considerable fraction of the available features, it generally requires only a small number of model terms, demonstrating its capability of compressing the information.

Table 3.2: *Average size of the obtained classifiers over the* 10 *folds.*

Dataset	N_f	$N_f^\star$	n_f	N_F	$N_F^\star$	n_F
Bupa	6	–	5.8	28	–	7.4
HillValley	100	100	8.3	5151	5151	3.7
Ionosphere	34	29	16.4	595	465	14.7
Iris	4	–	3.2	15	–	6.1
Musk1	166	165	46.2	14028	13860	23.2
Sonar	60	39	25.8	1891	820	18.7
WDBC	30	24	11.5	496	325	10.3
Wine	13	–	7.3	105	–	7.5

3.5.3 Comparative analysis

To assess the performance of the RFSC algorithm, we report in this section an extensive comparison with the results documented in [125], [112], [55], [23], [97], [83], on the datasets in Table 3.1. The comparison is carried out in terms of the average classification accuracy $\mathcal{J}_a$, the average Kappa rate K_a, and the average model size. The performance comparison is summarized in Tables 3.3-3.4 and the size comparison in Table 3.5.

The RFSC outperformed all other documented results on the Bupa, HillValley and WDBC datasets, both in terms of average accuracy and average Kappa rate. This has been achieved at the cost of using more attributes

[1] A dash in the $N_f^\star$ and $N_F^\star$ columns indicates that the pre-filtering stage has not been applied

compared to the other methods. On the other hand, the proposed algorithm was only slightly outperformed by the best competitor (which is different from case to case) on the remaining datasets, providing overall a good tradeoff between model complexity and accuracy. Considering more recent approaches such as [112], [55], and [97], the proposed RFSC dominates on 4 out of 8 tested benchmarks, and is only slightly outperformed in the other cases.

3.5.4 Computational time

A comparative analysis in terms of computational time is finally presented in Table 3.6. Though inherently time consuming due to the model exploration mechanism in the randomized MSS process, the RFSC achieves convergence in comparable time with competitor algorithms. Indeed, it outperforms the PSO4-2 method for the Wine and WDBC datasets, but is generally somewhat slower than PSO+LDA. In this respect, it is important to note that non-optimized Matlab code has been used to obtain the documented results, so that the reported figures must be considered gross upper bounds. Still, the computational time goes from a few seconds to a little more than a minute for all analyzed datasets.

In order to analyze the algorithm's computational effort as a function of the problem size, we report in Fig. 3.4 the elapsed time versus the number of extended features for the WDBC dataset. More precisely, the curves in Fig. 3.4 show, for different values of the maximum number of iterations N_i, the average computational time associated to the algorithm calculated over ten different subsets of extended features of a given size (drawn at random). The four curves are characterized by an initial increase of the computational time with the increase of the problem size, followed by a saturation. The latter indicates that all simulations exhaust the available number of iterations above a certain problem size. Notice that the initial TIPs are defined so as to result in the same initial AMS for any problem size, so that it is expected that the computational load of the algorithm is essentially independent of the problem size for a given N_i. As N_i increases, the saturation point shifts, indicating that early convergence is sometimes achieved.

Table 3.3: *Comparative performance analysis in terms of accuracy.*

FS Method + Classifier	Ref.	Bupa	HillValley	Ionosphere	Iris	Musk1	Sonar	WBCD	Wine
ACO + PMC	[112]	0.6725	-	0.9373	0.9600	–	0.9087	–	0.9755
Att. WV + DGC	[23]	0.6744	-	0.9311	0.9533	-	0.8487	0.9619	0.9731
Att. WV + DGC	[23]	0.6525		0.6724	0.9533	-	0.7694	-	0.9706
- + KNN	[23]	0.6066	-	0.8519	0.9400	-	0.8307	-	0.9549
- + KNN-A	[23]	0.6257	-	0.9372	0.9533	-	0.8798	-	0.9663
- + DW-KNN	[23]	0.6376	-	0.8747	0.9400	-	0.8648	-	0.9438
- + Cam-NN	[23]	0.5962	-	0.7379	0.9467	-	0.7743	-	0.9497
- + CNN	[23]	0.6316	-	0.8917	0.9267	-	0.8940	-	0.9663
SSMA + SFLDS	[23]	0.6426	-	0.9088	0.9533	-	0.8079	-	0.9438
forward FS + LDA	[83]	0.6110	-	0.8530	0.9630	-	0.7610	-	0.9660
backward FS + LDA	[83]	0.6430	-	0.9090	0.9370	-	0.8550	-	0.9990
PSO + LDA	[83]	0.6520	-	0.9220	0.9700	-	0.9050	-	1.000
PSO(4-2) + 5NN	[125]	-	0.5777	0.8727	-	0.8494	0.7816	0.9398	0.9526
FFW-DGC	[97]	-	-	0.9461	0.9667	-	0.9173	0.9525	0.9831
MDis ABC	[55]	-	0.5508	-	-	0.8529	-	-	-
DCF + RFSC		0.7800	0.9277	0.9330	0.9666	0.8132	0.8806	0.9827	0.9944

Table 3.4: *Comparative performance analysis in terms of Kappa rate.*

FS Method + Classifier	Ref.	Bupa	HillValley	Ionosphere	Iris	Musk1	Sonar	WBCD	Wine
ACO + PMC	[112]	0.3259	-	0.8604	0.9400	-	0.8164	-	0.9659
Att. WV + DGC	[23]	0.3076	-	0.8487	0.9300	-	0.6943	-	0.9590
Att. WV + DGC	[23]	0.2220	-	0.1142	0.9300	-	0.5187	0.9619	0.9552
- + KNN	[23]	0.1944	-	0.6494	0.9100	-	0.6554	-	0.9318
- + KNN-A	[23]	0.2021	-	0.8595	0.9300	-	0.7549	-	0.9491
- + DW-KNN	[23]	0.2645	-	0.7083	0.9100	-	0.7248	-	0.9152
- + Cam-NN	[23]	0.1024	-	0.5145	0.9200	-	0.5364	-	0.9228
- + CNN	[23]	0.2571	-	0.7526	0.8900	-	0.7861	-	0.9491
SSMA + SFLDS	[23]	0.2731	-	0.7986	0.9300	-	0.6100	-	0.9145
PSO(4-2) + 5NN	[125]	-	-	-	-	-	-	0.9398	-
FFW-DGC	[97]	-	-	0.8958	0.9500	-	0.6252	0.8984	0.9745
MDis ABC	[55]	-	-	-	-	-	-		-
DCF + RFSC		0.4950	0.8552	0.8541	0.9500	0.6201	0.8101	0.9621	0.9916

Table 3.5: *Comparative model size analysis.*

FS Method + Classifier	Ref.	Bupa	HillValley	Ionosphere	Iris	Musk1	Sonar	WDBC	Wine
FW FS + LDA	[83]	3.6	-	4.8	2.3	-	10.7	-	7.1
BW FS + LDA	[83]	4.7	-	30.4	3.9	-	56.4	-	12.8
PSO + LDA	[83]	4.6	-	21.7	3.6	-	38.1	-	12.3
PSO(4-2) + 5NN	[125]	-	12.22	3.26	-	76.54	11.24	3.46	6.84
MDis ABC	[55]	-	30.53	5.76	-	75.76	-	11.86	5.76
DCF + RFSC		5.8	8.3	16.4	3.1	46.2	25.8	11.5	3.3

Table 3.6: *Computation time* $[s]$.

FS Method + Classifier	Ref.	Bupa	HillValley	Ionosphere	Iris	Musk1	Sonar	WDBC	Wine
PSO(4-2) + 5NN	[125]	-	1210.2	61.8	-	620.4	32.4	172.8	18.6
PSO + LDA	[83]	-	-	27.6	-	-	36.6	-	5.4
DCF + RFSC (Avrg.)		14.6	18.6	57	10.01	51.6	72	66	12

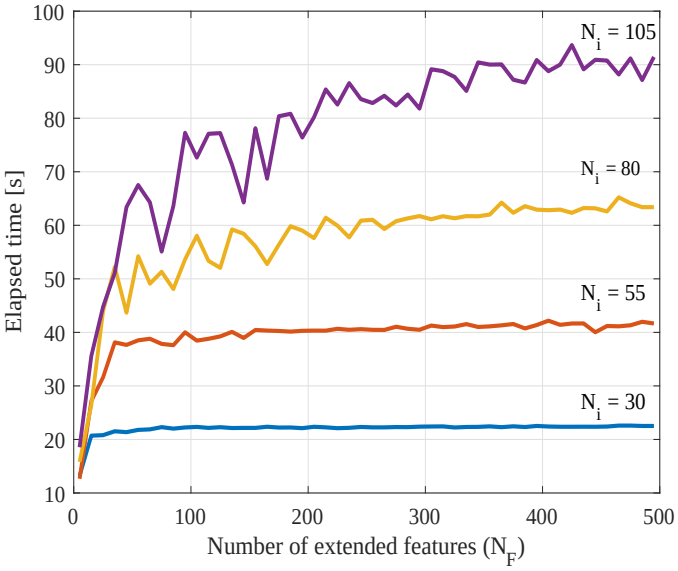

Figure 3.4: *Computational time of the RFSC algorithm for the WDBC dataset with* $N_i = [30\ 55\ 80\ 105]$.

3.6 RFSC application to Big Data problems

Nowadays, the implementation of Big Data applications increases constantly [66]. Apache Hadoop is a software-framework based on the MapReduce (MR) parallel programming model, which is used for the distributed storage and processing of Big Data. MR has become the most widespread programming model for processing and generating large datasets [34] due to its simplicity, maturity and generality [128]. The MR framework is based on two stages, i.e., map and reduce. The map job, takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs). The reduce job takes the output from a map as input and combines those data tuples into a smaller set of tuples. Its extensions, $e.g.$ Tez or Spark, split a complex application into a Directed Acyclic Graph (DAG) of stages so that all tasks are performed on corresponding input data partitions. The MR/Tez/Spark routine is repeated constantly in the following order: (i) read the input data set or data from a previous stage, (ii) execute an operation in parallel, (iii) send intermediate results to the next stage or write the final results on the cluster distributed file system (usually HDFS [130] or a cloud data storage, e.g., Amazon S3). Among the above mentioned technologies, the Spark framework is the most

promising today since it is able to cache in memory intermediate results and implements multiple transformations in a single step, reducing the disk and network I/O required by copy/shuffle operations. It reads data from the cluster in the beginning and once all required operations are completed it writes the final result on the cluster. Compared to Hadoop it can speed up by a factor 3x to 100x on certain scenarios.
In this context, one of the main challenges is the prediction of the execution time of Big Data applications, which is essential in providing and guaranteeing the required Quality of Service (QoS). Usually, it is done empirically through experimental tests, but this turns out to be very costly [46]. A convenient way to address this problem is by developing a predictive model.
In the literature there are two main approaches addressing the modeling of Big Data applications, namely *Analytical Models* (AMs) [6] and *Hybrid Models* (HMs) ([7], [36], [65]). AMs can be based on Queuing Networks, Petri Nets and Stochastic Activity networks. However, modelling is difficult especially in environments where resources are dynamically allocated ($e.g.$ Hadoop 2.x). Besides, resource contention and performance degradation in cloud applications, introduced by the underlying virtualization layer, might induce variations in the application execution time [24], adding additional complexity to the modeling task. HMs exploit the benefits of AM and Machine Learning (ML). More in detail, AM are characterized by good *extrapolation capabilities*, $i.e.$ good generalization properties, while ML by good *interpolation capabilities*, $i.e.$ good predictive capabilities for regions of the feature space learned on sufficiently big number of training samples. Here we present a modeling approach fully based on a ML approach. More specifically, the RFSC algorithm was employed to learn the model for predicting an *application completion time*. The original RFSC was adjusted to regression problems so that the MSE given by Eq. 2.20 is employed as criterion function for the model evaluation. The method is tested on both the Hadoop 2.x and the Spark frameworks.

3.6.1 Performance analysis on Hadoop 2.x

This section reports the analysis carried out to assess the performance of the RFSC algorithm with application to the Hadoop 2.x environment. More precisely, the data are generated on PICO [2], a Big Data cluster available at CINECA, composed of 74 nodes, each of them boasting two Intel Xeon 10- core 2670 v2@2.5GHz, with 128 GB RAM per node. Out of these 74 nodes, up to 66 are available for computation. In our experiments on PICO,

[2] http://www.hpc.cineca.it/hardware/pico

we used several configurations ranging from 40 to 120 cores and set up the scheduler to provide one container per core.

The cluster is shared among different users, and the resources are managed by the Portable Batch System (PBS). PBS Professional allows to submit jobs and check their progress, configuring at a fine-grained level the computational requirements. For each submission it is possible to request a number of nodes and to define how many CPUs and what amount of memory are needed. Since the cluster is shared among different users, the performance of single jobs depends on the overall system load, even though the PBS tries to split the resources. Due to this, it is possible to have large variations in performance according to the total usage of the cluster. In particular, the storage is not handled directly by the PBS, thus leading to an even greater impact on performance. To mitigate the variability, first of all entire nodes of the cluster are requested for the execution of our experiments. This ensures that nobody else could run other jobs on the same nodes, thus interfering with the performance measurement. An ephemeral Hadoop cluster was created at the beginning of each experiment on the allocated nodes. The PICO cluster provides the myHadoop tool for setting up a Hadoop 2.5.1 cluster, upon which we used Hive 1.2.1. HDFS is kept locally on the selected nodes, in order to experience lower variability than the one observed using the centralized storage. In spite of these settings, the experiments still showed high variability, with a few runs characterized by an extremely high execution time. To further reduce this variability, in our analyses we discarded runs with an anomalous execution time, taking out all the experiments that lie more than three standard deviations away from the average computed for the same configuration. The dataset used for testing has been generated using the TPC-DS benchmark [3] data generator, that creates several files ranging from 250 GB to 1 TB, directly used as external tables by Hive. We chose the TPC-DS benchmark as it is the industry standard for benchmarking data warehouses. Furthermore, we performed experiments on three Hive queries, dubbed $R1$, $R3$ and $R4$. These are ad hoc queries that Hive runs as a single MapReduce job. The profiling phase has been conducted by extracting average task durations from at least twenty runs of each query. The numbers of map and reduce tasks varied, respectively, in the ranges [2, 1560] and [2, 1009].

The main characteristics of the queries, in terms of data size, are presented in Tab. 3.7. N_f denotes the size of the original feature set, N_F is the size of the extended feature set and N is the total number of available samples.

[3] http://www.tpc.org/tpcds/

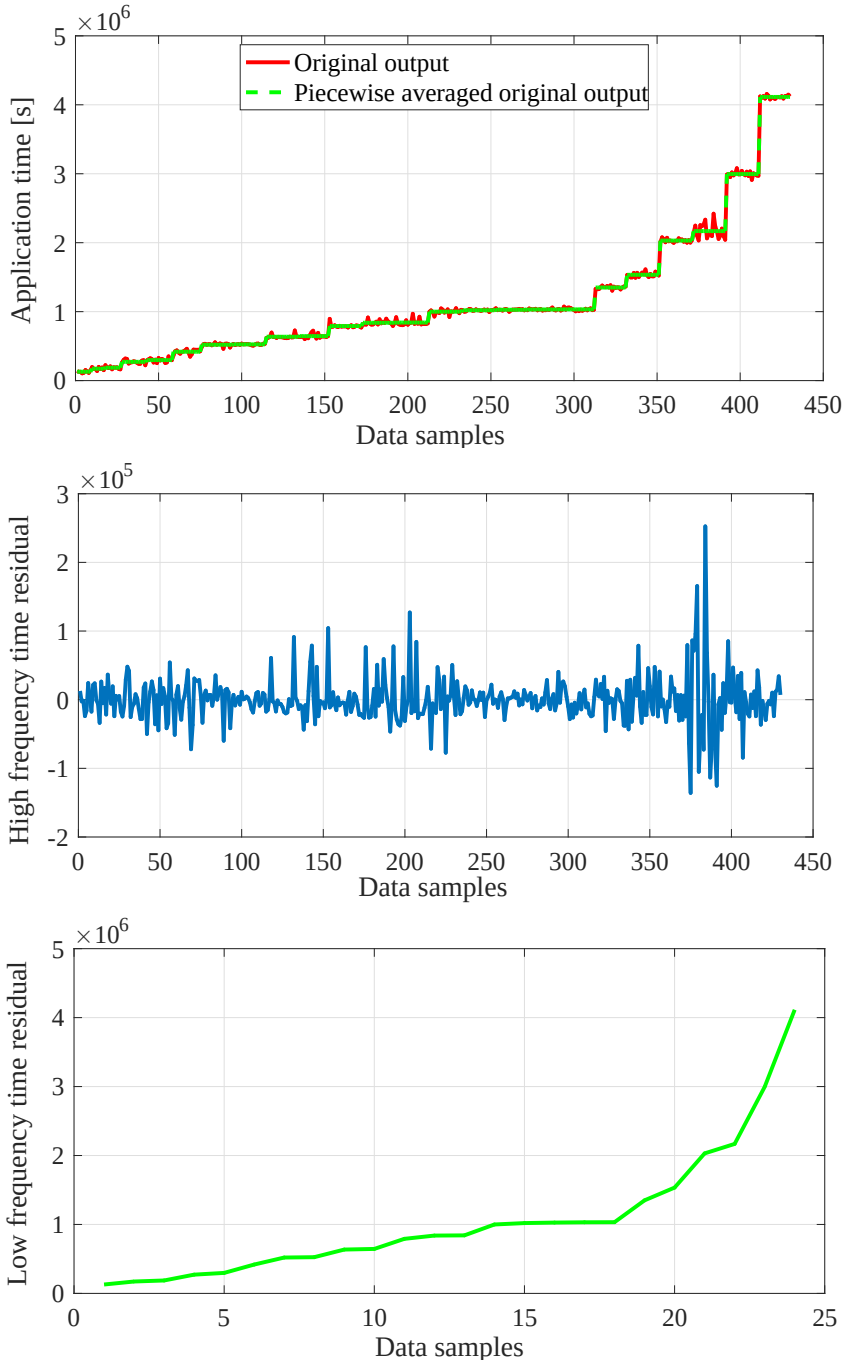

Figure 3.5: *Output data for query R3: Original output signal and piecewise averaged original output (top), high frequency residual of the original output (middle), resampled low frequency residual of the original output (bottom).*

The features represent the main characteristics of the MR phases, that are inverse of the number of cores (u_1) [4], the data size (u_2), the maximum and average bytes sent (u_3, u_4), the maximum and average shuffling time (u_5, u_6), the maximum and average reduce time (u_7, u_8), the maximum and average mapping time (u_9, u_{10}), the number of reducers (u_{11}) and the number of mappers (u_{12}).

Table 3.7: *Main characteristics of the datasets used for the experiments.*

$Query$	N_f	N_F	N
$R1$	12	91	395
$R3$	12	91	430
$R4$	12	91	264

Several preprocessing steps are applied on the original dataset. The output data are filtered so that the samples exceeding $+ - 20\%$ of the output (i.e. *application completion time*) median are removed. The original features are normalized with the same procedure explained in Section 3.5, while the output is recomputed in logarithmic scale. In Fig. 3.5 (top) the original output signal and the piecewise averaged original signal are presented for query $R3$ in increasing order. The latter one is obtained after the filtering by averaging the output on the samples corresponding to the same setup of the number of cores. It is used as a baseline of the original signal and characterizes the low frequency dynamics of the output. Thus the original signal is decomposed into two components: *the low frequency dynamics* (see Fig. 3.5 , bottom) and *the high frequency dynamics* (see Fig. 3.5, middle). Since the low frequency dynamics signal is piecewise constant, it was resampled so that just one sample is retained for each data slot of the constant values. Based on this rationale the RFSC was employed to extract two different linear regression models, which explain the low and high frequency dynamics. The final predicted completion time is obtained by summing up the predicted outputs of the two models. To account for the algorithm randomness, the algorithm is executed five times on all three queries. The final performance is evaluated in terms of the Normalized Root Mean Square Error (NRMSE) computed as:

$$NRMSE = \frac{\sqrt{MSE}}{y_{max} - y_{min}} \tag{3.11}$$

Regarding the initial setup of the design parameters, the number of iterations was set to $N_i = 100$, the maximum nonlinearity degree to $L = 2$,

[4] Since the application completion time is reciprocal to the number of employed cores, we used $1/nCores$ as an input instead of $nCores$.

the number of generated models to $N_p = 100$, the significance confidence interval to $\alpha = 0.8$, all initial TIPs to $1/50$, and $\epsilon = 0.001$. The proposed algorithm was implemented in Matlab (version 2016a) and executed on an Intel(R) Core i7-6700K machine, with 4.00GHz CPU, 32GB of RAM, and a 64-bit Windows Operating System.

Table 3.8 reports the results obtained with 5 independent runs for each query ($R1$, $R3$ and $R4$), in terms of the extracted models and the corresponding NRMSE. As a benchmark for the performance analysis we used an AM for the evaluation the execution time of Big Data applications. According to this model, the execution time is given by:

$$A_{time} = \sum_{s=1}^{k} \lceil \frac{nTask_s}{nCores} \rceil avgTask_s \tag{3.12}$$

where $nTask_s$ is the number of tasks, $avgTask_s$ the average execution time of each task associated with a stage s, $nCores$ the number of available cores during the jobs execution and k the total number of stages. The rationale behind the model given by (3.12) is as follows: the number of waves required to execute the tasks sequentially at a stage s equals $\lceil \frac{nTasks_s}{nCores} \rceil$. During the first $\lceil \frac{nTasks_s}{nCores} \rceil - 1$ waves, the tasks keep the $nCores$ cores busy, while during the very last wave, the final tasks complete the execution of a stage s using fewer cores.
Inspecting Table 3.8, the behavior of the high frequency dynamics behavior is mostly described by the maximum and average values of shuffling, mapping and reduce tasks and their combinations. Features related to the mean of the parameters ($e.g.$ u_6, u_8, and u_{10}) indicate a small variability in the data and thus they are sufficient to explain the overall behavior of the system. On the contrary, the presence of features related to maximum values indicate more variability due to the degradation of the performance at the synchronization points by slowest tasks. Considering that these variables belong to the network layer, the latter case can be justified by the fact that the experiments were performed in a data center accessing the remote data disk while the other tasks from other users were performed simultaneously, so that average values of the variables were insufficient to capture the system behavior properly. The low dynamics model frequently includes second order monomials, involving the reciprocal of the number of cores (u_1), the number of reducers (u_{11}) or the number of mappers (u_{12}), in line with the AM model. Considering the performance, the RFSC outperforms significantly the AM on query $R3$, while being outperformed on query $R2$.

Table 3.8: *High and Low frequency dynamics models and the corresponding NRMSE obtained in 5 independent runs on queries $R1$, $R3$ and $R4$ respectively with RFSC, averaged NRMSE, and the NRMSE of the AM model.*

Run	High dynamics model	Low dynamics model	$NRMSE$	$\overline{NRMSE}$	$NRMSE_{AM}$
1	$\{u_6,u_7,u_8,u_9,u_{10},u_8u_8,u_8u_9\}$	$\{1,u_9,u_1u_{12}\}$	0.06	0.07	0.07
2	$\{u_6,u_7,u_8,u_9,u_{10},u_6u_9$, $u_7u_9,u_8u_8,u_8u_9,u_9u_9\}$	$\{1,u_9,u_1u_6\}$	0.08		
3	$\{u_6,u_7,u_{10},u_6u_9,u_7u_8,u_7u_9,$ $u_9u_9\}$	$\{1,u_9,u_1u_6\}$	0.08		
4	$\{u_5,u_6,u_7,u_9,u_{10},u_5u_5$ $u_5u_8,u_6u_8,u_6u_9,u_7u_9,u_8u_8,$ $u_9u_9\}$	$\{1,u_9,u_1u_9\}$	0.09		
5	$\{u_6,u_7,u_{10},u_6u_9,u_7u_9,u_8u_8,u_9u_9\}$	$\{1,u_6,u_5u_8\}$	0.07		
1	$\{u_{10},u_9u_{10}\}$	$\{1,u_1u_{11}\}$	0.13	0.14	0.07
2	$\{u_{10},u_9u_{10}\}$	$\{1,u_{10},u_1u_2\}$	0.11		
3	$\{u_{10},u_9u_{10}\}$	$\{1,u_{12},u_1u_{10}\}$	0.08		
4	$\{u_{10}\}$	$\{1,u_3u_5,u_6u_{10}\}$	0.19		
5	$\{u_{10}\}$	$\{1,u_5u_6,u_6u_{10}\}$	0.19		
1	$\{u_{10},u_6u_6,u_6u_7,u_7u_9,u_{10}u_{10}\}$	$\{1,u_9u_9,u_{11}u_{12}\}$	0.13	0.13	0.22
2	$\{u_7,u_{10},u_6u_6,u_6u_7,u_7u_8$ $u_9u_{10}\}$	$\{1,u_2u_9\}$	0.12		
3	$\{u_{10},u_5u_8,u_5u_{12},u_6u_6,u_6u_7$ $u_7u_9,u_7u_{12},u_{10}u_{10}\}$	$\{1,u_9u_{11}\}$	0.15		
4	$\{u_6,u_7,u_{10},u_6u_7,u_7u_8,u_9u_{10}\}$	$\{1,u_2,u_{12},u_2u_{11},u_{11}u_{12}\}$	0.13		
5	$\{u_6,u_7,u_{10},u_6u_7,u_7u_8,u_9u_9$ $u_{10}u_{10}\}$	$\{1,u_{10}u_{12}\}$	0.13		

3.6.2 Performance analysis on Spark

In this section we report the main results obtained with the estimated models in terms of accuracy. The synthetic datasets used for running the experiments has been generated using the TPC-DS industry standard benchmark. Multiple experiments were conducted and three different cluster configurations were used, i.e., two different deployments on Microsoft Azure HDInsight and a cluster based on IBM Power8 (P8) available at the Politecnico di Milano premises. Concerning Microsoft Azure, the HDInsight PaaS offer which includes Spark 1.6.2 release on Ubuntu 14.04 was used. The experiments were performed on two different deployments where two different types of virtual Machines (VMs), namely A3 and D12v2 were tested. A3 VMs includes 4 cores with 7 GB of RAM and 250 GB disk.

Regarding the second deployment, the D12v2 nodes include 4 cores, 28 GB of RAM and 200 GB local SSD. In the A3 case, the workers configuration consisted of 6 up to 48 cores, while in the D12v2 case workers used between 12 and 52 cores. Each Spark executor had 2 cores with 2GB RAM while 4GB were allocated to the driver. The P8 cluster is based on Spark 1.4.1 running on Red Hat 7.3 and includes six VMs with 11 cores and 60GB of RAM each. Fiber channel disks up to 12TB of physical storage were available. Spark executors were configured with 2 cores and 4GB RAM while 8GB were allocated to the driver. Workers' runs were supported by 4 VMs and used between 6 and 44 cores. The driver was running on a dedicated master node with the same VM configuration. Tests were performed on four SQL queries: $Q26$, $Q32$, $Q40$ and $Q52$ coming form the official TPC-DS specification. The data set was generated also in this case through the TPC-DS generator and we considered a 500 GB size on Azure while on the P8 platform the data set was varied between 250 GB and 1TB with step 250 GB. For every query we repeated the profiling process ten times, considering that the profiles collect statistical information about jobs.

The main characteristics of the employed queries in terms of data size, are presented in Table 3.9, where stg denotes the number of stages in a given query, N_f the size of the original feature set, and N_s the total number of available samples for the experiments carried out on datasets with $DataSize = \{250 \quad 750 \quad 1000\}GB$. The original feature set was extended with the features obtained by combining the original features, $i.e.$ $DataSize * nCores$ (u_2), $(DataSize^2)/nCores$ (u_3), $DataSize/nCores$ (u_4), $1/nCores$ (u_5).

Table 3.9: *Main characteristics of the datasets used for the experiments.*

$Query$	stg	N_f	N_{250}	N_{750}	N_{1000}
$Q26$	9	54	115	105	115
$Q32$	8	52	111	105	115
$Q40$	9	54	113	105	115
$Q52$	4	30	111	105	115

The original features which characterize the Spark jobs at every stage are the average and maximal completion time of the tasks. The average and maximal shuffle time and the average and maximal bytes sent characterize spark jobs from stage 4 for the $Q26$, stage 2 for the $Q32$, stage 4 for the $Q40$ and stage 1 for the $Q26$. The data are normalized as explained in the previous sections ($e.g.$ Section 3.5).

To obtain a reliable assessment of the method and to test also the model interpolation capabilities, tests are carried out on six different training-test partitions of the original datasets' for every dataset, which will be denoted as a cases $c1$ to $c6$ in the following.

Fig. 3.6 presents the output, $i.e.$ the *application completion time* for query $Q26$ and $DataSize = 1000GB$. Unlike query $R3$ which we presented for the MR framework, it does not contain high frequency signal component. Hence, it was sufficient to extract a single model in the form of a liner regression which captures the overall output dynamics.

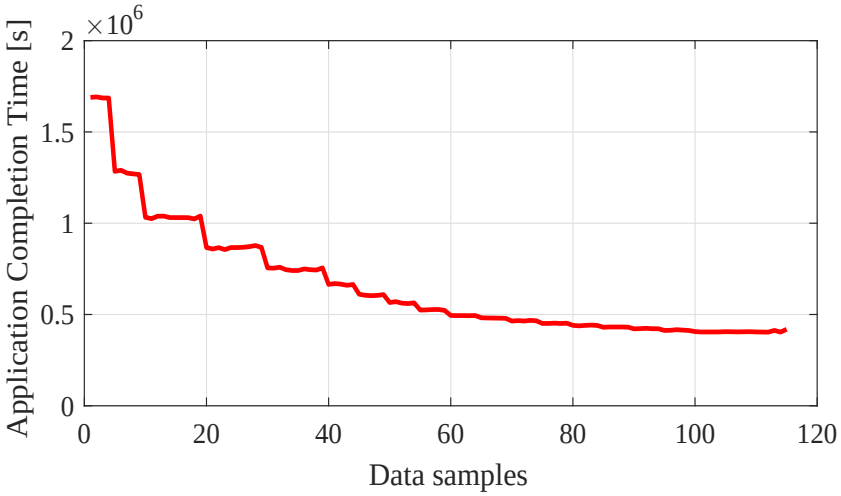

Figure 3.6: *Application completion time for query* $Q26$ *and* $DataSize = 1000GB$.

Regarding the initial parameter setup of the RFSC, the number of iterations was set to $N_i = 100$, the maximum nonlinearity degree to $L = 1$, the number of generated models to $N_p = 100$, the significance confidence interval to $\alpha = 0.9$, all initial TIPs to $1/1000$, and $\epsilon = 0.001$.

Tables 3.10 and 3.13 report the results in terms of the selected models and the corresponding accuracy on the training and test data for all six cases for each dataset. The list of model terms and the corresponding features for all four queries ($Q26$, $Q32$, $Q40$, $Q52$) is reported in 3.11. The performance of the obtained models is measured in terms of the NRMSE and compared to the one obtained by AM with the same experiment setup.

Inspecting the results presented in Tab. 3.10, features u_2, u_3, u_4, and u_5 constitute the most frequent model terms. Beside these, u_{13}, u_{27} u_{33} (representing the average time of tasks for the stages) and u_{40} (the number of cores) also frequently appear. The selection of feature u_4 is in line with the AM (3.12), where it is used to provide the expected number of stages in order to estimate the application execution time. u_2 is representative of the shuffle activities which are implemented as a *many to many* communication in the network, when intermediate results are sent form some nodes to the others to create next stage Resilient Distributed Dataset (RDD)[5]. Hence, the intermediate data size is also proportional to $DataSize$. For what concerns the number of cores (u_{40}) term, shuffle communication is *many to many* and grows with the cluster size. Though terms u_4 and u_5 do not have physical interpretation nor can they be compared the AM, the algorithm recognizes them as valuable for obtaining a good predictive performance.
In general the RFSC has a better performance compared to the AM. For what concerns the model structure, the RFSC provides more compact models containing on average 5-8 elements, while the size of the AM varies depending on the total number of stages.

[5]RDD are the most important concept in Spark: a collection of reliable objects partitioned across the nodes of the cluster that can be processed in parallel.

Table 3.10: *Models obtained with RFSC for different training-test data partitionings and* $DataSize = [250 \quad 750 \quad 1000]GB$.

		$Modelterms$		
	$DataSize[GB]$	250	750	1000
$Query$	c1	{ 1,u_3,u_4,u_5,u_{13},u_{40} }	{ 1,u_2,u_3,u_4,u_5,u_{40} }	{ u_3,u_4,u_5 }
	c2	{ 1,u_2,u_3,u_4,u_5,u_{23},u_{40} }	{ 1,u_2,u_3,u_4,u_5,u_{12},u_{40} }	{ u_3,u_4,u_5,u_{31},u_{34} }
	c3	{ u_3,u_4,u_5,u_{12},u_{13},u_{23},u_{27} }	{ 1,u_4,u_5,u_{33} }	{ u_4 }
$Q26$	c4	{ u_2,u_3,u_4,u_5,u_{18},u_{21},u_{27} }	{ u_3,u_5,u_{13} }	{ u_3,u_4,u_5,u_{35},u_{38} }
	c5	{ u_3,u_4,u_5 }	{ u_2,u_3,u_4,u_5,u_{19},u_{23},u_{27},u_{32},u_{40} }	{ 1,u_2,u_3,u_5, u_6,u_{23},u_{40} }
	c6	{ 1,u_3,u_4,u_5 }	{ u_3,u_4,u_5,u_{11},u_{13},u_{18} }	{ u_3,u_4 }
	c1	{ u_3,u_4,u_5,u_{22},u_{23},u_{24} }	{ u_2,u_3,u_4,u_5,2u_5,u_{40} }	{ 1,u_2,u_3,u_4,u_{11},u_{40} }
	c2		{ u_3,u_4,u_5,u_{10},u_{15},u_{35},u_{40} }	{ 1,u_2,u_3,u_4,u_5,u_{20},u_{40} }
$Q32$	c3	{ 1,u_2,u_3,u_4,u_5,u_{27},u_{40} }	{ 1,u_2,u_3,u_4,u_5,u_{16},u_{30},u_{40} }	{ u_3 }
	c4	{ 1,u_3,u_4,u_5,u_{11},u_{25} }	{ 1,u_3,u_4,u_5,u_{15},u_{25},u_{40} }	{ u_2,u_3,u_4,u_5,u_7,u_{25},u_{29} }
	c5	{ u_3,u_4,u_5 }	{ u_2,u_3,u_4,u_5,u_{26},u_{27},u_{33},u_{40} }	{ u_3,u_5,u_{32} }
	c6	{ u_2,u_3,u_4,u_5,u_{11},u_{25},u_{34},u_{40} }	{ u_2,u_3,u_4,u_5,1u_4,u_{35},u_{37},u_{40} }	{ u_4,u_5 }
	c1	{ 1,u_3,u_4,u_5,u_{31} }	{ u_3,u_4,u_5,1u_3,u_{19} }	{ 1,u_2,u_3,u_4,u_{27},u_{32},u_{40} }
$Q40$	c2	{ 1,u_3,u_4,u_5,u_{23},u_{33},u_{40} }	{ u_2,u_3,u_4,u_5,u_{11},u_{40} }	{ u_2,u_4,u_5,u_{35},u_{40} }
	c3	{ 1,u_2,u_3,u_4,u_5,u_{19},u_{27},u_{36},u_{40} }	{ 1,u_3,u_5,u_{11},u_{30} }	{ 1,u_3,u_4,u_5,u_{26},u_{30},u_{40} }
	c4	{ u_2,u_3,u_4,u_5,u_{26},u_{27},u_{31} }	{ 1,u_3,u_4,u_5,2u_2,u_{27},u_{28},u_{30} }	{ 1,u_2,u_3,u_4,u_5,u_{23},u_{30},u_{31},u_{40} }
	c5	{ u_3,u_4,u_5 }	{ 1,u_2,u_3,u_4,u_5,u_{23},u_{30},u_{231},u_{40} }	{ u_3,u_5,u_{27},u_{30},u_{39} }
	c6	{ 1,u_2,u_3,u_4,u_5,u_{36},u_{40} }	{ u_2,u_3,u_4,u_5,u_{20},u_{27},u_{31},u_{33},u_{40} }	{ u_3,u_4 }
$Q52$	c1	{ 1,u_2,u_3,u_4,u_5,u_8,u_{13},u_{16},u_{24} }	{ 1,u_2,u_3,u_4,u_5,u_{23} }	{ u_4,u_5 }
	c2	{ u_2,u_3,u_4,u_5,u_{17},u_{24} }	{ u_3,u_4,u_5,u_8,u_{13},1u_5,u_{21},u_{23} }	{ u_3,u_4,u_5,u_8,u_{14},u_{15},u_{19},u_{23} }
	c3	{ u_3,u_4,u_5,u_{13},u_{22},u_{24} }	{ 1,u_4,u_5,u_9 }	{ u_3 }
	c4	{ u_3,u_4,u_5,u_{13},u_{17} }	{ u_3,u_4,u_5,u_{13},u_{17},u_{18} }	{ 1,u_2,u_3,u_4,u_5,u_9,u_{10},u_{11},u_{13},u_{15},u_{23} }
	c5	{ u_3,u_4,u_5 }	{ u_2,u_3,u_4,u_5,u_{13},u_{17},u_{22},u_{23} }	{ u_2,u_3,u_5,u_{12},u_{13},u_{14},u_{22},u_{23} }
	c6	{ u_4,u_5,u_{16} }	{ 1,u_5 }	{ 1,u_4,u_5,u_{21} }

Table 3.11: *Model terms and the corresponding features for queries $Q26$, $Q32$ and $Q40$ (top) and $Q52$ (bottom).*

Model term	$Feature$	Index of the model term	$Feature$
u_1	$constant$	u_{23}	$avgTask_S6$
u_2	$datasize * nCore$	u_{24}	$SHmax_S6$
u_3	$(datasize^2)/nCore$	u_{25}	$SHavg_S6$
u_4	$datasize/nCore$	u_{26}	$maxTask_S7$
u_5	$1/nCore$	u_{27}	$avgTask_S7$
u_6	$maxTask_S0$	u_{28}	$SHmax_S7$
u_7	$avgTask_S0$	u_{29}	$SHavg_S7$
u_{10}	$maxTask_S2$	u_{30}	$Bmax_S7$
u_{11}	$avgTask_S2$	u_{31}	$Bavg_S7$
u_{12}	$maxTask_S3$	u_{32}	$maxTask_S8$
u_{13}	$avgTask_S3$	u_{33}	$avgTask_S8$
u_{14}	$maxTask_S4$	u_{34}	$SHmax_S8$
u_{15}	$avgTask_S4$	u_{35}	$SHavg_S8$
u_{16}	$SHmax_S4$	u_{36}	$Bmax_S8$
u_{18}	$maxTask_S5$	u_{37}	$Bavg_S8$
u_{19}	$avgTask_S5$	u_{38}	$maxTask_S9$
u_{20}	$SHmax_S5$	u_{39}	$avgTask_S9$
u_{21}	$SHavg_S5$	u_{40}	$nContainers$
u_{22}	$maxTask_S6$		

Model term	$Feature$
u_1	$constant$
u_2	$datasize * nCore$
u_3	$(datasize^2)/nCore$
u_4	$datasize/nCore$
u_5	$1/nCore$
u_6	$maxTask_S0$
u_8	$maxTask_S1$
u_9	$avgTask_S1$
u_{10}	$SHmax_S1$
u_{11}	$SHavg_S1$
u_{12}	$maxTask_S2$
u_{13}	$avgTask_S2$
u_{14}	$SHmax_S2$
u_{15}	$SHavg_S2$
u_{16}	$maxTask_S3$
u_{17}	$avgTask_S3$
u_{18}	$SHmax_S3$
u_{19}	$SHavg_S3$
u_{21}	$maxTask_S4$
u_{22}	$avgTask_S4$
u_{23}	$nCore$

Table 3.12: *NRMSE of the models presented in Tab. 3.10 for the corresponding training-test data partitionings and $DataSize$.*

RFSC		$NRMSE_{TR}$			$NRMSE_{TE}$		
	$DataSize[GB]$	250	750	1000	250	750	1000
$Query$	c1	0.01	0.01	0.04	0.01	0.01	0.05
	c2	0.01	0.01	0.03	0.01	0.01	0.04
$Q26$	c3	0.01	0.01	0.04	0.03	0.01	0.06
	c4	0.01	0.03	0.03	0.04	0.04	0.05
	c5	0.04	0.01	0.00	0.10	0.07	0.01
	c6	0.03	0.02	0.04	0.04	0.03	0.06
	c1	0.02	0.02	0.00	0.04	0.02	0.01
	c2	0.01	0.01	0.00	0.03	0.02	0.01
$Q32$	c3	0.01	0.01	0.04	0.02	0.01	0.05
	c4	0.01	0.01	0.01	0.02	0.01	0.03
	c5	0.03	0.03	0.03	0.08	0.06	0.05
	c6	0.01	0.01	0.03	0.07	0.04	0.05
	c1	0.02	0.01	0.01	0.03	0.02	0.01
	c2	0.01	0.02	0.02	0.01	0.02	0.03
$Q40$	c3	0.01	0.01	0.01	0.02	0.03	0.01
	c4	0.01	0.01	0.00	0.01	0.03	0.05
	c5	0.03	0.00	0.01	0.09	0.07	0.02
	c6	0.01	0.00	0.04	0.01	0.02	0.05
	c1	0.01	0.01	0.03	0.01	0.01	0.05
	c2	0.02	0.01	0.01	0.02	0.02	0.02
$Q52$	c3	0.02	0.02	0.04	0.02	0.04	0.05
	c4	0.03	0.03	0.00	0.05	0.06	0.01
	c5	0.03	0.01	0.01	0.08	0.05	0.04
	c6	0.03	0.03	0.01	0.05	0.03	0.02

Table 3.13: *NRMSE of the AM model for the corresponding training-test data partitionings and $DataSize$.*

AM		$NRMSE_{TR}$			$NRMSE_{TE}$		
	$DataSize[GB]$	250	750	1000	250	750	1000
$Query$	c1	0.04	0.01	0.02	0.05	0.03	0.02
	c2	0.04	0.02	0.02	0.05	0.03	0.02
$Q26$	c3	0.04	0.02	0.02	0.05	0.03	0.02
	c4	0.04	0.03	0.03	0.05	0.02	0.02
	c5	0.04	0.02	0.02	0.07	0.04	0.03
	c6	0.04	0.02	0.02	0.05	0.03	0.02
	c1	0.04	0.02	0.02	0.06	0.03	0.02
	c2	0.04	0.02	0.02	0.06	0.02	0.02
$Q32$	c3	0.04	0.02	0.02	0.05	0.02	0.02
	c4	0.04	0.03	0.02	0.05	0.02	0.02
	c5	0.04	0.02	0.02	0.08	0.03	0.03
	c6	0.04	0.02	0.02	0.06	0.02	0.02
	c1	0.03	0.02	0.02	0.05	0.03	0.03
	c2	0.03	0.01	0.02	0.04	0.03	0.03
$Q40$	c3	0.03	0.01	0.02	0.04	0.03	0.03
	c4	0.04	0.02	0.03	0.04	0.03	0.03
	c5	0.03	0.02	0.02	0.06	0.04	0.04
	c6	0.03	0.02	0.02	0.04	0.03	0.03
	c1	0.05	0.02	0.02	0.08	0.03	0.02
	c2	0.05	0.02	0.02	0.08	0.03	0.02
$Q52$	c3	0.05	0.03	0.02	0.07	0.03	0.02
	c4	0.05	0.03	0.03	0.07	0.03	0.02
	c5	0.05	0.03	0.02	0.11	0.04	0.03
	c6	0.05	0.03	0.02	0.08	0.03	0.02

CHAPTER 4

Distributed Model Selection Strategy

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 methods based on incremental model building. More importantly, besides the obvious increase in computational complexity, the ability of FS algorithms to reach the optimal feature subset diminishes with the increase of the number of features, 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 introduce 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. It is important to notice that any FS method can be employed in the proposed optimization scheme to perform the local model selection tasks. This distributed FS architecture, denoted distributed MSS (dMSS), is used to develop algorithms applicable

to both the frameworks considered in this thesis, *i.e.* distributed algorithm for the identification of NARX models, denoted dRaMSS, and two distributed feature selection algorithms for classification, namely the DFS and the D^{2}CORFS algorithms.

The dRaMSS [8] is a direct extension of the RaMSS method that uses the distributed scheme to divide the MSS task among different processors, each operating on a portion of the regressors with the RaMSS algorithm. The DFS [22] is a generic distributed scheme for FS. As such, it is applicable in combination with any FS method of choice: here we present results obtained using the distributed scheme with the SFS, the RFSC, and the ReliefF FS algorithms (the former two as representatives of wrapper methods, as the latter as representative of filter methods). The D^{2}CORFS [21] is a novel multivariate filter method which employs the distance correlation index as a model evaluation metric and the DFS scheme. In all cases, compared to its non-distributed counterpart and to other competitor methods as well, the proposed distributed scheme yields significant savings in computational time, as well as more accurate and more robust results.

4.1 Distributed model structure selection

The general idea behind the proposed dMSS scheme is borrowed from an algorithm developed for solving distributed LP-type optimization programs, called the Constraints Consensus algorithm [93]. These optimization problems are characterized by a large set of given constraints. Elaborating on the observation that only a tiny fraction of the latter are actually active in correspondence to the optimal solution, it is proposed in [93] to address the optimization problem in a distributed fashion, redistributing the constraints over a network of processors. Each of these processors solves a local optimization problem (of smaller complexity, since it is subject only to a fraction of the constraints) and finds the subset of active constraints that characterizes the solution. All these subsets of (locally) active constraints are then shared among the network and added to the local constraint sets of *all* the processors. When the communication round is finished, each processor verifies that its local solutions do not violate any of the newly added constraints. If this is the case, the processor must repeat the optimization process to update the solution. After a finite number of rounds, the exchanged constraints are not violated by any of the local solutions, *i.e.* a *consensus* on the constraints has been achieved, and the algorithm stops.

In analogy to LP-type optimization programs, the MSS problem is characterized by many potential model terms, only a fraction of which char-

acterize the optimal solution. Accordingly, one can address the MSS task using a distribution approach over the model terms, thus breaking the complexity of the original MSS problem into a number of smaller problems (*i.e.*, with fewer terms), and alternating a solution phase where all the local problems are solved, to a communication phase where the local solutions (*i.e.*, the locally selected model terms) are shared among the different local MSS problems. The optimal solution to any of the local problems is either the best model found so far (since the corresponding terms are among those shared at the end of the previous iteration) or an improving model that combines some of the newly added terms to the previously available ones in the local set. The procedure is iterated until convergence of all the solutions to a unique model. The overall dMSS scheme is explained in detail in the following.

A dMSS problem is defined as a tuple $(\mathcal{D}, \mathcal{R}, \mathcal{J}, \mathcal{U}, \mathcal{B})$, where:

1. $\mathcal{D}$ is a sequence of N input-output samples;
2. $\mathcal{R}$ is a finite set of N_F model terms;
3. $\mathcal{J}$ is a performance index $\mathcal{J} : 2^{\mathcal{R}} \rightarrow [0 \quad 1]$;
4. $\mathcal{U}$ is a set of N_u processing units;
5. $\mathcal{B} : \mathcal{R} \rightarrow \mathcal{U}$ is a surjective function that associates each model term to one of the local MSS processors (referred to as the term distribution map).

The triple ($\mathcal{D}$, $\mathcal{R}$, $\mathcal{J}$) identifies the MSS problem, whereas $\mathcal{U}$ and $\mathcal{B}$ define the structure of the distributed problem. In general, the latter should include a description of the communication network (generally, a time-dependent directed graph), but we here assume for simplicity that all processors are able to communicate with each other at every communication round. Accordingly, it is only necessary to specify the node set $\mathcal{U}$ of the communication network. The method can be generalized to problems where the local units are not all pair-wise connected (provided that the communication graph is strongly connected).

At a given round r the n_uth processing unit is assigned a local subset of model terms $\mathcal{R}_{n_u}(r) \subseteq \mathcal{R}$, according to the following formula:

$$\mathcal{R}_{n_u}(r) = \mathcal{R}_{n_u}(0) \cup \mathcal{E}(r-1), \tag{4.1}$$

for $n_u = 1, \ldots, N_u$, where $\mathcal{R}_{n_u}(0)$ is the initial local term set for the n_uth processing unit, and $\mathcal{E}(r-1)$ denotes the subset of model terms selected at

the end of the previous round collectively by all the processors. Then, the n_uth processing unit solves the local MSS task using a suitable algorithm and returns the local best model term subset $\mathcal{E}_{n_u}(r) \in \mathcal{R}_{n_u}(r)$. Finally, $\mathcal{E}(r) = \bigcup_{n_u=1}^{N_u} \mathcal{E}_{n_u}(r)$ (as commented below, it is sometimes convenient to limit this aggregation to a portion of the processors, *e.g.*, those returning the best results).

The iterative procedure is initialized by applying a strict partition of $\mathcal{R}$ to define the initial local model term sets:

$$\bigcup_{n_u=1}^{N_u} \mathcal{R}_{n_u}(0) = \mathcal{R}, \tag{4.2}$$

where $\mathcal{R}_{n_m}(0) \cap \mathcal{R}_{n_n}(0) = \varnothing$, for $m \neq n$.

A balanced-size random allocation of the model terms to the sets $\mathcal{R}_{n_u}(0)$, $n_u = 1, \ldots, N_u$, is here adopted, such that the size of the resulting sets is either $\lfloor N_F/N_u \rfloor$ or $\lceil N_F/N_u \rceil$. Regarding the subdivision of model terms to the various processors, another policy is also experimented (see Section 4.4, later on), that consists in re-allocating randomly the terms to the processors at the beginning of *every* iteration. This reshuffling of the bins improves the exploration capabilities of the algorithm by adding a further layer of randomization.

The initial common model term set $\mathcal{E}(0)$ is typically an empty set, unless some *a priori* information about the model structure is available that suggests some promising terms. Assigning those model terms to all processors may accelerate the convergence of the algorithm.

Let $\mathcal{C}(r)$ denote the cumulative set of shared model terms until round r (referred to as the *common model term set*), which can be calculated recursively as:

$$\mathcal{C}(r) = \mathcal{C}(r-1) \cup \mathcal{E}(r), \tag{4.3}$$

with $\mathcal{C}(0) = \varnothing$. The dMSS algorithm stops when no new model terms are returned by the set of processors, *i.e.* when $\mathcal{C}(r^\star) = \mathcal{C}(r^\star - 1)$, $r^\star$ denoting the round at which convergence is reached. Ideally, at convergence all processors should return the same model. In practice, however, this does not necessarily always happen, since the employed MSS algorithms are based on some heuristic, and hence may fail to achieve optimality. This effect may be further amplified if the MSS algorithm employed at each processor is randomized. To deal with this potential multiplicity of solutions we here adopt a majority voting approach and return the model structure that has been selected by most of the local MSS processors. Alternatively, one can

perform a final MSS round limited to the model terms in $\mathcal{E}(r^\star)$.

During the selection process, one must take care that the size of the local model term sets $\mathcal{R}_{n_u}(r)$, $n_u = 1, \ldots, N_u$, does not increase too much by way of the shared regressors $\mathcal{E}(r-1)$. Indeed, this would defy the very purpose of the distribution procedure (*i.e.* that of breaking the original MSS problem into much smaller ones). This "overloading" problem is particularly significant at the early stages of the procedure, and especially when N_u is large, since then many of the locally selected models will typically contain irrelevant terms and thus have poor performance. Therefore, sharing all the N_u different models with each of the local model term bins would greatly enlarge its set of model terms, and mostly with useless content. This problem is automatically mitigated as the procedure goes on, since when some useful model terms have been detected and shared among the local model term bins then they are typically selected in all local models. Consider also that the information sharing stage is always performed starting from the initial model term subsets assigned to the local model term bins so that term accumulation is avoided. Clearly, it is important to limit the number of terms that are actually shared among the local model term bins, especially at the first stages of the algorithm. This can be simply done by restricting attention to the first $N_u^\star \ll N_u$ local models ranked by performance.

Algorithm 4 The dMSS scheme

$(\mathcal{R}_1(0), \ldots, \mathcal{R}_{N_u}(0)) \leftarrow \mathrm{BRP}(\mathcal{R}, N_u)$
$\mathcal{C}(0) \leftarrow \varnothing$
$\mathcal{E}(0) \leftarrow \varnothing$
$r \leftarrow 0$
repeat
    $r \leftarrow r + 1$
    for $n_u \leftarrow 1, \ldots, N_u$ **do**
        $\mathcal{R}_{n_u}(r) \leftarrow \mathcal{R}_{n_u}(0) \cup \mathcal{E}(r-1)$
        $(\mathcal{E}_{n_u}(r), \mathcal{J}_{n_u}) \leftarrow \mathrm{MSS}(\mathcal{D}, \mathcal{R}_{n_u}(r), \mathcal{J})$
    end for
    $U^\star \leftarrow \mathrm{RANK}(\mathcal{J}_1, \ldots, \mathcal{J}_{N_u}, N_u^\star)$
    $\mathcal{E}(r) \leftarrow \bigcup_{n_u \in U^\star} \mathcal{E}_{n_u}(r)$
    $\mathcal{C}(r) \leftarrow \mathcal{C}(r-1) \cup \mathcal{E}(r)$
until $\mathcal{C}(r) = \mathcal{C}(r-1)$
$(\mathcal{E}^\star, c^\star) \leftarrow \mathrm{MV}(\mathcal{E}_1(r), \ldots, \mathcal{E}_{N_u}(r))$

A pseudocode of the dMSS procedure is described in Algorithm 4. The main procedure uses three accessory routines, namely BRP, RANK and MV

which are omitted for brevity. BRP performs the balanced-size random partition of $\mathcal{R}$, RANK outputs the indices of the top $N_u^\star$ local models, ranked in terms of performance, and MV returns the most selected model among the different processors ($c^\star$ is the consensus percentage, *i.e.* the percentage of processors returning exactly that model, which provides a measure of the reliability of the algorithm result). The MSS function performs the selection over a given subset of model terms. The preferred MSS algorithm can be adopted for this purpose.

A flowchart of the dMSS scheme is shown in Figure 4.1.

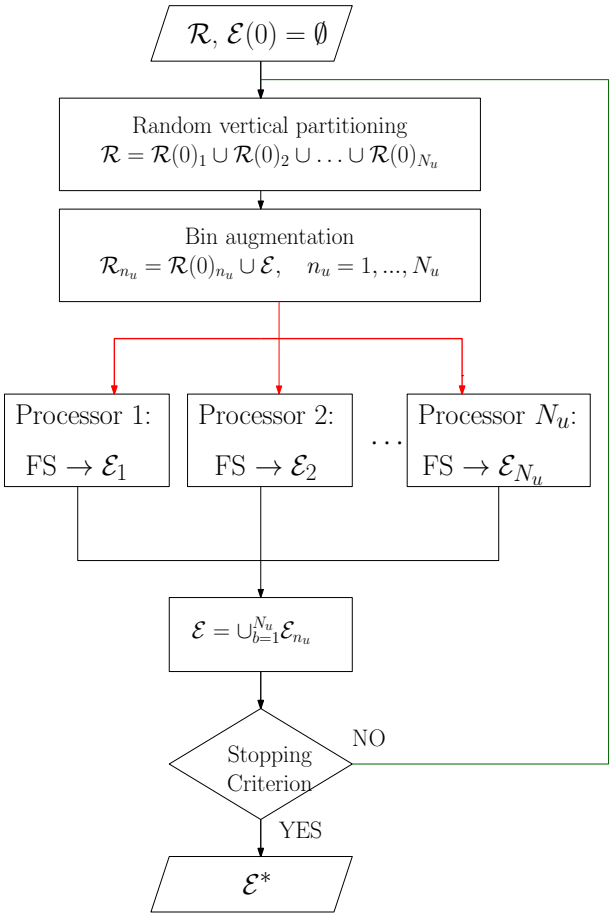

Figure 4.1: *A flowchart of the proposed dMSS scheme.*

4.2 The distributed RaMSS

The RaMSS has proven to be a very effective MSS strategy for the identification of NARX models, capable of outperforming both classical MSS algorithms, such as the FROE and the iOFR, and probabilistic ones, such as the RJMCMC [4]. As such, it is a good candidate for application within the dMSS scheme. However, some of the parameters that govern its execution must be specifically set for the dMSS scheme, in order to fully leverage the advantages provided by the distributed framework. In the sequel we will employ the acronym dRaMSS (distributed RaMSS) to refer to the dMSS scheme employing the RaMSS as local MSS algorithm.

As explained in Section 3.2, at each iteration the RaMSS extracts N_p samples from the current model distribution and uses them to evaluate the importance indices $\mathcal{I}_j$, $j = 1, \ldots, N_F$, associated to the regressors. The reliability of these indices depends on the size of the sample (N_p), and ultimately on the relative frequency with which the various regressors are extracted. In general, the higher the number of the regressors in the search space $\mathcal{R}$, the higher the number of generated models should be, in order to sample the space of model structures with sufficient resolution and to gather sufficiently vast information to evaluate all regressors. Unfortunately, the computational effort increases with N_p and becomes easily prohibitive for large problems. Thanks to its distribution mechanism, the dMSS procedure can overcome this limitation by applying the RaMSS algorithm to sub-problems of manageable size.

The experimental results reported in section 4.3.2 show that the accuracy and elapsed time are nondecreasing with respect to N_p. They are also affected by the initial value of the TIPs, which sets the exploration rate of the algorithm (high initial TIPs imply that each regressor will be sampled more frequently and that, consequently, larger models will be extracted). Combining a small N_p with large initial TIPs provides a good setting for the dRaMSS, resulting in quite accurate selections at an affordable computational cost. In most of the documented experiments we set $N_p = 10$ and $\mu_j(0) = 0.5$ for $j = 1, \ldots, N_F$. Since with these settings quite large models are initially selected, the role of the statistical test in detecting the redundant terms is crucial.

Other important settings are the threshold for selecting regressors μ_{thresh} and the maximum number of iterations. Indeed, especially in the first rounds of the dRaMSS, many processors will operate on regressor subsets containing few or even none of the true regressors, which may result in an incomplete convergence and a stopping of the local RaMSS execu-

tions due to an exhaustion of the available iterations. As a side effect, these executions often return large regressor subsets as a result of the selection, with mostly useless regressors. A possible, effective solution to this problem is to reduce the maximum number of iterations regardless of the actual convergence of the algorithm. In this way, the pointless exploration of local spaces that contain little or no useful information is contained. Furthermore, stopping prematurely these executions prevents the algorithm from selecting many useless regressors for further sharing. As already commented in the previous section, these local selections can be automatically rejected by applying the rule that only the top-ranked local models are to be considered for sharing. In the following, we used the threshold $\mu_{\text{thresh}} = 0.75$ for regressor selection, and a number of maximum iterations equal to 50.

4.3 Experimental analysis of the dRaMSS

4.3.1 An illustrative example

Consider the following system taken from [10]:

$$\begin{aligned}\mathcal{S}_1 : y(k) = {} & 0.7y(k-1)u(k-1) - 0.5y(k-2) \\ & + 0.6u(k-2)^2 - 0.7y(k-2)u(k-2)^2 + e(k)\end{aligned}$$

with $u(k) \sim \text{WUN}(-1, 1)$ and $e(k) \sim \text{WGN}(0, 0.004)$, where $\text{WUN}(a, b)$ denotes a white uniform distribution defined on the interval $[a,\, b]$, whereas $\text{WGN}(\rho, \sigma^2)$ is a white Gaussian distribution with mean ρ and variance σ^2. A sequence of 2000 input/output data is used for identification purposes. The candidate regressor set $\mathcal{R}$ contains all the monomials obtained as combinations of lagged inputs and outputs with maximum lags $n_u = n_y = 4$ and maximum degree $L = 3$, for a total of $N_F = 165$ regressors. The dRaMSS is applied with $N_u = 10$, so that each processor starts with 16 or 17 regressors. As a result of the initial random distribution of the candidate regressors, $y(t-1)u(t-1) \in \mathcal{R}_4(0)$, $y(t-2) \in \mathcal{R}_5(0)$, $y(t-2)u(t-2)^2 \in \mathcal{R}_5(0)$, and $u(t-2)^2 \in \mathcal{R}_7(0)$.

The parameter settings of the local RaMSS algorithms are set as follows: the maximum number of iterations is equal to $N_i = 50$, the number of extracted models is equal to $N_p = 20$, the initial TIPs are set to $\mu_j(0) = 0.25$, $j = 1, \ldots, N_F$, the significance level for the statistical test is set to $\alpha = 0.001$, and $K = 1$ in the exponential form of the cost function. Model performances are evaluated using index (2.34) with $\alpha_J = 1/2$. The TIP

threshold for final regressor selection is set to $\mu_{\text{thresh}} = 0.75$. The models extracted by the local MSS processors are all redistributed.

The evolution of the TIP vectors for all processors is shown in Figure 4.2, where the rows are associated to the processors and the columns to the iterations of the dMSS scheme. The TIP trajectories associated to the correct model terms are emphasized. At the 1st round, all processors select a model with 2 to 6 regressors and performance ranging from 0.91 to 0.95. Only 3 of the 4 correct regressors are selected by processors 4, 5 and 7 ($y(t-2)u(t-2)^2$ is missed). The best performing model is obtained by the 7th processor. A total of 33 regressors are shared for the next dMSS round. At the beginning of the 2nd round all processors have 3 correct regressors in their local regressor sets, and the 5th one has the complete model structure. Unsurprisingly, processor 5 converges to the correct model structure after only 23 iterations. All the other processors select all their correct terms and their local solutions are characterized by better performance indices compared to the previous round (the performances range from 0.98662 to 0.99178, the latter being the performance of the 5th processor). All these solutions contain the term $y(t-4)u(t-2)^2$ which resembles the missing correct one $y(t-2)u(t-2)^2$. As one can see from the evolution of the TIPs (see Figure 4.2, 2nd column), the limit on the number of iterations stops prematurely the execution of all the processors that do not contain the complete true model structure (*i.e.*, all except processor 5), and prevents many irrelevant regressors from being selected. At the end of the execution, the only new regressor that is added to the common set is the missing correct regressor. At the beginning of the 3rd round, the search space of all the processors finally contains the correct model structure. All the processors are able to select the correct model structure (100% consensus) and the algorithm terminates (no new regressors are selected). The three rounds have lasted around 3 s, overall.

4.3.2 Sensitivity of the RaMSS to its design parameters

The number of generated models N_p and the initial TIP values $\boldsymbol{\mu}_0$ are crucial design parameters for the RaMSS algorithm both in terms of accuracy and speed of convergence. It is therefore important to understand their impact, in order to determine appropriate parameter settings for the dRaMSS. We here illustrate a Monte Carlo analysis on the $\mathcal{S}_1$ system for varying values of the mentioned parameters. The initial TIP values are expressed as a fraction of the total number of regressors N_F, to easily determine the average size of the models extracted during the first iteration. All the other

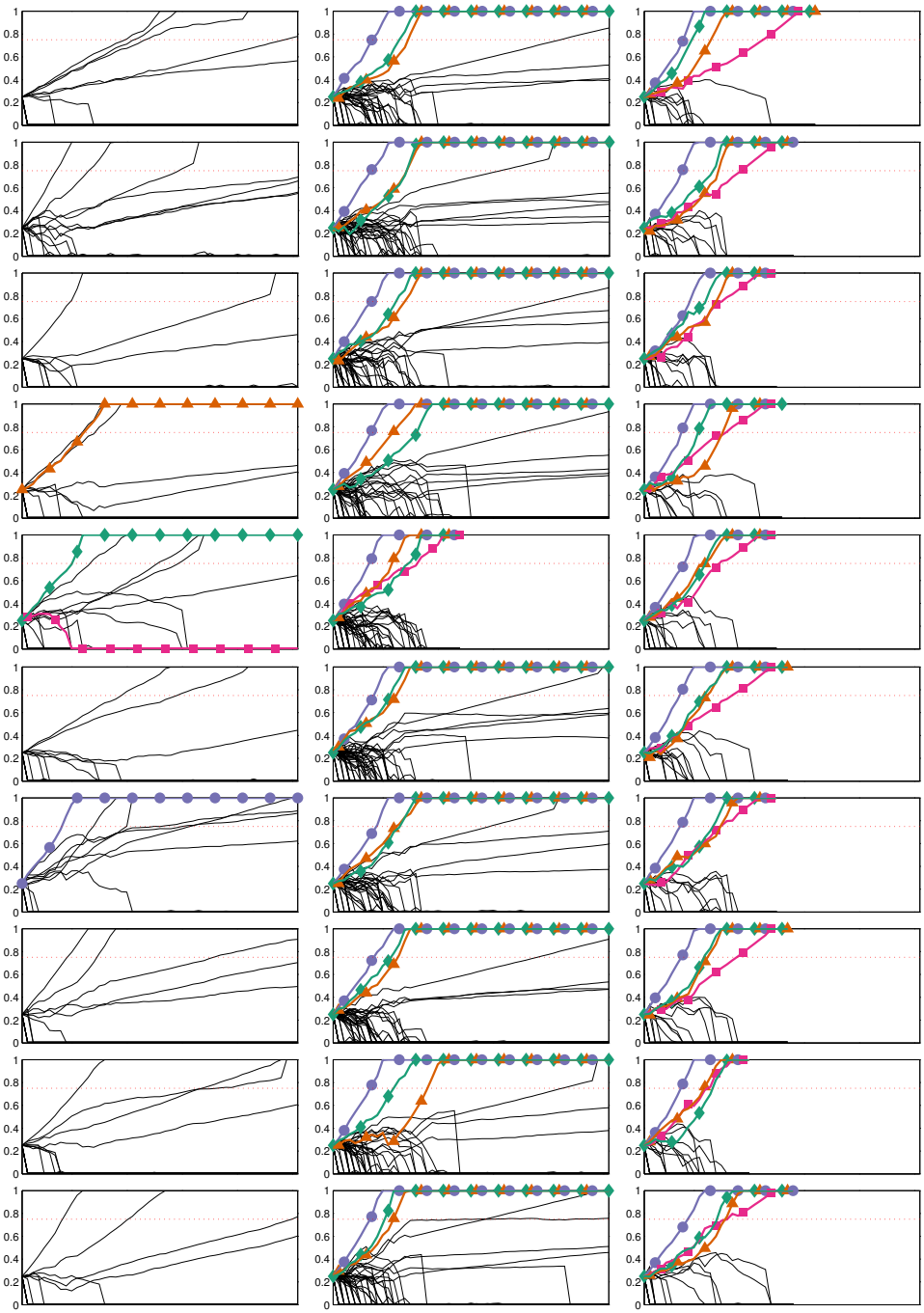

Figure 4.2: *Evolution of the TIP vectors for processors 1 to 10 (from top to bottom), over three consecutive rounds (from left to right). Trajectories associated to true regressors are emphasized.*

algorithm parameters are set as in Section 4.3.1, except that model evaluation is carried out only in prediction ($\alpha_J = 0$).

Figure 4.3 shows the average accuracy of the RaMSS algorithm for increasing N_p, each curve being calculated with a different initial TIP value. The results support the general expectation that increasing the number of explored models improves the robustness of the selection process (the regressor importance is evaluated based on more samples), with a consequent benefit in terms of accuracy as well. For example, with $\mu_j(0) = 1/N_F$, $j = 1, \ldots, N_F$, a correct model selection is achieved only 13% of the times if only 10 models are generated per iteration. This percentage grows to 44% with $N_p = 15$ and to 99% with $N_p = 30$.

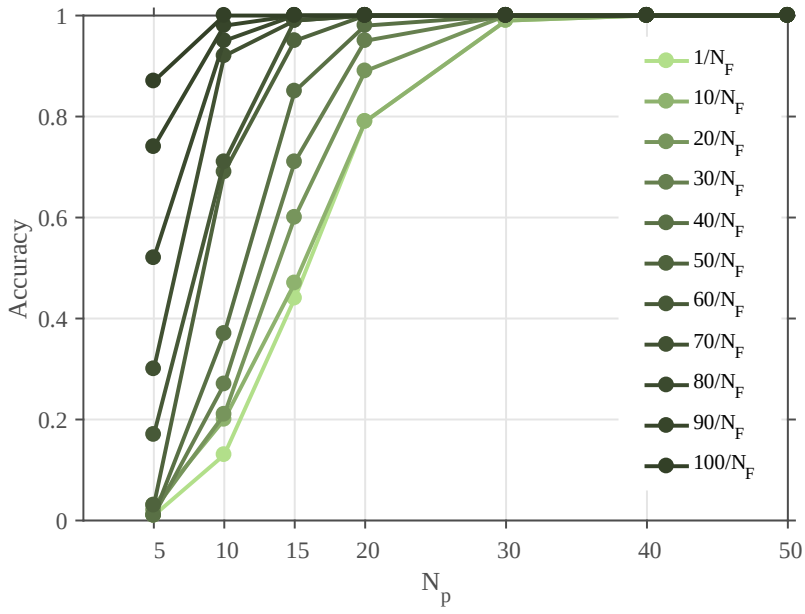

Figure 4.3: *Illustrative example: Average accuracy of the RaMSS for increasing N_p and different initial TIP values ($\mu_j = \beta/N_F$, $j = 1, \ldots, N_F$, with $\beta = 1, 10, 20, \ldots, 100$).*

More interestingly, the simulations also show that the initial TIP value greatly affects the accuracy achieved by the RaMSS algorithm. Apparently, one can achieve a high selection accuracy even with a low N_p, provided that a sufficiently large initial TIP value is used. For example, setting $\mu_j(0) = 100/N_F$, $j = 1, \ldots, N_F$, allows one to reach 100% accuracy with as few as 10 generated models per iteration. The rationale is that, since more regressors are tested for each generated model, the terms that belong to the true model structure have a higher probability to emerge. This occurs

even if they are combined with irrelevant regressors, since the latter ones are removed by the statistical significance test.

Needless to say, $\boldsymbol{\mu}_0$ and N_p also affect the computational load involved in the execution of the RaMSS algorithm. Overall, the elapsed time grows with both N_p and $\boldsymbol{\mu}_0$ (see Figure 4.4), given that the two parameters affect respectively the number and size of the models to be processed. On the contrary, both the number of explored models and the number of iterations (not represented for brevity) decrease as $\boldsymbol{\mu}_0$ grows. In fact, increasing the number of regressors that are simultaneously tested allows the RaMSS to explore the search space more efficiently, requiring the exploration of fewer models and the execution of fewer iterations to reach convergence. This effect partially compensates the increase in computational load due to the fact that larger models are processed.

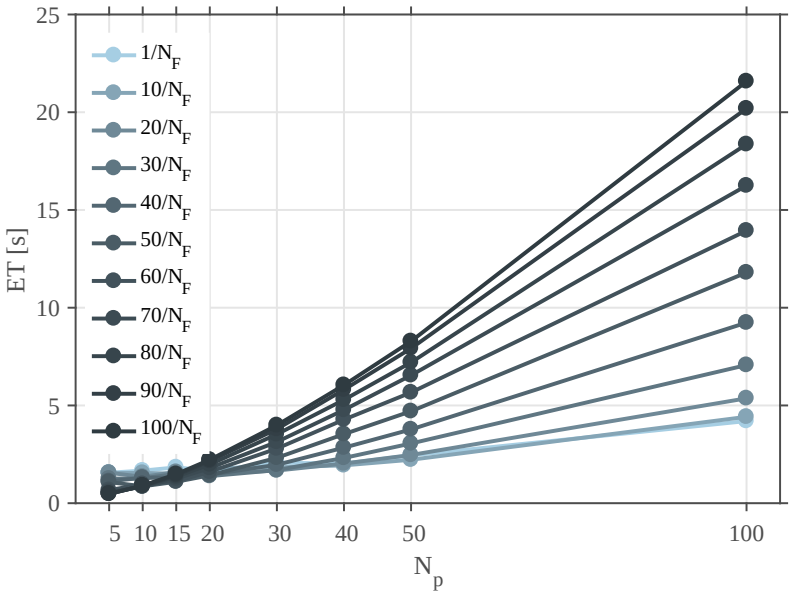

Figure 4.4: *Illustrative example: Elapsed time of the RaMSS for increasing N_p and different initial TIP values ($\mu_j = \beta/N_F$, $j = 1, \ldots, N_F$, with $\beta = 1, 10, 20, \ldots, 100$).*

This analysis suggests that a large N_p should be adopted in combination with small initial TIPs or viceversa. In the latter case, we are leveraging the statistical test to remove the vast majority of irrelevant terms in the generated models. In the considered example, the best results were obtained by setting $\mu_j(0) = 100/N_F$, $j = 1, \ldots, N_F$ and $N_p = 10$. The algorithm converged, on average, in 0.899 seconds after 14.61 iterations, with

275.1 models explored on average during each run. Notice, however, that this strategy increases the risk of incurring in ill-conditioned parameter estimations, especially if the total number of terms is much greater than the number of available samples.

4.3.3 An MSS problem with a large candidate regressor set: case 1

This example compares the basic RaMSS and its distributed version on an MSS problem with a large number of regressors. Consider the following system[1]:

$$\begin{aligned}\mathcal{S}_2 : y(k) &= 0.7y(k-1)u(k-1)^3 - 0.5y(k-2) \\ &+ 0.6u(k-2)^6 - 0.7y(k-2)u(k-2)^6 \\ &+ 0.2e(k-1) - 0.3u(k-1)^3e(k-2) + e(k)\end{aligned}$$

with $u(k) = \text{sign}(w(k)) \cdot \sqrt[3]{|w(k)|}$, $w(k) \sim \text{WUN}(-1, 1)$, $e(k) \sim \text{WGN}(0, 0.01)$. System $\mathcal{S}_2$ has a nonlinearity degree equal to 7. Accordingly, we have included in the regressor set monomials up to lags $n_u = n_y = 4$ with a maximum degree of $L = 7$, for a total of $N_F = 6435$ regressors.

Table 4.1 reports the results of a Monte Carlo analysis of the performance of the RaMSS and dRaMSS algorithms on this example. Here, the convergence rate CR denotes the fraction of runs that converged before the maximum allowed number of iterations (for the RaMSS) or rounds (for the dRaMSS). A is an accuracy indicator calculated as the fraction of runs resulting in the selection of the correct model structure. The consensus on the winner model is measured by parameter CW, which reports the fraction of processors that have selected the same model (applies only to the dRaMSS). Two elapsed time indices are used: ET_t is simply the sum of all processor times, while ET_p denotes the elapsed time assuming that the processing units can actually operate in parallel (only the former applies to the RaMSS case). Both indices are in seconds. Finally, EN_F is the total number of extracted models, I is the total number of RaMSS iterations (all processors, all rounds), and R denotes the number of rounds required by the distributed algorithm to converge (applies only to the dRaMSS). All indices are averaged over the Monte Carlo runs.

Given the large size of the candidate regressor set, using large initial TIP values to increase the algorithm's exploration capabilities turns out to be inapplicable in the non-distributed setting, both because the number of available samples is smaller than the number of regressors (so that

[1] This system is equivalent to system $\mathcal{S}_4$ in [4], albeit with a higher polynomial degree.

ill-conditioning issues may easily occur), and because of the large computational load involved. This, in turn, calls for a setting of N_p to a relatively large value to allow for a sufficient regressor exploration. A first set of results reported in Table 4.1 has been obtained with $\mu_j(0) = 1/N_p$, $j = 1, \ldots, N_F$ (this setting implies that on average a given regressor should appear at least in one model at the onset of the algorithm) and values of N_p ranging from 500 to 2000. In this case 100 Monte Carlo runs were carried out. Apparently, perfect selection accuracy is achieved only at the cost of extracting a large number of models at each iteration ($N_p = 2000$), for a significant computational cost.

A second set of results (5 runs) of the non-distributed RaMSS was obtained with larger initial TIPs ($\mu_j(0) = 0.25$, $j = 1, \ldots, N_F$). Not surprisingly, smaller N_p values can be employed in this case. However, perfect selection accuracy was obtained only for $N_p = 100$, which resulted in a much higher elapsed time compared to the previous case.

As for the dRaMSS, ten processor units have been employed in the distributed scheme and all the local selected models are redistributed at the end of each round. Given that the size of the candidate regressor set used in each MSS task is 10 times smaller than with the plain RaMSS, much higher initial TIP values can be tolerated. The reported results were obtained with $\mu_j(0) = 0.5$, $j = 1, \ldots, N_F$. Not surprisingly, perfect selection can be achieved with much fewer extracted models for each local processor ($N_p = 10$). Less than 3 MSS rounds were required to achieve convergence. Notice that on average each processor performs I/N_u MSS iterations, which is less than the number of iterations of the centralized RaMSS. This can be partially ascribed to the fact that a lower threshold on the number of MSS iterations can be adopted in the distributed case. Accuracy being equal, the distributed algorithm can converge to the solution faster not only in terms of ET_p, which is one order of magnitude lower, but even in terms of the *total* elapsed time ET_t. This demonstrates that the strategy of splitting and distributing the regressors really allows a more efficient exploration of the model space.

4.3.4 An MSS problem with large candidate regressor set: case 2

With larger MSS problems both computational and selection accuracy problems are experienced with the plain RaMSS, while the dRaMSS still pro-

Table 4.1: *Performance of the RaMSS and dRaMSS algorithms on system $\mathcal{S}_2$.*

Method	N_u	N_p	$\mu_j(0)$	CR	A	CW	ET_t	ET_p	EM	I	R
RaMSS	1	500	$1/N_p$	1.00	0.80	-	144.73	-	101155.82	103.50	-
RaMSS	1	1000	$1/N_p$	1.00	0.93	-	243.08	-	163577.16	83.62	-
RaMSS	1	1500	$1/N_p$	1.00	0.97	-	348.36	-	224748.10	76.56	-
RaMSS	1	2000	$1/N_p$	1.00	1.00	-	476.50	-	285511.15	72.72	-
RaMSS	1	20	0.25	1.00	0.00	-	140.67	-	5994.20	159.80	-
RaMSS	1	50	0.25	1.00	0.60	-	320.41	-	3610.40	37.00	-
RaMSS	1	100	0.25	1.00	1.00	-	615.63	-	9721.17	49.50	-
dRaMSS	10	5	0.5	0.93	0.80	0.40	104.42	11.32	7479.89	991.54	5.14
dRaMSS	10	10	0.5	1.00	1.00	0.81	112.51	12.56	14779.17	850.28	2.43
dRaMSS	10	20	0.5	1.00	1.00	0.95	257.94	30.69	37683.10	982.88	2.39

vides a viable solution. Consider the following system:

$$\mathcal{S}_3 : y(k) = 0.5y(k-1) + 0.8u(k-2)^3 + u(k-2)^6 - 0.05y(k-2)^2 + 0.5 + e(k)$$

with $u(k) = \text{sign}(w(k)) \cdot \sqrt[3]{|w(k)|}$, $w(k) \sim \text{WGN}(0, 0.3)$, $e(k) \sim \text{WGN}(0, 0.01)$. The candidate regressor set includes all monomials with maximum lags $n_u = n_y = 6$ and maximum degree $L = 6$, for a grand total of $N_F = 18564$ terms. The results are reported in Tables 4.2. For each entry, 30 Monte-Carlo runs have been executed. In order to tackle the huge regressor set, 100 processors have been employed in the distributed scheme, and, at the end of each round, only the best 10 locally selected models are redistributed to the other units. As previously mentioned, by limiting the distribution of the local MSS solutions we prevent the local regressor subsets from being overloaded with irrelevant terms, so that their sizes do not grow up to an unmanageable level.

Table 4.2: *Performance of the RaMSS and dRaMSS algorithms on system $\mathcal{S}_3$.*

Method	N_u	N_p	$\mu_j(0)$	CR	A	CW	ET_t	ET_p	EM	I	R
RaMSS	1	100	$1/N_p$	1.00	0.37	-	194.47	-	48388.20	244.50	-
RaMSS	1	500	$1/N_p$	1.00	0.50	-	993.92	-	186469.00	187.80	-
RaMSS	1	1000	$1/N_p$	1.00	0.50	-	1924.01	-	273515.53	137.53	-
dRaMSS	100	10	0.5	1.00	1.00	0.80	922.73	14.03	255807.85	14806.09	4.18

In this challenging scenario the RaMSS algorithm clearly shows its limitations: the model space is too large to be accurately explored, even if many models are extracted at each iteration. In fact, even with $N_p = 1000$ the plain RaMSS algorithm has been able to select the true model structure only half of the times, spending in doing so more than 30 minutes to

converge on average, the slowest execution requiring 2 hours. On the other hand, the distributed algorithm appears to be capable of dealing with this kind of situations thanks to its *divide et impera* approach that breaks the problem into many smaller ones. With a number of generated models as low as 10, the dRaMSS algorithm has been successful in selecting the true model every time, reaching also a very high consensus rate: the true model structure has indeed been selected on average by 80% of the processors. The algorithm converges approximately in 4 rounds, with an average total elapsed time which is slightly greater than 15 minutes (with a maximum of 28 minutes) and an average parallel time of 14 seconds. Each elementary MSS task converges in less than 36 iterations on average.

4.3.5 A system with non-polynomial model structure

A full assessment of the presented method requires that it is also evaluated when the system generating the data does not belong to the model family used in the model structure identification procedure. For this reason, consider now the non-polynomial NARX system

$$\mathcal{S}_4 : y(k) = \exp(-y(k-1)) - 1 + 0.3u(k-1) + e(k),$$

with $u(k) \sim \text{WUN}(-1, 1)$ and $e(k) \sim \text{WGN}(0, 0.01)$. For identification purposes we generated a sequence of 3000 input-output data from system $\mathcal{S}_4$, the first 2000 being used for training and the rest for validation. The candidate regressor set $\mathcal{R}$ contains all the monomials obtained as combinations of the lagged input and output variables with maximum lags $n_u = n_y = 4$ and maximum degree $L = 3$, for a total of $N_F = 165$ regressors. The significance level is set to $\alpha = 0.001$, the performance index is set to $\mathcal{J} = \mathcal{J}_p$, with $K = 1$. The final TIP threshold is equal to $\mu_{\text{thresh}} = 0.75$.

Two different experiments were carried out with the dRaMSS algorithm, both with $N_u = 10$ processors. In the first case, at each round of the distributed scheme all the 10 returned models are shared among the local regressor subsets, while in the second case only the top 5 models are redistributed. Regarding the local RaMSS processes, $N_p = 50$, $N_i = 100$, and the initial TIPs are set to 0.5. One hundred Monte Carlo runs have been computed for each experiment. Table 4.3 summarizes the results (f_{sel} denotes the frequency with which a given model structure is selected). The structures of the selected models are reported in Table 4.4, where $f_{\text{sel}}^{(1)}$ and $f_{\text{sel}}^{(2)}$ denote the selection frequency for all regressors in the two experiments.

Although there does not exist a unique target model in this case, the dRaMSS tends to select combinations of regressors belonging to a quite

small subset of regressors (containing just 8 elements), and yields models with a size ranging from 4 to 6 regressors and all with comparable accuracy (on the training set). In the first experiment the selection looks sharper, with only two different model structures being selected by the dRaMSS in 100 runs. The second case is slightly more various, although the 3 most selected models account for 94% of the runs. Notice that for models $\mathcal{M}_5$, $\mathcal{M}_6$, and $\mathcal{M}_7$, the threshold on the number of RaMSS iterations has been reached, suggesting that convergence may have not been obtained yet.

Table 4.3: *System $\mathcal{S}_4$: dRaMSS model selections.*

Exp.	Model	f_{sel}	$\mathcal{J}_p^{\text{training}}$	$\mathcal{J}_p^{\text{test}}$	CW	ET_t	ET_p	EM	I	R
1	$\mathcal{M}_1$	63%	0.9902	0.9903	0.61	154.28	21.53	18020.83	218.22	2.43
1	$\mathcal{M}_2$	37%	0.9901	0.9902	0.78	143.74	20.19	16045.00	189.51	2.24
2	$\mathcal{M}_2$	56%	0.9901	0.9902	0.71	294.44	38.06	45357.73	281.05	2.86
2	$\mathcal{M}_1$	19%	0.9902	0.9903	0.42	249.34	30.82	35992.63	215.79	2.16
2	$\mathcal{M}_3$	19%	0.9903	0.5631	0.43	235.31	28.92	34776.16	210.53	2.11
2	$\mathcal{M}_4$	3%	0.9901	0.7525	0.42	307.13	38.40	43887.33	266.67	2.67
2	$\mathcal{M}_5$	1%	0.9900	0.9809	0.40	329.53	41.85	51597.00	300.00	3.00
2	$\mathcal{M}_6$	1%	0.9899	0.9901	1.00	219.62	26.31	32215.00	200.00	2.00
2	$\mathcal{M}_7$	1%	0.9882	0.9884	0.30	377.08	44.62	48533.00	300.00	3.00

Table 4.4: *System $\mathcal{S}_4$: structure of the selected models.*

Regressor	$\mathcal{M}_1$	$\mathcal{M}_2$	$\mathcal{M}_3$	$\mathcal{M}_4$	$\mathcal{M}_5$	$\mathcal{M}_6$	$\mathcal{M}_7$	$f_{\text{sel}}^{(1)}$	$f_{\text{sel}}^{(2)}$
$u(k-1)$	✓	✓	✓	✓	✓	✓	✓	100%	100%
$y(k-1)$	✓	✓	✓	✓	✓	✓	✓	100%	100%
$y(k-2)$			✓	✓				0%	22%
$y(k-1)^2$	✓	✓	✓	✓	✓	✓		100%	99%
$y(k-1)y(k-2)$					✓		✓	0%	2%
$y(k-2)^2$							✓	0%	1%
$y(k-1)^3$	✓	✓	✓					100%	94%
$y(k-1)^2y(k-2)$	✓		✓	✓	✓	✓		63%	43%

4.4 The DFS algorithm

The proposed distributed scheme for MSS was also employed to develop an FS method for classification problems, denoted DFS. In principle, the distributed selection scheme can be combined with any FS method of choice [22]. In the work presented here, we employ the SFS and the RFSC algorithms as representatives of wrapper methods, and the ReliefF algorithm as a representative of filter methods.

A pseudocode of the proposed DFS scheme is given in Algorithm 5. The inputs are the set of input/output observation pairs $\mathcal{D} = \{d_1(k), \cdots, d_N(k)\}$, with $d(k) = (\boldsymbol{u}(k), c(k))$, the full set of extended features $\mathcal{R} = \{\varphi_1, \ldots, \varphi_{N_F}\}$ (that includes the original features u_j, $j = 1, \ldots N_f$, as well as products of them up to a given order), the number of feature bins N_u, and the maximum number iterations N_i. As the proposed scheme can be combined with the FS method of choice (represented by a placeholder function FS$(\cdot)$ in the pseudocode, which returns the selected feature subset and the associated performance index), we synthetically denote with Ψ the corresponding vector of input parameters (which is algorithm-dependent). The main loop goes from line 2 to 19. The vertical partitioning in N_u bins is carried out at line 3, by means of function DISTRIBUTE$(\cdot)$. Lines 4 to 12 describe the local selection processes, while at line 13 the N_u^* top ranked models are selected, and the aggregation stage is at line 14. Finally, the termination conditions are given from line 15 to 19 (plus line 9). Though 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 iterations or the exceeding of a prescribed number of iterations (r_{max}) are also employed as premature termination conditions. To avoid the overloading of irrelevant terms, the algorithm provides the option to share only the top ranked local best models at each iteration among the processors in the subsequent iteration. It is important to note that the feature distribution is reshuffled at every iteration to allow a richer exploration of the search space by the algorithm. The algorithm returns the selected feature subset $\mathcal{E}^\star$ along with the corresponding classification performance $\mathcal{J}^\star$.

4.4.1 Discussion on the algorithm convergence

As already explained, at the onset of each DFS iteration 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

Algorithm 5 DFS

Input: $\mathcal{D}$, $\mathcal{R}$, N_u, r_{max}, Ψ.
Output: $\mathcal{E}^\star$, $\mathcal{J}^\star$.

$\mathcal{E}(0) = \emptyset$, $\mathcal{J}^\star = 0$, $\mathcal{E}^\star = \emptyset$
for $r = 1$ **to** r_{max} **do**
  $(\bar{\mathcal{R}}_1, \cdots, \bar{\mathcal{R}}_{N_u}) = \text{DISTRIBUTE}(\mathcal{R}, N_u)$
  for $n_u = 1$ **to** N_u **do**
    $\mathcal{R}_{n_u} = \bar{\mathcal{R}}_{n_u} \cup \mathcal{E}(r-1)$
    $(\mathcal{E}_{n_u}, \mathcal{J}_b)$ = FS(D, $\mathcal{R}_b$, Ψ)
    if $\mathcal{J}_{n_u} > \mathcal{J}^\star$ **then**
      $\mathcal{E}^\star \leftarrow \mathcal{E}_{n_u}$, $\mathcal{J}^\star \leftarrow \mathcal{J}_{n_u}$
      if $\mathcal{J}^\star = 1$ **then return end if** ▷ The current best cannot be improved.
    end if
  end for
  $\mathcal{J}^\star_{vec}(r) = \mathcal{J}^\star$
  $U^\star \leftarrow \text{RANK}(\mathcal{J}_1, \ldots, \mathcal{J}_{N_u}, N_u^\star)$
  $\mathcal{E}(r) \leftarrow \bigcup_{n_u \in U^\star} \mathcal{E}_{n_u}(r)$
  if $\mathcal{E}(r) = \cap_{n_u=1}^{N_u} \mathcal{E}_{n_u}$ **then return end if** ▷ All local models are equal.
  if $r = r_{max}$ **then return** ▷ Maximum number of iterations reached.
  else if $r \geq 3$ **then**
    if $(\mathcal{J}^\star_{vec}(r-1) = \mathcal{J}^\star) \wedge (\mathcal{J}^\star_{vec}(r-2) = \mathcal{J}^\star)$ **then return end if**
  end if ▷ No appreciable improvement over the last 3 iterations.
end for

local processors operate on a relatively small subset of features, so that the corresponding local search space is easily manageable.

Theorem 2. *Assume that Alg. 5 employs an FS method that guarantees the optimality of the solutions of the local problems. Then, the DFS will converge to a locally optimal solution in a finite number of iterations, not greater than the number of tested samples.*

Proof. *In view of the iteration mechanism described above,* $\mathcal{J}^\star(r) \geq \mathcal{J}^\star(r-1)$*, where* $\mathcal{J}^\star(r)$ *denotes the performance of the best local model obtained at the* i*-th iteration. Since* $\mathcal{J}$ *can take only a finite number of values (*$\mathcal{J}$ *equals the percentage of correctly classified samples), and the sequence* $\mathcal{J}^\star(r)$ *is limited from above (it cannot exceed* 1*), it will converge in a finite number of iterations (actually, not greater than the number of tested samples).*

Notice that, due to the discrete nature of the performance index $\mathcal{J}$, there might be multiple models with equal accuracy. Thanks to the convergence condition $\mathcal{J}^\star(r) = \mathcal{J}^\star(r-1)$ (the last termination condition of Alg. 5,

which is extended to 3 consecutive iterations for greater robustness), the DFS will converge to one of the equivalent locally optimal solutions. Notice also that the alternative termination conditions (included in Alg. 5 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 iteration 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 reshuffling of the feature bins that is carried out at the beginning of each iteration. Although the previous discussion on the convergence properties of the DFS scheme holds regardless of this operation, the reshuffling proves to be useful in escaping from local minima of the combinatorial problem, by enhancing the exploration capabilities of the algorithm. As such it greatly enhances the probability of finding the actual global optimum.

4.5 Analysis of the DFS algorithm

This section reports the results of various tests carried out to assess the performance of the proposed distributed FS architecture. Ten numerical data-sets, collected from the UCI machine learning repository [91], are used in the experiments. In the following tests, the original data-sets are preprocessed as follows.

All original features are first normalized in the $[0, 1]$ range, according to (3.9). Then, as already mentioned, the original features are polynomially expanded. A different maximum nonlinearity degree is adopted depending on the size of the considered problem. More precisely, we used $L = 3$ for small data-sets (Bupa and Iris) and $L = 2$ for medium- and big-sized data-sets, with the exception of the Colon, Madelon, Ovarian data-sets for which $L = 1$. The FS task was carried out on the resulting extended feature set $\mathcal{R}$. The main characteristics of these data-sets are presented in Table 4.5.

To evaluate the performance of the proposed algorithm and provide a fair comparison with the literature, two validation methods are used, namely 10-FCV and hold out validation with random $70 - 30$ horizontal data partitioning (70% of the samples used for training and 30% for testing).

For the $70 - 30$ validation, 10-FCV was performed as an inner loop on the training data. The best of the 10 obtained models is selected and tested on the test data. To account for the non-determinism of the algorithm the procedure is further repeated 10 times and the performance averaged.

Besides evaluating the performance in terms of classification accuracy,

we also considered the Cohen's Kappa rate.

Table 4.5: *Main characteristics of the considered data-sets.*

Data-set	N	N_f	N_F	N_c
Bupa	345	6	84	2
Colon	62	2000	2000	2
HillValley	606	100	5151	2
Ionosphere	351	34	595	2
Iris	150	4	70	3
Madelon	4400	500	500	2
Musk1	476	166	14028	2
Ovarian	253	15154	15154	2
Sonar	208	60	1891	2
Vehicle	846	18	190	4
WDBC	569	30	496	2
Wine	178	13	105	3

4.5.1 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. More precisely, the 10-FCV is performed in the inner loop for each candidate feature subset so that the feature whose inclusion improves the performance the most within the 10-FCV is included in the model. 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 [20], the initial model distribution was defined so that the probability of selecting any given feature is equal to $\mu_0 = 1/N_F$. 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 $\bar{\mu} = 0.7$.

The only parameter to design for the ReliefF algorithm is the number of features to be retained after ranking. This parameter is set to $\frac{N_F}{N_u}\frac{r_{perc}}{100}$, where r_{perc} denotes the percentage of retained features. Parameter r_{perc} was set to 35 for all data-sets, *i.e.* the 35% top ranked features are selected from each bin at every iteration.

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_F, 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 data-sets with $N_F \leq 1000$, in order to get less than 100 features per bin. Larger N_u values were used for the other data-sets. 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 $N_u \geq N_F/N$.

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 data-sets, 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 ($\mu_0 = 1/N_F$), 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 data-sets the number of bins was set so as to generate bins of approximately size 25 and 50 for small and medium cases, respectively. Larger bins are used for the other data-sets.

Table 4.6 reports settings for N_u that were reported to perform success-

fully in this study.

Table 4.6: *Number of bins (N_u) employed for each data-set, as a function of the FS algorithm.*

Data-set	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

When 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.

4.5.2 Sensitivity analysis

Effects of the randomized nature of the DFS scheme

The DFS scheme involves a random shuffling and the distribution of the features in the N_u 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. We considered the WDBC data-set employing the same training-test partitioning of the data for this purpose, and tested both the non-distributed and distributed architectures for all 3 FS methods. In the latter case 10 feature bins are employed. The total feature set amounts to $N_F = 496$ terms (30 original features, polynomial expansion of degree $L = 2$). 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 itself. 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 view of this, we used $N_p = 100$ for the non-distributed 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 4.5 shows the distribution of the classification error on the test set with the non-distributed (RFSC, SFS, ReliefF) and the distributed (dRFSC, dSFS, dReliefF) approaches. A strong dominance of the distributed approach over the non-distributed 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 non-distributed 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 non-distributed RFSC operates on a search space of 2^{N_F} 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 non-distributed 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.

Effects of the partitioning of the data for training and testing

The classification accuracy depends on how the partitioning of the data-set in the training and testing sets is performed. To analyze this effect we carried out 100 Monte Carlo simulations on the WDBC data-set, regenerating every time the training-testing partition. In this analysis, we compared the results obtained with the distributed dRFSC, dSFS and dReliefF architectures. To provide a fair comparison, all 3 algorithms were trained and tested on the same data for each Monte Carlo simulation. Fig. 4.6 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.

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

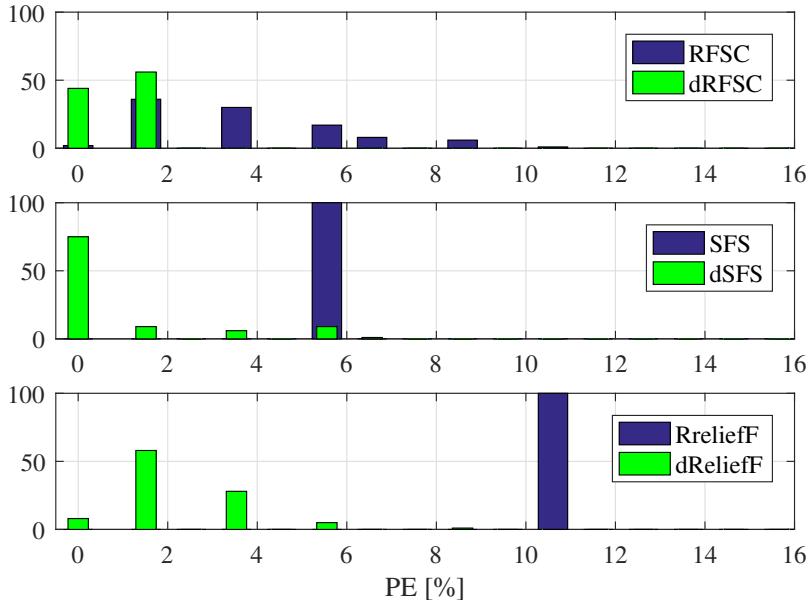

Figure 4.5: *Test error performance of the RFSC, dRFSC, SFS, dSFS, ReliefF and dReliefF over 100 Monte Carlo runs performed on the same training/test data split of the WDBC data-set.*

performance.

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 data-set using a fixed training-test data split, but different bin settings, $N_u = 10, 20, \ldots, 80$). Five simulations were carried out for each case, assuming a fixed maximum number of iterations. The average results are reported in Fig. 4.7. Apparently, the error first decreases with the number of bins, but after some point it starts to increase again. In other words, one obtains worse performance when there are either too few bins (because their size is too large) or too many (because many redundant features are shared at the end of each DFS round).

4.5.3 Comparative analysis

Non-distributed vs. distributed FS scheme

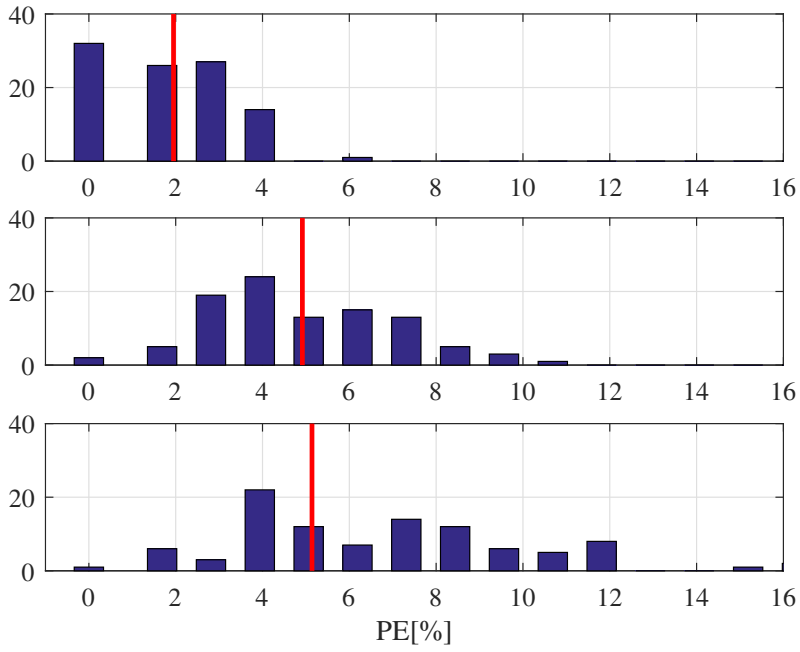

Figure 4.6: *Monte Carlo evaluation of the test error with dRFSC (top), dSFS (middle) and dReliefF (bottom) for 100 different training/test data partitions.*

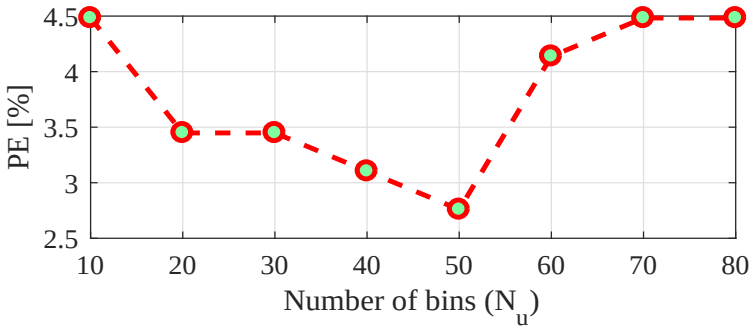

Figure 4.7: *Test error dependence on N_u.*

Tables 4.7 and 4.8 report extensive simulation results obtained on the twelve considered data-sets using all the mentioned non-distributed and

Table 4.7: *Comparative analysis of the non-distributed and distributed schemes for the RFSC, SFS and ReliefF algorithms (Part 1)*

Method	Data-set	N_{fs}	N_{Fs}	$\mathcal{J}$	K	ET
RFSC	Bupa	5.8	7.4	0.7884	0.4950	14.6
dRFSC		5.8	4.3	0.7945	0.5701	1.9
SFS + 5NN		3.9	4.2	0.6228	0.2104	24.2
dSFS + 5NN		3.4	2.7	0.6576	0.3424	12.6
ReliefF + 5NN		3	6	0.5704	0.1154	0.4
dReliefF + 5NN		5	6	0.6724	0.4348	0.2
RFSC	Colon	-	-	-	-	-
dRFSC		2.3	2.3	0.8421	0.6209	0.3
SFS + 5NN		3	3	0.7579	0.2963	605
dSFS + 5NN		1.25	1.25	0.8026	0.4673	15.8
ReliefF + 5NN		21	21	0.5529	0.5529	0.37
dReliefF + 5NN		21	21	0.8526	0.6211	0.01
RFSC	HillValley	8.3	3.7	0.9277	0.8552	18.6
dRFSC		5.6	2.8	0.9859	0.9846	7.7
SFS + 5NN		5.6	4.5	0.5549	0.1095	658.7
dSFS + 5NN		4.3	4.5	0.5604	0.1214	250.4
ReliefF + 5NN		39	120	0.4890	-0.0207	30.4
dReliefF + 5NN		39	120	0.5274	0.0616	2.8
RFSC	Ionosphere	16.4	14.7	0.9330	0.8541	57.0
dRFSC		13.5	11.8	0.9487	0.8861	2.4
SFS + 5NN		4.7	2.8	0.9028	0.7818	129.8
dSFS + 5NN		3.9	2.1	0.9198	0.8215	21.6
ReliefF + 5NN		10	20	0.9004	0.7699	2.3
dReliefF + 5NN		18	20	0.9403	0.8653	0.5
RFSC	Iris	3.2	6.1	0.9666	0.9500	10.0
dRFSC		2.8	4.5	0.9902	0.9900	1.3
SFS + 5NN		2.6	2.2	0.9600	0.9480	6.8
dSFS + 5NN		1.7	1.7	0.9800	0.9745	2.10
ReliefF + 5NN		2	5	0.9600	0.9457	0.08
dReliefF + 5NN		4	5	0.9800	0.9740	0.06
RFSC	Madelon	3.5	3.5	0.6160	0.2212	370.3
dRFSC		3.5	3.5	0.6347	0.2692	34.5
SFS + 5NN		7.5	7.5	0.8966	0.7933	326.1
dSFS + 5NN		7.4	7.4	0.9021	0.8043	60.3
ReliefF + 5NN		18	18	0.8910	0.7820	42.2
dReliefF + 5NN		18	18	0.9083	0.8166	4.7

distributed FS algorithms. For each case we report the number of selected original (N_{fs}) and extended (N_{Fs}) features (clearly, $N_{fs} \leq N_f$ and $N_{Fs} \leq N_r$), the accuracy performance index $\mathcal{J}$, the Kappa rate K, and the elapsed time ET.

Table 4.8: *Comparative analysis of the non-distributed and distributed schemes for the RFSC, SFS and ReliefF algorithms (Part 2)*

Method	Data-set	N_{fs}	N_{Fs}	$\mathcal{J}$	K	ET
RFSC	Musk1	46.2	23.2	0.8132	0.6201	51.6
dRFSC		48	22.5	0.8216	0.6372	40.7
SFS + 5NN		20	11	0.8531	0.7049	11712.2
dSFS + 5NN		15	7.5	0.8671	0.7360	1328.9
ReliefF + 5NN		84	327	0.8461	0.6937	41.9
dReliefF + 5NN		84	327	0.8559	0.7143	4.2
RFSC	Ovarian	-	-	-	-	-
dRFSC		5	5	0.9653	0.9251	2.23
SFS + 5NN		2	42	0.9868	0.9710	3344
dSFS + 5NN		2.25	2.25	0.9868	0.9710	130
ReliefF + 5NN		62	62	1	1	16.4
dReliefF + 5NN		62	62	1	1	14.4
RFSC	WDBC	11.5	10.3	0.9827	0.9621	66.0
dRFSC		8.1	5.2	0.9860	0.9674	8.5
SFS + 5NN		6.1	3.5	0.9507	0.8942	129.9
dSFS + 5NN		4.4	2.5	0.9597	0.9129	32.2
ReliefF + 5NN		10	17	0.9596	0.9116	4.1
dReliefF + 5NN		21	17	0.9825	0.9621	0.8
RFSC	Sonar	25.8	18.7	0.8806	0.8101	72.0
dRFSC		9.6	5.1	0.9090	0.8164	1.21
SFS + 5NN		8.5	4.8	0.7167	0.5346	624.6
dSFS + 5NN		6.9	3.7	0.8073	0.6093	131.1
ReliefF + 5NN		19	44	0.7753	0.5460	3.1
dReliefF + 5NN		42	44	0.8075	0.7640	1.4
RFSC	Vehicle	10.9	16.6	0.7888	0.7186	162.4
dRFSC		8.1	5.2	0.8003	0.7339	25.3
SFS + 5NN		10.3	8.5	0.7010	0.6558	130.2
dSFS + 5NN		9.6	7.8	0.7292	0.6886	57.6
ReliefF + 5NN		9	13	0.6913	0.6436	3.3
dReliefF + 5NN		10	13	0.7671	0.7314	1.2
RFSC	Wine	7.3	7.5	0.9944	0.9916	12.0
dRFSC		3.6	2.2	0.9944	0.9916	1.16
SFS + 5NN		6.7	4.6	0.9493	0.9306	40.1
dSFS + 5NN		5.6	3.6	0.9777	0.9698	11.2
ReliefF + 5NN		6	7	0.9323	0.9080	0.2
dReliefF + 5NN		8	7	0.9944	0.9916	0.1

Apparently, there is a systematic gain in using the distributed scheme as opposed to the non-distributed 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 Tables 4.7 and 4.8 reveals that the dSFS and dRFSC generally provide smaller models than their non-distributed 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 non-distributed and distributed schemes. The dSFS typically returns the smaller models, whereas the dRFSC and the 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 due to the adoption of the DFS scheme, implying that the searching mechanism is much more efficient than in the non-distributed case.

Comparison with state-of-the-art 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, [124], [125], [112], [23], [83], and [17]). Tables 4.10 and 4.11 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_{Fs}) of the obtained classifiers for 10 UCI datasets (the larger Colon and Ovarian data-sets are considered separately).

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 4.11) 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.

Finally, Table 4.12 reports the results (in terms of model size and classification performance) obtained for the two large microarray data-sets (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 [17].

Table 4.9: *Comparative analysis: performance (J and K).*

FS Method + Classifier	Bupa		HillValley		Ionosphere		Iris		Madelon	
	J	K	J	K	J	K	J	K	J	K
ACO + PMC [112]	0.6725	0.3259	-	-	0.9373	0.8604	0.9600	0.9400	-	-
Att.-Cls. WM + DGC + [23]	0.6744	0.3076	-	-	0.9311	0.8487	0.9533	0.9300	-	-
Att. WV + DGC [23]	0.6525	0.2220	-	-	0.6724	0.1142	0.9533	0.9300	-	-
- + KNN [23]	0.6066	0.1944	-	-	0.8518	0.6494	0.9400	0.9100	-	-
- + KNN-A [23]	0.6257	0.2021	-	-	0.9372	0.8595	0.9533	0.9300	-	-
- + DW-KNN [23]	0.6376	0.2645	-	-	0.8747	0.7083	0.9400	0.9100	-	-
- + Cam-NN [23]	0.5962	0.1024	-	-	0.7379	0.5145	0.9467	0.9200	-	-
- + CNN [23]	0.6316	0.2571	-	-	0.8917	0.7526	0.9267	0.8900	-	-
SSMA + SFLDS [23]	0.6426	0.2731	-	-	0.9088	0.7986	0.9533	0.9300	-	-
forward FS + LDA [83]	0.6110	-	-	-	0.8530	-	0.9630	-	-	-
backward FS + LDA [83]	0.6430	-	-	-	0.9090	-	0.9370	-	-	-
PSO + LDA [83]	0.6520	-	-	-	0.9220	-	0.9700	-	-	-
PSO(4-2) + 5NN [125]	-	-	0.5777	-	0.8727	-	-	-	0.7886	-
PSOMulti + 5NN [124]	-	-	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

Table 4.10: *Comparative analysis: performance (J and K).*

FS Method + Classifier	Musk1		Sonar		Vehicle		WDBC		Wine	
	J	K	J	K	J	K	J	K	J	K
ACO + PMC [112]	-	-	0.9087	0.8164	-	-	-	-	0.9755	0.9659
Att.-Cls. WM + DGC + [23]	-	-	0.8487	0.6943	0.7116	0.6152	-	-	0.9731	0.9590
Att. WV + DGC [23]	-	-	0.7694	0.5187	0.6572	0.5437	0.9619	-	0.9706	0.9552
- + KNN [23]	-	-	0.8307	0.6554	0.7175	0.6233	-	-	0.9549	0.9318
- + KNN-A [23]	-	-	0.8798	0.7549	0.6879	0.5844	-	-	0.9663	0.9491
- + DW-KNN [23]	-	-	0.8648	0.7248	0.7258	0.6342	-	-	0.9438	0.9152
- + Cam-NN [23]	-	-	0.7743	0.5364	0.6170	0.4905	-	-	0.9497	0.9228
- + CNN [23]	-	-	0.8940	0.7861	0.7423	0.6563	-	-	0.9663	0.9491
SSMA + SFLDS [23]	-	-	0.8079	0.6100	0.9145	0.5273	-	-	0.9438	0.9145
forward FS + LDA [83]	-	-	0.7610	-	0.7500	-	-	-	0.9660	-
backward FS + LDA [83]	-	-	0.8550	-	0.7900	-	-	-	0.9990	-
PSO + LDA [83]	-	-	0.9050	-	0.7940	-	-	-	1.0000	-
PSO(4-2) + 5NN [125]	0.8494	-	0.7816	-	-	-	0.9398	-	-	-
PSOMulti + 5NN [124]	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 4.11: *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 4.12: *Comparative analysis: performance (N_{fs} and J).*

FS Method + Classifier	Colon		Ovarian	
	N_{fs}	J	N_{fs}	J
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

4.6 The D^{2}CORFS Algorithm

The high dimensional nature of bioinformatic data poses a severe challenge on machine learning methods. For example, microarrays allow to simultaneously measure the expression levels of a large number of genes, so that the resulting datasets are characterized by a large number of features (tens of thousands of genes may be considered) and a very limited sample size [104]. Most of the genes provide little or no information useful for classification purposes, and it is particularly important to detect the smallest subset of features (referred to as *biomarkers*) that provide sufficient information to separate the classes represented in the dataset (which could distinguish cancerous and noncancerous samples, or identify different types of cancer [60]).

The highly unbalanced dimensions of microarray datasets greatly complicate the FS task, and unsatisfactory classification performances are often obtained with standard methods [67], [3]. Indeed, large feature vectors significantly slow down the learning process, since the complexity of the FS problem grows exponentially with the number of features. At the same time, in combination with the small number of samples, this may cause the classifier to overfit the training data, thus compromising model generalization [72]. Besides their unbalanced dimensions, microarray data are often

corrupted with noise, which further aggravates the analysis. For all these reasons, specialized FS techniques must be developed to appropriately handle this type of datasets.

The dMSS scheme provides a powerful approach for the exploration of the huge space of feature subsets, but we cannot complement it with a wrapper method to operate at each processor, due to the excessive computational effort required. In addition, by inspecting the results presented in Table 4.12 the filter method outperforms both wrapper methods. We therefore adopt here (see also [21]) a multivariate filter-based FS strategy based on a randomized model generation policy as in the RFSC and the distance correlation (dCor) criterion (see Section 2.8) for model evaluation.
An important feature of the dMSS scheme is that the FS procedure is carried out on smaller datasets with much less disproportionate dimensions, which has significant implications on the accuracy and robustness of the results. In particular, the bias effect of the dCor associated to high-dimensional problems is much de-emphasized thanks to this strategy. Due to the issue of dimensional imbalance, only the best local solution is shared with the feature bins at each iteration.

A pseudocode of the distributed scheme is provided below under the name D^{2}CORFS (Distributed dCor-based FS). The local FS method applied at each processor is denoted DCORFS algorithm (see Section 4.6.1). The input parameters N_p, N_i, $\mu^{(0)}$, $\bar{\mu}$, and ϵ are actually arguments of the DCORFS function and will be explained later. The other inputs are the set of input/output observation pairs $\mathcal{D} = \{(\mathbf{u}_s^{(k)}, c^{(k)}), k = 1, \ldots, N\}$, the full set of features $\mathcal{R}$, the number of feature bins N_u, and the maximum number of allowed rounds N_F. Notice that the problem addressed here (*i.e.*, microarrays) is too high-dimensional to allow a polynomial extension of the original features. Therefore, $\mathcal{R} = \mathcal{F}$. The algorithm returns the selected feature subset $\mathcal{E}^\star \subseteq \mathcal{R}$, along with its dCor value $\mathcal{R}_N^\star$. Notice that the procedure is terminated if a model with the maximum possible dCor is obtained (line 10), or if all local models are equal (line 13), or if the maximum number of rounds has been reached (line 15), or finally if no improvement is achieved for 3 consecutive rounds (line 18). A detailed experimental study of the proposed distributed algorithm will be presented in Section 4.7 after introducing the DCORFS algorithm.

Algorithm 6 D^{2}CORFS

Input: $\mathcal{D}$, $\mathcal{R}$, N_u, N_F, N_i, N_p, $\mu^{(0)}$, $\bar{\mu}$, ϵ.
Output: $\mathcal{E}^\star$, $\mathcal{R}^\star_N$.

$\mathcal{R} = \mathcal{R}^{(0)}_1 \cup \cdots \cup \mathcal{R}^{(0)}_{N_u}$
$\mathcal{E}(0) = \emptyset$, $\mathcal{E}^\star = \emptyset$, $\mathcal{R}^\star_N = 0$
for $r = 1$ **to** r_{max} **do**
    $\mathcal{E}_{n_u} \leftarrow s^\star$
    for $n_u = 1$ **to** N_u **do**
        $\mathcal{R}_{n_u} = \mathcal{R}^{(0)}_{n_u} \cup \mathcal{E}$
        $(\mathcal{E}_{n_u}, \mathcal{R}_{n_u})$ = DCORFS(D, $\mathcal{R}_b$, N_i, N_p, $\mu^{(0)}$, $\bar{\mu}$, ϵ)
        if $\mathcal{R}_{n_u} > \mathcal{R}^\star_N$ **then**
            $\mathcal{E}^\star \leftarrow \mathcal{E}_{n_u}$, $\mathcal{R}^\star_N \leftarrow \mathcal{R}_{n_u}$
            if $\mathcal{R}^\star_N = 1$ **then return end if**
        end if
    end for
    if $\cup_{n_u} \mathcal{E}_{n_u} = \cap_{n_u} \mathcal{E}_{n_u}$ **then return end if**
    $\mathcal{R}^\star_{Nvec}(r) = \mathcal{R}^\star_N$
    if $r = r_{max}$ **then**
        return
    else if $r \geq 3$ **then**
        if $(\mathcal{R}^\star_{Nvec}(r-1) = \mathcal{R}^\star_{Nvec}(r-2) = \mathcal{R}^\star_N)$ **then**
            return
        end if
    end if
end for

4.6.1 The DCORFS algorithm

The same randomized approach of the RFSC is employed in the DCORFS. A probabilistic distribution of the model structures is progressively refined based on the information on the individual extended features gathered from populations of models extracted from the same distribution. The main difference is in the criterion used to evaluate models, which does not require the development and assessment of a classifier but only the calculation of the unbiased dCor index given by (2.46). Unlike any other (multivariate) filter method and according to the randomized scheme, the features are evaluated not just individually or in pairs, but based on populations of extracted models. The objective of the DCORFS algorithm is to find the subset of features $\mathcal{E}^* \subseteq \mathcal{R}$ that maximizes the dCor index $\mathcal{R}_N(\varphi_s^{(k)}, c^{(k)})$.
A pseudocode of the DCORFS algorithm is given in Algorithm 7.

Algorithm 7 DCORFS

Input: $\mathcal{D}$, $\mathcal{R}_b$, N_i, N_p, $\mu^{(0)}$, $\bar{\mu}$, ϵ
Output: $\mathcal{E}^\star$, $\mathcal{R}_N^\star$

$\boldsymbol{\varphi}_{\mathcal{E}^\star} = \emptyset$,
for $j = 1$ **to** $|\mathcal{R}_b|$ **do**
    $\mu_j \leftarrow \mu^{(0)}$ TIP initialization
end for
for $t = 1$ **to** N_i **do**
    for $p = 1$ **to** N_p **do**
        $\phi_p \sim \mathcal{P}_\phi$ Extract sample feature subset
        $\mathcal{R}_N^p \leftarrow \mathcal{R}_N(\phi_p, c)$ Compute dCor with (2.46)
    end for
    $\mathcal{R}_{\text{N max}} \leftarrow \max(\mathcal{R}_N^1, \dots \mathcal{R}_N^{N_p})$
    $\bar{\mathcal{R}}_N = \frac{1}{N_p} \sum_{p=1}^{N_p} \mathcal{R}_N^p$
    $\gamma = \frac{1}{\lambda(\mathcal{R}_{\text{N max}} - \bar{\mathcal{R}}_N) + 0.1}$
    for $j = 1$ **to** $|\mathcal{R}_b|$ **do**
        $\mathcal{I}_j \leftarrow \frac{\sum_{p|\varphi \in \phi_p} \mathcal{R}_N^p}{\sum_{p|\varphi \in \phi_p} 1} - \frac{\sum_{p|\varphi \notin \phi_p} \mathcal{R}_N^p}{\sum_{p|\varphi \notin \phi_p} 1}$
        $\mu_j \leftarrow \text{sat}(\mu_j + \gamma \mathcal{I}_j)$ TIP update
    end for
    if $\max_{j=1,\dots,|\mathcal{R}_b|} |\mu_j(i) - \mu_j(i-1)| \leq \epsilon$ **then**
        break
    end if
end for
$\mathcal{E}^\star \leftarrow \emptyset$
for $j = 1$ **to** $|\mathcal{R}_b|$ **do**
    if $\mu_j \geq \bar{\mu}$ **then** $\mathcal{E}^\star \leftarrow \mathcal{E}^\star \cup \{\varphi_j\}$ **end if**
end for
$\mathcal{R}_N^\star = \mathcal{R}_N(\boldsymbol{\varphi}_{\mathcal{E}^\star}, c)$

The required inputs are the observations $\{\boldsymbol{u}_s^{(k)}, c^{(k)}\}$, $k = 1, \dots, N$, the set of features $\mathcal{R}_b$ on which to perform the search, the maximum number of iterations N_i, the number N_p of feature subsets to be extracted from the current distribution at each iteration, the initial value of the TIPs $\mu^{(0)}$, the acceptance threshold $\bar{\mu}$, and a convergence threshold ϵ. The algorithm returns the selected feature subset $\mathcal{E}^\star$, along with its dCor value $\mathcal{R}_N^\star$.

4.7 Analysis of the D^{2}CORFS algorithm

In this section we report the results of different experiments carried out to assess the performance of the proposed algorithm on four well known microarray benchmarks: Prostate, Lung, Leukemia and Ovarian cancer. All four datasets are biclass problems. For example, the Prostate dataset has

the *Relapse* and *non-Relapse* classes. The Lung data set contains 12533 genes and distinguishes between the *Mesothelioma* and *ADCA* class. The Leukemia data cover 7129 genes used to distinguish between *Acute Lymphoblastic Leukemia* (ALL) and *Acute myeloid leukemia* (AML). All datasets except the Ovarian cancer are originally divided into training and test data. To be comparable with the literature, we split the Ovarian data randomly assigning 70% of the samples for training and 30% for testing, preserving the original class distributions on both data sets. The main characteristics of the considered datasets are given in Table 4.13, where NP and NN are the total number of samples belonging to class 1 and 2, respectively, and σ_{TR} and σ_{TE} are the data skew ratios. The latter indicate the class imbalance, measured as $\sigma = \frac{NP}{NN}$ in the training and test data, respectively. This information is important, since it is related to the accuracy and reliability of classification algorithms across classes [59].

Table 4.13: *Dataset characteristics.*

Dataset	N_F	N_c	Training (*NP / NN*)	Test (*NP / NN*)	σ_{TR}	σ_{TE}
Prostate	12600	2	102 (52 / 50)	34 (25 / 9)	0.96	0.36
Lung	12533	2	32 (16 / 16)	149 (15 /134)	1	8.93
Leukemia	7129	2	38 (13 / 25)	34 (10 / 24)	1.92	2.4
Ovarian	15154	2	177 (113 / 64)	76 (49 / 27)	0.566	0.551

The original features have been normalized in the $[0, 1]$ range according to Eq. (3.9). For the Prostate, Lung and Ovarian datasets, a feature screening based on dCor was applied as a preprocessing step. The significance level was set to $\alpha_d = 0.95$ for the Prostate, $\alpha_d = 0.90$ for the Lung and to $\alpha_d = 0.99$ for Ovarian dataset, in order to reduce the features by a factor 4. To evaluate the performance of the proposed method we trained different classifiers on the selected features, namely support vector machines (SVM) with linear decision boundaries, the naive Bayes (NB) classifier and k-nearest neighbors (kNN, with $k = 5$). We employed alternative evaluation criteria ($e.g.$ TPR, TNR, $Gmean$, $Fscore$), especially designed to account for class imbalanced data.

The initial parameter setup for the D^{2}CORFS in the experiments is as follows: a maximum of $r_{max} = 5$ rounds is allowed for the distributed search scheme and the number of bins is set to $N_u = 2N_F/N$. As for the DCORFS algorithm operating on each feature bin, the number of iterations is limited to $N_i = 100$, the number of feature subset extractions at each

iteration is set to $N_p = 100$, the initial TIPs are set to $\mu_0 = 1/|\mathcal{R}_b|$, $\epsilon = 0.001$, and the acceptance threshold is $\bar{\mu} = 0.98$. The proposed algorithm was implemented in Matlab (version 2016a) and executed on an Intel(R) Core i7-3630QM machine, with 2.4GHz CPU, 8GB of RAM, and a 64-bit Operating System.

4.7.1 Detailed performance analysis

Table 4.14 reports the results obtained with the linear SVM, NB and kNN classifiers, assessed in terms of classification accuracy on the training (J_{TR}) and the test data (J_{TE}), as well as using the mentioned alternative metrics. Apparently, the FS procedure selected very compact models in all cases, with 4 features at most, and relatively high classification accuracy was obtained with SVMs (the results compare quite favorably with the literature, as shown later in Table 4.16). Interestingly enough, though the Leukemia data are imbalanced in favor of negative samples, the obtained classifier scores better on the TPR index, than on the TNR. The computational time is sufficiently low, the lowest computational cost having being observed for the Lung dataset. Indeed, the Lung dataset has the smallest number of samples, which in turn causes the size of the feature bins to be particularly small resulting in a very high computational efficiency. Conversely, the much higher computational time observed for the Ovarian data can be ascribed to the larger size of the local FS problems both in terms of samples and features.

4.7.2 Redundancy analysis of the obtained models

We also performed an *a posteriori* analysis on the obtained models, both in terms of the dCor measure and the classifier performance, to investigate the presence of redundant biological information. The test is performed by removing one gene at a time from $\mathcal{E}^\star$, and re-evaluating the reduced feature subset. Table 4.15 reports the obtained results. By inspecting Table 4.15, it is apparent that the dCor index indicates the absence of redundant terms in all selected models. Conversely, the performance index on the training set J_{TR} is not necessarily worse for reduced models. Still, the corresponding performance on the test data is always worse (or equal at most).

Table 4.14: *Performance analysis of the D^2CORFS on the tested datasets.*

Dataset	Model	Time [s]	Classifier	J_{TR}	J_{TE}	TNR_{TE}	TPR_{TE}	$Gmean_{TE}$	$Fscore_{TE}$
Prostate	$\mathcal{E}^\star = \{\varphi_{4282}, \varphi_{6185}, \varphi_{8965}, \varphi_{10494}\}$	876.2	SVM	0.9314	0.9706	1.00	0.9600	0.9798	0.9796
			KNN	0.9510	0.9118	0.8889	0.9200	0.9043	0.9042
			NB	0.9314	0.8529	1.00	0.8000	0.8944	0.8889
Lung	$\mathcal{E}^\star = \{\varphi_{7765}, \varphi_{12308}\}$	194.77	SVM	0.9688	0.9933	1.00	0.9333	0.9661	0.9655
			KNN	1.00	0.9866	0.9925	0.9333	0.9629	0.9620
			NB	1.00	0.9866	0.9925	0.9333	0.9629	0.9620
Leukemia	$\mathcal{E}^\star = \{\varphi_{4781}, \varphi_{4847}, \varphi_{5039}\}$	373.9	SVM	1.00	0.9419	0.9167	1.00	0.9574	0.9565
			KNN	1.00	0.9118	0.8750	1.00	0.9354	0.9333
			NB	1.00	0.9118	0.8750	1.00	0.9354	0.9333
Ovarian	$\mathcal{E}^\star = \{\varphi_{182}, \varphi_{1680}, \varphi_{2237}\}$	2582.1	SVM	1.00	1.00	1.00	1.00	1.00	1.00
			KNN	1.00	1.00	1.00	1.00	1.00	1.00
			NB	1.00	1.00	1.00	1.00	1.00	1.00

Table 4.15: *Redundancy analysis on the obtained models (obtained with D^2CORFS and SVM).*

Dataset	Feature subset	$\mathcal{R}_{n_u}$	J_{TR}	J_{TE}
Prostate	$\mathcal{E}^\star$	0.8253	0.9314	0.9706
	$\mathcal{E}^\star \setminus \{\varphi_{4282}\}$	0.8129	0.9216	0.8529
	$\mathcal{E}^\star \setminus \{\varphi_{6185}\}$	0.7875	0.9020	0.9118
	$\mathcal{E}^\star \setminus \{\varphi_{8965}\}$	0.8054	0.9412	0.9706
	$\mathcal{E}^\star \setminus \{\varphi_{10494}\}$	0.8169	0.9020	0.9412
Lung	$\mathcal{E}^\star$	0.9311	0.9688	0.9933
	$\mathcal{E}^\star \setminus \{\varphi_{7765}\}$	0.7809	0.7500	0.9329
	$\mathcal{E}^\star \setminus \{\varphi_{12308}\}$	0.8919	0.9063	0.9664
Leukemia	$\mathcal{E}^\star$	0.9816	1.0000	0.9412
	$\mathcal{E}^\star \setminus \{\varphi_{4781}\}$	0.9709	0.9737	0.9118
	$\mathcal{E}^\star \setminus \{\varphi_{4847}\}$	0.9809	1.0000	0.3529
	$\mathcal{E}^\star \setminus \{\varphi_{5039}\}$	0.9699	1.0000	0.9118
Ovarian	$\mathcal{E}^\star$	0.9413	1.0000	1.0000
	$\mathcal{E}^\star \setminus \{\varphi_{182}\}$	0.9194	0.9774	0.9868
	$\mathcal{E}^\star \setminus \{\varphi_{1680}\}$	0.8741	0.9831	1.0000
	$\mathcal{E}^\star \setminus \{\varphi_{2237}\}$	0.9261	1.0000	0.9868

Apparently, using the dCor index as a selection criterion provides models with good generalization properties, and, what is more, the proposed algorithm has better generalization properties than a wrapper method using SVM, since the search process of the latter would be driven by performance J_{TR}. The only exception to the mentioned behavior is observed for the Prostate case, where a reduced model slightly outperformed the complete model on the training data, but without any apparent benefit on the test data.

4.7.3 Distribution of the feature values

A more detailed analysis of the obtained models, with focus on the selected features, reveals several interesting aspects. Fig. 4.8 shows the values of the selected features for all samples, divided by class and training/test subset. If the data distribution of the extracted features on the training set remains the same or similar on the test part, then it is expected to have comparable performance of the extracted model on the training and test splits. On the contrary, if the distribution of the features within the extracted model on the test part differs (significantly) with respect to the training data, then it is expected to have different performances on the training and the test parts.

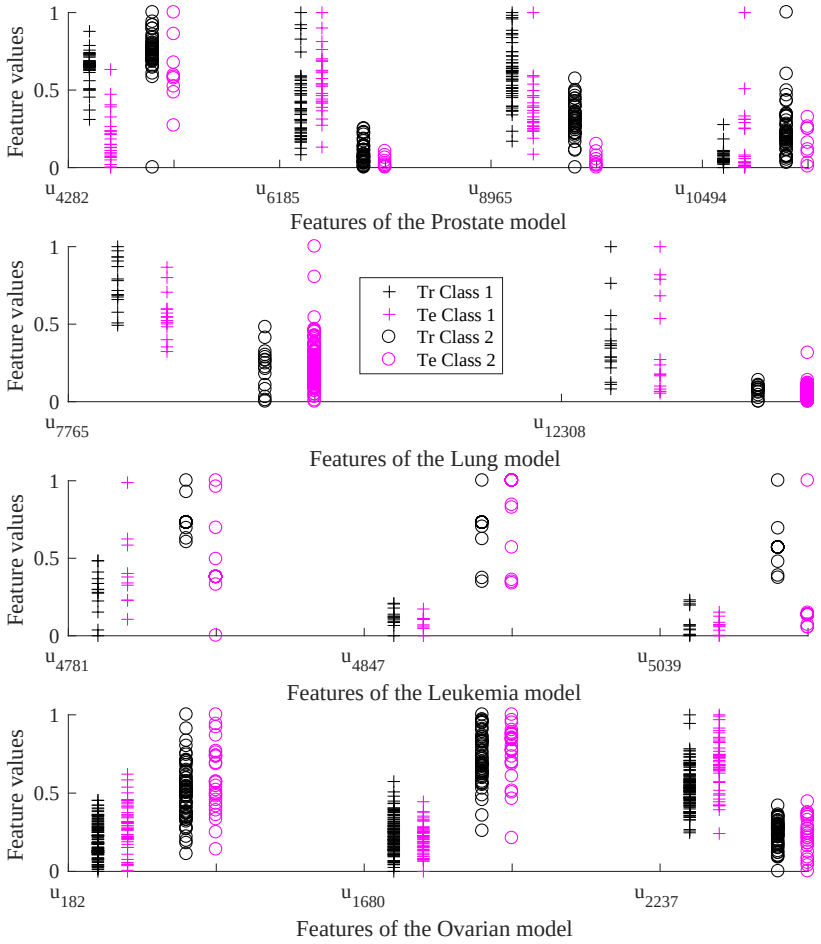

Figure 4.8: *Distribution of the feature values of the models presented in Table 4.14.*

Models characterized by perfect performance on the training set have features with little or no overlap between different classes (see, *e.g.*, Lung, Leukemia, and Ovarian datasets). This indicates that a perfectly legitimate model selection was operated based on the available information (training set). Unfortunately, for some of these datasets (Lung, Leukemia) the feature value distributions over classes turns out to be different on the test set, resulting in some small classification error.

Such imprecision could not have been avoided based on information

gathered from the training set, if not by luck. In other words, if a subset of features provides good class discrimination on the training set, it will provide good generalization only if the feature value distributions on the training and the test subsets are similar. It may well happen that better generalization is achieved through a model which is not optimal on the training set.

4.7.4 Complexity analysis

We here analyse the computational complexity of the proposed algorithm as a function of the problem size (*i.e.*, the number of features N_F and samples N), and of some crucial design parameters (*e.g.*, the number of rounds of the D^{2}CORFS algorithm r_{max}, the number of iterations of the DCORFS algorithm N_i, and the number of the feature bins N_u). Let $\mathcal{F}$ be the full set of N_F features. Clearly, the model space to be explored grows exponentially with the number of features (the number of possible non-empty subsets of $\mathcal{F}$ is $2^{N_F} - 1$. At every iteration, each processor executes the DCORF algorithm on its feature bin, which performs three tasks: feature subset extraction and evaluation, regressor evaluation, and RIP update. The first task requires $N_p N'_F$ operations, where N_p is the number of feature subsets to be evaluated, and $N'_F \simeq N_F/N_u$ is the number of features in each feature bin. The evaluation of a feature subset by means of Equation (2.46) is of order $O(N^2 N'_F)$, whereas the calculation of all the indices $\mathcal{I}_j$, $j = 1, \ldots, N'_F$ requires an order of $N_p N'_F$ operations. Finally, the RIP update is linear in the number of features in the bin, *i.e.* $O(N'_F)$. The complexity of the DCORFS is then $O(N_i N'_F(N^2 + N_p))$, which is typically dominated by the first term. The complexity of the overall distributed scheme D^{2}CORFS is dominated by its main cycle which repeats up to N_r times the DCORFS on N_b feature bins, for an overall complexity of $O(r_{max} N_u N_i N'_F(N^2 + N_p)) \simeq O(r_{max} N_F N_i N^2))$. An experimental characterization of the algorithm time complexity has also been carried out, the results of which are shown in Fig. 4.9. More precisely, Fig. 4.9 (top) reports the elapsed time for the DCORF algorithm, averaged over ten runs (such that the features are reshuffled at random in each run), as a function of the number of features in the bin and the number of iterations. The curves are characterized by an initial increase of the computational time with the growth of the feature space, followed by a saturation associated with the reaching of the maximum allowed number of iterations. Indeed, as N_i increases, the saturation point shifts to the right. These curves can be used to properly set N_i with respect to the number of features in the bin, in order to

obtain convergence prior to the saturation point.

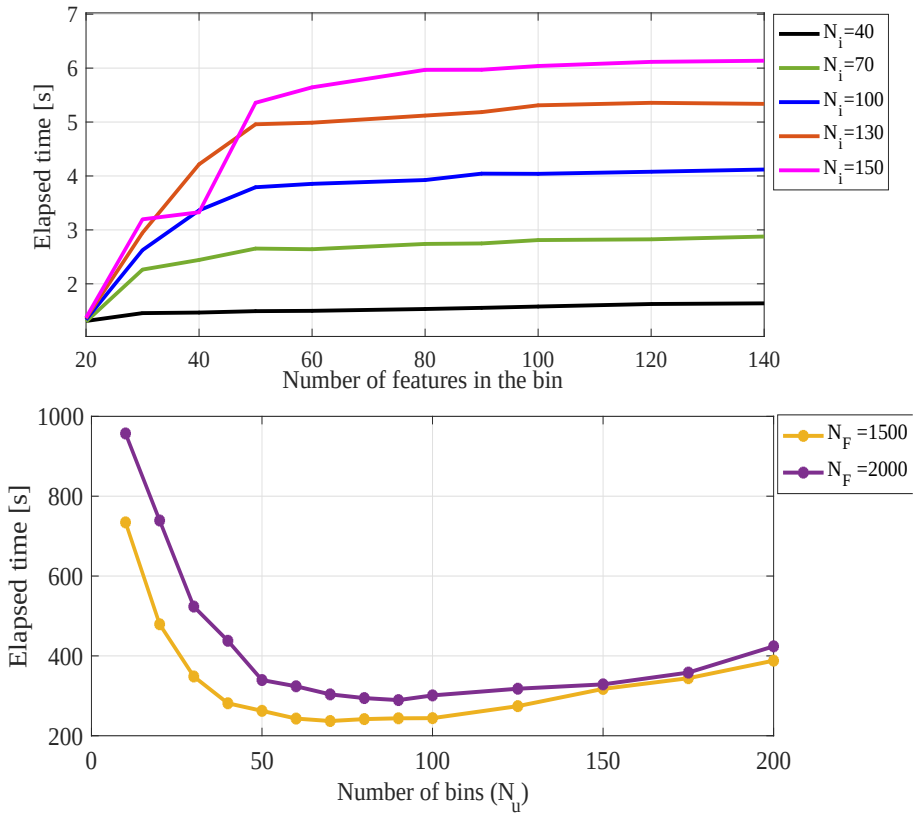

Figure 4.9: *Complexity analysis: Time dependency of the DCORFS execution time on the number of features in the bin and the number of iterations (top), time dependency of the D^{2}CORFS execution time on the problem size and the maximal number of bins (bottom).*

Fig. 4.9 (bottom) analyses the elapsed time of the overall D^{2}CORFS algorithm, as a function of the problem sizes N_F and the number of bins N_u. As can be seen from the figure, it is quite apparent that the execution time decreases rapidly as the number of bins increases, but at a certain point it starts increasing again, though at a slower rate. This result emphasizes the importance of the N_u design parameter. If the number of bins is chosen too sparingly, the bin size will be too large, thus leading to insufficient search space reduction. This ultimately defies the very purpose of the distributed scheme, *i.e.* to break the problem complexity, and thus slows down the convergence of the algorithm. Conversely, if one employs too many small bins, most of these will initially not contain any useful feature and will pre-

sumably 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 of this, the algorithm will require more rounds and thus more time to converge.

4.7.5 Model sensitivity on the data subdivision

To analyse the model sensitivity on the data subdivision in training and test data, we took the best model (see Tab. 4.14) obtained using the nominal training-test subdivision of the Leukemia dataset, and evaluated its performance with a Monte Carlo test over 1000 random training-test data subdivisions (generated so as to preserve the distribution among classes). On each run, the selected features are the same but the classifier is re-estimated on the corresponding training subset and evaluated on the test subset. The results are presented in Fig. 4.10, and show a non-neglectable sensitivity to the training-test data subdivisions. Indeed, the same performance of the nominal case is re-obtained less than 50% of the times, and on more than 10% of the runs a performance as low as $J_{TE} = 0.91$ is achieved (corresponding to 3 errors over the 34 test samples). One possible explanation of this phenomenon is that there are some isolated samples which cannot be learnt by the model if they fall in the test portion of the data.

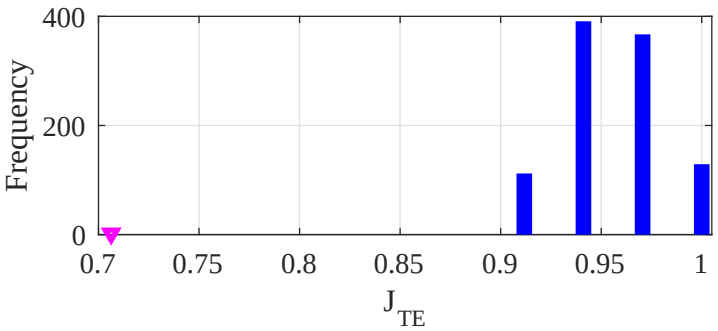

Figure 4.10: *Accuracy of the Leukemia model obtained with the nominal training-test data split over* 1000 *alternative data-splits.*

4.7.6 Comparative analysis

Table 4.16 reports a comparison with the results documented in [17] and [79], which consider exactly the same datasets with the same data splitting. To account for the randomized nature of the D^{2}CORFS algorithm (both regarding the splitting of the features in the feature bins and the DCORFS

algorithm employed on each of these), we present the averaged results of 5 independent runs besides the best ones. Both the classification accuracy on the test set and the model size are reported ($\bar{J}_{TE}$ and $\overline{|\mathcal{E}|}$ denoting the averages, and $J^{\star}_{TE}$ and $|\mathcal{E}^{\star}|$ the values associated to the best models, respectively). Compared to the results taken from the literature, the proposed method selects compact models capable of providing good performance. Moreover, the selected feature subsets result in comparable performance on all four datasets, regardless of the classifier employed, which is not the case for the methods discussed *e.g.* in [17].
Some further results are available for the Leukemia dataset regarding other multivariate filter methods, namely the MRMR [38], the CFS [119], and the MBF [122]. These results cannot be directly compared to the preceding ones since they have been obtained with the leave-one-out cross validation (LOOCV). Using this approach the MRMR, the MBF, and the D^{2}CORFS all score $\bar{J} = 1$, whereas the CFS only obtains $\bar{J} = 0.91$. Interestingly enough, the D^{2}CORFS obtains this performance with the smallest feature subset with respect to the other methods, with just 3 features, compared to the 6 of MRMR and the 8 of MBF.

Table 4.16: *Comparative analysis in terms of classification accuracy and model size.*

	Prostate		Lung		Leukemia		Ovarian	
Method	$\bar{J}_{TE}(J^\star_{TE})$	$\overline{\lvert\mathcal{E}\rvert}(\lvert\mathcal{E}^\star\rvert)$	$\bar{J}_{TE}(J^\star_{TE})$	$\overline{\lvert\mathcal{E}\rvert}(\lvert\mathcal{E}^\star\rvert)$	$\bar{J}_{TE}(J^\star_{TE})$	$\overline{\lvert\mathcal{E}\rvert}(\lvert\mathcal{E}^\star\rvert)$	$\bar{J}_{TE}(J^\star_{TE})$	$\overline{\lvert\mathcal{E}\rvert}(\lvert\mathcal{E}^\star\rvert)$
D^{2}CORFS	0.94 (0.97)	4 (4)	0.97 (0.99)	3.6 (2)	0.94 (0.94)	3 (3)	1.00 (1.00)	3.8 (3)
DRF+SVM [17]	0.97	30	0.98	2	0.91	15	1.00	45
DRF+NB [17]	0.26	30	0.98	2	0.94	15	1.00	45
DRF+kNN [17]	0.62	30	0.98	2	0.94	15	1.00	45
L1-norm SVM [79]	-	-	0.83	29.1	0.84	24.9	-	-
Elastic Net [79]	-	-	0.84	39.4	0.84	36.7	-	-
PEAN [79]	-	-	0.85	26.3	0.84	21.9	-	-
DrSVM [79]	-	-	0.86	54.4	0.85	67.7	-	-
WDRSVM [79]	-	-	0.86	23.8	0.86	19.9	-	-

4.8 Randomized FS as a mechanism for generating ensemble of classifiers

It is well-known that an ensemble of classifiers can perform better in terms of accuracy with respect to a single classifier [37]. This is possible just if the classifiers make uncorrelated decisions, that is just if the individual classifiers are different among each other. In this section we present the preliminary results of a new ensemble method denoted E-RFSC. Namely, the random nature of the RFSC algorithm in combination with the discrete loss function and sample manipulation are exploited to serve as a mechanism for generating diversity among the classifiers. Once all classifiers are constructed, the final decision is made by a so called *meta-classifier*, which takes the form of a linear regression and is trained on the whole training dataset.

4.8.1 Preliminaries on ensemble-based classification methods

One of the main aims in supervised ML is to improve accuracy. In this direction, an active research area on this problem is the study of methods for constructing ensembles of classifiers, which combine the decisions od different classifiers by adding a further decision layer based on voting, weighted voting, or other criteria. Ensembles are more accurate than the individual classifiers with which they are built if the individual classifiers disagree with one another [56]. That is, diversity among the classifiers leads to uncorrelated class predictions, which results in improved classification accuracy. This is in line with the well known Condorcet's jury theorem which refers to the problem of a group of individuals making a correct final decision. Namely, let us assume that the voting is independent and that there are only two possible outcomes. If an individual of a group has a probability p of being correct and the probability of the majority of being correct is T_p, then $p > 0.5$ implies $T_p > p$. If $p > 0.5$ and the number of individuals approaches infinity, then $T_p \to 1$. Hence, the key factor for making a successful ensemble is to find uncorrelated classifiers, with individual classification performance above 0.5.
Still, the theorem is based on the strong assumption that the voting is independent and holds just for biclass classification problems.

In general, the ensemble construction involves four main components: i) the available training data pairs $\{\boldsymbol{u}(k), y(k)\}$, ii) a *diversity generator* responsible for generating diversity among the classifiers, iii) a *base inducer*,

$i.e.$ an algorithm for classifier design and iv) a *combiner block*, $i.e.$ a policy for combining the decisions of the individual classifiers into the final outcome of the ensemble.

There are several methods in the literature for obtaining diversity: i) *Sampling methods* manipulate the training samples so that multiple hypotheses can be generated; ii) *Methods based on input manipulation* manipulate the features to generate different feature subsets for different learning processes; iii) *Methods based on output manipulation* manipulate the outputs to generate different training subsets on which the learning process is performed; and finally iv) *Methods based on injecting randomness* into the learning algorithm repeat the learning process multiple times on the same training data but with different initial feature weights.

Given the set of classifiers which construct an ensemble, it is necessary to apply some policy to combine their individual results into a unique decision. Combining methods are generally classified into *voting* and *meta-combination methods*. Voting methods can be either unweighted or weighted. Majority voting is the simplest (unweighted) vote approach. In case of classifiers with probabilistic outcome, the ensemble outcome is the class with the highest average probability obtained by averaging the predicted probabilities of the individual classifiers independently for each class. Weighted voting methods weigh the decisions of the individual classifiers with a certain policy to make the final decision. In [56] least square regression is applied to find the weights which maximize the ensemble accuracy on the training data. Another common method is to weigh the classifiers' decisions based on their individual accuracy ($e.g.$ [5]). The approach proposed in [68] uses features as inputs to compute weights, which are later used to weigh the hypothesis of the individual classifiers.
Meta-combination methods learn meta-classifiers from the individual classifiers and their corresponding outputs. In the literature, several methods based on meta-combinations are proposed [121], [106], [26], [25]). In this work we exploit the basic idea of stacking [121], which uses the predictions of the individual classifiers as inputs to a $meta-level$ classifier which combines the individual decisions into the final one.

4.8.2 The E-RFSC algorithm

The discrete nature of the loss function used in the RFSC algorithm in combination with its random nature causes different runs on the same data to ex-

tract different models with similar or equivalent performance as discussed in Section 3.5.2. This behaviour of the algorithm suggested the development of a method for generating ensembles of classifiers, which will be denoted as E-RFSC in the sequel. Considering that diversity is essential in the construction of successful ensemble classifiers, we exploited the mentioned behavior of the RFSC algorithm together with sample manipulation as a diversity generator (in the work we present here we used 10-fold cross-validation for this purpose). More precisely, training data are divided initially into 10 non-overlapping sample subsets. Learning is performed 10 times, so that nine slots are used for training and the remaining one for testing. Once all classifiers are obtained, the corresponding predictions serve as inputs to a meta-classifier. The meta-classifier takes the form of a linear regression:

$$\hat{c}_E(k) = \boldsymbol{\varphi}_E(k)^T \boldsymbol{\vartheta}_E. \tag{4.4}$$

where $\boldsymbol{\varphi}_E(k) = [\hat{y}_1(k),\ \ldots,\ \hat{y}_{10}(k)]^T$ is vector of regressors representing the prediction values of the individual classifiers, while $\boldsymbol{\vartheta}_E = [\vartheta_{E_1},\ \ldots,\ \vartheta_{E_{10}}]^T$ is the parameter ($i.e.$ weight) vector. The logistic loss function is applied to find the weights that maximize the accuracy of the meta-classifier on the whole training dataset.

4.8.3 Performance analysis

This section reports the results of an analysis carried out to assess the accuracy of the proposed ensemble classification algorithm. It is tested on four numerical datasets, two obtained from the UCI machine learning repository (HillVally and Musk1) and two micro-array datasets (Colon and Ovarian). The original features of the HillValley and the Musk1 datasets are polynomially expanded with $L = 2$. The main characteristics of the datasets are represented in Table 4.17. As with the RFSC, all the original features are normalized in the $[0, 1]$ range according to Eq.(3.9).
The design parameters of the algorithm were set to the same values as for the RFSC for the HillValley and Musk1 datasets (number of iterations was set to $N_i = 300$, the maximum nonlinearity degree to $L = 2$, the number of generated models to $N_p = 100$, the significance confidence interval to $\alpha = 0.99$ and all initial TIPs to $1/N_F$), and the to the initial setting used in dRFSC for two microarray datasets (the nonlinearity degree $L = 1$, the number of generated models $N_p = 10$, the significance confidence interval $\alpha = 0.998$. This parameter influences the model size, in that the closer it is to 1, the more terms are rejected by the statistical test, 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 $\bar{\mu} = 0.7$)).

Table 4.17: *Main characteristics of the considered data-sets.*

Data-set	N	N_f	N_F	N_c
Colon	62	2000	2000	2
HillValley	606	100	5151	2
Musk1	476	166	14028	2
Ovarian	253	15154	15154	2

The performance is evaluated by random $70 - 30$ horizontal data partitioning, so that 70% of the samples served for learning and the remaining 30% for testing. The results in terms of the performance index $\mathcal{J}$ are presented in Table 4.18. As can be noticed, the proposed ensemble method scores the same or improved results with respect to the other two methods which take decisions based on single classifiers.

Table 4.18: *Performance comparison E-RFSC vs. RFSC and dRFSC in terms of accuracy.*

Data-set	Method	$\mathcal{J}$
Colon	E-RFSC	0.8421
	RFSC [20]	-
	dRFSC [22]	0.8421
HillValley	E-RFSC	1.0000
	RFSC [20]	0.9277
	dRFSC [22]	0.9859
Musk1	E-RFSC	0.8670
	RFSC [20]	0.8132
	dRFSC [22]	0.8216
Ovarian	E-RFSC	1.0000
	RFSC [20]	-
	dRFSC [22]	1.0000

CHAPTER 5

Conclusions

In this thesis we presented the outcome of the research on randomized model selection for NL identification and supervised ML.

The first outcome of the research is the RFSC algorithm, which applies a randomized model search strategy to the feature selection and classification problem. The FS problem is reformulated as a model structure selection problem, where suitable nonlinear functions of the original features are evaluated for insertion in a linear regression model. Differently from commonly adopted methods, the importance of each candidate regressor is not evaluated with reference to a specific model, but to a population of models, which appears to provide more reliable information regarding the actual significance of the term. A distribution of the models is used to extract the population of models and is then updated based on the aggregate information gathered from the extracted models, reinforcing the probability to extract the most promising regressors. Upon convergence a limit distribution is obtained, which in practice identifies a single model structure. The proposed algorithm was tested also in the context of Big Data problems to build models capable of predicting the completion time of an application. The discrete nature of the loss function used in the RFSC al-

gorithm in combination with its random nature and sample manipulation is exploited to develop a method for generating ensembles of classifiers, denoted E-RFSC. The outputs of several individual classifiers serve as inputs to a meta-classifier, in the form of a linear regression, which takes the final decision.

The observed limitations of the proposed algorithms on large scale problems motivated the development of a distributed scheme for FS applicable to both frameworks. The proposed algorithm is based on vertical data partitioning, the distribution of the FS task among several processing units and on the exchange of their local solutions until a consensus is reached. The proposed distributed scheme can be used in combination with any FS algorithm of choice. The method produces small and compact models, easily amenable to interpret, while at the same it obtains improvements on performance and efficiency. This concept is exploited also for developing multivariate filter algorithm based on the distance correlation index, targeting micro-arrays problems. Actually, the distributed scheme in combination with the RFSC algorithm was found to be ineffective when dealing with extra-large search spaces (e.g. microarrays), due to computational issues related to parameter estimation and classifier design. More in detail, we introduced a novel feature selection (FS) approach which employs the distance correlation (dCor) as a criterion for evaluating the dependence of the class on a given feature subset. The dCor index provides a reliable dependence measure among random vectors of arbitrary dimension, without any assumption on their distribution. Moreover, it is sensitive to the presence of redundant terms. The proposed FS method is based on a probabilistic representation of the feature subset model, which is progressively refined by a repeated process of model extraction and evaluation. A key element of the approach is a distributed optimization scheme based on a vertical partitioning of the dataset, which alleviates the negative effects of its unbalanced dimensions.

The proposed methods have been evaluated and compared to the other well-known FS and MSS algorithms on standard benchmarks, obtaining promising and competitive results, especially in terms of the tradeoff between model complexity and prediction accuracy.

Bibliography

[1] L.A. Aguirre and A. P. Barbosa, B.H.G. Braga. Prediction and simulation errors in parameter estimation for nonlinear systems. *Mechanical Systems and Signal Processing*, 24(8):2855–2867, nov 2010.

[2] D. W Aha, D. Kibler, and M. K Albert. Instance-based learning algorithms. *Machine learning*, 6(1):37–66, 1991.

[3] E. Alba, J. Garcia-Nieto, L. Jourdan, and E. G. Talbi. Gene selection in cancer classification using PSO/SVM and GA/SVM hybrid algorithms. In *IEEE Congress on Evolutionary Computation (CEC 2007)*, pages 284–290, 2007.

[4] F. Alessandro, L. Piroddi, and M. Prandini. A randomized algorithm for nonlinear model structure selection. *Automatica*, 60:227–238, oct 2015.

[5] K. M Ali and M. J Pazzani. Error reduction through learning multiple descriptions. *Machine learning*, 24(3):173–202, 1996.

[6] D. Ardagna, S. Bernardi, E. Gianniti, S. K. Aliabadi, D. Perez-Palacin, and José I. Requeno. Modeling performance of hadoop applications: A journey from queueing networks to stochastic well formed nets. In *Algorithms and Architectures for Parallel Processing*, pages 599–613. Springer, 2016.

[7] E. Ataie, E. Gianniti, D. Ardagna, and A. Movaghar. A combined analytical modeling machine learning approach for performance prediction of mapreduce jobs in cloud environment. In *Symbolic and*

Numeric Algorithms for Scientific Computing (SYNASC), 2016 18th International Symposium on, pages 431–439. IEEE, 2016.

[8] M. Avelina, A. Brankovic, and L. Piroddi. Distributed randomized model structure selection for NARX models. *International Journal of Adaptive Control and Signal Processing*, 31(12):1853–1870, 2017.

[9] T. Baldacchino, S. R. Anderson, and V. Kadirkamanathan. Structure detection and parameter estimation for NARX models in a unified EM framework. *Automatica*, 48(5):857–865, may 2012.

[10] T. Baldacchino, S. R. Anderson, and V. Kadirkamanathan. Computational system identification for bayesian NARMAX modelling. *Automatica*, 49(9):2641–2651, sep 2013.

[11] M. Banerjee and S. Chakravarty. Privacy preserving feature selection for distributed data using virtual dimension. In *Proceedings of the* 20^{th} *ACM international conference on Information and knowledge management*, pages 2281–2284, 2011.

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

[13] F. Bianchi, A. Falsone, M. Prandini, and L. Piroddi. A randomised approach for NARX model identification based on a multivariate Bernoulli distribution. *International Journal of Systems Science*, 48(6):1203–1216, 2017.

[14] S. A. Billings. *Nonlinear System Identification*. Wiley-Blackwell, jul 2013.

[15] S. A. Billings, S. Chen, and M. J. Korenberg. Identification of MIMO non-linear systems using a forward-regression orthogonal estimator. *International Journal of Control*, 49(6):2157–2189, jun 1989.

[16] C. M. Bishop. Pattern recognition. *Machine Learning*, 128:1–58, 2006.

[17] V. Bolón-Canedo, N. Sánchez-Maroño, and A. Alonso-Betanzos. Distributed feature selection: An application to microarray data classification. *Applied soft computing*, 30:136–150, 2015.

[18] V. Bolón-Canedo, N. Sánchez-Marono, and J. Cerviño-Rabuñal. Toward parallel feature selection from vertically partitioned data. In *ESANN*, 2014.

[19] M. Bonin, L. Piroddi, and V. Seghezza. NARX model selection based on simulation error minimisation and LASSO. *IET Control Theory & Applications*, 4(7):1157–1168, jul 2010.

[20] A. Brankovic, A. Falsone, M. Prandini, and L. Piroddi. A feature selection and classification algorithm based on randomized extraction of model populations. *IEEE Transactions on Cybernetics*, 2017.

[21] A. Brankovic, M. Hosseini, and L. Piroddi. A distributed feature selection algorithm based on distance correlation with an application to microarrays. *submitted paper, available on request*, 2017.

[22] A. Brankovic and Piroddi L. A distributed feature selection scheme with partial information sharing. *submitted paper, available on request*, 2017.

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

[24] G. Casale, M. Tribastone, and P. G. Harrison. Blending randomness in closed queueing network models. *Performance Evaluation*, 82:15–38, 2014.

[25] P. K Chan and S. J Stolfo. Experiments on multistrategy learning by meta-learning. In *Proceedings of the second international conference on information and knowledge management*, pages 314–323. ACM, 1993.

[26] P. K Chan and S. J Stolfo. Toward scalable learning with non-uniform class and cost distributions: A case study in credit card fraud detection. In *KDD*, volume 1998, pages 164–168, 1998.

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

[28] S. Chen, S. A. Billings, and W. Luo. Orthogonal least squares methods and their application to non-linear system identification. *International Journal of Control*, 50(5):1873–1896, nov 1989.

[29] C. Chu, S. K. Kim, Y.A. Lin, Y.Y. Yu, G. Bradski, A. Y. Ng, and K. Olukotun. Map-reduce for machine learning on multicore. *Advances in neural information processing systems*, 19:281, 2007.

[30] P. Connally, K. Li, and G. W. Irwin. Prediction-and simulation-error based perceptron training: solution space analysis and a novel combined training scheme. *Neurocomputing*, 70(4):819–827, 2007.

[31] M. Cord and P. Cunningham. *Machine learning techniques for multimedia: case studies on organization and retrieval*. Springer Science & Business Media, 2008.

[32] M. Dash and H. Liu. Feature selection for classification. *Intelligent data analysis*, 1(1-4):131–156, 1997.

[33] J. T. de Souza, S. Matwin, and N. Japkowicz. Parallelizing feature selection. *Algorithmica*, 45(3):433–456, 2006.

[34] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

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

[36] D. Didona, F. Quaglia, P. Romano, and E. Torre. Enhancing performance prediction robustness by combining analytical modeling and machine learning. In *Proceedings of the 6th ACM/SPEC international conference on performance engineering*, pages 145–156. ACM, 2015.

[37] T. G Dietterich. Machine-learning research. *AI magazine*, 18(4):97, 1997.

[38] C. Ding and H. Peng. Minimum redundancy feature selection from microarray gene expression data. *Journal of bioinformatics and computational biology*, 3(02):185–205, 2005.

[39] S. A Dudani. The distance-weighted k-nearest-neighbor rule. *IEEE Transactions on Systems, Man and Cybernetics*, (4):325–327, 1976.

[40] P. G Espejo, S. Ventura, and F. Herrera. A survey on the application of genetic programming to classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 40(2):121–144, 2010.

[41] M. Farina and L. Piroddi. Identification of polynomial input/output recursive models with simulation error minimisation methods. *International Journal of Systems Science*, 43(2):319–333, feb 2012.

[42] M. Farina and L. Piroddi. Identification of polynomial input/output recursive models with simulation error minimisation methods. *International Journal of Systems Science*, 43(2):319–333, 2012.

[43] F. Ferri, P. Pudil, M. Hatef, and J. Kittler. Comparative study of techniques for large-scale feature selection. *Pattern Recognition in Practice IV*, pages 403–413, 1994.

[44] J. H. Friedman. Multivariate adaptive regression splines. *Ann. Statist.*, 19(1):1–67, mar 1991.

[45] Q. B. Gao and Z. Z. Wang. Center-based nearest neighbor classifier. *Pattern Recognition*, 40(1):346–349, 2007.

[46] G. P. Gibilisco, M. Li, L. Zhang, and D. Ardagna. Stage aware performance modeling of dag based in memory analytic platforms. In *Cloud Computing (CLOUD), 2016 IEEE 9th International Conference on*, pages 188–195. IEEE, 2016.

[47] P. J. Green. Reversible jump markov chain monte carlo computation and bayesian model determination. *Biometrika*, 82(4):711–732, 1995.

[48] A. Guillén, A. Sorjamaa, Y. Miche, A. Lendasse, and I. Rojas. Efficient parallel feature selection for steganography problems. In *International Work-Conference on Artificial Neural Networks*, pages 1224–1231. Springer, 2009.

[49] S. R. Gunn et al. Support vector machines for classification and regression. *ISIS technical report*, 14, 1998.

[50] Y. Guo, L.Z. Guo, S. A. Billings, and H. L. Wei. An iterative orthogonal forward regression algorithm. *International Journal of Systems Science*, 46(5):776–789, nov 2014.

[51] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *Journal of machine learning research*, 3(Mar):1157–1182, 2003.

[52] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik. Gene selection for cancer classification using support vector machines. *Machine learning*, 46(1):389–422, 2002.

[53] R. Haber and H. Unbehauen. Structure identification of nonlinear dynamic systems—a survey on input/output approaches. *Automatica*, 26(4):651–677, jul 1990.

[54] M. A. Hall. *Correlation-based feature selection for machine learning*. PhD thesis, The University of Waikato, 1999.

[55] E. Hancer, B. Xue, D. Karaboga, and M. Zhang. A binary abc algorithm based on advanced similarity scheme for feature selection. *Applied Soft Computing*, 36:334–348, 2015.

[56] L. K. Hansen and P. Salamon. Neural network ensembles. *IEEE transactions on pattern analysis and machine intelligence*, 12(10):993–1001, 1990.

[57] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer New York, 2009.

[58] W. K. Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.

[59] H. He and E. A. Garcia. Learning from imbalanced data. *IEEE Transactions on knowledge and data engineering*, 21(9):1263–1284, 2009.

[60] Z. M. Hira and D. F. Gillies. A review of feature selection and feature extraction methods applied on microarray data. *Advances in bioinformatics*, 2015, 2015.

[61] A. Holzinger and I. Jurisica. *Knowledge Discovery and Data Mining in Biomedical Informatics: The Future Is in Integrative, Interactive Machine Learning Solutions*, pages 1–18. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

[62] X. Hong, R. J. Mitchell, S. Chen, C. J. Harris, K. Li, and G. W. Irwin. Model selection approaches for non-linear system identification: a review. *International Journal of Systems Science*, 39(10):925–946, 2008.

[63] J. Huang, Y. Cai, and X. Xu. A hybrid genetic algorithm for feature selection wrapper based on mutual information. *Pattern Recognition Letters*, 28(13):1825–1844, 2007.

[64] H. H. Inbarani, A. T. Azar, and G. Jothi. Supervised hybrid feature selection based on PSO and rough sets for medical diagnosis. *Computer methods and programs in biomedicine*, 113(1):175–185, 2014.

[65] F. Isaila, P. Balaprakash, S. M. Wild, D. Kimpe, R. Latham, R. Ross, and P. Hovland. Collective i/o tuning using analytical and machine learning models. In *Cluster Computing (CLUSTER), 2015 IEEE International Conference on*, pages 128–137. IEEE, 2015.

[66] H. V. Jagadish, J. Gehrke, A. Labrinidis, Y. Papakonstantinou, J. M. Patel, R. Ramakrishnan, and C. Shahabi. Big data and its technical challenges. *Communications of the ACM*, 57(7):86–94, 2014.

[67] T. Jirapech-Umpai and S. Aitken. Feature selection and classification for microarray data analysis: Evolutionary methods for identifying predictive genes. *BMC bioinformatics*, 6(1):148, 2005.

[68] M. I Jordan and R. A Jacobs. Hierarchical mixtures of experts and the em algorithm. *Neural computation*, 6(2):181–214, 1994.

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

[70] D. Koller and M. Sahami. Toward optimal feature selection. Technical report, Stanford InfoLab, 1996.

[71] M. Korenberg, S. A. Billings, Y. P. Liu, and P. J. McIlroy. Orthogonal parameter estimation algorithm for non-linear stochastic systems. *International Journal of Control*, 48(1):193–210, jul 1988.

[72] S. Kotsiantis. Feature selection for machine learning classification problems: a recent overview. *Artificial Intelligence Review*, pages 1–20, 2011.

[73] C. Lazar, J. Taminau, S. Meganck, D. Steenhoff, A. Coletta, C. Molter, V. de Schaetzen, R. Duque, H. Bersini, and A. Nowe. A survey on filter techniques for feature selection in gene expression microarray analysis. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 9(4):1106–1119, 2012.

[74] I. J. Leontaritis and S. A. Billings. Input-output parametric models for non-linear systems part i: deterministic non-linear systems. *International Journal of Control*, 41(2):303–328, feb 1985.

[75] I. J. Leontaritis and S. A. Billings. Input-output parametric models for non-linear systems part II: stochastic non-linear systems. *International Journal of Control*, 41(2):329–344, feb 1985.

[76] Y. Leung and Y. Hung. A multiple-filter-multiple-wrapper approach to gene selection and microarray data classification. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 7(1):108–117, 2010.

[77] B. C. Levy and M. Zorzi. A contraction analysis of the convergence of risk-sensitive filters. *SIAM Journal on Control and Optimization*, 54(4):2154–2173, 2016.

[78] B. Li, Y. W. Chen, and Y. Q. Chen. The nearest neighbor algorithm of local probability centers. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 38(1):141–154, 2008.

[79] J. Li, Y. Wang, Y. Cao, and C. Xu. Weighted doubly regularized support vector machine and its application to microarray classification with noise. *Neurocomputing*, 173:595–605, 2016.

[80] K. Li, J. X. Peng, and G. W. Irwin. A fast nonlinear model identification method. *IEEE Transactions on Automatic Control*, 50(8):1211–1216, aug 2005.

[81] R. Li, W. Zhong, and L. Zhu. Feature screening via distance correlation learning. *Journal of the American Statistical Association*, 107(499):1129–1139, 2012.

[82] Y. Li, G. Wang, H. Chen, L. Shi, and L. Qin. An ant colony optimization based dimension reduction method for high-dimensional datasets. *Journal of Bionic Engineering*, 10(2):231–241, 2013.

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

[84] S.W. Lin, K.C. Ying, S.C. Chen, and Z.J. Lee. Particle swarm optimization for parameter determination and feature selection of support vector machines. *Expert systems with applications*, 35(4):1817–1824, 2008.

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

[86] K. Z. Mao and S. A. Billings. Variable selection in non-linear system modelling. *Mechanical Systems and Signal Processing*, 13(2):351–366, mar 1999.

[87] A. Miller. *Subset selection in regression*. CRC Press, 2002.

[88] D. P. Muni, N. R Pal, and J. Das. Genetic programming for simultaneous feature selection and classifier design. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 36(1):106–117, 2006.

[89] S. Mylavarapu and A. Kaban. Random projections versus random selection of features for classification of high dimensional data. In *2013 13th UK Workshop on Computational Intelligence (UKCI)*, pages 305–312. IEEE, 2013.

[90] K. Neshatian and M. Zhang. Pareto front feature selection: using genetic programming to explore feature space. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 1027–1034, 2009.

[91] D.J. Newman, S. Hettich, C.L. Blake, and C.J. Merz. UCI repository of machine learning databases, 1998.

[92] A. Y. Ng. Feature selection, l 1 vs. l 2 regularization, and rotational invariance. In *Proceedings of the twenty-first international conference on Machine learning*, page 78. ACM, 2004.

[93] G. Notarstefano and F. Bullo. Distributed abstract optimization via constraints consensus: Theory and applications. *IEEE Transactions on Automatic Control*, 56(10):2247–2261, oct 2011.

[94] I.S. Oh, J.S. Lee, and B.R. Moon. Hybrid genetic algorithms for feature selection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(11):1424–1437, 2004.

[95] M. Paliwal and U. A. Kumar. Neural networks and statistical techniques: A review of applications. *Expert systems with applications*, 36(1):2–17, 2009.

[96] L. Peng, B. Yang, Y. Chen, and A. Abraham. Data gravitation based classification. *Information Sciences*, 179(6):809–819, 2009.

[97] L. Peng, H. Zhang, H. Zhang, and B. Yang. A fast feature weighting algorithm of data gravitation classification. *Information Sciences*, 2016.

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

[99] L. Piroddi and W. Spinelli. A pruning method for the identification of polynomial NARMAX models. In *13th IFAC Symposium on System Identification, Rotterdam, The Netherlands, August 27-28*, pages 1108–1113, aug 2003.

[100] P. Pudil, J. Novovičová, and J. Kittler. Floating search methods in feature selection. *Pattern recognition letters*, 15(11):1119–1125, 1994.

[101] P. Pudil and P. Somol. Current feature selection techniques in statistical pattern recognition. *Computer Recognition Systems*, pages 53–68, 2005.

[102] C. C.O. Ramos, A. N. Souza, G. Chiachia, A. X. Falcão, and J. Papa. A novel algorithm for feature selection using harmony search and its application for non-technical losses detection. *Computers & Electrical Engineering*, 37(6):886–894, 2011.

[103] R. Rodriguez-Vazquez, C. M. Fonseca, and P. J. Fleming. Identifying the structure of NonLinear dynamic systems using multiobjective genetic programming. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 34(4):531–545, jul 2004.

[104] Y. Saeys, I. Inza, and P. Larrañaga. A review of feature selection techniques in bioinformatics. *bioinformatics*, 23(19):2507–2517, 2007.

[105] S. Saha, S. Rajasekaran, and R. Ramprasad. Novel randomized feature selection algorithms. *International Journal of Foundations of Computer Science*, 26(03):321–341, 2015.

[106] A. K Seewald and J. Fürnkranz. An evaluation of grading classifiers. In *International Symposium on Intelligent Data Analysis*, pages 115–124. Springer, 2001.

[107] D. Shalon, S. J. Smith, and P. O. Brown. A dna microarray system for analyzing complex dna samples using two-color fluorescent probe hybridization. *Genome research*, 6(7):639–645, 1996.

[108] R. Sheikhpour, M. A. Sarram, and R. Sheikhpour. Particle swarm optimization for bandwidth determination and feature selection of

kernel density estimation based classifiers in diagnosis of breast cancer. *Applied Soft Computing*, 40:113–131, 2016.

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

[110] T. Söderström and P. Stoica, editors. *System Identification*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1988.

[111] J. Speyer, J. Deyst, and D. Jacobson. Optimization of stochastic linear systems with additive measurement and process noise using exponential performance criteria. *IEEE Transactions on Automatic Control*, 19(4):358–366, 1974.

[112] N.K. Sreeja and A. Sankar. Pattern matching based classification using ant colony optimization based feature selection. *Applied Soft Computing*, 31:91–102, 2015.

[113] S. Stearns. On selecting features for pattern classifiers. In *International Conference on Pattern Recognition*, 1976.

[114] G. J. Székely and M. L. Rizzo. The distance correlation t-test of independence in high dimension. *Journal of Multivariate Analysis*, 117:193–213, 2013.

[115] G. J. Székely, M. L. Rizzo, N. K Bakirov, et al. Measuring and testing dependence by correlation of distances. *The Annals of Statistics*, 35(6):2769–2794, 2007.

[116] I. Triguero, S. García, and F. Herrera. Differential evolution for optimizing the positioning of prototypes in nearest neighbor classification. *Pattern Recognition*, 44(4):901–916, 2011.

[117] V. Venkatraman, A. R. Dalby, and Z. R. Yang. Evaluation of mutual information and genetic programming for feature selection in QSAR. *Journal of chemical information and computer sciences*, 44(5):1686–1692, 2004.

[118] J. Wang, P. Neskovic, and L. N. Cooper. Improving nearest neighbor rule with a simple adaptive distance measure. *Pattern Recognition Letters*, 28(2):207–213, 2007.

[119] Y. Wang, I. V. Tetko, M. A. Hall, E. Frank, A. Facius, K. F. X. Mayer, and H. W. Mewes. Gene selection from microarray data for cancer

classificationa machine learning approach. *Computational biology and chemistry*, 29(1):37–46, 2005.

[120] H. L. Wei and S. A. Billings. Model structure selection using an integrated forward orthogonal search algorithm assisted by squared correlation and mutual information. *IJMIC*, 3(4):341, 2008.

[121] D. H Wolpert. Stacked generalization. *Neural networks*, 5(2):241–259, 1992.

[122] E. P Xing, M. I. Jordan, R. M. Karp, et al. Feature selection for high-dimensional genomic microarray data. In *ICML*, volume 1, pages 601–608. Citeseer, 2001.

[123] Y. Xu. Maximum margin of twin spheres support vector machine for imbalanced data classification. *IEEE transactions on cybernetics*, 47(6):1540–1550, 2017.

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

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

[126] J. Yang and V. Honavar. Feature subset selection using a genetic algorithm. In *Feature extraction, construction and selection*, pages 117–136. Springer, 1998.

[127] C. D. Yenigün and M. L Rizzo. Variable selection in regression using maximal correlation and distance correlation. *Journal of Statistical Computation and Simulation*, 85(8):1692–1705, 2015.

[128] Y. Zhang, S. Chen, Q. Wang, and G. Yu. i ^{2} mapreduce: Incremental mapreduce for mining evolving big data. *IEEE transactions on knowledge and data engineering*, 27(7):1906–1919, 2015.

[129] Y. Zhang, S. Wang, P. Phillips, and G. Ji. Binary PSO with mutation operator for feature selection using decision tree applied to spam detection. *Knowledge-Based Systems*, 64:22–31, 2014.

[130] Z. Zhang, L. Cherkasova, and B. T. Loo. Exploiting cloud heterogeneity to optimize performance and cost of mapreduce processing. *ACM SIGMETRICS Performance Evaluation Review*, 42(4):38–50, 2015.

[131] Z. Zhao, R. Zhang, J. Cox, D. Duling, and W. Sarle. Massively parallel feature selection: an approach based on variance preservation. *Machine learning*, 92(1):195–220, 2013.

[132] C. Y. Zhou and Y. Q. Chen. Improving nearest neighbor classification with cam weighted distance. *Pattern Recognition*, 39(4):635–645, 2006.