IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received July 26, 2019, accepted August 1, 2019, date of publication August 5, 2019, date of current version August 20, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2933261

Experimental Study on 164 Algorithms Available in Software Tools for Solving Standard Non-Linear Regression Problems

MARÍA JOSÉ GACTO[1], JOSE MANUEL SOTO-HIDALGO[2], (Member, IEEE), JESÚS ALCALÁ-FDEZ[3], (Member, IEEE), AND RAFAEL ALCALÁ[3], (Member, IEEE)

[1]Department of Computer Science, University of Jaén, 23071 Jaén, Spain
[2]Department of Electronics and Computer Engineering, University of Córdoba, 14071 Córdoba, Spain
[3]Department of Computer Science and Artificial Intelligence, University of Granada, E-18071 Granada, Spain

Corresponding author: Jose Manuel Soto-Hidalgo (jmsoto@uco.es)

This work was supported in part by the University of Córdoba under the project PPG2019-UCOSOCIAL-03, and in part by the Spanish Ministry of Science, Innovation and Universities under Grant TIN2015- 68454-R and Grant TIN2017-89517-P.

ABSTRACT In the specialized literature, researchers can find a large number of proposals for solving regression problems that come from different research areas. However, researchers tend to use only proposals from the area in which they are experts. This paper analyses the performance of a large number of the available regression algorithms from some of the most known and widely used software tools in order to help non-expert users from other areas to properly solve their own regression problems and to help specialized researchers developing well-founded future proposals by properly comparing and identifying algorithms that will enable them to focus on significant further developments. To sum up, we have analyzed 164 algorithms that come from 14 main different families available in 6 software tools (Neural Networks, Support Vector Machines, Regression Trees, Rule-Based Methods, Stacking, Random Forests, Model trees, Generalized Linear Models, Nearest Neighbor methods, Partial Least Squares and Principal Component Regression, Multivariate Adaptive Regression Splines, Bagging, Boosting, and other methods) over 52 datasets. A new measure has also been proposed to show the goodness of each algorithm with respect to the others. Finally, a statistical analysis by non-parametric tests has been carried out over all the algorithms and on the best 30 algorithms, both with and without bagging. Results show that the algorithms from Random Forest, Model Tree and Support Vector Machine families get the best positions in the rankings obtained by the statistical tests when bagging is not considered. In addition, the use of bagging techniques significantly improves the performance of the algorithms without excessive increase in computational times.

INDEX TERMS Data mining, supervised learning, regression algorithms, experimental study.

I. INTRODUCTION

Regression is one of the most classic statistical techniques for predictive data mining [1]. Regression consists in designing a model from available training data that allows to predict the value of a continuous output variable from new given values of a set of input variables. Nowadays, a large number of proposals have been published for solving regression problems, such as financial forecasting [2], marketing [3] or drup response modeling [4] among others.

In the specialized literature, researchers can find proposals that come from different areas of research. When new approaches are published in any of these areas, researchers usually tend to use the same category of algorithms historically applied in the area of research in which they are experts, probably due to their partial knowledge about the available algorithms. In addition, this problem is made worse because of only a few number of researchers make the software and/or source code associated with their proposals public and sometimes authors provide vague or even ambiguous descriptions in the specialized literature. This issue, along with the high complexity of some proposals, makes the widespread use of

The associate editor coordinating the review of this manuscript and approving it for publication was Jing Bi.

these algorithms difficult. With the aim of tackling with these drawbacks, a great effort has been made by the data mining research community and a large number of regression algorithms have been included in well-known and used software tools, such as Matlab [5], R [6] and Weka [7], [8], among others.

The main objective of this paper is to analyze the performance of a large number of regression algorithms in order to help both non-expert users and specialized researchers. In this sense, non-expert users from other areas could properly solve their own regression problems and specialized researchers could develop well-founded future proposals by properly comparing and identifying algorithms that will enable them to focus on significant further developments. To accomplish this, we have analyzed 164 regression algorithms that come from 14 different families (Neural Networks, Support Vector Machines, Regression Trees, Rule-Based Methods, Stacking, Random Forests, Model trees, Generalized Linear Models, Nearest Neighbor methods, Partial Least Squares and Principal Component Regression, Multivariate Adaptive Regression Splines, Bagging, Boosting, and Other Methods) and that are available in the software tools Java Statistical Analysis Tool (JSAT) [9], KEEL [10], Matlab [5], R [6], Scikit-learn [11] and Weka [7], [8]. Moreover, a new measure has also been presented to assess the goodness of an algorithm with respect to the rest of analyzed algorithms in each dataset.

Notice that, it is intended to drive on algorithms that are correctly implemented and publicly available (so that we consider reference algorithms such as those included in the mentioned software tools). These algorithms have already been tested by many users, or even by the authors themselves, and corrected by experts in the case of presenting problems, understanding therefore that their implementations are reliable. Also, it is not intended to cover Big Data problems or problems with strong hardware requirements, which are not usually available to all users, but standard regression problems that even without such requirements are still difficult to solve. Of course, there are more recent algorithms in the specialized literature [12]–[14], but they are still not included in the said software tools, so that for non-expert users (without programming abilities) it is somehow difficult to implement and apply them. Even though we recommend their consideration when it is possible, we will focus here only on those available in the mentioned software tools. The underlying idea is therefore, to be able to guide on the algorithms available in some of the most well-known and used software tools, so that any non-specialized researcher, student, company, etc., that needs to use these algorithms could know which algorithms can be used and what could be expected from their application. Likewise, we would also like to ease further well-founded comparisons and studies from the specialized research community.

In order to assess the performance of these algorithms, an experimental study collecting 52 real-world datasets, with a number of variables within the interval $[2, 60]$ and a number of examples within the interval $[43, 45730]$ has been performed. Moreover, a new absolute metric together with different quality categories (based on this metric domain) are proposed in this contribution for assessing the regression algorithms goodness over different datasets. We have also developed a double study. Firstly, we have studied the performance of all the algorithms over the 52 datasets. Secondly, we have compared the performance of the 30 best algorithms including bagging application, and the 30 best algorithms without considering bagging in order to analyze the influence of bagging on those algorithms for which the software tools allow us to apply it. In both studies, we have used some non-parametric statistical tests for multiple comparison [15], [16] over the average performance values obtained on the 52 datasets. Additionally, we analyze the variety of data and the different algorithms' behavior related with the curse of dimensionality, as well as the tuning of the algorithmic parameters by nested cross-fold validation when using the ''train'' function from caret in R, in order to provide some further insights on the behavior of the most promising approaches.

Please, take it into account that we do not try to discard any algorithm based on the results obtained but only to find possible potentialities. A model explaining well a certain situation may fail in another situation. The ''No Free Lunch'' theorem states that ''there is no one model that works best for every problem'' [17]. And of course, this is still true after our particular study.

Finally, a web page associated with this paper (i.e., http://www4.ujaen.es/~mgacto/regression/study/http://www4.ujaen.es/~mgacto/regression/study/), which contains complementary material to this study, has been also developed. It includes, the datasets collected and used in this study (the 5-fold cross-validation partitions) together with the generated results (errors and times) per algorithm and dataset (164×52), which can be found in a downloadable spreadsheet. Furthermore, it also includes the complete results by types of datasets on the 164 algorithms for the curse of dimensionality study. These public materials will ease further well-founded comparisons and studies from the specialized research community.

This paper is organized as follows. The next section describes the set-up of the experimental study considered in this paper and proposes a new absolute metric and quality categories for assessing regression algorithms goodness over different datasets. Section III analyzes and discusses the obtained results. Finally, in Section IV we draw some conclusions.

II. EXPERIMENTAL SETUP

Several experiments have been performed to evaluate the performance of the analyzed algorithms. In the following, we firstly show the datasets used in the experimental study; second, we introduce a new quality measure proposed for these kinds of studies; third, we present the widely known and used software tools including the public regression algorithms analyzed in this contribution; fourth, we introduce a

TABLE 1. Datasets used for the experimental study.

Datasets								
Name	Var	Examples	Name	Var	Examples	Name	Var	Examples
2DPLANES	10	40768	DELTAAIL	5	7129	MPG8	7	392
ABA	8	4177	DELTAELV	6	9517	MV	10	40768
ADD10	10	9792	DIABETES	2	43	PLA	2	1650
AIL	40	13750	DIAMOND	18	308	POLE	26	14998
AIRFOIL	5	1503	ELE1	2	495	PUMA32	32	8192
ANA	7	4052	ELE2	4	1056	PUMA8	8	8192
AUTOPRICE	15	159	ELV	18	16599	PYRIM	27	74
BANK32	32	8192	FAT	14	252	QUA	3	2178
BANK8	8	8192	FOR	12	517	STO	9	950
BAS	16	337	FRIED	5	1200	STRIKES	6	625
BOSTON	13	506	HOUSE16	16	22784	TRE	15	1049
CA	21	8192	HOUSE8	8	22784	TRIAZ	60	186
CAL	8	20640	KINE32	32	8192	WA	9	1609
CASP	9	45730	KINE8	8	8192	WI	9	1461
CCPP	4	9568	LASER	4	993	WPBC	32	194
CONCRETE	8	1030	MACHINECPU	6	209	YH	6	308
CPU_SMALL	12	8192	MOR	15	1049			
DEE	6	365	MPG6	5	392			

brief description of the studied algorithms and their configurations; and finally we describe the statistical analysis that is performed in this study.

A. DATASETS

The experiments have been carried out over 52 real-world datasets available in the well-known public repositories, with a number of variables within the interval [2, 60] and a number of examples within the interval [43, 45730]. These datasets have been downloaded from the following repositories: https://archive.ics.uci.edu/ml/datasets.html?format=&task=reg&att=&area=&numAtt=&numIns=&type=&sort=nameUp&view=tableUCI Machine Learning Repository [18], http://sci2s.ugr.es/keel/category.php?cat=regKEEL-dataset [19], http://www.cs.waikato.ac.nz/ml/Weka/datasets.htmlDataset Collections of Weka [7], [8], http://www.cs.toronto.edu/~delve/data/datasets.htmlDelve Datasets [20], [21], http://www.dcc.fc.up.pt/~ltorgo/Regression/DataSets.htmlLuis Torgo Repository [22], and http://ww2.amstat.org/publications/jse/jse_data_archive.htmJournal of Statistics Education Data Archive [23].

We have included all the available standard regression datasets from these repositories. To the best of our knowledge no study has been performed previously on this quantity of standard regression datasets since it is quite difficult to find them public (with 28 being the highest number previously considered in a particular comparison from our knowledge to the date [24]). Table 1 summarizes the main characteristics of the datasets, where *Name* is the short name, *Var* is the number of input variables, and *Examples* is the number of examples.

In all the experiments, we adopted a *5-fold cross-validation model*, i.e., we randomly split the dataset into 5 folds, each containing 20% of the examples of the dataset, where four folds have been used for training and one for testing. These datasets and their 5-fold crosvalidation partitions are available in the complementary material web page associated to this paper (http://www4.ujaen.es/~mgacto/regression/study/). Finally, for each of the five partitions, we executed three trials of the algorithms (same 3 different seeds for all), of course, only when they are non deterministic approaches.

B. QUALITY MEASURES CONSIDERED: RegM PROPOSAL

To evaluate each algorithm we have used the well-known Mean Square Error (MSE):

$$MSE = \frac{1}{N}\sum_{l=1}^{N}(alg(x^l) - y^l)^2, \quad (1)$$

where N is the number of examples of the dataset, $alg(x^l)$ is the output obtained from the model generated by the algorithm when the l-th example is considered and y^l is the known desired output.

In regression problems, the average MSEs obtained by an algorithm may not represent its real performance magnitude when it is compared to any other algorithm, since the domain of the output variable is different for each dataset. Therefore, MSE is a non absolute value depending on the range of each dataset outputs (estimated continuous values). MSE normalization could be a solution but it is very difficult to know the minimum and maximum possible MSE values on each given dataset. However, from the results in this paper we will have not only good estimations of the minimum and maximum possible MSE, but also a distribution with 164 values per dataset becoming a well-supported sample representation on what we could expect from regression algorithms.

Based on this distribution for each of the addressed datasets, we have also proposed a new absolute measure (*RegM*) in order to identify a comparable average goodness value of the results obtained by an algorithm on different datasets (as in classification, where we can easily compute the average correct classification percentage, from 0 to 100%). In order to address this problem, we could properly identify the median point over 164 MSE values as the MSE expected for a normal (on or over 50% of the studied algorithms), or at

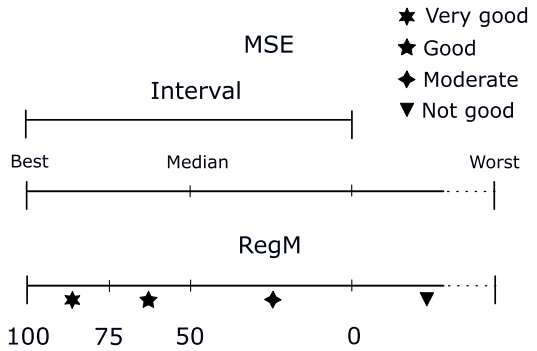

FIGURE 1. Normalization interval, performance categories, associated symbols and labels, and *RegM* domains for dataset *d*.

least reasonably, well-performing algorithm (i.e., not particularly good or bad). We will fix this median value as the 50% performance scoring (in a range from 0 to 100%).

Since we have also the best and worst MSEs, we could now define intervals for normalization. However, while considering from median MSE to best MSE seems appropriate (these values are determined by well-performing algorithms), considering from median to worst MSE could be somehow as throwing the dice, since any algorithm performing bad could obtain unexpectedly high errors. In this sense, we have defined as appropriate interval for normalization twice the difference between the best MSE and the median MSE obtained for the dataset. It is, we fix the measurable loss of performance as equal to the possible improvement. See Figure 1 top for a graphical representation of this interval definition. Taking it into account, this interval is defined as:

$$Interval = (MSE^d_{Median} - MSE^d_{Best}) * 2 \tag{2}$$

where MSE^d_{Best} is the best MSE obtained by the analyzed algorithms in the dataset d, MSE^d_{Median} is the median of the MSEs obtained by the analyzed set of algorithms in the dataset d (concretely, in this study, 164 algorithms). Thus, the new measure *RegM* for the algorithm *alg* is defined as:

$$RegM^d_{alg} = max(0, 1 - \frac{MSE^d_{alg} - MSE^d_{Best}}{Interval}) \tag{3}$$

$$RegM_{alg} = \frac{100}{N_{Dat}} \sum_{d=1}^{N_{Dat}} RegM^d_{alg} \tag{4}$$

where MSE^d_{alg} is the MSE obtained by the algorithm *alg* in the dataset d, and N_{Dat} is the number of datasets (52 datasets in this study).

This measure takes values in the interval [0, 100], where we define four qualitative categories with values in [75, 100] representing algorithms with a very good performance, values in [50, 75) representing algorithms with a good performance, values in [0, 50) representing algorithms with a moderate performance, and values bellow 0 representing a not good performance. Figure 1 shows the definition of the performance categories and their interpretation (symbols and labels) together with the associated *RegM* value domains for a given dataset d. From this study, these domains could be taken as reference values for each of the 52 datasets, thus making easier testing a new proposal.

C. SOFTWARE USED FOR THE EXPERIMENTS

In this paper we have considered 6 public software tools including the analyzed algorithms in the experimental study. A brief description of these software tools can be found in the following:

Java Statistical Analysis Tool (JSAT) [9] is a "library for quickly getting started with Machine Learning problems" written in Java. The library has no external dependencies, and almost all of the algorithms are independently implemented using an Object-Oriented framework. JSAT is suitable for small and medium size problems and it is made available for use under the GPL 3. JSAT. Version 0.0.9 has been employed.

Knowledge Extraction based on Evolutionary Learning (KEEL) [10] is a "Java software tool that can be used for a large number of different knowledge data discovery tasks. KEEL provides a simple GUI based on data flows to design experiments with different datasets and computational intelligence algorithms (paying special attention to evolutionary algorithms). It contains a wide variety of classical knowledge extraction algorithms, preprocessing techniques, computational intelligence based learning algorithms, hybrid models, statistical methodologies for contrasting experiments and so forth". KEEL is open source for use under the GPL 3. We have used the current KEEL version with date of creation 2018-04-09.

MATLAB (MATrix LABoratory) [5] is a commercial multiparadigm numerical computing environment developed by MathWorks. This environment allows "matrix manipulations, functions and data plotting, implementation of algorithms (including machine learning methods". In the paper, we have used the version R2016a. Moreover, we have used several toolboxes implemented by Gints Jekabsons. The toolbox codes are open source regression software for Matlab/Octave and are licensed under the GNU GPL license. These toolboxes are: ARESLab [25] version 1.13.0, M5PrimeLab [26] version 1.7.0, and PRIM [27] version 2.2.

Scikit-learn [11] is a library for Machine Learning in Python. "It features a rich number of supervised and unsupervised learning algorithms and builds on NumPy, SciPy, and matplotlib". Sciki-learn is open source software issued under BSD 3 license. We have used Scikit-learn version 0.18.1 included in Anaconda 3-4.3.1 (Python distribution 3.6).

R [6] is a "free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS". Many useful R functions come in packages and free libraries of code written by the R's active users community. The software can redistribute it and/or modify it under the terms of the GNU GPL as published by the Free Software Foundation. It is available on the web page https://cran.r-project.org/. We have considered the R version 3.5. Moreover, when it is possible

we use the *train* function from caret package in R to set up a grid of tuning parameters for regression routines, to fit each model and calculate a resampling based performance measure. This allows learning which is the best parameter for the algorithm, for example, to set the correct value for k parameter in the k-Nearest Neighbors (kNN) method.

Weka [7], [8] is a data mining software tool ''including a collection of machine learning algorithms for data mining tasks'' developed by University of Waikato. ''The algorithms can either be applied directly to a dataset or called from your own Java code. Weka contains tools for data pre-processing, classification, regression, clustering, association rules, and visualization''. It is also well-suited for developing new machine learning schemes. Weka is open source software issued under the GNU General Public License. We have used Weka version 3.9.1.

D. ALGORITHMS AND PARAMETERS CONSIDERED IN THE EXPERIMENTS

In the experiments, 164 algorithms for regression problems available in the software tools JSAT, KEEL, Matlab, R, Scikit-learn, and Weka have been used. Notice that although a few additional algorithms are available in these tools, they have not been included in the experiments since they were not able to run in all the considered datasets due to scalability problems.

These 164 algorithms (actually different ones or some particular implementations of the same ones) have been grouped into 14 families by unifying the own tools categorization: Neural Networks (NNET): 21 algorithms; Support Vector Machines (SVM): 16 algorithm; Regression Trees (RT): 17 algorithms; Rule-based Methods (RL): 9 algorithms; Stacking (STA): 2 algorithms; Random Forests (RF): 10 algorithms; Model Trees (MT): 8 algorithms; Generalized Linear Models (GLM): 29 algorithms; Nearest Neighbor methods (NN): 7 algorithms; Partial Least Squares and Principal Component Regression (PLSR): 4 algorithms; Multivariate Adaptive Regression Splines (MARS): 4 algorithms; Bagging (BAG): 18 algorithms; Boosting (BST): 5 algorithms; and Other Methods (OM): 14 algorithms. In order to facilitate the analysis, we have used the same terminology used by the authors in [28], where a similar study was performed for classification techniques.

A brief description of these algorithms is as follows (sorted by category and alphabetically). Each algorithm has been identified by its name in the software tool in which it is available followed by the short form of the software tool. For example, JSAT:J; Weka:W; KEEL:K; Matlab:M; Scikit-learn:P; R:R; R using caret with implicit use of the caret *Train* function):T.

In relation with parameters, the standard ones recommended by authors (those included in each tool as recommended parameters by default), except in R for those that are tuneable with train have been considered. In some cases, for different implementations of the same algorithm any of the parameters proposed in each software tool by default are different. In these cases, we have determined the best parameter value experimentally, and set it the same for all the cases. For example, the number of total trees in the Random Forest based algorithms is not 500 by default in all the software tools but only in some of them. Setting up all of them to 500 improved their results systematically, without significant improvements over this value, so that for fair comparison with the different software tools we fix it to 500 for all the cases. It was the same for the bagging application, where in some cases 25 bags are recommended and for some others 50. We fixed it to 50 without significant changes over this value.

Moreover, as we previously said, when it is possible we use the *train* function from caret package in R to set up a grid of tuning parameters for regression routines, to fit each model and calculate a resampling based performance measure. It involves a nested cross-fold validation, where only training data is used for inner cross-fold validation and selection of the best parameters, and test data is lately used on the final application of these selected parameters. This is applicable to some methods that are external to caret. In these cases, we have included both versions, without algorithmic parameter tuning (as they are recommended in their own packages) and with algorithmic parameter tuning by *train* (in order to also check this possibility). However, it must be said that the use of the *train* function imposes high computational restrictions due to the cost timing required in tuning the parameters slowing down the overall operation of the algorithm, so that in some cases it is impossible performing this type of parameter tuning since the computational time needed goes beyond the wise (see Sections III-D and III-F, where a computational time analysis and the effects of the tuning of algorithmic parameters are briefly studied).

Therefore, even though using this type of automatic nested cross-fold validation could be a non biased way to perform for the best algorithmic parameters selection (not by hand trial and error, where you could check the test error), in the analyzed software tools it is only directly available for a reduced set of algorithms (in order to ease non-expert users, to whom this contributions is particularly focused, who are not usually able to implement the corresponding scripts or to modify the available implementations). Moreover, the much higher computational cost on $164 \times 52 \times 5$ (algorithms, datasets and folds... even seeds for non-deterministic methods) makes it impossible to apply for the whole study, i.e., the application of 42,640 nested cross-fold validations over different combinations of algorithmic parameters.

On the other hand, from the brief study on the *train* parameter tuning in section III-F, we can see as it is only reporting slight improvements in general and that even sometimes the test errors (generalization ability) worsen significantly. It is the problem when we are prohibited to check test errors while tuning or developing a method in a given single problem, that we do not know whether it is overfitting or not.

All these reasons are why for these type of studies, standard parameters are even recommended [15], [16]. In our case, we are applying the standard ones, which also represents

the real situation when non-expert users need to apply data mining techniques to the problems they face in their respective areas, in order to show a general (of course non perfect) estimation on how they perform, which is one of the main objectives of this contribution.

1) NEURAL NETWORKS (NNET): 21 ALGORITHMS

1) **avNNet-T** from the caret package, creates a committee of multi-layer perceptrons (MLPs) from the nnet package (the number of MLPs is given by parameter repeat) trained with different random weight initializations. The tunable parameters are the #hidden neurons (size) in {1, 3, 5} and the weight decay (values {0, 0.1, 10^{-4}}). This low number of hidden neurons is to reduce the computational cost of the ensemble.
2) **BackPropagationNet-J** is an implementation of a feed forward neural network trained by back propagation. NNets are powerful classifiers and regressors, but can suffer from slow training time and overfitting.
3) **elm-M** is a extreme learning machine [29] implemented in Matlab using the code freely available in the elm Web (http://www3.ntu.edu.sg/home/egbhuang/elm_codes.html), using sigmoidal function for activation functions and 20 as the value for #hidden neurons.
4) **elmNN-R** [29], [30] trains of a generic single hidden-layer feed forward neural network using ELM algorithm, from the elmNN package.
5) **EnsembleR-K** [31] is an ensemble neural network for regression problems. The method employs an ensemble construction based on the use of nonlinear projections to achieve both accuracy and diversity of individual regressors. It also uses the philosophy of boosting for difficult instances.
6) **iRProp+-K** [32] is a regression model by means of product unit neural networks or multilayer perceptrons trained with the iRProp+ algorithm.
7) **MLP-BP-K** [33] is an MLP for regression problems, with back-propagation as learning technique. The networks apply a sigmoid function as an activation function.
8) **mlp-R** creates a MLP and learns it with backpropagation, by using the RSNNS package.
9) **MLPRegressor-P** produces an MLP regressor which optimizes the squared-loss using stochastic gradient-based proposed by Kingma [34].
10) **mlpWeightDecay-T** trains MLP networks using caret to access the RSNNS package with #hidden neurons and the weight decay parameter tuning.
11) **MultilayerPerceptron-W** is an MLP network with sigmoid hidden neurons, unthresholded linear output neurons, learning rate 0.3, momentum 0.2, 500 training epochs, and #hidden neurons equal to (#inputs)/2.
12) **newff-M** creates a feed-forward backpropagation network implemented in Matlab with hidden neurons 3:3:30. Matlab v. 7.9.0.529 (R2009b) with Neural Network Toolbox v. 6.0.3
13) **NNEP-K** [33] consists of obtaining the neural network architecture and simultaneously estimating the weights of the model coefficients with an algorithm of evolutionary computation.
14) **nnet-R** [35] fits single-hidden-layer neural network possibly with skip-layer connections using nnet package, considering 10 as the number of hidden layer.
15) **nnet-T** uses caret as interface to function nnet in the nnet package, training an MLP network. The tunable parameters are the #hidden neurons (size) with 1:2:9 and the weight decay values {0,0.1,0.01, 0.001,0.0001}.
16) **pcaNNet-T** trains the MLP using caret and the nnet package, and running principal component analysis (PCA) previously on the data set. The tunable parameters are the size with 1:2:9 and weight decay values{0,0.1,0.01,0.001,0.0001}.
17) **rbf-R** creates a radial basis function (RBF) network in the RSNNS package considering default values. The number of hidden neurons takes values 5 or 3 (for smaller datasets) depending on the datasets.
18) **rbfDDA-R** [36] creates incrementally from the scratch a RBF network with dynamic decay adjustment (DDA), in the RSNNS package.
19) **RBFNet-J** produces a RBF neural network which uses K-means to select the RBF centers. Using a number of clusters (or hidden neurons) equal to 25.
20) **RBFNR-K** [36] builds a RBF neural network composed of one hidden layer and one output layer. This hidden layer contains neurons, each one being activated when the input to the network falls close to a point that is considered the center of that neuron. The final result of the network is provided by the neurons of the output layer that perform a weighted sum using the outputs coming from hidden neurons.
21) **RBFRegressor-W** implements a normalized gaussian RBF network. It uses the k-means clustering algorithm to provide the basis functions.

2) SUPPORT VECTOR MACHINES (SVM): 16 ALGORITHMS

22) **DCD-J** implements Dual Coordinate Descent (DCD) [37], [38] training algorithms for a Linear L1 or L2 SVM for binary classification and regression (in our case, we use the default L1), without the shrinkage optimization.
23) **DCDs-J** creates a linear SVM trained by DCD.
24) **EPSILON-SVR-K** builds regression models by means of EPSILON-SVM [39] in libSVM [40] library.
25) **fitrsvm-M** fits an SVM regression model on a low-through moderate-dimensional predictor data set.
26) **ksvmEpsilon-R** uses the function ksvm [41] in the kernlab package with epsilon regression.
27) **ksvmNu-R** uses the function ksvm [41] (kernlab package) with Nu regression.
28) **LibLINEAR-R** [42] creates Linear predictive models estimation based on the LIBLINEAR C/C++ Library

in the LibLinear package, with type 11. We have been testing with three values for type parameter:
- 11 L2-regularized L2-loss support vector regression (primal)
- 12 L2-regularized L2-loss support vector regression (dual)
- 13 L2-regularized L1-loss support vector regression (dual)

The best result was obtained by type 11 that we finally use in this contribution.

29) **LinearSVR-P** is a scalable linear SVM for regression implemented using liblinear [40].
30) **NuSVR-K** creates regression model by means of NU-SVM [39] based on libSVM [40] library.
31) **NuSVR-P** is an SVM for regression implemented with libsvm with a parameter to control the number of support vectors.
32) **SMOreg-W** [43] is an SVM for regression. The parameters are learned using RegSMOImproved with C=1 and polynomial kernel. RegSMOImproved learns SVM using Sequential Minimal Optimization (SMO) with adaption of the stopping criterion.
33) **svm-R** creates an SVM with the kernel used in training and predicting by radial basis, using the library LibSVM [40] in the e1071 package.
34) **svmLinear-R** [40] uses the function SVM (e1071 package) with linear kernel.
35) **svmPoly-R** [40] uses the e1071 package to create a SVM with polynomial kernels.
36) **svmSigmoid-R** [40] trains an SVM with sigmoid kernel in e1071 package.
37) **SVR-P** implements an epsilon-support vector regression based on libsvm [40] library.

3) REGRESSION TREES (RT): 17 ALGORITHMS

38) **ctree-T** uses the function ctree [44], [45] in the party package, which creates conditional inference trees by recursive partitioning for continuous, censored, ordered, nominal and multivariate response variables in a conditional inference framework. The threshold in the association measure is given by the parameter mincriterion, tuned with the values 0.1:0.11:0.99 (10 values).
39) **ctree2-T** uses the function ctree tuning the maximum tree depth with values up to 10.
40) **DecisionStump-W** is a one-node regression tree which develops classification or regression based on just one input using entropy.
41) **DecisionTree-J** is a generic implementation, allowing the ability to mimic the behavior of many tree algorithms such as C4.5 [46] (for classification) and CART [47] (regression).
42) **DecisionTreeReg-P** [47], [48] is a simple decision tree regressor. It creates a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.
43) **ExtraTreesReg-P** [49] implements a meta estimator that fits a number of randomized regression trees (extra-trees) on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. Consider 500 as the number of default trees in the forest previous to the final meta estimator.
44) **fitrtree-M** [47], [50], [51] fits a binary regression decision tree.
45) **quantregForest-R** [52] infers conditional quantile functions from data based on previously obtained quantile regression forests. Included in quantregForest package.
46) **RandomSubSpace-W** [53] trains multiple REPTrees regressors selecting random subsets of inputs (random subspaces) to obtain a decision tree based classifier. Each REPTree is learnt using information gain/variance and error-based pruning with backfitting. Each subspace includes the 50% of the inputs. The minimum variance for splitting is 0.001, with at least 2 patterns per leaf.
47) **RandomTree-J** is a regression tree that chooses a random subset of features at each iteration to consider.
48) **RandomTree-W** is a non-pruned tree where each leaf tests log_2(#inputs+1) randomly chosen inputs, with at least 2 instances per leaf, unlimited tree depth and without backfitting.
49) **REPTree-W** learns a fast pruned regression tree using information variance and Reduced Error Pruning (REP). It uses at least 2 training patterns per leaf, 3 folds for reduced error pruning and unbounded tree depth. The minimum proportion of the variance on all the data for splitting is 0.001.
50) **rpart-R** [47] implements CART method using the function rpart in the rpart package, which develops recursive partitioning.
51) **rpart-T** uses the same previous function tuning the complexity parameter (threshold on the accuracy increasing achieved by a tentative split in order to be accepted) with 6 values from 0.1 to 0.11.
52) **rpart1SE-T** trains CART using caret and the rpart package with no tuning parameters.
53) **rpart2-T** uses the function rpart by tuning the tree depth with values up to 10.
54) **tree-R** [35], [47] grows a tree by binary recursive partitioning using the response in the specified formula and choosing splits from the terms of the right-hand-side.

4) RULE-BASED METHODS (RL): 9 ALGORITHMS

55) **ConjunctiveRule-W** uses a single rule whose antecedent is the AND of several antecedents and whose consequent is the mean for a numeric value. If the test instance is not covered by this rule, then it's predicted using the value of the data not covered by the rule in the training data. This learner selects an antecedent by computing the information gain of each antecedent and prunes the generated rule using REP.

56) **DecisionTable-W** [54] is a simple decision table majority regressor which uses BestFirst as search method.
57) **GFS-GPG-K** [55] is a fuzzy learning based on genetic programming grammar operators.
58) **GFS-GSP-K** [56] implements a symbolic fuzzy learning based on genetic programming grammar operators and simulated annealing.
59) **GFS-SAP-Sym-K** [56] is a symbolic fuzzy-valued data learning based on genetic programming grammar operators and simulated annealing.
60) **GFS-SP-K** [55] produces fuzzy rule learning grammar-GP based operators and simulated annealing-based algorithm.
61) **PRIM-M** implements the Patient Rule Induction Method (PRIM) [57] included in the PRIM toolbox [27]. This method is for finding ''interesting'' regions (bump hunting) in high-dimensional data. The regions are described by hyper-rectangles (boxes) containing simple decision rules.
62) **WM-K** [58] implements the fuzzy rule learning Wang-Mendel algorithm for generating fuzzy rules by learning from examples.
63) **ZeroR-W** predicts the mean for all the test patterns. Obviously, this regressor gives low accuracies, but it serves to give a lower limit on the accuracy.

5) STACKING (STA): 2 ALGORITHMS

64) **Stacking-J** is a stacking ensemble [59]. Stacking learns several base classifiers and a top level classifier learns to predict the target based on the outputs of all the ensemble models. A linear model is used, which translates to learning a weighted vote of the regressor outputs.
65) **Stacking-W** is a stacking ensemble [59] using ZeroR as meta and base regressors.

6) RANDOM FORESTS (RF): 10 ALGORITHMS

66) **cforest-R** is a version of random forest and bagging ensemble of conditional inference trees (ctrees) aggregated by averaging observation weights extracted from each ctree. The parameter mtry takes the value 1 with 500 trees. It uses the caret package to access the party package (no algorithmic parameter tuning is performed).
67) **RandomForest-J** [60] creates a collection of random trees with 500 trees.
68) **randomForest-R** creates a random forest [60] ensemble using the randomForest function in the randomForest package, with parameters ntree = 500 (number of trees in the forest) and mtry=#inputs.
69) **RandomForest-W** implements a forest of RandomTree base classifiers with 500 trees (except in 2dplanes and casp datasets with 300 trees since the method causes memory problems), the number of randomly chosen attributes as $log_2(\#inputs)+1$ and unlimited depth trees.
70) **RandomForestRegressor-P** implements a RF algorithm for regression problem with 500 trees.
71) **ranger-R** is a fast implementation of RF [60] or recursive partitioning, particularly suited for high dimensional data, with 500 trees. It is included in the package ranger.
72) **Rborist-R** is a rapid decision tree construction and evaluation, with 500 trees, provided by Rborist package. The method includes accelerated implementation of the random forest algorithm and it is tuned for multicore and GPU hardware.
73) **rf-T** creates a random forest using the caret interface to the function randomForest in the randomForest package, with ntree = 500 and tuning the parameter mtry with values 2:3:8.
74) **rfsrc-R** is the random forests for survival, regression and classification, with 500 trees, included in randomForestSRC package.
75) **RRF-R** [61], [62] implements regularized random forest algorithm, with 500 trees, included in the package RRF. It is based on the randomForest R package.

7) MODEL TREES (MT): 8 ALGORITHMS

76) **Cubist-R** is a rule-based model that is an extension of the classic Quinland's M5 model tree [63]. A tree is grown where the terminal leaves contain linear regression models.
77) **M5-K** implements M5 model tree.
78) **M5P-R** [64] implements M5 prime model tree using M5P function in the package RWeka.
79) **M5P-W** builds the M5 prime tree regression method.
80) **M5Prime-M** is the M5 prime regression method. This method is included in M5PrimeLab [26] toolbox.
81) **M5Rules-K** implements M5 model rules.
82) **M5Rules-R** creates a M5 model rules using M5Rules function in the package RWeka.
83) **M5Rules-W** builds the same M5 model rules.

8) GENERALIZED LINEAR MODELS (GLM) : 29 ALGORITHMS

84) **bam-mgcv-R** [65], [66] is a version of generalized additive models for very large datasets included in mgcv package.
85) **BayesianRidge-P** builds a bayesian ridge regression model to optimize the regularization parameters.
86) **ElasticNet-P** is a linear regression model trained with L1 and L2 prior as regularizer. It is useful when there are multiple features which are correlated with one another.
87) **fitrlinear-lsr-M** [38] implements linear regression model to high-dimensional data and uses least-squares regression method as learner.
88) **fitrlinear-svm-M** [37], [38], [67] fits linear regression models to high-dimensional data. The method includes regularized support vector machines (SVM) and minimizes the objective function using techniques that reduce computing time (e.g., stochastic gradient descent).
89) **fitlm-M** creates a simple linear regression model.

90) **fitlm-Robust-M** produces a linear robust regression model to reduce outlier effects.
91) **gam-R** is used to fit generalized additive models [68], [69], specified by giving a symbolic description of the additive predictor and a description of the error distribution. It uses the backfitting algorithm to combine different smoothing or fitting methods. The model is included in gam package.
92) **gam-mgcv-R** [70], [71] fits generalized additive models with integrated smoothness estimation in mgcv package.
93) **glm-R** [72] uses the function glm in the stats package with gaussian family.
94) **glmnet-R** trains a GLM via penalized maximum likelihood, with Lasso or elasticnet regularization parameter [73] (using glmnet function in the glmnet package).
95) **glmStepAIC-R** performs generalized linear regression with stepwise selection by Akaike information criterion [74] using the function stepAIC in the MASS package, which es executed by train from caret (no algorithmic parameter tuning is performed).
96) **HuberRegressor-P** [75], [76] is a linear regression model that is robust to outliers.
97) **lars-R** fits Least Angle Regression (LARS) [77] and is included in the package lars. With the ''lasso'' option, it computes the complete lasso solution simultaneously for all values of the shrinkage parameter in the same computational cost as a least squares fit.
98) **Lars-P** performs LARS [77] algorithm for high-dimensional data.
99) **Lasso-P** is a linear model that estimates sparse coefficients. It is useful in some contexts due to its tendency to prefer solutions with fewer parameter values, effectively reducing the number of variables upon which the given solution is dependent.
100) **LassoLars-P** is a lasso model implemented using the LARS algorithm.
101) **LinearRegression-P** produces an ordinary least squares linear regression.
102) **LinearRegression-R** builds suitable linear regression models, using the Akaike criterion for model selection. It is included in RWeka package.
103) **LinearRegression-W** learns a simple linear regression model. It picks the attribute that results in the lowest squared error.
104) **lm-R** [78], [79] is used to fit linear models in stats package.
105) **nnls-R** trains an algorithm for non-negative least squares executed from caret with no tuning of algorithmic parameters.
106) **PassiveAggressive-J** is a version of the passive aggressive algorithm [80] for regression. It is a type of online algorithm that performs the minimal update necessary to correct for a mistake.
107) **PassiveAggressiveRegressor-P** is a passive-aggressive regressor algorithm [80] for large-scale learning.
108) **randomGLM-R** is a random generalized linear model predictor included in the randomGLM package.
109) **SGDRegressor-P** is a linear model fitted by minimizing a regularized empirical loss with stochastic gradient descent.
110) **SimpleLinearRegression-W** implements a simple linear regression model.
111) **stepwiseglm-M** [72], [81], [82] creates generalized linear regression model by stepwise regression.
112) **TheilSenRegressor-P**s [83] implements a robust multivariate regression model. The algorithm uses a generalization of the median in multiple dimensions and it is robust to multivariate outliers.

9) NEAREST NEIGHBOR METHODS (NN): 7 ALGORITHMS

113) **IBk-W** [84] is a k-Nearest Neighbors regressor with linear neighbor search and euclidean distance, considering 7 as the number of nearest neighbors.
114) **IB1-W** is a simple 1-NN regressor.
115) **kknn-R** uses the function kknn in the kknn package considering 7 as the number of neighbors.
116) **knn-T** trains function knn in the caret package with 12 number of neighbors in the range 3:2:25.
117) **KMeans-P** is k-means clustering method. Consider 8 as the number of centroids.
118) **KNeighborsRegressor-P** implements regression algorithm based on k-nearest neighbors. Considering 5 as the number of neighbors by default.
119) **NearestNeighbour-J** is a nearest neighbor algorithm with 7 as the number of neighbors.

10) PARTIAL LEAST SQUARES AND PRINCIPAL COMPONENT REGRESSION (PLSR): 4 ALGORITHMS

120) **kernelpls-R** [85] performs partial least squares regression with the function plsr (in the pls package) and method=kernelpls.
121) **pcr-R** performs principal components regression using pls package.
122) **pls-R** uses the function mvr in the pls package to perform partial least squares regression, which es executed by train from caret (no algorithmic parameter tuning is performed).
123) **simpls-R** considers the same function plsr using the SIMPLS [86] method in the pls package.

11) MULTIVARIATE ADAPTIVE REGRESSION SPLINES (MARS): 4 ALGORITHMS

124) **earth-R** builds MARS [87], [88], in earth package.
125) **gcvEarth-R** uses the function earth in the earth package. It builds an additive MARS model without interaction terms using the fast MARS [48] method.
126) **mars-M** is the MARS method included in the ARESLab [25] toolbox.
127) **mars-R** fits a MARS [87] model using the function mars in the mda package.

12) BAGGING (BAG): 18 ALGORITHMS

It is one of the cases where some of the software tools use a different number of bags, it is 10, 20 or 50. In [89], the authors tested with different number of bags indicating that 50 or 25 bags were necessary or sufficient to obtain good results within reasonable execution times. In our experiments, 50 bags have been used for all the methods.

In the case of the ensemble based methods (RF based ones among others) it makes no sense the bagging application (for example method ExtraTreeReg-P). We tried it over a good amount of datasets and it did not significantly improved the results but it greatly increased the execution time. It is the case as said among others, when bagging is applied to randomForest-R, thus making its computational time cost extremely high without signficant improvement. Thereforre, such contributions have not been considered as cases of study.

128) **bagEarth-R** is a bagging ensemble of MARS method included in the earth package with 50 bagging iterations.
129) **bagEarthGCV-R** is a bagged MARS method from the earth package, using gCV pruning with 50 bagging iterations.
130) **bagging-DecisionStump-W** uses DecisionStump base regressor with 50 bagging iterations.
131) **bagging-DecisionTable-W** uses DecisionTable with BestFirst and forward search, leave-one-out validation and RMSE as measure used to evaluate the performance, with 50 bagging iterations.
132) **bagging-DecisionTree-J** is an ensemble technique for reducing variance that uses DecisionTree base regressors with 50 bagging iterations.
133) **bagging-DecisionTree-P** is an ensemble meta-estimator that fits decision tree base regressors each on random subsets of the original dataset and then aggregate their individual predictions (the number of estimators in the ensemble is 50) to form a final prediction.
134) **bagging-IBk-W** uses IBk base classifiers, which develop kNN regressor tuning K using cross-validation with linear neighbor search and Euclidean distance, with 50 bagging iterations.
135) **bagging-MultilayerPerceptron-W** is a bagging with 50 iterations using the same configuration as the single MultilayerPerceptron-w method.
136) **bagging-M5P-R** uses M5P base regressor with 50 bagging iterations.
137) **bagging-M5P-W** applies bagging with 50 iterations to the same M5P base regressor.
138) **bagging-M5Rules-R** uses M5Rules base regressor with 50 bagging iterations.
139) **bagging-M5Rules-W** builds a bagging with 50 iterations using M5Rules method as base regressor.
140) **bagging-RandomTree-J** is a bagging ensemble that uses RandomDecisionTree base regressor with 50 bagging iterations.
141) **bagging-RandomTree-W** applies bagging with RandomTree base regressor without backfitting, with unlimited tree depth, considering $[log_2(\#inputs) + 1]$ as the number of random inputs, and 2 as the number of instances per leaf, with 50 bagging iterations.
142) **bagging-REPTree-W** uses REPTree with 2 instances per leaf, minimum class variance 0.001, 3-fold for reduced error pruning and unlimited tree depth, with 50 bagging iterations.
143) **bagging-Rpart-R** [89] is a bagging ensemble with 50 bagging iterations of decision trees (rpart method) using the function bagging (in the ipred package).
144) **treebag-R** trains a bagging ensemble of linear discriminant analysis with option bagControl=ldaBag and 50 bagging iterations.
145) **treeBagger-M** creates a bag of regression trees with 50 trees. TreeBagger grows the decision trees in the ensemble using bootstrap samples of the data. Also, the method selects a random subset of predictors to use at each decision split as in the random forest algorithm.

13) BOOSTING (BST): 5 ALGORITHMS

146) **bstls-R** uses a gradient boosting for optimizing loss functions with component wise linear models as base learners, with the function bst (from the bst package), learner=ls and number of boosting iterations equals 50.
147) **bsttree-R** fits a boosting for regression using the tree regression models, with the function bst (from the bst package), learner=tree and the same number of iterations of the bstls method.
148) **glmboost-R** is the gradient boosting for optimizing arbitrary loss functions where component-wise linear models are utilized as base-learners. It is included in mboost package and uses 100 as number of boosting iterations.
149) **fitensembleBst-M** is a regression tree ensemble using LSBoost and 100 learning cycles. LSBoost is the gradient boosting strategy applied for least squares from Friedman [90].
150) **GradientBoostingRegressor-P** is a Gradient Boosting for regression. It builds an additive model in a forward stage-wise fashion; it allows for the optimization of arbitrary differentiable loss functions. In each stage a regression tree is fit on the negative gradient of the given loss function. It uses 100 as the number of boosting stages to perform.

14) OTHER METHODS (OM): 14 ALGORITHMS

151) **AdditiveRegression-W** [91] is a method that helps to improve the performance of the regression where each iteration adjusts a model to the residuals left by the regressor on the previous iteration. The prediction is obtained by adding the predictions of each regressor. This, avoids overfitting but increases the learning time.
152) **AttributeSelectedClassifier-W** uses M5P trees to classify patterns reduced by attribute selection. The CfsSubsetEval method [92] selects the best group of attributes weighting their individual predictive ability and their degree of redundancy, preferring groups with high

correlation within outputs. The BestFirst forward search method is used, stopping the search when five non-improving nodes are found.

153) **foba-R** is a greedy variable selection for ridge regression using a forward greedy, backward greedy and the Adaptive Forward-Backward Greedy (FoBa) [93] methods. This method is included in foba package.
154) **KernelRLS-J** [94] implements Kernel Recursive Least Squares (RLS) for online regression learning. This is a kernelization of the RLS algorithm, and it uses projection for bounded learning.
155) **KStar-W** [95] is an instance-based regressor which uses entropy based similarity to assign a test instance to the output of its nearest training instances.
156) **LWL-J** [96] is a Local Weighted Learning (LWL) that builds a local model for every query, and uses that local model to make predictions.
157) **LWL-W** [97] is an ensemble of Decision-Stump base regressors. Each training instance is weighted with a linear weighting kernel, using the Euclidean distance for a linear search of the nearest neighbor.
158) **MultiScheme-W** selects a regressor among several ZeroR regressors using cross validation on the training set.
159) **ppr-R** builds a projection pursuit regression model [98]. It is included in stats library.
160) **RandomCommittee-W** is an ensemble of RandomTrees (each one built using a different seed) whose output is the average of the base regressor outputs.
161) **relaxo-R** builds relaxed lasso solutions [99] included in the relaxo package.
162) **Ridge-P** performs linear least squares with l2 regularization. The model solves a regression model where the loss function is the linear least squares function and regularization is given by the l2-norm.
163) **RidgeRegression-J** creates a simple batch implementation of ridge regression.
164) **spikeslab-R** fits a rescaled spike and slab model [100], [101] using a continuous bimodal prior in the spikeslab package. A generalized elastic net estimator is used for variable selection and estimation. It can be used for prediction and variable selection in low and high-dimensional linear regression models.

E. STATISTICAL ANALYSIS

In order to assess whether significant differences exist among the results, we have adopted statistical analysis [15], [16], concretely non-parametric tests. According to the recommendations made in [15] and [16], a set of simple, safe and robust non-parametric tests for statistical comparisons of regressors has been considered. We have employed the Friedman's test [102] in order to rank the studied algorithms and to find out whether at least a significant difference exists among any of the mean values. And then, we have proceed with the post-hoc Holm's test [103] in order to find the concrete pairwise comparisons which produce differences.

Notice that the Holm's test have only been applied to the results obtained by the best 30 algorithms Friedman's ranking since the total number of algorithms is too high to compute this test. A detailed description of these tests and explanations of the use of non-parametric tests for data mining and Computational Intelligence can be found at the Website at http://sci2s.ugr.es/sicidm/.

III. RESULTS AND DISCUSSION

In order to evaluate the performance of the analyzed algorithms, several analyzed have been performed in this paper, which are organized in this section as follows:

- In Subsection III-A, we present the rankings and average RegM results and we analyze the performance of the 164 algorithms studied.
- In Subsection III-B, we analyze the best 30 algorithms in rank without considering the algorithms that make use of bagging.
- In Subsection III-C, we analyze the best 30 algorithms in rank by considering the algorithms that make use of bagging.
- In Subsection III-D, we analyze the scalability of the studied algorithms.
- In Subsection III-E, we analyze the variety of data and the different algorithms' behavior related with the curse of dimensionality.
- In Subsection III-F, we analyze the tuning of the algorithmic parameters by ''train''.
- In Subsection III-G, we analyze the results obtained grouped by algorithm family.

A. ANALYSIS OF THE 164 ALGORITHMS AVAILABLE IN THE STUDIED SOFTWARE TOOLS

Several executions have been carried out on different datasets in order to analyze the performance of the 164 algorithms (see Subsections III-A, II-D and III-G). Tables 2, 3 and 4 sum up the average RegM results obtained by each algorithm (sorted by rankings) where *Rank* represents the Friedman's ranking for the averaged error (MSE) obtained over the test data; *RegM* represents the values obtained for the new measure proposed in this paper (see Subsection II-B); *Win* is the number of datasets in which the algorithm obtains the best MSE over the test data; ✱ represents the number of datasets in which the algorithm has obtained a value in [75, 100] for the measure *RegM*; ★ represents the number of datasets in which the algorithm has obtained a value in [50, 75) for the measure *RegM*; ✦ represents the number of datasets in which the algorithm has obtained a value in [0, 50) for the measure *RegM*; and ▼ represents the number of datasets in which the algorithm would obtain a value bellow 0 for *RegM*. Finally, *AvTime* is the average computational cost in seconds.

All these values have been computed over the particular MSE results which can be found in a downloadable spreadsheet at the web page associated with this paper (i.e., http://www4.ujaen.es/ mgacto/regression/study/http://www4.ujaen.es/~mgacto/regression/study/). It includes the

TABLE 2. Results obtained by the studied methods. (I/III).

Algorithm	Rank	RegM	Win	✸	★	✦	▼	AvTime
bagging-M5P-R (BAG)	**18.51**	**85.52**	1	**40**	12	0	0	136.83
bagging-M5P-W (BAG)	19.17	84.92	1	39	12	1	0	33.71
bagging-M5Rules-W (BAG)	19.93	82.99	0	37	13	1	1	166.18
bagging-M5Rules-R (BAG)	20.77	83.76	0	35	16	1	0	411.92
bagging-MultilayerPerceptron-W (BAG)	26.16	80.34	3	36	13	1	2	626.59
randomForest-R (RF)	27.27	81.37	2	39	11	0	2	401.07
RandomForest-W (RF)	27.47	81.13	0	38	11	1	2	19.12
RRF-R (RF)	28.22	80.04	1	35	14	2	1	396.69
bagging-REPTree-W (BAG)	28.53	82.65	0	38	14	0	0	4.19
ExtraTreesReg-P (RT)	28.57	78.98	9	35	11	3	3	13.21
RandomForestRegressor-P (RF)	30.17	78.37	0	35	11	3	3	28.25
bagging-RandomTree-W (BAG)	31.83	79.99	0	36	13	1	2	3.25
treeBagger-M (BAG)	32.43	80.47	0	35	14	3	0	2.12
ranger-R (RF)	32.75	79.53	0	36	14	1	1	86.77
M5P-R (MT)	32.97	80.36	0	34	15	3	0	4.22
GradientBoostingRegressor-P (BST)	34.22	77.43	4	33	13	3	3	0.71
bagging-DecisionTree-P (BAG)	34.58	77.31	0	35	11	3	3	5.95
M5Rules-R (MT)	35.10	77.89	0	31	17	4	0	20.63
Rborist-R (RF)	35.26	75.68	0	32	16	1	3	35.32
M5Rules-W (MT)	35.45	77.70	0	33	14	5	0	5.53
cubist-R (MT)	36.64	74.58	1	32	12	5	3	20.12
rf-T (RF)	37.96	74.13	1	30	16	4	2	4701.09
M5-K (MT)	38.78	76.46	0	29	18	5	0	0.87
M5Rules-K (MT)	39.27	74.44	0	29	16	6	1	16.23
svm-R (SVM)	40.29	74.49	1	31	15	5	1	494.01
bagEarth-R (BAG)	41.63	73.51	0	30	18	3	1	26.28
bagEarthGCV-R (BAG)	41.96	72.49	0	30	16	4	2	549.31
mars-M (MARS)	42.24	74.44	0	28	17	7	0	175.36
ppr-R (OM)	42.53	71.78	1	25	21	2	4	1.77
ksvmEpsilon-R (SVM)	44.42	70.47	0	29	14	6	3	121.62
ksvmNu-R (SVM)	44.59	70.74	0	29	15	7	1	184.47
mars-R (MARS)	46.02	71.34	0	27	20	3	2	2.26
stepwiseglm-M (GLM)	46.62	69.14	1	22	22	7	1	89.91
RBFRegressor-W (NNET)	47.39	70.86	1	24	23	5	0	2.36
RandomCommittee-W (OM)	48.30	71.84	0	33	8	8	3	0.77
earth-R (MARS)	49.92	71.05	0	27	18	5	2	1.38
gcvEarth-R (MARS)	49.92	71.05	0	27	18	5	2	68.05
AttributeSelectedClassifier-W (OM)	50.89	69.17	0	25	17	9	1	0.49
bagging-DecisionTree-J (BAG)	51.52	68.14	0	28	11	10	3	19.60
knn-T (NN)	58.28	63.24	1	23	15	10	4	971.36
ctree-T (RT)	58.85	64.51	0	19	21	11	1	277.48
RandomForest-J (RF)	60.12	60.57	0	16	21	12	3	6.70
fitensembleBst-M (BST)	60.32	62.95	0	16	25	9	2	2.14
kknn-R (NN)	61.29	60.69	0	20	16	9	7	7.80
IBk-W (NN)	61.46	61.06	0	22	15	9	6	14.77
spikeslab-R (OM)	63.75	61.08	0	16	22	13	1	7.98
bagging-Rpart-R (BAG)	63.91	58.13	1	18	18	10	6	5.74
glmStepAIC-R (GLM)	65.09	58.67	0	14	24	12	2	410.35
treebag-R (BAG)	65.18	57.48	0	17	19	10	6	126.87
REPTree-W (RT)	65.50	61.23	0	17	20	14	1	0.09
fitlm-M (GLM)	65.73	58.89	0	14	20	16	2	0.94
LinearRegression-P (GLM)	66.35	58.14	0	14	19	16	3	0.10
BayesianRidge-P (GLM)	66.65	56.31	0	14	19	16	3	0.06
foba-R (OM)	66.72	58.68	0	15	24	11	2	0.54
RidgeRegression-J (OM)	66.80	57.45	0	11	23	17	1	0.04

generated results (errors and times) per algorithm and dataset (164×52).

The following facts can be highlighted from the results presented in the Tables 2, 3 and 4:

- The bagging methodology have been applied to several algorithms belonging to different families (MT, MARS, NNET, RT, among others). From the results shown in the tables; it can be drawn that applying bagging to simple algorithms allows obtaining quite improved results with a reasonable computational cost. Individual regression methods tend to overfit but bootstrap-aggregated (bagged) regression combine the results from many regressors, reducing the effects of overfitting and improving the accuracy. Nevertheless, we can still find some bagging based algorithms remaining in the last ranking positions. The best one is bagging-M5P-R, with the best RegM value (85.52) and with zero results within the moderate/bad result interval zones, represented by ✦ and ▼, respectively. There is also another algorithm with zero results within

TABLE 3. Results obtained by the studied methods. (II/III).

Algorithm	Rank	RegM	Win	✸	★	✦	▼	AvTime
LinearRegression-W (GLM)	66.96	56.96	0	12	23	15	2	0.04
bam-mgcv-R (GLM)	67.09	58.66	0	15	24	11	2	6.47
NNEP-K (NNET)	67.15	60.30	0	18	22	7	5	1720.27
Ridge-P (OM)	67.24	55.19	0	10	22	16	4	0.02
gam-mgcv-R (GLM)	67.26	58.67	0	15	24	11	2	1.21
LinearRegression-R (GLM)	69.05	56.37	0	11	**26**	13	2	1.21
bagging-IBk-W (BAG)	69.44	55.47	0	17	15	13	7	591.00
gam-R (GLM)	69.77	56.78	0	14	25	10	3	1.74
glm-R (GLM)	69.77	56.78	0	14	25	10	3	0.69
lm-R (GLM)	69.77	56.78	0	14	25	10	3	0.69
RandomSubSpace-W (RT)	70.09	55.38	0	13	21	13	5	0.53
randomGLM-R (GLM)	70.17	55.42	0	14	22	11	5	145.83
M5P-W (MT)	70.74	52.99	0	15	15	13	9	0.64
MultilayerPerceptron-W (NNET)	72.91	56.39	0	19	14	11	8	12.89
M5Prime-M (MT)	72.99	55.10	0	17	14	12	9	54.45
fitrtree-M (RT)	73.82	53.92	0	14	21	7	10	1.30
Kstar-W (OM)	73.84	50.48	0	16	14	10	12	2283.74
bagging-RandomTree-J (BAG)	74.09	48.50	0	12	19	13	8	1.06
glmboost-R (BST)	75.03	55.66	0	11	15	25	1	2.16
NearestNeighbour-J (NN)	76.14	47.43	0	11	17	14	10	15.57
bsttree-R (BST)	76.23	51.67	0	9	18	21	4	10.21
DecisionTable-W (RL)	77.04	47.00	0	12	17	15	8	0.66
BackPropagationNet-J (NNET)	77.11	45.92	2	17	11	8	16	1899.83
KNeighborsRegressor-P (NN)	78.93	46.97	0	10	20	12	10	0.61
bagging-DecisionTable-W (BAG)	79.27	48.19	0	11	16	17	8	41.35
DecisionTree-J (RT)	80.29	48.71	0	10	20	10	12	0.51
svmLinear-R (SVM)	81.63	53.51	0	11	18	19	4	99.12
iRProp+-K (NNET)	81.70	46.29	0	5	23	16	8	257.96
DecisionTreeReg-P (RT)	81.96	48.64	0	17	10	14	11	0.08
KernelRLS-J (OM)	82.37	47.72	0	9	14	20	9	0.06
SMOreg-W (SVM)	82.90	52.60	0	10	15	23	4	262.38
rpart-T (RT)	83.62	46.68	0	11	14	18	9	50.11
rpart-R (RT)	84.22	44.86	0	11	13	17	11	1.55
rpart1SE-T (RT)	84.22	44.86	0	11	13	17	11	31.48
fitlm-Robust-M (GLM)	84.85	48.24	0	11	12	23	6	1.10
Lars-P (GLM)	85.95	45.11	0	10	16	13	13	**0.01**
TheilSenRegressor-P (GLM)	86.84	45.56	0	12	7	22	11	4.63
AdditiveRegression-W (OM)	86.94	44.67	0	6	15	23	8	0.19
tree-R (RT)	87.33	43.80	0	9	13	20	10	1.03
ctree2-T (RT)	88.64	42.69	0	8	11	25	8	40.47
rpart2-T (RT)	88.86	41.92	0	10	8	24	10	13.93
IB1-W (NN)	89.38	42.00	0	13	9	17	13	11.90
GFS-SP-K (RL)	90.66	41.99	0	7	11	24	10	636.32
RandomTree-W (RT)	91.22	43.15	0	12	11	15	14	0.08
pls-R (PLSR)	91.33	36.33	0	5	13	20	14	4.58
GFS-GSP-K (RL)	91.71	39.27	1	6	16	15	15	3520.77
GFS-SAP-Sym-K (RL)	91.71	39.27	0	6	16	15	15	3855.58
HuberRegressor-P (GLM)	94.41	38.97	0	3	13	27	9	0.23
Stacking-J (STA)	95.17	41.05	0	4	14	24	10	2.05
newff-M (NNET)	95.33	34.79	0	8	11	13	20	549.57
kernelpls-R (PLSR)	96.02	35.35	0	4	10	28	10	1.09
LWL-J (OM)	96.42	30.41	1	11	7	9	25	261.44
Lasso-P (GLM)	97.43	35.41	0	6	11	19	16	0.18
simpls-R (PLSR)	99.32	32.32	0	3	8	**31**	10	2.56
ElasticNet-P (GLM)	99.83	33.28	1	6	9	23	14	0.91

the moderate/bad zones, i.e., bagging-REPTree-W (82.65 RegM). In this sense, both of them could be considered as quite robust algorithms without any registered bad result.

- The algorithms obtaining the best values for the Friedman ranking and the new measure *RegM* are the algorithms M5 and M5Rules available in the software tools R and Weka and making use of 50 bagging iterations (Bagging-M5P-R). Notice that both of them are algorithms that belong to the MT family to which 50 bagging iterations have been applied. Close to the results obtained by these algorithms we can also find several algorithms of the RF family.
- Analyzing the values obtained for the *RegM* measurement, we can see how these values present a coherent correlation with the values for Friedman's ranking, decreasing the value of the measurement as the ranking value increases.
- Finally, from our point of view and taking into account the RegM distribution of results into the four quality

TABLE 4. Results obtained by the studied methods. (III/III).

Algorithm	Rank	RegM	Win	★	★	✦	▼	AvTime
RBFNet-J (NNET)	100.13	31.98	0	3	14	21	14	0.19
glmnet-R (GLM)	102.15	30.71	0	2	8	**31**	11	0.74
NuSVR-K (SVM)	102.47	29.53	0	8	6	14	24	652.20
fitrlinear-lsr-M (GLM)	103.69	26.35	0	4	9	16	23	0.98
DCD-J (SVM)	108.13	25.03	0	1	12	16	23	14.28
SimpleLinearRegression-W (GLM)	108.23	21.31	0	3	6	20	23	**0.01**
lars-R (GLM)	108.67	25.07	0	4	5	23	20	0.76
LWL-W (OM)	109.09	22.65	0	1	7	22	22	610.48
EnsembleR-K (NNET)	109.20	26.66	0	5	10	10	27	2.42
NuSVR-P (SVM)	109.44	24.19	1	6	8	8	30	36.90
nnls-R (GLM)	109.45	28.69	0	5	10	15	22	4.21
GFS-GPG-K (RL)	109.51	20.63	0	0	3	29	20	1334.98
pcr-R (PLSR)	110.08	23.08	0	3	5	24	20	0.91
fitrsvm-M (SVM)	110.70	29.75	0	2	10	18	22	276.86
quantregForest-R (RT)	110.77	25.09	0	6	3	22	21	456.71
DCDs-J (SVM)	111.18	22.35	0	2	9	14	27	14.44
PRIM-M (RL)	111.87	19.31	0	1	8	16	27	2.01
WM-K (RL)	111.99	21.76	0	3	10	11	28	331.31
bagging-DecisionStump-W (BAG)	112.66	20.03	0	2	4	23	23	0.86
elm-M (NNET)	112.71	28.03	**12**	14	0	2	36	0.04
EPSILON-SVR-K (SVM)	113.23	22.56	0	5	8	10	29	167.88
svmPoly-R (SVM)	113.88	21.89	0	1	11	13	27	101.13
bstls-R (BST)	114.62	18.25	0	3	7	13	29	4.94
MLPRegressor-P (NNET)	116.17	22.64	0	8	3	8	33	32.59
ConjunctiveRule-W (RL)	116.30	18.03	0	1	5	19	27	3.36
SVR-P (SVM)	116.84	20.42	0	4	7	10	31	30.97
fitrlinear-svm-M (GLM)	117.08	21.44	0	1	9	13	29	0.95
RandomTree-J (RT)	118.21	17.06	0	0	4	24	24	0.09
cforest-R (RF)	118.28	16.13	0	3	1	21	27	15124.35
MLP-BP-K (NNET)	120.67	19.35	0	3	7	11	31	0.96
DecisionStump-W (RT)	120.99	16.71	0	1	4	21	26	0.03
LibLineaR-R (SVM)	121.97	18.92	0	1	3	18	30	0.40
LassoLars-P (GLM)	122.12	18.13	0	3	8	9	32	0.02
LinearSVR-P (SVM)	123.15	15.35	0	0	4	18	30	0.60
elmNN-R (NNET)	125.63	15.07	0	2	3	16	31	1.87
mlpWeightDecay-T (NNET)	127.07	15.07	0	4	3	8	37	10672.82
rbf-R (NNET)	127.23	14.83	1	4	2	10	36	63.21
rfsrc-R (RF)	128.90	13.40	2	2	5	4	41	80.30
RBFNR-K (NNET)	129.38	12.74	0	3	3	9	37	8.61
PassiveAggressive-J (GLM)	134.77	7.04	0	0	1	14	37	0.06
MultiScheme-W (OM)	134.89	9.29	0	2	1	11	38	**0.01**
Stacking-W (STA)	134.89	9.29	0	2	1	11	38	**0.01**
ZeroR-W (RL)	134.89	9.29	0	2	1	11	38	**0.01**
pcaNNet-T (NNET)	136.43	11.07	1	3	4	3	42	1555.87
nnet-T (NNET)	137.05	10.35	0	3	3	3	43	1065.43
PassiveAggressiveRegressor-P (GLM)	137.64	5.40	0	1	1	11	39	**0.01**
avNNet-T (NNET)	138.74	9.92	1	3	2	4	43	533.92
mlp-R (NNET)	142.28	9.02	0	3	1	5	43	58.00
nnet-R (NNET)	150.96	3.47	0	0	0	7	45	1.32
SGDRegressor-P (GLM)	151.13	8.73	0	0	1	10	41	**0.01**
Kmeans-P (NN)	153.29	1.53	0	0	1	1	**50**	0.67
relaxo-R (OM)	155.22	1.62	0	0	0	4	48	1.26
svmSigmoid-R (SVM)	155.56	2.59	0	0	2	1	49	78.85
rbfDDA-R (NNET)	157.95	1.25	0	0	0	2	**50**	49.11

intervals, we recommend as highly promising those whose RegM average values is around and over 60. Moreover, as a particular singularity, we can find the elm-M algorithm, which gets the best results in 12 datasets and very bad results (according to RegM) in 35 datasets, by which it is ranked within the last algorithms. Therefore, we think elm-M is also a promising one to consider for tackling real problems.

In the following subsections, we perform statistical analysis on the 30 best algorithms according to Friedman's ranking, with and without bagging.

B. ANALYSIS OF THE BEST 30 ALGORITHMS WITHOUT BAGGING CONSIDERATION

We have only analyzed the 30 best algorithms according to Friedman's ranking without considering the BAG family algorithms in order to study the algorithms without this additional methodology. Table 5 shows the 30 best algorithms according to the Friedman's ranking (recalculated for only these 30 algorithms) and the adjusted p-value (APV_{Holm}) obtained by the Holm's test when we compare the best ranking algorithm ($ExtraTreesReg - P$) with the remaining algorithms. As a summary: 7 RFs, 6 MTs, 4 MARSs, 3 SVMs,

TABLE 5. The best 30 algorithms without bagging (including recalculated Friedman's test and Holm's Adjusted P-Value).

Algorithm	Rank	APV_{Holm}	Algorithm	Rank	APV_{Holm}
ExtraTreesReg-P (RT)	9.70		M5-K (MT)	15.37	0.347
RandomForest-W (RF)	10.58	39.838	ksvmEpsilon-R (SVM)	16.01	0.093
randomForest-R (RF)	11.02	39.838	mars-M (MARS)	16.02	0.091
RRF-R (RF)	11.30	37.086	ppr-R (OM)	16.26	0.053
RandomForestRegressor-P (RF)	11.67	32.965	ksvmNu-R (SVM)	16.61	**0.024**
GradientBoostingRegressor-P (BST)	12.20	25.389	mars-R (MARS)	17.40	**0.003**
cubist-R (MT)	12.88	13.771	RandomCommittee-W (OM)	17.48	**0.003**
ranger-R (RF)	13.02	12.193	stepwiseglm-M (GLM)	17.96	**6.98E-04**
M5P-R (MT)	13.06	11.637	earth-R (MARS)	18.44	**1.70E-04**
M5Rules-W (MT)	13.88	4.115	gcvEarth-R (MARS)	18.44	**1.70E-04**
Rborist-R (RF)	14.12	2.973	AttributeSelectedClassifier-W (OM)	18.61	**1.04E-04**
M5Rules-R (MT)	14.18	2.703	RBFRegressor-W (NNET)	19.11	**2.16E-05**
rf-T (RF)	14.68	1.198	knn-T (NN)	20.36	**2.91E-07**
M5Rules-K (MT)	15.29	0.403	ctree-T (RT)	21.81	**1.02E-09**
svm-R (SVM)	15.30	0.396	fitensembleBst-M (BST)	22.26	**1.52E-10**

3 OMs, 2 RTs, 2BSTs, 1 GLM, 1 NNET and 1 NN. Notice that no algorithms from the RL, STA, and PLSR families have been included among the 30 best algorithms. Taking into account the results shown in Table 5 we can highlight:

- The equality hypothesis to the first one is not rejected for the remaining first 18 algorithms with a significance level of 0.05. Among them, we can find algorithms from 7 different families (7 algorithms from RF, 6 from MT, 1 from RT, 1 from BST, 2 from SVM, 1 from MARS and 1 from OM) but most of them belong to the RF and MT families, which shows the potential of tree based algorithms. SVMs also appears twice, which shows a significantly good performance taking into account that there are 146 algorithms without bagging.
- The best Friedman's ranking is obtained by the single tree-based algorithm *ExtraTreesReg* − *P*. Moreover, we can see how the following 4 top-ranked algorithms belong to the RF family, which are ensemble algorithms with multiples trees in the forest (500 trees in this study).
- Finally, single-based MT family algorithm also seen to compete with RF ones. It is quite interesting since they are quite simpler and therefore they should be easier to understand/interpret. It also shows potentiality in order to be considered as base algorithms for bagging or new RF proposals.

C. ANALYSIS OF THE BEST 30 ALGORITHMS WITH BAGGING

We have only focused the 30 best algorithms of the 164 algorithms analyzed in this study according to Friedman's ranking (see Table 2). It makes it able testing the equality hypothesis (first ranked as reference) according to Holm's test when we compare the results obtained by all the algorithms including bagging. Notice that we have not applied bagging to the ensemble algorithms (RF family and ExtraTrees Reg-P) because these algorithms already perform an internal bagging-like process (see Section II-D - Bagging, for extended explanation).

Table 6 shows the results obtained by Friedman's test recalculated for these 30 algorithms (this type of table was described in the previous subsection). As a summary: 11 BAGs,7 RFs, 6 MTs, 2 SVMs, 1 RT, 1BST, 1 MARS and 1 OM. Notice that the best Friedman's ranking is obtained by M5P, but in this case with its implementation available in Weka. In addition, this implementation also got the best value for the measure RegM (see Table 2). Analyzing the results presented in Table 6, we can highlight the following facts:

- The bagging methodology allows to improve considerably the precision of the algorithms, being 11 of the best 30 algorithms from BAG family (from a total of 18 BAG algorithms).
- Holm's test with bagging-M5P-W as reference algorithm rejects the equality hypothesis with a significance level of 0.05 with the last 10 algorithms in the table, which include several implementations of the M5 algorithm. It should be noted that the implementation of M5 without bagging available in Weka is not even among Friedman's top 30 ranking algorithms. This shows how the use of bagging can significantly improve the algorithm's performance without a high computational cost (see Tables 2, 3 and 4).
- The remaining algorithms (those with p-values over 0.05) are distributed in families as follows: 8 algorithms from BAG, 7 from RF family (from total of 10 RF algorithms), 2 from MT and 1 from BST. Even when there are no SVM, MARS or OM in the not rejected set of algorithms, we should take into account that there are no implemented versions on these algorithms in combination with bagging. It depicts on open framework for including these types of combinations as a part of the studied software tools.

D. ANALYSIS OF SCALABILITY

In this section, we include an analysis of the computational cost of the first ranked algorithms and the average times by families. Figure 2 shows the graph of the time in seconds of the best 30 algorithms sorted by their rankings. The methods using the "train" function from caret to tune algorithm parameters need significant extra time to obtain the results, thus slowing down the overall operation of the algorithms.

TABLE 6. The best 30 algorithms with bagging (including recalculated Friedman's test and Holm's Adjusted P-Value).

Algorithm	Rank	APV_{Holm}	Algorithm	Rank	APV_{Holm}
bagging-M5P-W (BAG)	10.92		bagging-REPTree-W (BAG)	15.59	2.198
Bagging-M5P-R (BAG)	10.93	46.833	treeBagger-M (BAG)	15.83	1.465
ExtraTreesReg-P (RT)	11.13	46.833	Rborist-R (RF)	16.38	0.542
Bagging-M5Rules-R (BAG)	11.22	46.833	M5P-R (MT)	16.91	0.189
bagging-M5Rules-W (BAG)	11.45	46.833	rf-T (RF)	16.95	0.175
RandomForest-W (RF)	12.27	44.792	M5Rules-W (MT)	17.59	**0.044**
RandomForestRegressor-P (RF)	12.84	38.046	M5Rules-R (MT)	17.84	**0.024**
randomForest-R (RF)	13.11	33.808	svm-R (SVM)	18.37	**6.56E-03**
RRF-R (RF)	13.38	28.787	M5-K (MT)	18.68	**2.88E-03**
bagging-MultilayerPerceptron-W (BAG)	13.39	28.643	M5Rules-K (MT)	18.70	**2.75E-03**
GradientBoostingRegressor-P (BST)	14.51	9.480	ppr-R (OM)	18.86	**1.82E-03**
cubist-R (MT)	14.99	5.156	ksvmEpsilon-R (SVM)	18.94	**1.43E-03**
bagging-RandomTree-W (BAG)	15.13	4.311	mars-M (MARS)	19.37	**4.34E-04**
bagging-DecisionTree-P (BAG)	15.18	4.031	bagEarth-R (BAG)	19.53	**2.69E-04**
ranger-R (RF)	15.41	2.901	bagEarthGCV-R (BAG)	19.61	**2.14E-04**

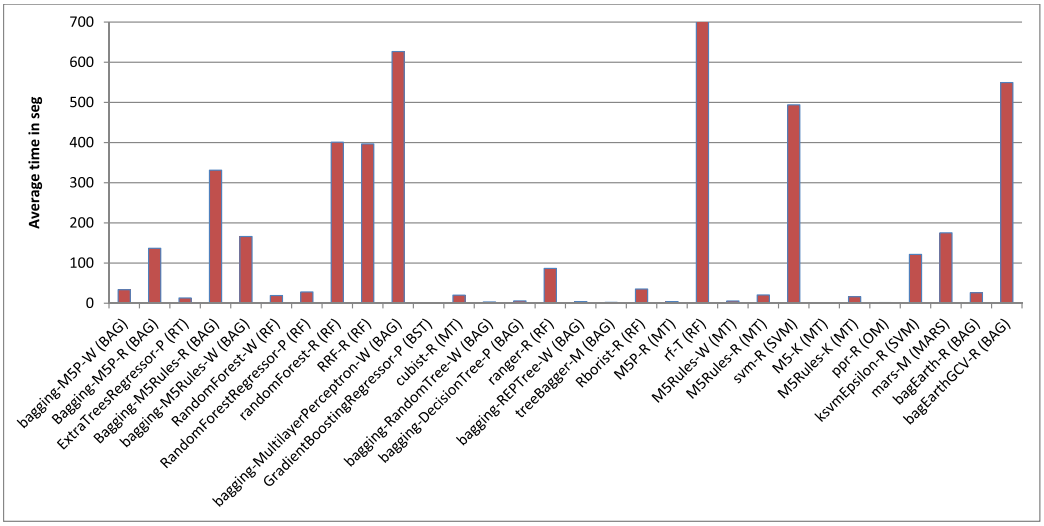

FIGURE 2. Average computational cost of the 30 best ranked algorithms.

In this figure, we can see how the algorithms from BAG family (12 among the 30 best) have a reasonable time except Bagging-M5Rules-R, bagging-MultilayerPerceptron-W and bagEarthGCV-R algorithms. The algorithms of the RF family are slower except for those versions specifically designed to be fast, such as ranger-R and Rborist-R methods, the Weka versions and the python versions. In general, the algorithms implemented in python are quite fast.

The average times from each family are displayed in Figure 3. In this figure, we have excluded algorithms executed with "train" (-T) since the way to execute the algorithms looking for an adjustment of the parameters penalize the own family times. This figure shows that RL, NNET and OM families are quite slow. The algorithms from the BAG family are not the fastest ones but they are quite competitive in average times and they get pretty good precision results. The best results in time, with an acceptable and very competitive precision, are the algorithms of the MT family (as we saw in the previous section, they obtain acceptable results even equivalent to the best algorithm). Therefore, applying

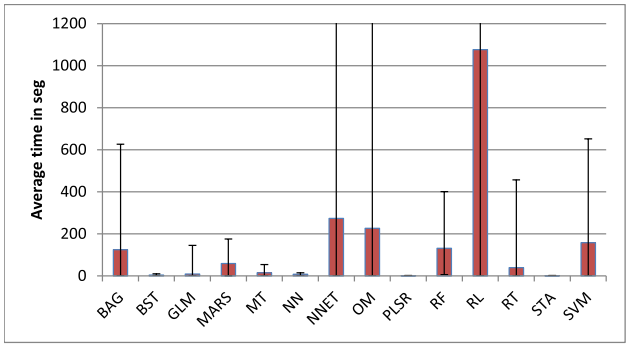

FIGURE 3. Average time for each family.

bagging to MT methods allows not only obtaining good results but also good run times.

E. ANALYSIS ON THE CURSE OF DIMENSIONALITY

In this paper we have considered a wide variety of data - particularly relative to number of attributes and number of

examples. This section helps to analyze some algorithms in different situations, as for datasets with a large number of attributes compared to other methods, datasets with a small number of attributes, or the combined effect depending on the number of examples. In order to do so, we have divided the datasets into two different groups depending on their dimensionality (i.e., number of attributes). The first one includes the datasets with the higher dimensionality, High Dimensional (HD) group with only >= 9 variables. The second one includes the datasets with the lower dimensionality, Low Dimensional (LD) group with only < 9 variables. Thus, we can compute all the measures again separately by group (Friedman's ranking, *RegM*, and number of results in the quality intervals) in order to check differences with the general results and to contrast both groups.

On the other hand, there are some data complexity measures that were proposed or used in the classification framework where most of them were proposed based on the existence of classes. Even though, some of them can be directly used for regression. That is the case when in discussions on *curse of dimensionality*, the number of patterns is compared to the number of variables, which seems more interesting than considering number of patterns only. In [104], [105], the authors introduced a very simple index, denoted T2, defined as the average number of patterns per variable. This measure also represents an interesting characteristic for the datasets in the regression framework. In this section, we also apply this measure as in the previous case for analyzing the dimensionality. Thus, we have again divided the datasets in a group with High T2 (HT2) values (good distributions with T2 >= 250) and a group with Low T2 (LT2) values (bad distributions with T2 < 250).

The complete results in the four groups (HD, LD, HT2 and LT2) sorted by ranking on the 164 algorithms are available in the complementary material web page associated to this paper (http://www4.ujaen.es/~mgacto/regression/study/http://www4.ujaen.es/~mgacto/regression/study/). For the sake of simplicity and due to it is not possible to show and analyze all the 164 algorithms in the mentioned four groups in the manuscript, we show and analyze here only the best 20 algorithms for each group.

Table 7 shows these results, where PosHD is the position sorted by Friedman's ranking obtained on the HD group of datasets (only >= 9 variables), PosLD is the position sorted by ranking obtained on the LD group of datasets (only < 9 variables), PosGlobal is the position sorted by ranking obtained when all datasets are considered and the remaining columns where previously explained in Section III-A for Tables 2, 3 and 4. Analogously, Table 8 contains the same columns but for the corresponding division on T2, High HT2 group of datasets (only T2 >= 250) and LT2 group of datasets (only T2 < 250), respectively.

From the results shown in these tables we can highlight the following facts:

- There is a group of algorithms, the bagging of the M5P versions, that reach a good behaviour in both the HD and the LD groups of datasets, also obtaining the same behaviour in T2. They rank the best positions in all the considered datasets. Notwithstanding, this aspect was shown in Table 2 where their RegM average scorings are quite high but, most importantly, their individual RegM results are mainly located in the very good and good quality ranges, represented by ✸ and ★, respectively. For this reason, the use of these algorithms as a first approximation on a given real problem could be considered as a starting recommendation for non-expert users whenever there are no special restrictions such as interpretability, real time computing, etc. And then, if it is possible (depending on their programming abilities and data mining knowledge) to try to improve the obtained results with more recent techniques.
- Despite including feature selection as part of the learning process itself, the base algorithms of these combinations (versions of M5P) suffer more in the LD than in the HD, so it seems that bagging behavior is improved in small problems, where maybe overlearning occurs, without affecting its behavior in the HD too much. This aspect is even more prominent when we focus on T2, where the base versions do not appear in the top 20 in the LT2 group (datasets with low data density), showing that bagging really solves the problem of low data density (tendency to overlearning). As a conclusion or recommendation, these base algorithms are also good alternatives for solving problems with high T2 values, which allow obtaining simpler models (just one tree, or even a set of rules). Anyway, they also maintain their individual RegM results, even in the LT2 group, mainly located in the very good and good quality ranges, represented by ✸ and ★, respectively.
- The family of RF algorithms seems to behave in a contrary way than M5P versions. Focusing on T2, it seems to be that, although they work well in both types of problems, they are generally better in problems with poor data density/distribution than with good ones. They also provide a better behavior in the HD than LD, and vice versa, depending on the RF version, so they are actually more dependent on the distribution of data than on the dimensionality itself (although logically they are related).

F. TUNING OF THE ALGORITHMIC PARAMETERS BY "TRAIN"

As we explained before, the setting/tuning of algorithmic parameters is an important aspect on which we had to take a decision based on the reasons discussed in Section II-D. Even though it is not the objective in this contribution, because of the reasons previously stated, in this section we show the performance of those algorithms for which "train" (from Caret in R) has been applied on recommended nested cross-fold validation for tuning some of their most relevant algorithmic parameters (based on Caret recommendations). Of course, it is not a taxative demonstration on how the

TABLE 7. The best 20 algorithms for the HD datasets and for LD datasets (including recalculated performance metrics, where more than 10 position differences are boldfaced).

PosHD	PosLD	PosGlobal	Algorithm	Rank HD/LD	RegM HD/LD	Win HD/LD	✸ HD/LD	★ HD/LD	✦ HD/LD	▼ HD/LD
			Sorted by results in Higher Dimensional (HD) datasets only							
1	6	2	bagging-M5P-W (BAG)	16.63 / 21.92	88.09 / 81.5	1 / 0	23 / 16	4 / 8	0 / 1	0 / 0
2	1	1	bagging-M5P-R (BAG)	16.72 / 20.44	88.15 / 82.69	0 / 1	23 / 17	4 / 8	0 / 0	0 / 0
3	4	3	bagging-M5Rules-W (BAG)	18.52 / 21.46	85.25 / 80.55	0 / 0	22 / 15	4 / 9	0 / 1	1 / 0
4	7	4	bagging-M5Rules-R (BAG)	18.91 / 22.78	86.16 / 81.17	0 / 0	21 / 14	5 / 11	1 / 0	0 / 0
5	19	10	**ExtraTreesReg-P (RT)**	22.41 / 35.22	84.45 / 73.06	6 / 3	20 / 15	5 / 6	1 / 2	1 / 2
6	13	8	RRF-R (RF)	26.7 / 29.86	84.12 / 75.64	1 / 0	21 / 14	5 / 9	0 / 2	1 / 0
7	15	11	RandomForestRegressor-P (RF)	26.83 / 33.78	82.29 / 74.14	0 / 0	20 / 15	5 / 6	1 / 2	1 / 2
8	34	21	**cubist-R (MT)**	29.59 / 44.26	80.21 / 68.5	1 / 0	18 / 14	6 / 6	2 / 3	1 / 2
9	31	20	**M5Rules-W (MT)**	29.83 / 41.52	81.35 / 73.74	0 / 0	19 / 14	6 / 8	2 / 3	0 / 0
10	27	18	**M5Rules-R (MT)**	30.39 / 40.18	81.32 / 74.18	0 / 0	17 / 14	9 / 8	1 / 3	0 / 0
11	3	5	bagging-MultilayerPerceptron-W (BAG)	30.54 / 21.44	79.85 / 80.86	2 / 1	19 / 17	5 / 8	1 / 0	2 / 0
12	8	7	RandomForest-W (RF)	30.78 / 23.9	83.11 / 79	0 / 0	20 / 18	6 / 5	0 / 1	1 / 1
13	16	15	M5P-R (MT)	31.85 / 34.18	83.01 / 77.49	0 / 0	19 / 15	7 / 8	1 / 2	0 / 0
14	24	17	**bagging-DecisionTree-P (BAG)**	32.11 / 37.24	81.12 / 73.2	0 / 0	20 / 15	5 / 6	1 / 2	1 / 2
15	2	6	**randomForest-R (RF)**	32.94 / 21.14	82.22 / 80.45	0 / 2	20 / 19	6 / 5	0 / 0	1 / 1
16	35	23	**M5-K (MT)**	33.24 / 44.76	80.86 / 71.68	0 / 0	17 / 12	8 / 10	2 / 3	0 / 0
17	18	16	GradientBoostingRegressor-P (BST)	33.48 / 35.02	78.64 / 76.12	1 / 3	19 / 14	5 / 8	1 / 2	2 / 1
18	22	19	Rborist-R (RF)	33.69 / 36.96	82.24 / 68.6	0 / 0	20 / 12	6 / 10	0 / 1	1 / 2
19	11	12	bagging-RandomTree-W (BAG)	34.37 / 29.08	81.92 / 77.91	0 / 0	19 / 17	7 / 6	0 / 1	1 / 1
20	5	9	**bagging-REPTree-W (BAG)**	34.7 / 21.86	83.5 / 81.73	0 / 0	21 / 17	6 / 8	0 / 0	0 / 0
			Sorted by results in Lower Dimensional (LD) datasets only							
2	1	1	bagging-M5P-R (BAG)	16.72 / 20.44	88.15 / 82.69	0 / 1	23 / 17	4 / 8	0 / 0	0 / 0
15	2	6	**randomForest-R (RF)**	32.94 / 21.14	82.22 / 80.45	0 / 2	20 / 19	6 / 5	0 / 0	1 / 1
11	3	5	bagging-MultilayerPerceptron-W (BAG)	30.54 / 21.44	79.85 / 80.86	2 / 1	19 / 17	5 / 8	1 / 0	2 / 0
3	4	3	bagging-M5Rules-W (BAG)	18.52 / 21.46	85.25 / 80.55	0 / 0	22 / 15	4 / 9	0 / 1	1 / 0
20	5	9	**bagging-REPTree-W (BAG)**	34.7 / 21.86	83.5 / 81.73	0 / 0	21 / 17	6 / 8	0 / 0	0 / 0
1	6	2	bagging-M5P-W (BAG)	16.63 / 21.92	88.09 / 81.5	1 / 0	23 / 16	4 / 8	0 / 1	0 / 0
4	7	4	bagging-M5Rules-R (BAG)	18.91 / 22.78	86.16 / 81.17	0 / 0	21 / 14	5 / 11	1 / 0	0 / 0
12	8	7	RandomForest-W (RF)	30.78 / 23.9	83.11 / 79	0 / 0	20 / 18	6 / 5	0 / 1	1 / 1
29	9	22	**rf-T (RF)**	47.87 / 27.26	71.31 / 77.18	0 / 1	14 / 16	8 / 8	4 / 0	1 / 1
22	10	14	**ranger-R (RF)**	37.31 / 27.82	80.67 / 78.29	0 / 0	20 / 16	6 / 8	0 / 1	1 / 0
19	11	12	bagging-RandomTree-W (BAG)	34.37 / 29.08	81.92 / 77.91	0 / 0	19 / 17	7 / 6	0 / 1	1 / 1
21	12	13	treeBagger-M (BAG)	35.46 / 29.16	82.83 / 77.93	0 / 0	20 / 15	6 / 8	1 / 2	0 / 0
6	13	8	RRF-R (RF)	26.7 / 29.86	84.12 / 75.64	1 / 0	21 / 14	5 / 9	0 / 2	1 / 0
30	14	25	**svm-R (SVM)**	47.89 / 32.08	75.07 / 73.86	1 / 0	15 / 16	9 / 6	3 / 2	0 / 1
7	15	11	RandomForestRegressor-P (RF)	26.83 / 33.78	82.29 / 74.14	0 / 0	20 / 15	5 / 6	1 / 2	1 / 2
13	16	15	M5P-R (MT)	31.85 / 34.18	83.01 / 77.49	0 / 0	19 / 15	7 / 8	1 / 2	0 / 0
31	17	26	**bagEarth-R (BAG)**	48.02 / 34.74	71.98 / 75.17	0 / 0	16 / 14	7 / 11	3 / 0	1 / 0
17	18	16	GradientBoostingRegressor-P (BST)	33.48 / 35.02	78.64 / 76.12	1 / 3	19 / 14	5 / 8	1 / 2	2 / 1
5	19	10	**ExtraTreesReg-P (RT)**	22.41 / 35.22	84.45 / 73.06	6 / 3	20 / 15	5 / 6	1 / 2	1 / 2
28	20	27	bagEarthGCV-R (BAG)	47.5 / 35.98	71.85 / 73.17	0 / 0	16 / 14	7 / 9	3 / 1	1 / 1

parametric tuning influences the all the algorithms in general, but it could represent a glimpse whether repetitive trends can be found on some different algorithms. Moreover, it could lead to some recommendations when users/researchers perform parametric tuning in real applications/problems they need to solve (especially when they are non-expert users).

In Table 9, we show the average results obtained by the available methods with fixed standard parameters (without parametric tuning) and their recommended versions with parametric tuning. It includes the same columns that where previously explained in Section III-A for Tables 2, 3 and 4, but also their obtained global positions in these tables (positions from 1 to 164).

Taking into account the values of the different metrics shown in Table 9, we can stress the following facts:

- The highest differences in algorithm positions with respect to their tuned versions are found for random Forest-R and its tuned version rf-T (positions 6 to 22, respectively); for mlp-R and mlpWeightDecay-T (positions 158 to 146, respectively); and for rpart-R and rpart2-T (positions 88 to 96, respectively). Contrary to the expected, two of them (those with the best ranking positions) are even obtaining worse results (of course, in their test errors). Checking the RegM intervals, we can observe that some datasets are changing their quality classification moving to a worse category, showing the overfitting effects on some datasets/problems. However, even in these cases, they are not highly relevant changes over a total of 164 algorithms.
- The remaining changes look not so relevant, by changing no more than four positions and only getting slight improvements in general. Moreover the frequency in the RegM quality intervals is quite similar, involving only small changes in the shown distributions.
- In general, it seems that significant changes on performance could depend more on the design of the algorithms itself than on the subsequent tuning of their parameters.

Finally, we recommend for non-expert users the use of standard parameters in principle, unless they have previous knowledge on the corresponding parameters effects, so that some alternative combinations could be explored. In principle, without expert knowledge, probably not so high improvements and taking into account the high risk of overfitting,

TABLE 8. The best 20 algorithms for the HT2 datasets and for LT2 datasets (including recalculated performance metrics, where more than 10 position differences are boldfaced).

PosHT2	PosLT2	PosGlobal	Algorithm	Rank HT2/LT2	RegM HT2/LT2	Win HT2/LT2	★ HT2/LT2	★ HT2/LT2	✦ HT2/LT2	▼ HT2/LT2
			Sorted by results in Higher T2 (HT2) datasets only							
1	5	1	bagging-M5P-R (BAG)	13.13 / 24.32	88.43 / 82.38	1 / 0	23 / 17	4 / 8	0 / 0	0 / 0
2	2	2	bagging-M5P-W (BAG)	15.96 / 22.64	86.97 / 82.7	1 / 0	22 / 17	4 / 8	1 / 0	0 / 0
3	4	4	bagging-M5Rules-R (BAG)	17.67 / 24.2	85.96 / 81.38	0 / 0	19 / 16	8 / 8	0 / 1	0 / 0
4	1	3	bagging-M5Rules-W (BAG)	18.87 / 21.08	84.85 / 80.98	0 / 0	19 / 18	7 / 6	1 / 0	0 / 1
5	8	5	bagging-MultilayerPerceptron-W (BAG)	24.69 / 27.76	83.84 / 76.56	1 / 2	18 / 18	9 / 4	0 / 1	0 / 2
6	13	9	bagging-REPTree-W (BAG)	24.8 / 32.48	85.8 / 79.24	0 / 0	22 / 16	5 / 9	0 / 0	0 / 0
7	33	24	**M5Rules-K (MT)**	24.83 / 54.7	83.46 / 64.7	0 / 0	18 / 11	9 / 7	0 / 6	0 / 1
8	10	7	RandomForest-W (RF)	26.54 / 28.48	82.47 / 79.69	0 / 0	21 / 17	4 / 7	1 / 0	1 / 1
9	21	15	**M5P-R (MT)**	27 / 39.46	84 / 76.42	0 / 0	20 / 14	6 / 9	1 / 2	0 / 0
10	7	6	randomForest-R (RF)	27.44 / 27.08	83.02 / 79.59	2 / 0	21 / 18	5 / 6	0 / 0	1 / 1
11	25	20	**M5Rules-W (MT)**	27.96 / 43.74	81.73 / 73.34	0 / 0	17 / 16	10 / 4	0 / 5	0 / 0
12	24	18	**M5Rules-R (MT)**	28.91 / 41.82	81.72 / 73.74	0 / 0	17 / 14	10 / 7	0 / 4	0 / 0
13	16	13	treeBagger-M (BAG)	29.28 / 35.88	83.34 / 77.38	0 / 0	20 / 15	5 / 9	2 / 1	0 / 0
14	29	23	**M5-K (MT)**	29.65 / 48.48	81.7 / 70.79	0 / 0	16 / 13	11 / 7	0 / 5	0 / 0
15	6	8	RRF-R (RF)	30.09 / 26.2	80.32 / 79.74	1 / 0	18 / 17	7 / 7	2 / 0	0 / 1
16	19	16	GradientBoostingRegressor-P (BST)	30.31 / 38.44	78.44 / 76.34	1 / 3	18 / 15	7 / 6	0 / 3	2 / 1
17	36	29	**ppr-R (OM)**	30.54 / 55.52	80.13 / 62.77	1 / 0	13 / 12	14 / 7	0 / 2	0 / 4
18	15	14	ranger-R (RF)	31.37 / 34.24	81.45 / 77.44	0 / 0	20 / 16	6 / 8	1 / 0	0 / 1
19	12	12	bagging-RandomTree-W (BAG)	31.59 / 32.08	81.36 / 78.51	0 / 0	21 / 15	4 / 9	1 / 0	1 / 1
20	30	25	**svm-R (SVM)**	31.78 / 49.48	80.34 / 68.17	1 / 0	20 / 11	6 / 9	0 / 5	1 / 0
			Sorted by results in Lower T2 (LT2) datasets only							
4	1	3	bagging-M5Rules-W (BAG)	18.87 / 21.08	84.85 / 80.98	0 / 0	19 / 18	7 / 6	1 / 0	0 / 1
2	2	2	bagging-M5P-W (BAG)	15.96 / 22.64	86.97 / 82.7	1 / 0	22 / 17	4 / 8	1 / 0	0 / 0
22	3	10	**ExtraTreesReg-P (RT)**	33.09 / 23.68	76.59 / 81.56	6 / 3	18 / 17	5 / 6	2 / 1	2 / 1
3	4	4	bagging-M5Rules-R (BAG)	17.67 / 24.2	85.96 / 81.38	0 / 0	19 / 16	8 / 8	0 / 1	0 / 0
1	5	1	bagging-M5P-R (BAG)	13.13 / 24.32	88.43 / 82.38	1 / 0	23 / 17	4 / 8	0 / 0	0 / 0
15	6	8	RRF-R (RF)	30.09 / 26.2	80.32 / 79.74	1 / 0	18 / 17	7 / 7	2 / 0	0 / 1
10	7	6	randomForest-R (RF)	27.44 / 27.08	83.02 / 79.59	2 / 0	21 / 18	5 / 6	0 / 0	1 / 1
5	8	5	bagging-MultilayerPerceptron-W (BAG)	24.69 / 27.76	83.84 / 76.56	1 / 2	18 / 18	9 / 4	0 / 1	0 / 2
21	9	11	**RandomForestRegressor-P (RF)**	31.89 / 28.32	76.82 / 80.04	0 / 0	18 / 17	5 / 6	2 / 1	2 / 1
8	10	7	RandomForest-W (RF)	26.54 / 28.48	82.47 / 79.69	0 / 0	21 / 17	4 / 7	1 / 0	1 / 1
29	11	19	**Rborist-R (RF)**	39.61 / 30.56	73.1 / 78.47	0 / 0	17 / 15	7 / 9	1 / 0	2 / 1
19	12	12	bagging-RandomTree-W (BAG)	31.59 / 32.08	81.36 / 78.51	0 / 0	21 / 15	4 / 9	1 / 0	1 / 1
6	13	9	bagging-REPTree-W (BAG)	24.8 / 32.48	85.8 / 79.24	0 / 0	22 / 16	5 / 9	0 / 0	0 / 0
25	14	17	**bagging-DecisionTree-P (BAG)**	35.63 / 33.44	75.91 / 78.82	0 / 0	18 / 17	5 / 6	2 / 1	2 / 1
18	15	14	ranger-R (RF)	31.37 / 34.24	81.45 / 77.44	0 / 0	20 / 16	6 / 8	1 / 0	0 / 1
13	16	13	treeBagger-M (BAG)	29.28 / 35.88	83.34 / 77.38	0 / 0	20 / 15	5 / 9	2 / 1	0 / 0
27	17	22	**rf-T (RF)**	39.04 / 36.84	75.26 / 72.92	0 / 1	17 / 13	7 / 9	2 / 2	1 / 1
35	18	27	**bagEarthGCV-R (BAG)**	45.3 / 38.4	70.52 / 74.61	0 / 0	13 / 17	11 / 5	2 / 2	1 / 1
16	19	16	GradientBoostingRegressor-P (BST)	30.31 / 38.44	78.44 / 76.34	1 / 3	18 / 15	7 / 6	0 / 3	2 / 1
34	20	26	**bagEarth-R (BAG)**	44.19 / 38.96	72.81 / 74.28	0 / 0	13 / 17	12 / 6	2 / 1	0 / 1

TABLE 9. Available methods with standard parameters (without parametric tuning, boldfaced) and their recommended versions with parametric tuning.

Algorithm	Position	Rank	RegM	Win	★	★	✦	▼	AvTime
randomForest-R (RF)	6	27.27	81.37	2	39	11	0	2	401.07
rf-T (RF)	22	37.96	74.13	1	30	16	4	2	4701.09
nnet-R (NNET)	159	150.96	3.47	0	0	0	7	45	1.32
avNNet-T (NNET)	157	138.74	9.92	1	3	2	4	43	533.92
nnet-T (NNET)	155	137.05	10.35	0	3	3	3	43	1065.43
mlp-R (NNET)	158	142.28	9.02	0	3	1	5	43	58.00
mlpWeightDecay-T (NNET)	146	127.07	15.07	0	4	3	8	37	10672.82
rpart-R (RT)	88	84.22	44.86	0	11	13	17	11	1.55
rpart-T (RT)	87	83.62	46.68	0	11	14	18	9	50.11
rpart2-T (RT)	96	88.86	41.92	0	10	8	24	10	13.93
kknn-R (NN)	44	61.29	60.69	0	20	16	9	7	7.80
knn-T (NN)	40	58.28	63.24	1	23	15	10	4	971.36

it would be better trying on different types of algorithms than adjusting and adjusting on only one or two of them. But in any case, in order to provide a good assessment/estimation of the real system error, please avoid checking the test errors before fixing all the tentative combinations of parameters. And once they are fixed then compute the test errors without repeating the process (since, maintaining the test data hidden for all the learning process is the only way to properly estimate the generalization ability of the models obtained).

It should be the same for expert users, which are supposed to be able to also apply a nested cross-fold validation (and/or even consider some fixed combinations of parameters). Again, it should be always fixed and performed before checking the test errors, without repeating the process if test

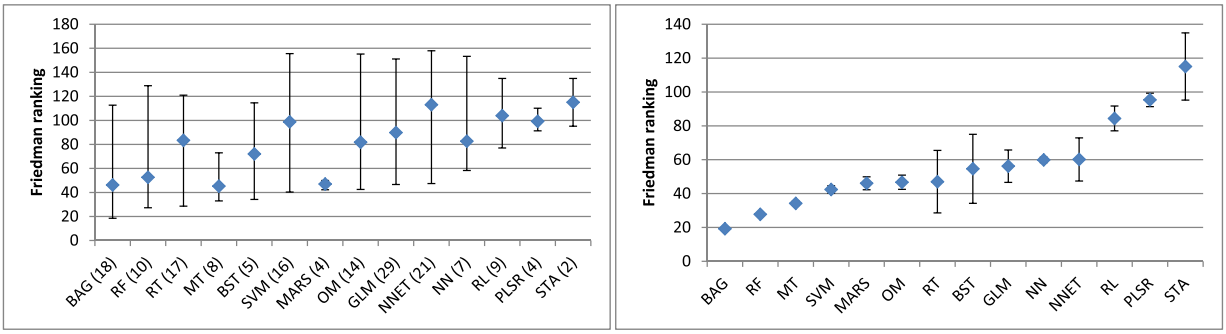

FIGURE 4. a) Friedman's rank average and range for the regressors by family; b) Friedman's average and range for only the 3 best ranked regressors by family.

errors are not as good as the expected, in order to avoid possible overfitting on the final real system application.

G. DISCUSSION BY ALGORITHM FAMILY

In this section, we discuss the results by regressor family (see Tables 2, 3 and 4). Figures 4.a) and 4.b) show respectively: the minimum, maximum and average ranking values by family considering all the algorithms (even those with extremely bad results) and the same by only considering the 3 best ranked algorithms by family (since we think that those most competitive ones could represent better each family potentiality). From now on, we will follow the order in Figure 4.b) to briefly analyze the algorithm families.

We can find the most accurate family in BAG. Checking Figure 4.a) we can see it gets almost the best average, while its best algorithm gets the first ranking position. When checking Figure 4.b) we can infer that BAG global average, see also Figure 4.a), is affected by a few bad algorithms but most of the BAG family are in the best ranking positions. In fact, the best regressor is bagging-M5P-R (ensemble technique for reducing variance that uses M5 base regressors with 50 bags in R). Followed by bagging-M5P-W (bagging of M5 model tree in Weka), bagging-M5Rules-W (bagging of M5 model rules in Weka), bagging-M5Rules-R (bagging of M5 model tree in R) and bagging-MultilayerPerceptron-W (bagging of MLP networks in Weka) with ranks about 19.17-26.16. These five first best algorithms belong to the BAG family. Bagging could be considered as a meta family or additive family, since it is applied over existent implementations of the remaining families, so that it also depends on the base algorithms from the other families.

The following family of regressors is RF. The best three ones from this family are randomForest-R (RF in R), RandomForest-W (RF in Weka) and RRF-R (Regularized RF in R) with ranks about 27.27-28.22. Among the 10 best ranked algorithms we can find several algorithms from the RF family, all of them with a fairly good ranking. RF are ensemble algorithms that perform an internal bagging-like process. This is one of the main reasons for them to be highly competitive with respect to the BAG family.

Two of the best ranked families that are not directly based on ensembles are MT and SVM. MT-based regressors work relatively well, being M5-R (M5 method in R) still equivalent to the best ranked method (see Table 6). The following methods from the MT family are M5Rules-R, M5Rules-W, cubist-R (advanced version of M5 method in R), M5-K and M5Rules-K, i.e., different versions of the classic M5. In fact, it should be noted that the method with the best ranking from the total of 164 algorithms is a bagging of the old and classic M5 algorithm. The SVM family is also quite competitive being the best method svm-R which is a simple implementation of a SVM using the library LibSVM [40]. Many of the methods from the SVM family are based on this library. They were not able to be used in combination with bagging in their current implementations at the studied software tools, which seems a potential open framework for the data mining software developers.

The methods from the MARS family obtain quite acceptable results, being mars-M the one with the best performance. MARS methods get results over the median value (i.e., very good ✭ and good ★ categories) in almost all the datasets, and only in a small percentage of datasets the results outweigh the median.

OM family includes very different algorithms and the rankings are not too good on global average but acceptable on the 3 best ranked average. ppr-R method (pursuit regression model in R) achieves good results (42.53 ranks) among the OM family and it is within the 30 best algorithms. The next method of the family is RandomCommittee-W with a ranking very close to ppr-R.

RT family presents intermediate results in general. ExtraTreesReg-P is the algorithm with the best ranking of the RT family with a rank value of 28.57. ExtraTreesReg-P is the algorithm with the best ranking of the RT family, surprisingly getting the best ranked position when bagging is not considered. The remaining algorithms of the RT family have a quite superior ranking, higher than 58. The second best algorithm of the family is ctree-T followed by REPTree-W. Both, REPTree-W and ExtraTreesReg-P, are quite fast.

Next, we are going to analyze the results of the group of intermediate-low ranked families. GradientBoosting

Regressor-P regressor (Gradient Boosting in Python) is the first algorithm of the BST family, the rest of the algorithms of this family are glmboost-R and bsttree-R but they have a worse ranking higher than 75. The family of GLM is the one with the higher number of algorithms 29, being the best one the stepwiseglm-M method. In this family, there are a great variety of methods: generalized linear models, least angle regression, passive aggressive algorithm, etc.

The algorithms of the NNET family are quite competitive even though they present some overfitting. They obtain very good results in some datasets (those that does not suffers from overfitting) whereas in other datasets (those that suffer from overfitting) they obtain results above the acceptable values. It can be said that depending on the problem these algorithms may be the best or the worst option. An example is the elm-M algorithm, which gets the best results in 12 datasets and very bad results in 35 datasets. RBFRegressor-W is the best ranked method from the NNET family with an acceptable ranking of 47.39.

Finally, the NN methods are classic methods, being most of them applicable to both regression and classification problems. They are quite fast in general (there is no learning stage) but their results are not excessively good. The best algorithm is the knn-T with k tuning by Caret ''train''. The RL methods does not obtained the best results, achieving the classical method DecisionTable-W the best results compared to evolutionary fuzzy rule learning (GFS-SP-K, GFS-GSP-K and GFS-SAP-Sym-K). However, these methods were particularly designed with explainability/interpretability purposes, so that they mainly try to obtain clear and simple models (when it is possible). M5Rules versions can also be considered as a part of the RL family. The PLSR family seems not too competitive at all. The pls-T method archives the best ranking value with a high rank (91.33). The STA family is the last one, probably motivated because of only two algorithms of this family have been included (this family is underrepresented in the studied software tools).

IV. CONCLUSIONS

In this paper, we have performed an experimental study with 164 algorithms for regression problems that come from 14 different families (BAG, RF, MT, BST, RT, SVM, MARS, OM, GLM, NNET, NN, PLSR, RL, and STA) that are available in several software tools (JSAT, KEEL, Matlab, R, Scikit-learn, and Weka) over 52 datasets. A statistical study with non-parametric tests has been carried out on all the algorithms and on the best 30 algorithms including the algorithms of the BAG family and without them. Moreover, the new measure RegM based on MSE has also been proposed to show the performance of an algorithm with respect to an interval of the MSE, which allows us to represent when an algorithm has a suitable performance. The objective of this study is to analyze the performance of a large number of regression algorithms to help non-expert users from other areas to properly solve their own regression problems and to help specialized researchers developing well-founded future proposals by properly comparing and identifying algorithms that will enable them to focus on significant further developments. In this context, a key aspect is related with the parameter setting. We have highlighted the importance of this aspect and detailed why for this type of studies, the standard parameters recommended by the authors are considered despite that the variety of datasets. Notice that, applying the standard parameters, a real situation when non-expert users need to apply data mining techniques to the problem they face in their areas is represented.

The results obtained over the 52 datasets collected show how the implementation of the M5 algorithm combined with bagging, which is available in the software tools R and Weka, obtains the best Friedman's ranking and the best value for the measure RegM when we analyze the 164 algorithms. On the other hand, the implementation of the ensemble-based algorithms ExtraTreesReg available in Scikit-learn and Random Forest available in R, Weka and Scikit-learn, present the best performance when we only analyze the best 30 algorithms according to Friedman's ranking without considering the algorithms of the BAG family. Notice that the ensemble algorithms perform an internal bagging-like process. This highlights the potential of tree-based methods to solve regression problems.

Results analyzed by families show that the algorithms from RF, MT and SVM get the best positions in the rankings obtained by the statistical tests when bagging is not considered. Finally, from the results obtained in the analyses with and without bagging we can see how the use of bagging can significantly improve the algorithm's performance without a high computational cost in general, so that BAG becomes the most accurate family.

REFERENCES

[1] J. Han and M. Kamber, *Data Mining: Concepts and Techniques*, 2nd ed. San Mateo, CA, USA: Morgan Kaufmann, 2006.
[2] V. Ravi, D. Pradeepkumar, and K. Deb, "Financial time series prediction using hybrids of chaos theory, multi-layer perceptron and multi-objective evolutionary algorithms," *Swarm Evol. Comput.*, vol. 36, pp. 136–149, Oct. 2017.
[3] Q. Madera, O. Castillo, M. García-Valdez, and A. Mancilla, "A method based on interactive evolutionary computation and fuzzy logic for increasing the effectiveness of advertising campaigns," *Inf. Sci.*, vol. 414, pp. 175–186, Nov. 2017.
[4] R. Rahman, S. R. Dhruba, S. Ghosh, and R. Pal, "Functional random forest with applications in dose-response predictions," *Sci. Rep.*, vol. 9, Feb. 2019, Art. no. 1628.
[5] MathWorks. (2018). *MATLAB Relase 2018b*. [Online]. Available: https://www.mathworks.com/products/matlab.html
[6] R Core Team. (2018). *R: A Language and Environment for Statistical Computing*. [Online]. Available: http://www.R-project.org
[7] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: An update," *ACM SIGKDD Explor. Newslett.*, vol. 11, no. 1, pp. 10–18, 2009. [Online]. Available: http://www.cs.waikato.ac.nz/ml/weka/
[8] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, *Data Mining: Practical Machine Learning Tools and Techniques*, 4th ed. San Francisco, CA, USA: Morgan Kaufmann, 2016.
[9] E. Raff, "JSAT: Java statistical analysis tool, a library for machine learning," *J. Mach. Learn. Res.*, vol. 18, no. 23, pp. 1–5, 2017. [Online]. Available: https://github.com/EdwardRaff/JSAT

[10] I. Triguero, S. González, J. M. Moyano, S. García, J. Alcalá-Fdez, J. Luengo, A. Fernández, M. del Jesús, L. Sánchez, and F. Herrera, "KEEL 3.0: An open source software for multi-stage analysis in data mining," *Int. J. Comput. Intell. Syst.*, vol. 10, pp. 1238–1249, Sep. 2017. [Online]. Available: http://www.keel.es

[11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, Oct. 2011. [Online]. Available: http://scikit-learn.org/stable/

[12] W. Dong and M. Zhou, "A supervised learning and control method to improve particle swarm optimization algorithms," *IEEE Trans. Syst., Man, Cybern. Syst.*, vol. 47, no. 7, pp. 1135–1148, Jul. 2017.

[13] J. C. Gámez, D. García, A. González, and R. Pérez, "An approximation to solve regression problems with a genetic fuzzy rule ordinal algorithm," *Appl. Soft Comput.*, vol. 78, pp. 13–28, May 2019.

[14] J. Cardoso-Silva, G. Papadatos, L. G. Papageorgiou, and S. Tsoka, "Optimal piecewise linear regression algorithm for QSAR modelling," *Mol. Informat.*, vol. 38, no. 3, 2019, Art. no. 1800028.

[15] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *J. Mach. Learn. Res.*, vol. 7, pp. 1–30, Jan. 2006.

[16] S. García, A. Fernández, J. Luengo, and F. Herrera, "Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power," *Inf. Sci.*, vol. 180, no. 10, pp. 2044–2064, 2010.

[17] D. H. Wolpert, "The lack of a priori distinctions between learning algorithms," *Neural Comput.*, vol. 8, no. 7, pp. 1341–1390, 1996. doi: 10.1162/neco.1996.8.7.1341.

[18] D. Dua and C. Graff. (2017). *UCI Machine Learning Repository*. [Online]. Available: http://archive.ics.uci.edu/ml

[19] J. Alcalá-Fdez, A. Fernández, J. Luengo, J. Derrac, S. García, L. Sánchez, and F. Herrera, "KEEL data-mining software tool: Data set repository, integration of algorithms and experimental analysis framework," *J. Multiple-Valued Logic Soft Comput.*, vol. 17, pp. 255–287, Jan. 2010.

[20] *Delve Datasets(Collections of Data for Developing, Evaluating, and Comparing Learning Methods)*. [Online]. Available: http://www.cs.toronto.edu/~delve/data/datasets.html

[21] U. Akujuobi and X. Zhang, "Delve: A dataset-driven scholarly search and analysis system," *SIGKDD Explor. Newsl.*, vol. 19, no. 2, pp. 36–46, Nov. 2017. doi: 10.1145/3166054.3166059.

[22] L. Torgo. *Luís Torgo Repository*. Accessed: Jul. 2019. [Online]. Available: http://www.dcc.fc.up.pt/~ltorgo/Regression/DataSets.html

[23] *Journal of Statistics Education Data Archive*. Accessed: Jul. 2019. [Online]. Available: http://jse.amstat.org/jse_data_archive.htm

[24] R. Alcalá, M. J. Gacto, and F. Herrera, "A fast and scalable multiobjective genetic fuzzy system for linguistic fuzzy modeling in high-dimensional regression problems," *IEEE Trans. Fuzzy Syst.*, vol. 19, no. 4, pp. 666–681, Aug. 2011.

[25] G. Jekabsons. (2016). *ARESLab: Adaptive Regression Splines Toolbox for MATLAB/Octave*. [Online]. Available: http://www.cs.rtu.lv/jekabsons/

[26] G. Jekabsons. (2016). *M5PrimeLab: M5' Regression Tree, Model Tree, and Tree Ensemble Toolbox for MATLAB/Octave*. [Online]. Available: http://www.cs.rtu.lv/jekabsons/

[27] G. Jekabsons. (2015). *Bump Hunting Using Patient Rule Induction Method for MATLAB/Octave*. [Online]. Available: http://www.cs.rtu.lv/jekabsons/

[28] M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim, "Do we need hundreds of classifiers to solve real world classification problems?" *J. Mach. Learn. Res.*, vol. 15, pp. 3133–3181, 2014.

[29] G.-B. Huang, H. Zhou, X. Ding, and R. Zhang, "Extreme learning machine for regression and multiclass classification," *IEEE Trans. Syst. Man, Cybern. B, Cybern.*, vol. 42, no. 2, pp. 513–529, Apr. 2012.

[30] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: Theory and applications," *Neurocomputing*, vol. 70, nos. 1–3, pp. 489–501, 2006.

[31] N. García-Pedrajas, C. García-Osorio, and C. Fyfe, "Nonlinear boosting projections for ensemble construction," *J. Mach. Learn. Res.*, vol. 7, pp. 1–33, Jan. 2007.

[32] C. Igel and M. Hüsken, "Empirical evaluation of the improved Rprop learning algorithms," *Neurocomputing*, vol. 50, pp. 105–123, Jan. 2003.

[33] R. Rojas and J. Feldman, *Neural Networks: A Systematic Introduction*. Berlin, Germany: Springer-Verlag, 1996.

[34] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, pp. 1–15, Dec. 2014.

[35] B. D. Ripley, *Neural Networks for Pattern Recognition*. Cambridge, U.K.: Cambridge Univ. Press, 1996.

[36] M. R. Berthold and J. Diamond, "Boosting the performance of RBF networks with dynamic decay adjustment," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 7, G. Tesauro, D. S. Touretzky, and T. K. Leen, Eds., 1995, pp. 521–528.

[37] C.-J. Hsieh, K.-W. Chang, C.-J. Lin, S. S. Keerthi, and S. Sundararajan, "A dual coordinate descent method for large-scale linear SVM," in *Proc. ACM Int. Conf. Mach. Learn. (ICML)*, 2008, pp. 408–415.

[38] C.-H. Ho and C.-J. Lin, "Large-scale linear support vector regression," *J. Mach. Learn. Res.*, vol. 13, pp. 3323–3348, Nov. 2012.

[39] R.-E. Fan, P.-H. Chen, and C.-J. Lin, "Working set selection using second order informationfor training support vector machines," *J. Mach. Learn. Res.*, vol. 6, pp. 1889–1918, Dec. 2005.

[40] C.-C. Chang and C.-J. Lin. (2008). *LIBSVM—A Library for Support Vector Machines*. [Online]. Available: http://www.csie.ntu.edu.tw/~cjlin/libsvm

[41] J. C. Platt, "Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods," in *Advances in Large Margin Classifiers*, A. J. Smola, P. L. Bartlett, B. Scholkopf, and D. Schuurmans, Eds. Cambridge, MA, USA: MIT Press, 2000, pp. 61–74.

[42] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, "LIBLINEAR: A library for large linear classification," *J. Mach. Learn. Res.*, vol. 9, pp. 1871–1874, Jun. 2008.

[43] S. K. Shevade, S. S. Keerthi, C. Bhattacharyya, and K. R. K. Murthy, "Improvements to the SMO algorithm for SVM regression," *IEEE Trans. Neural Netw.*, vol. 11, no. 5, pp. 1188–1193, Sep. 2000.

[44] T. Hothorn, K. Hornik, M. A. van de Wiel, and A. Zeileis, "A lego system for conditional inference," *Amer. Stat.*, vol. 60, no. 3, pp. 257–263, 2006.

[45] T. Hothorn, K. Hornik, and A. Zeileis, "Unbiased recursive partitioning: A conditional inference framework," *J. Comput. Graph. Statist.*, vol. 15, no. 3, pp. 651–674, Sep. 2006.

[46] J. R. Quinlan, *C4.5: Programs for Machine Learning*. San Mateo, CA, USA: Morgan Kaufmann, 1993.

[47] L. Breiman, J. Friedman, R. Olshen, and C. J. Stone, *Classification and Regression Trees*. London, U.K.: Chapman & Hall, 1984.

[48] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*. Berlin, Germany: Springer, 2009.

[49] P. Geurts, D. Ernst, and L. Wehenkel, "Extremely randomized trees," *Mach. Learn.*, vol. 63, no. 1, pp. 3–42, 2006.

[50] W.-Y. Loh and Y.-S. Shih, "Split selection methods for classification trees," *Statist. Sinica*, vol. 7, no. 4, pp. 815–840, 1997.

[51] W.-Y. Loh, "Regression tress with unbiased variable selection and interaction detection," *Statist. Sinica*, vol. 12, pp. 361–386, Apr. 2002.

[52] N. Meinshausen, "Quantile regression forests," *J. Mach. Learn. Res.*, vol. 7, pp. 983–999, Jun. 2006.

[53] T. K. Ho, "The random subspace method for constructing decision forests," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 20, no. 8, pp. 832–844, Aug. 1998.

[54] R. Kohavi, "The power of decision tables," in *Proc. Eur. Conf. Mach. Learn. (ECML)*, N. Lavrac and S. Wrobel, Eds. Berlin, Germany: Springer, 1995, pp. 174–189.

[55] L. Sánchez, I. Couso, and J. Corrales, "Combining GP operators with SA search to evolve fuzzy rule based classifiers," *Inf. Sci.*, vol. 136, nos. 1–4, pp. 175–191, 2001.

[56] L. Sánchez and I. Couso, "Fuzzy random variables-based modeling with GA-P algorithms," in *Information, Uncertainty and Fusion*, B. Bouchon-Meunier, R. R. Yager, and L. A. Zadeh, Eds. Boston, MA, USA: Springer, 2000, pp. 245–256.

[57] J. H. Friedman and N. I. Fisher, "Bump hunting in high-dimensional data," *Statist. Comput.*, vol. 9, no. 2, pp. 123–143, 1999.

[58] L.-X. Wang and J. M. Mendel, "Generating fuzzy rules by learning from examples," *IEEE Trans. Syst., Man, Cybern.*, vol. 22, no. 6, pp. 1414–1427, Nov. 1992.

[59] D. H. Wolpert, "Stacked generalization," *Neural Netw.*, vol. 5, no. 2, pp. 241–259, 1992.

[60] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.

[61] H. Deng and G. Runger, "Gene selection with guided regularized random forest," *Pattern Recognit.*, vol. 46, no. 12, pp. 3483–3489, 2013.

[62] H. Deng and G. Runger, "Feature selection via regularized trees," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, 2012, pp. 1–8.

[63] R. J. Quinlan, "Combining instance-based and model-based learning," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 1993, pp. 236–243.
[64] R. J. Quinlan, "Learning with continuous classes," in *Proc. Austral. Joint Conf. Artif. Intell. (AI)*, 1992, pp. 343–348.
[65] S. N. Wood, Y. Goude, and S. Shaw, "Generalized additive models for large data sets," *J. Roy. Stat. Soc. C, Appl. Statist.*, vol. 64, no. 1, pp. 139–155, 2015.
[66] S. N. Wood, Z. Li, G. Shaddick, and N. H. Augustin, "Generalized additive models for gigadata: Modeling the U.K. black smoke network daily data," *J. Amer. Stat. Assoc.*, vol. 112, no. 519, pp. 1199–1210, 2017.
[67] L. Xiao, "Dual averaging methods for regularized stochastic learning and online optimization," *J. Mach. Learn. Res.*, vol. 11, pp. 2543–2596, Oct. 2010.
[68] T. Hastie and R. Tibshirani, *Generalized Additive Models*. London, U.K.: Chapman & Hall, 1990.
[69] T. Hastie, "Generalized additive models," in *Statistical Models in S*, J. M. Chambers and T. Hastie, Eds. London, U.K.: Chapman & Hall, 1990, ch. 7.
[70] S. N. Wood, "Fast stable restricted maximum likelihood and marginal likelihood estimation of semiparametric generalized linear models," *J. Roy. Stat. Soc. B, Stat. Methodol.*, vol. 73, no. 1, pp. 3–36, 2011.
[71] S. Wood, "Stable and efficient multiple smoothing parameter estimation for generalized additive models," *J. Amer. Stat. Assoc.*, vol. 99, no. 467, pp. 673–686, 2004.
[72] A. J. Dobson, *An Introduction to Generalized Linear Models*, 2nd ed. London, U.K.: Chapman & Hall, 2001.
[73] J. Friedman, T. Hastie, and R. Tibshirani, "Regularization paths for generalized linear models via coordinate descent," *J. Statist. Softw.*, vol. 33, no. 1, pp. 1–22, 2010.
[74] W. N. Venables and B. D. Ripley, *Modern Applied Statistics With S*. Springer, 2002.
[75] P. J. Huber, "Regression," in *Robust Statistics*, P. J. Huber, Ed. Hoboken, NJ, USA: Wiley, 2005, pp. 153–198.
[76] A. B. Owen, "A robust hybrid of lasso and ridge regression," in *Prediction and Discovery* (Contemporary Mathematics), vol. 443. Salt Lake City, UT, USA: American Mathematical Society, 2007, pp. 59–71.
[77] B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani, "Least angle regression," *Ann. Statist.*, vol. 32, no. 2, pp. 407–499, 2004.
[78] J. M. Chambers, "Linear models," in *Statistical Models in S*, J. M. Chambers and T. J. Hastie, Eds. Belmont, CA, USA: Wadsworth, 1992, ch. 4.
[79] G. N. Wilkinson and C. E. Rogers, "Symbolic description of factorial models for analysis of variance," *J. Roy. Stat. Soc. C, Appl. Statist.*, vol. 22, no. 3, pp. 392–399, 1973.
[80] K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer, "Online passive-aggressive algorithms," *J. Mach. Learn. Res.*, vol. 7, pp. 551–585, Dec. 2006.
[81] D. Collett, *Modelling Binary Data*, 2nd ed. London, U.K.: Chapman & Hall, 2002.
[82] P. McCullagh and J. A. Nelder, *Generalized Linear Models*, 2nd ed. London, U.K.: Chapman & Hall, 1989.
[83] X. Dang, H. Peng, X. Wang, and H. Zhang, "Theil-sen estimators in a multiple linear regression model," Ole Miss Education, Tech. Rep., 2009.
[84] D. W. Aha, D. Kibler, and M. K. Albert, "Instance-based learning algorithms," *Mach. Learn.*, vol. 6, no. 1, pp. 37–66, 1991.
[85] B. S. Dayal and J. F. MacGregor, "Improved PLS algorithms," *J. Chemom.*, vol. 11, no. 1, pp. 73–85, Jan. 1997.
[86] S. de Jong, "SIMPLS: An alternative approach to partial least squares regression," *Chemometrics Intell. Lab. Syst.*, vol. 18, no. 3, pp. 251–263, 1993.
[87] J. H. Friedman, "Multivariate adaptive regression splines," *Annu. Statist.*, vol. 19, no. 1, pp. 1–67, Mar. 1991.
[88] J. H. Friedman, "Fast MARS," Dept. Statist., Stanford Univ., Stanford, CA, USA, Tech. Rep. 101, 1993.
[89] L. Breiman, "Bagging predictors," *Mach. Learn.*, vol. 24, no. 2, pp. 123–140, 1996.
[90] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Ann. Statist.*, vol. 29, no. 5, pp. 1189–1232, Oct. 2001.
[91] J. H. Friedman, "Stochastic gradient boosting," Stanford Univ., Stanford, CA, USA, 1999. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167947301000652
[92] M. Hall, "Correlation-based feature subset selection for machine learning," Ph.D. dissertation, Dept. Comput. Sci., Univ. Waikato, Hamilton, New Zealand, 1998.
[93] T. Zhang, "Adaptive forward-backward greedy algorithm for learning sparse representations," *IEEE Trans. Inf. Theory*, vol. 57, no. 7, pp. 4689–4708, Jul. 2011.
[94] Y. Engel, S. Mannor, and R. Meir, "The kernel recursive least-squares algorithm," *IEEE Trans. Signal Process.*, vol. 52, no. 8, pp. 2275–2285, Aug. 2004.
[95] J. G. Cleary and L. E. Trigg, "K*: An instance-based learner using an entropic distance measure," in *Proc. Int. Conf. Int. Conf. Mach. Learn. (ICML)*. San Francisco, CA, USA: Morgan Kaufmann, 1995, pp. 108–114.
[96] C. Atkeson, A. Moore, and S. Schaal, "Locally weighted learning," *Artif. Intell. Rev.*, vol. 11, no. 1, pp. 11–73, Apr. 1997.
[97] E. Frank, M. Hall, and B. Pfahringer, "Locally weighted Naive Bayes," in *Proc. Conf. Uncertainty Artif. Intell. (UAI)*. San Francisco, CA, USA: Morgan Kaufmann, 2003, pp. 249–256.
[98] J. H. Friedman and W. Stuetzle, "Projection pursuit regression," *J. Amer. Statist. Assoc.*, vol. 76, no. 376, pp. 817–823, Dec. 1981.
[99] N. Meinshausen, "Relaxed lasso," *Comput. Stat. Data Anal.*, vol. 52, no. 1, pp. 374–393, 2007.
[100] H. Ishwaran and J. S. Rao, "Spike and slab variable selection: Frequentist and Bayesian strategies," *Ann. Statist.*, vol. 33, no. 2, pp. 730–773, 2005.
[101] H. Ishwaran and J. S. Rao, "Spike and slab gene selection for multigroup microarray data," *J. Amer. Stat. Assoc.*, vol. 100, no. 471, pp. 764–780, 2005.
[102] M. Friedman, "The use of ranks to avoid the assumption of normality implicit in the analysis of variance," *J. Amer. Stat. Assoc.*, vol. 32, no. 200, pp. 675–701, Dec. 1937.
[103] S. Holm, "A simple sequentially rejective multiple test procedure," *Scandin. J. Statist.*, vol. 6, no. 2, pp. 65–70, 1979.
[104] T. K. Ho and M. Basu, "Complexity measures of supervised classification problems," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, no. 3, pp. 289–300, Mar. 2002.
[105] M. Basu and T. K. Ho, Eds., *Data Complexity in Pattern Recognition* (Advanced Information and Knowledge Processing), vol. 16. London, U.K.: Springer-Verlag, 2006.

MARÍA JOSÉ GACTO received the M.Sc. degree in computer science and the Ph.D. degree in computer science from the University of Granada, Spain, in 1999 and 2010, respectively. She is a member of the Intelligent Systems and Data Mining Research Group, Department of Computer Science, University of Jaén. She has over 40 international publications. She has worked on several research projects supported by the Spanish government and the European Union. Her research interests include multi-objective genetic algorithms and genetic fuzzy systems, particularly the learning/tuning of fuzzy systems for modeling and control with a good trade-off between accuracy and interpretability.

JOSE MANUEL SOTO-HIDALGO received the M.S. and Ph.D. degrees in computer science from the University of Granada, Spain, in 2004 and 2014, respectively. Since 2007, he has been a member of the Department of Electronics and Computer Engineering, University of Córdoba, Spain, where he is currently an Associate Professor. He is a member of the "Software Fuzzy Systems" Task Force (TF-FSS), the IEEE Computational Intelligence Society (CIS), since 2016. His current research interests include soft-computing applied to image processing, data science and development of free software, and data sensors management tools. He has authored more than 50 papers in international journals, book chapters, and conferences.

JESÚS ALCALÁ-FDEZ received the M.Sc. and Ph.D. degrees in computer science from the University of Granada, Spain, in 2002 and 2006, respectively. He is currently an Associate Professor with the Department of Computer Science and Artificial Intelligence, University of Granada, and he is a member of the Andalusian Research Institute DaSCI. He has published more than 80 papers in international journals, book chapters, and conferences, with a H-index of 20 (source: ISI WoK). His current research interests include association rules, genetic fuzzy systems, multiobjective evolutionary algorithms, data mining software, data science, bioinformatics, and big data. He acts as an editorial member of several international journals. He was the Chair of the Software Fuzzy Systems Task Force (TF-FSS) and the Fuzzy Systems Technical Committee, the IEEE Computational Intelligence Society, from 2011 to 2017. He is currently the Vice-Chair of the TF-FSS.

RAFAEL ALCALÁ received the M.Sc. degree in computer science and the Ph.D. degree in computer science from the University of Granada, Spain, in 1998 and 2003, respectively, where he is currently a Full Professor with the Department of Computer Science and A.I.

He has published over 100 papers in international journals, book chapters, and conferences. His current research interests include multiobjective genetic algorithms and genetic fuzzy systems, particularly the learning/tuning of fuzzy systems for regression and control with a good trade-off between accuracy and interpretability, imbalanced regression, as well as fuzzy association rules. He currently serves as a member of the editorial board of the IEEE TRANSACTIONS ON FUZZY SYSTEMS INTERNATIONAL JOURNAL. He was the President of the FSTC "Genetic Fuzzy Systems" Task Force at the IEEE Computational Intelligence Society (2009–2014), and the Vice-President (2014–2018). He was the Program Co-Chair at GEFS 2010, the Area Co-Chair at the FUZZ-IEEE 2011, and the General Co-Chair at GEFS 2011 and 2013.