An extensive experimental survey of regression methods

M. Fernández-Delgado[1,*], M.S. Sirsat[1,2], E. Cernadas[1], S. Barro[1], M. Febrero Bande[3]

Abstract

Regression is one of the most relevant problems in machine learning, with many different available approaches. The current work presents a comparison of a large collection composed by 77 popular regressors which belong to 19 families: linear and generalized linear models, generalized additive models, least squares, projection methods, LASSO and ridge regression, Bayesian models, Gaussian processes, quantile regression, nearest neighbors, regression trees and rules, random forests, bagging and boosting, neural networks, deep learning and support vector regression. These methods are evaluated using all the regression datasets of the UCI machine learning repository (83 datasets), with some exceptions due to technical reasons. The experimental work identifies several outstanding regressors: the M5 rule-based model with corrections based on nearest neighbors (`cubist`), the gradient boosted

*M. Fernández-Delgado
Email address: `manuel.fernandez.delgado@usc.es` (M. Fernández-Delgado)
URL: `http://citius.usc.es` (M. Fernández-Delgado)
[1]Centro Singular de Investigación en Tecnoloxías da Información da USC (CiTIUS), University of Santiago de Compostela, Campus Vida, 15782, Santiago de Compostela, Spain.
[2]Laboratory of Artificial Intelligence and Decision Support (LIAAD), INESC Tec., Campus da FEUP, Porto, Portugal.
[3]Dept. of Statistics, Mathematical Analysis and Optimization, University of Santiago de Compostela, Campus Vida, 15782, Santiago de Compostela, Spain.

machine (gbm), the boosting ensemble of regression trees (bstTree) and the M5 regression tree. Cubist achieves the best squared correlation (R^2) in 15.7% of datasets being very near to it, with difference below 0.2 for 89.1% of datasets, and the median of these differences over the dataset collection is very low (0.0192), compared e.g. to the classical linear regression (0.150). However, cubist is slow and fails in several large datasets, while other similar regressors as M5 never fail and its difference to the best R^2 is below 0.2 for 92.8% of datasets. Other well-performing regresors are the committee of neural networks (avNNet), extremely randomized regression trees (extraTrees, which achieves the best R^2 in 33.7% of datasets), random forest (rf) and ε-support vector regression (svr), but they are slower and fail in several datasets. The fastest regressor is least angle regression lars, which is 70 and 2,115 times faster than M5 and cubist, respectively. The regressor which requires least memory is non-negative least squares (nnls), about 2 GB, similarly to cubist, while M5 requires about 8 GB. For 97.6% of datasets there is a regressor among the 10 bests which is very near (difference below 0.1) to the best R^2, which increases to 100% allowing differences of 0.2. Therefore, provided that our dataset and regressor collection are representative enough, the main conclusion of this study is that, for a new regression problem, some regressor in our top-10 should achieve R^2 near to the best attainable for that problem.

Keywords: Regression, UCI machine learning repository, cubist, M5, gradient boosted machine, extremely randomized regression tree, support vector regression penalized linear regression.

1. Introduction

The objective of this paper is to provide a “road map” for researchers who want to solve regression problems and need to know how well work the currently available regression methods. In machine learning, regression methods are designed to predict continuous numeric outputs where an order relation is defined. Regression has been widely studied from the statistics field, which provides different approaches to this problem: linear and generalized linear regression, least and partial least squares regression (LS and PLS), least absolute shrinkage and selection operator (LASSO) and ridge regression, multivariate adaptive regression splines (MARS), least angle regression (LARS), among others. Furthermore, several methods arising from the field of machine learning were designed to be universal function aproximators, so they can be applied both for classification and regression: neural networks, support vector machines, regression trees and rules, bagging and boosting ensembles, random forests and others. The current work develops an empirical quantitative comparison of a very large collection of regression techniques which is intended to provide the reader: 1) a list of the currently available regressors, grouped by families of related methods; 2) a brief description and list of references about each approach, alongside with technical details about its execution such as software implementation, list of tunable hyperparameters and recommended values; 3) a ranking of the available regressors according to its performace and speed, identifying the best performing approach and the performance level which can be expected for it; and 4) the code to run all the regressors considered in this study for any regression problem[4]. In this comparison, we use the whole collection of regression datasets provided of the UCI machine learning repository (excepting some datasets excluded by technical reasons), which is the largest available collection of regression problems, and it should allow to develop a realistic and significant evaluation of the regression methods. As we explained in a previous paper comparing classifiers [1], provided that the size of the regressor collection used in the current comparison is large enough, we can assume that the best performance, measured in terms of squared correlation (R^2), achieved by some regressor for each dataset (denoted as R^2_{best}) is the highest attainable performance for that dataset. For a regressor which achieves a given R^2 in that dataset, the difference $\Delta = R^2_{best} - R^2$, averaged over the dataset collection, can be used as an estimation of the expectable Δ for that regressor and a new dataset D, not included in the collection. For the best regressor X on the current comparison, it is expected that $\Delta \gtrsim 0$, i.e., the R^2 achieved by X should not be too far from R_{best} in average over the data collection. Thus, although by the No-Free-Lunch theorem [2] we can not guarantee that X will be the best regressor for D, we can expect that X will achieve $R^2 > R^2_{best} - \Delta$, so that X will not be very far from R^2_{best} for dataset D. Consequently, the current paper may be useful for researchers who want to know how far a given regressor (e.g. the best regressor X) will be from the best available performance (which is, of course, unknown) for a new dataset. On the other hand, in general the best regressors in the current comparison achieve the best, or very near to the best, performances for most datasets in

[4]http://persoal.citius.usc.es/manuel.fernandez.delgado/papers/regression.tar.gz (Available upon paper acceptance).

the collection. Therefore, although X will not be the best regressor for a new dataset D, we can expect that some of the best regressors in our comparison will achieve the best R^2. Thus, the current comparison may be also useful to provide to the reader a reduced list (e.g., the 10 best performing regressors of the collection) which is expected to include the one which provides the highest available performance for a new dataset D.

The section 2 describes the materials and methods used for this comparison, which include the list of datasets and regression methods, grouped by families. The description of regressors and issues related to their execution (software implementation, number of tunable hyperparameters and their values) are included in Appendix A. The section 3 reports the results of the experimental work and discusses them globally, by regressor families and datasets, best regressor for each dataset, elapsed time and memory. Finally, the section 4 compiles the conclusions of this study.

2. Materials and methods

This section describes the scope of the current work, defined by the collection of datasets used in this comparison (subsection 2.1) and by the regression methods that will be compared (subsection 2.2).

2.1. Datasets

In the current research, we selected 48 of the 82 datasets (81 because the *Air Quality* dataset is repeated) listed as regression problems[5] by the UCI Machine Learning Repository [3]. The remaining 33 datasets were discarded

[5]http://archive.ics.uci.edu/ml/datasets.html?task=reg (visited Feb, 5, 2018).

Excluded dataset	Reason
Amazon access samples	Huge number of inputs (20,000)
Breast cancer Wisconsin (Prognostic)	Too few recurrent patterns (47)
Cargo 2000 Freight Tracking and Tracing	Less than 10 different output values (3)
Challenger USA space shuttle O-ring	Too few patterns (23) and inputs (3)
Condition based maintenance of naval propulsion plants (`compress` output)	Less than 10 different output values (9)
Container crane controller	Too few patterns (15)
DrivFace	Less than 10 different output values (4 subjects)
Early biomarkers of Parkinsons disease based on natural connected speech	Data are not available
Educational process mining	Inputs and output for regression are not clear
ElectricityLoadDiagrams	Huge number of inputs (140,256)
Fertility	Less than 10 different output values (2)
Gas sensor array drift dataset at different concentrations	Less than 10 different output values (7)
Gas sensor array exposed to turbulent gas mixtures	Huge number of inputs (150,000)
Gas sensor array under flow modulation	Less than 10 different output values (4)
Geo-Magnetic field and WLAN	Data format very complex
Improved spiral test using digitized graphics tablet for monitoring parkinsons disease	Data are not available
Insurance Company Benchmark (COIL 2000)	Less than 10 different output values (3)
KDC-4007 dataset Collection	Less than 10 different output values (8)
KDD cup 1998	Format too complex
Las Vegas Strip	Less than 10 different output values (5)
News popularity in multiple social media platforms	Data are text instead of numbers
Noisy office	Format too complex (PNG images)
Open university learning analytics	Format too complex
Paper Reviews	Less than 10 different output values (5)
Parkinson disease spiral drawings using digitized graphics tablet	Less than 10 different output values (3)
Skillcraft1 master table	Less than 10 different output values (7)
Solar flare	Less than 10 different output values
Tamilnadu electricity board hourly readings	Less than 10 different output values (2)
Tennis major tournament match statistics	Format problems
Twin gas sensor arrays	Less than 10 different output values (4)
UJIIndoorLoc-Mag	Output almost constant, format very complex
wiki4HE	Less than 10 different output values (7)
Wine quality (white/red)	Less than 10 different output values (7/6)

Table 1: List of the UCI regression datasets which were excluded from this study with the reason to be discarded. In datasets with discrete outputs the number of different output values is between parentheses.

due to the reasons listed in Table 1. The reason which leaded to discard a larger amount (17) of datasets was the reduced number of output values, because a regression problem is expected to have a continuous output. We

Original UCI name	Datasets	#patterns	#inputs
3D Road network	3Droad	434,874	4/3
Airfoil self-noise	airfoil	1,503	5
Air quality	air-quality-CO, air-quality-NMHC air-quality-NO2, air-quality-NOx air-quality-O3	1,230	8
Appliances energy prediction	appliances-energy	19,735	28/26
Auto MPG	auto-MPG	398	8/23
Automobile	automobile	205	26/66
Beijing PM2.5	beijing-pm25	41,758	12
Bike sharing	bike-day	731	13/30
	bike-hour	17,379	14/42
Blog feedback	blog-feedback	60,021	280/13
Buzz in social media	buzz-twitter	583,250	77
Combined cycle power plant	combined-cycle	9,568	4
Communities & crime	com-crime	1,994	122
Communities & crime unnormalized	com-crime-unnorm	2,215	124/126
Computer hardware	com-hd	209	7
Concrete compressive strength	compress-stren	1,030	8
Concrete slump test	slump	103	9
	slump-comp, slump-flow		7
Condition based maintenance of naval propulsion plants	cond-turbine	11,934	13
Conventional and Social Media Movies 14/15	csm1415	231	12/11
Cuff-less blood pressure estimation	cuff-less	61,000	3/2
Daily Demand Forecasting Orders	daily-demand	60	13/12
Dynamic features of VirusShare Executables	dynamic-features	107,856	482/265
Energy efficiency	energy-cool, energy-heat	768	8/7
Facebook comment volume	facebook-comment	40,949	54/48
Facebook metrics	facebook-metrics	500	19
Forestfires	forestfires	517	12/39
Gas sensor array under dynamic gas mixtures	gas-dynamic-CO	58	438/57
	gas-dynamic-methane		
Geographical original of music	geo-lat, geo-long	1,059	116/72
	geo-music-lat, geo-music-long		68
GPS trajectories	gps-trajectory	163	10
Greenhouse gas observing network	greenhouse-net	955,167	15
Housing	housing	452	13
Individual household electric power consumption	household-consume	2,049,280	6/5

Table 2: Collection of 83 datasets from the UCI repository. Each column reports: original name in the UCI repository; datasets created from the original one; number of patterns and inputs, before and after preprocessing. Continued in Table 3.

decided to exclude those datasets whose outputs have less than 10 values. These datasets should be considered as classification instead of regression problems. Tables 2 and 3 report the collection of 83 datasets which we use

Original UCI name	Datasets	#patterns	#inputs
Istanbul stock exchange	stock-exchange	536	8
KEGG metabolic reaction network (undirected)	KEGG-reaction	65,554	27/25
KEGG metabolic relation network (directed)	KEGG-relation	54,413	22/17
Online news popularity	online-news	39,644	59/55
Online video characteristics and transcoding time dataset	video-transcode	68,784	20/8
Parkinson speech dataset with multiple types of sound recordings	park-speech	1,040	26
Parkinsons telemonitoring	park-motor-UPDRS, park-total-UPDRS	5,875	16
	pm25-beijing-dongsi	24,237	
	pm25-beijing-dongsihuan	20,166	
	pm25-beijing-nongzhanguan	24,137	
	pm25-beijing-us-post	49,579	
	pm25-chengdu-caotangsi	22,997	
	pm25-chengdu-shahepu	23,142	
	pm25-chengdu-us-post	27,368	
PM2.5 Data 5 Chinese Cities	pm25-guangzhou-city-station	32,351	13
	pm25-guangzhou-5th-middle-school	21,095	
	pm25-guangzhou-us-post	32,351	
	pm25-shanghai-jingan	22,099	
	pm25-shanghai-us-post	31,180	
	pm25-shanghai-xuhui	23,128	
	pm25-shenyang-taiyuanji	22,992	
	pm25-shenyang-us-post	20,452	
	pm25-shenyang-xiaoheyan	23,202	
Physicochemical properties of protein tertiary structure	physico-protein	45,730	9
Relative location of CT slices on axial axis	CT-slices	53,500	385/355
Servo	servo	167	4/15
SML2010	SML2010	4,137	20/18
Stock portfolio	stock-abs, stock-annual, stock-excess stock-rel, stock-systematic, stock-total	252	6
Student performance	student-mat	395	32/77
	student-por	649	32/56
UJIIndoorLoc	UJ-lat, UJ-long	21,048	528/373
Yacht hydrodynamics	yacht-hydro	308	6
YearPredictionMSD	year-prediction	2,000	90

Table 3: Continuation of Table 2.

in the current work, with their numbers of patterns and inputs (attributes). Some of the 48 original UCI regression datasets selected for this work generated several regression problems, one for each data column which can be used as output for regression. Thus, some UCI datasets (whose original names are listed in the column 1 of the tables) give several datasets in column 2 (e.g., the *Air quality* dataset gives five datasets named by us *air-quality-CO*, *air-quality-NMHC*, etc.). There are some discrepancies with respect to the documentation of the UCI ML repository. The *beijing-pm25* dataset has 41,757 patterns, despite its description in the UCI documentation specifies 43,824 because 2067 patterns whose output is missing were removed. The *cuff-less* dataset has 73,200,000 patterns, while its description specifies 12,000. Instead of discarding it, we used the first 61,000 patterns. The *greenhouse-net* dataset has 2,921 files with 327 patterns per file, which gives 955,167 patterns instead of 2,921 in the dataset description. The *household-consume* dataset has 2,049,280 patterns instead of 2,075,259 as listed in the UCI documentation, because 25,979 patterns have missing values (labeled as '?') in the original data. The *online-news* dataset has 39,644 patterns instead of 39,797 as listed in the UCI documentacion. For the *UJIIndoorLoc* datasets, output `floor` was discarded and did not give a separate regression dataset because it has only tree different values.

Since many datasets are large, often the regressors are not able to train and test with them due to the limited RAM memory, although we set a maximum size of 128 GB. Besides, some regressors spend a long time to finish, so we fixed a maximum run-time of 48 hours and labeled any regressor that could not finish within this time lapse as failing for this dataset. As usual, the output was pre-processed using the Box-Cox transformation [4] in order to make it more similar to a symmetric uni-modal distribution, with the `boxcox` function (`MASS` package) of the R statistical computing language [5]. In the *greenhouse-net* and *com-crime-unnorm* datasets, the decimal logarithm of

Dataset and details	Dataset and details
3Droad: 4: altitude	geo-long: 118: longitude; same file
airfoil : 6: scaled sound pressure	geo-music-lat : 69: latitude; default file
air-quality-CO : 3: PT08.S1; 1,2,7,9,11,12	geo-music-long : 70: longitude; same file
air-quality-NMHC : 7: PT08.S2; 1,2,4,9,11,12	gps-trajectory* : 2: speed; 1,9,12,13; tracks file
air-quality-NO2 : 10: PT08.S4; 1,2,4,7,9,12	greenhouse-net: 16: synthetic; pasted all files
air-quality-NOx : 9: PT08.S3; 1,2,4,7,11,12	household-consume* : 3: global_active_power
air-quality-O3 : 11: PT08.S5; 1,2,4,7,9,11	housing: 14: MEDV
appliances-energy : 2: appliances; 1	KEGG-reaction* : 29: edgeCount; 1
auto-MPG* : 1: mpg	KEGG-relation : 24: ClusteringCoefficient; 1
automobile : 26: price :	online-news : 60: shares
bike-day : 16: cnf; 1,2; day.csv	park-motor-UPDRS : 5: motor_UPDRS; 1 2, 3, 4, 6
bike-hour : 17: cnf; 1,2; hour.csv	park-speech : 28: UPDRS; train_data.txt
blog-feedback : 281: target; pasted all files	park-total-UPDRS : 6: total_UPDRS; 1, 2, 3, 4, 5
buzz-twitter : 78: discussions : Twitter.data	physico-protein : 1: RMSD
combined-cycle : 5: PE; Folds5x2_pp.csv	servo : 5: class
com-crime* : 128: ViolentCrimesPerPop;1-5	slump : 8: slump
com-crime-unnorm* : 146: ViolentCrimesPerPop; 1-5,130-145,147	slump-comp : 10: comp. strength
com-hd : 10: ERP; 1,2	slump-flow : 9: flow
compress-stren : 9: ccs; Concrete_data.xls	SML2010* : 3: dining-room temperature; 1,2,4; both files
cond-turbine : 18: GT Turbine; 17; data.txt	stock-exchange : 10: EM; 1
CT-slices : 386: reference	student-mat : 33: G3; G1, G2
cuff-less* : 2: ABP	student-por : 33: G3; G1, G2
energy-cool : 10: cool	UJ-lat : 522: latitude; both files
energy-heat : 9: heat	UJ-long : 521: longitude; both files
facebook-comment : 54; Features_Variant_1.csv	
facebook-metrics : 1	
forestfires : 13: area	video-transcode : 21: utime; transcoding_mesurment.tsv
gas-dynamic-CO : 2: CO conc; 1	yacht-hydro : 7: resistance
gas-dynamic-methane : 2: Methane; 1	year-prediction : 1: year
geo-lat : 117: latitude; chromatic file	

Table 4: Information about datasets used in the current work: column number and name (if exists) used as output; removed columns (e.g., time marks or other outputs) where corresponds; files used, in datasets where several files are available; *: means that dataset contains missing patterns, which we replaced by the column mean.

the inputs are used, due to the wide range of many inputs. The constant, repeated and collinear inputs are removed from all the datasets. Specifically, the `lm` function in the `stats` R package is used to calculate the coefficients of the linear model trained on the whole dataset, and the inputs with `NA` (not available) coefficients in the linear model are removed. This reason leads e.g. the *Blog feedback* dataset to reduce its inputs from 280 to 13. The rationale behind this is that constant, repeated or collinear inputs lead many regressors to develop calculations with singular matrices, so it is useful to remove these inputs in order to avoid the subsequent errors. On the other hand, the inputs with discrete values are replaced by dummy (also named indicator) inputs. For each discrete input with n values, it is replaced by $n-1$ dummy binary inputs. The first value of the original discrete input is codified as zero values for the $n-1$ dummy inputs; the second value is codified as 1 in the first dummy variable and zero in the remaining ones, and so on. Therefore, those datasets with discrete inputs increase the number of inputs, so that e.g. the *student-mat* dataset (Table 3, second column) increases its inputs from 32 to 77 due to the presence of discrete inputs. In both tables 2 and 3, the datasets whose "#inputs" column shows two numbers (i.e. 8/23), the first is the number of inputs of the original UCI dataset, and the second is the number of inputs used effectively in our experiments, after removing constant, repeated and collinear inputs, and after replacing discrete inputs by their corresponding dummy variables. Those datasets with only one number in the #inputs column means that no input was removed nor added. Table 4 reports the name and number of the attribute used as output for each dataset. It also specifies the numbers of the columns that were discarded (if any), due to being useless (e.g., times, dates, names, etc.) or because they are alternative outputs (in datases with several outputs to be predicted) which can not be used as inputs (e.g., `latitude` can not be used as input for *UJ-long* dataset in Table 3). In those datasets with more than one file, the table specifies the files used. An asterisk (*) identifies datasets with missing values, which are replaced by the mean of the non-missing values of that column.

2.2. Regressors

We apply a wide collection of 77 regressors which belong to several families. All the files (data, programs and results) are publicly available[6]. The majority of them (74 regressors) are selected from the list of models[7] included in the Classification and Regression Training (`caret`) R package [6]. We discarded 52 `caret` regressors listed in Table 5, either because they are equivalent to other regressors already included in our study (which are listed in the "Equivalence" columns of the upper part of the table), due to run-time errors or because they can not be used for regression (listed in the lower part of the table). Instead of using the `train` function of the `caret` package, we ran the regressors directly using the corresponding R packages (see the detailed list of regressors below), in order to control the execution of each single model. Besides, the direct execution allows us to use the same configuration (e.g., the same training and test patterns) as other four popular regressors, implemented in other platforms, that we included in our study although they do not belong to the `caret` model list (see the link in the above footnote). These regressors are the deep learning neural network (named `dlkeras` in our study), using the module `Keras`, configured for Theano [7], in the Python programming language [8]; the ε-support vector regression (named `svr`), implemented by the `LibSVM` library [9] and accessed via the C++ interface; the generalized regression neural network and extreme learning machine with

[6]http://persoal.citius.usc.es/manuel.fernandez.delgado/papers/regression.tar.gz (Available upon paper acceptance. Visited May 10, 2017).

[7]http://topepo.github.io/caret/available-models.html (visited April 27, 2017).

Gaussian kernel (named `grnn` and `kelm`, respectively) in Matlab [10].

Equivalence	Equivalence	Equivalence	Equivalence
bagEarthGCV $\rightarrow$ bagEarth	ctree $\rightarrow$ ctree2	gamLoess, gamSpline $\rightarrow$ gam	enpls $\rightarrow$ enpls.fs
gcvEarth $\rightarrow$ earth	glm.nb $\rightarrow$ bayesglm	glmnet_h2o $\rightarrow$ glmnet	knn $\rightarrow$ kknn
lars2 $\rightarrow$ lars	lmStepAIC $\rightarrow$ glmSAIC	M5Rules $\rightarrow$ M5	pls $\rightarrow$ simpls
nnet,mlpWD, mlpSGD, neuralnet $\rightarrow$ mlpWDml		RRFglobal $\rightarrow$ RRF	rbfDDA $\rightarrow$ rbf
parRF, ranger, Rborist, rfRules $\rightarrow$ rf		rpart1SE, rpart2 $\rightarrow$ rpart	xyf $\rightarrow$ bdk

Regressor not used	Reason
bam	Version of gam for very large datasets
krlsPoly	Polynomial kernel is not implemented
ordinalNet	It requires a discrete output
blasso, blassoAveraged, bridge	Not valid for regression
leapBackward, leapForward, leapSeq	Run-time errors
logicBag, logreg	Only for logic regression (binary outputs)
svmLinear, svmPoly, rvmLinear, rvmPoly	Replaced by their versions with radial kernel
svmBoundrangeString, svmExpoString	Only for text classification
ANFIS, DENFIS, FIR.DM, GFS.LT.RS,HYFIS	Run-time errors
GFS.FR.MOGUL, GFS.THRIFT, WM, FS.HGD	

Table 5: Upper part: Regressors of the `caret` model list which are not used because an equivalent model is already included in our study (`mlpWD` and `mlpWDml` refer to `mlpWeightDecay` and `mlpWeightDecayML`, respectively, in the `caret` model list). Lower part: regressors of the `caret` model list excluded from this study due to run-time errors and other reasons.

The model operation is optimized by tuning the set of hyperparameters specified in the `caret` model list. Almost all the regressors that we used have from one to four tunable hyperparameters. We specify the number of values tried for each hyperparameter (defined in the file `values.txt`, placed in the folder `programs/R` of the file `regression.tar.gz`), which are listed in the regressor description below. However, the specific hyperparameter values are calculated by the `getModelInfo` function of the `caret` package, being in some cases different for each dataset. Note that for some regressors (e.g. `gprRad`) and datasets, this function returns a value list with less items than the number specified in `values.txt`, and even sometimes just one value is used. In

Family	Regressors	Family	Regressors
Linear regression (LR)	1. lm [11]	Least absolute shrinkage and selection operator (LASSO)	21. lasso [12]
	2. rlm [13]		22. relaxo [14]
Generalized linear regression (PLM)	3. penalized [15]		23. lars [16]
	4. enet [12]	Ridge	24. ridge [12]
	5. glmnet [17]		25. spikeslab [18]
	6. glmSAIC [19]		26. foba [20]
Additive models (AM)	7. gam [21]	Bayesian models (BYM)	27. bayesglm [22]
	8. earth [23]		28. brnn [24]
Least squares (LS)	9. nnls [25]		29. bMachine [26]
	10. krlsRadial [27]	Gaussian processes (SGP)	30. gprLin [28]
Projection methods (PRJ)	11. spls [29]		31. gprRad [28]
	12. simpls [30]		32. gprPol [28]
	13. kpls [31]	Quantile regression (QTR)	33. rqlasso [32]
	14. wkpls [33]		34. rqnc [34]
	15. enpls.fs [35]		35. qrnn [36]
	16. plsRglm [37]	Nearest neighbors (NN)	36. kknn [38]
	17. ppr [39]	Regression trees (RGT)	37. rpart [40]
	18. pcr [41]		38. nodeHarvest [42]
	19. icr [43]		39. ctree2 [44]
	20. superpc [45]		40. partDSA [46]
			41. evtree [47]

Table 6: List of regressors and references grouped by families (see Appendix A for a brief description of each regressor).

these cases, although the `caret` model list specifies that hyperparameter as tunable, in the practice only one value is used. The list of hyperparameter values which are used in our experiments for a regressor and dataset is included in the file `results_regressor_dataset.dat`, where `regressor` and `dataset` stand for the names of regressor and dataset, respectively, which is placed in the directory `results/dataset/regressor_implem`, where `implem` may be `R`, `C`, `Python` or `Matlab`. For some regressors (`ridge`, `rlm`, `mlpWD`, `mlpWDml`, `dnn`, `krlsRad` and `icr`), the value list provided by the `getModelInfo` function was not valid due to several reasons, so we directly specify the hyperparameter values used for tuning in the file `programs/R/initialize.R`. The regressors in Matlab, C++ and Python use pre-specified values, listed

Family	Regressors	Family	Regressors
Regression rules (RGR)	42. M5 [48]	Boosting (BST) (continued)	60. gbm [49]
	43. cubist [50]		61. blackboost [51]
	44. SBC [52]		62. xgbTree [53]
Random forests (RF)	45. rf [54]		63. xgbLinear [53]
	46. Boruta [55]	Neural networks (NET)	64. mlWD [56]
	47. RRF [57]		65. mlWDml [56]
	48. cforest [58]		66. avNNet [6]
	49. qrf [59]		67. rbf [56]
	50. extraTrees [60]		68. grnn [61]
Bagging (BAG)	51. bag [62]		69. elm [63]
	52. bagEarth [6]		70. kelm [63]
	53. treebag [64]		71. pcaNNet [6]
Boosting (BST)	54. rndGLM [65]		72. bdk [66]
	55. BstLm [53]	Deep learning (DL)	73. dlkeras [67]
	56. bstSm [53]		74. dnn [68]
	57. bstTree [53]	Support vector regression (SVR)	75. svr [9]
	58. glmboost [69]		76. svmRad [70]
	59. gamboost [69]		77. rvmRad [71]

Table 7: Continuation of Table 6.

in the script `run_regressor.sh`, which are the same for all datasets. Tables 6 and 7 list the the collection of 77 regressors used in this work, grouped by families, which are described in the Appendix A, specifying the software implementation (R package or other platforms), their tunable hyperparameters and the values used.

3. Results and discussion

The experimental work [72] uses the following methodology: for each dataset with less than 10,000 patterns, $N = 500$ random partitions are generated, using the 50% of the patterns for training, 25% for validation (in hyperparameter tuning) and 25% for test. For each dataset with more than 10,000 patterns, a 10-fold cross validation is developed, so there are $N = 10$ training, validation and test percentages. The rationale is to limit the computational overhead of 500 trials to smaller datasets, using a lighter methodology (10-fold), although statistically less significant, for larger datasets. Each regressor is trained on the training partitions for each combination of its hyperparameter values, and it is tested on its corresponding validation partition. The performance measures used are the root mean square error (RMSE), the squared correlation (R^2) and the mean absolute error (MAE). For each combination of hyperparameter values, the average RMSE over the validation partitions is calculated, and the combination with the lowest average RMSE is selected for testing (quantile regressors as `rqlasso`, `rqnc` and `qrnn` are designed to optimize the quantile error, which is used instead of RMSE). Finally, the regressor is trained on the training partitions using the selected combination of its hyperparameter value and tested on the test partitions. The performance measurements are the RMSE, R^2 and MAE between the true and predicted output values concatenated for the N test sets. Note that the R^2 is calculated using the predicted and true outputs for the test patterns, while it is often used to measure the percentage of variance explained by the regressor on the training patterns. Those regressors which lack tunable hyperparameters are trained on the training partitions and tested on the corresponding test partitions, and the average RMSE, R^2 and MAE over the test partitions are the quality measurements. Some regressors which are specially sensitive to collinear inputs are trained, for each partition, using only those inputs which are not collinear. Although collinear inputs have been removed from the dataset in the initial preprocessing, for certain partitions some inputs in the training set may be collinear despite of being not collinear considering the whole dataset. To avoid the subsequent errors, these inputs are discarded for these regressors. All the inputs and the output are standarized to have zero mean and standard deviation one, using the mean and deviation calculated in each training partition.

We run this collection of 77 regressors over 83 datasets, developing a total of 6,391 experiments, which were developed on a cluster whose nodes are equipped with 64 Intel Xeon E5-2650L processors and 4 GB of RAM memory each processor, although those regressors which required more memory with large data sets were run using several processors and up to 128 GB of RAM memory. Since certain regressors failed for some datasets, we developed a preliminar study to evaluate the datasets according to their size, given by its population, and "difficulty", estimated by the R^2 achieved by the linear regressor (`lm`). We selected `lm` because it is a classical approach which can be considered as a baseline reference for other regressors and it does not require large time nor memory, so it does not fail in any dataset. The Figure 1 plots R^2_{lm} for all the datasets vs. their populations N_p. According this plot, we divided the datasets into four groups: group SD includes 20 datasets with $N_p < 5,000$ and $R^2_{lm} < 0.6$, i.e., small and difficult datasets; group SE includes 23 datasets with $R^2_{lm} \geq 0.6$ and $N_p < 5,000$ (small and easy datasets); group LD with 33 datasets where $R^2_{lm} < 0.6$ and $N_p \geq 5,000$ (large difficult datasets); and group LE with 7 datasets where $R^2_{lm} \geq 0.6$ and $N_p \geq 5,000$ (large easy datasets). Table 8 lists the datasets of each group.

In order to compare the R^2 values achieved by regressors over all the datasets, averaging would weight more those datasets with high R^2, favouring regressors which perform well in easy datasets and biasing the results. In order to do a neutral comparison, the solution is to average over all the

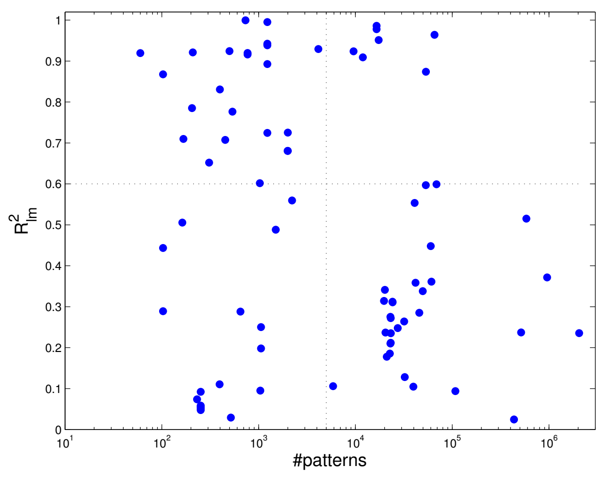

Figure 1: Values of R^2 achieved by `lm` for all the datasets plotted against their populations (in logarithmic scale), dividing datasets in groups small difficult (SD, lower left quarter of the figure), small easy (SE, upper left), large difficult (LD, lower right) and large easy (LE, upper right).

datasets the regressor positions in a ranking sorted by decreasing R^2, instead of directly averaging R^2 values, because the regressor positions belong to the same range for all the datasets. This is done using the Friedman ranking [73] of the R^2 coefficient, which evaluates the position where each regressor is placed, in average over all the datasets, when R^2 is sorted by decreasing values. The R^2 Friedman ranking of the $n = 77$ regressors over the $m = 83$ datasets can be calculated as follows. For each dataset $j = 1, \ldots, m$, the R^2 values of all the regressors are sorted decreasingly. For each regressor $i = 1, \ldots, n$ let be p_{ij} its position in dataset j. The Friedman rank F_i of regressor i is defined as $F_i = \frac{1}{m}\sum_{j=1}^{m} p_{ij}$, i.e., the average position of regressor i over all the sortings of R^2 for the different datasets. For example, a regressor with rank 5 achieves, in average over all the datasets, the 5th highest R^2 coefficient.

A number of run-time errors happened for certain regressors and datasets. There are more errors in large datasets, because some regressor implementations may not be designed to process large amounts of data. When a regressor fails for a dataset (because it overcomes the maximum allowed time of 48 hours, because it requires more than 128 GB of RAM, or due to other reasons), and in order to calculate the Friedman ranking, its R^2 is intended to be zero, while its RMSE and MAE are assigned as:

$$RMSE = \max\left\{\max_{j\in\mathcal{R}}\left[RMSE_j\right], \sqrt{\frac{1}{N}\sum_{i=1}^{N}(t_i-\bar{t}_i)^2}\right\} \tag{1}$$

$$MAE = \max\left\{\max_{j\in\mathcal{R}}\left[RMSE_j\right], \frac{1}{N}\sum_{i=1}^{N}|t_i-\bar{t}|\right\} \tag{2}$$

where $\mathcal{R}$ is the set of regressors which did not fail in that dataset, t_i is the true output for test pattern i and N is the number of test patterns. Besides, denoting by k the test partition to which pattern i belongs, $\bar{t}_i$ is the mean of the true output values over the patterns in the k-th training partition. The rationale behind this is that a regressor which fails behaves as if it would predict the mean of the true output values for all the test patterns, so it should be the last of the list. For some regressors, the errors happen only during tuning for some partitions, which are not considered to calculate the average RMSE corresponding to that combination of hyper-parameter values. When a regressor fails for a given combination of hyper-parameter values and all the partitions, that combination is not selected for testing. When a regressor fails for all the combinations of hyper-parameter values, or when it fails for some test partition, the regressor is considered that fails for that dataset. Overall, the number of experiments where the regressor failed is 1,205 and represents 18.85% of the 6,391 experiments.

Group (#datasets)	Datasets
SD (20): small difficult	airfoil com-crime-unnorm csm1415 forestfires geo-lat geo-long geo-music-lat geo-music-long gps-trajectory park-speech slump slump-flow stock-abs stock-annual stock-excess stock-rel stock-systematic stock-total student-mat student-por
SE (23): small easy	air-quality-CO air-quality-NMHC air-quality-NO2 air-quality-NOx air-quality-O3 automobile auto-mpg bike-day com-crime com-hd compress-stren daily-demand energy-cool energy-heat facebook-metrics gas-dynamic-CO gas-dynamic-methane housing servo slump-comp SML2010 stock-exchange yacht-hydro
LD (33): large difficult	3Droad appliances-energy beijing-pm25 blog-feedback buzz-twitter cuff-less dynamic-features facebook-comment greenhouse-net household-consume KEGG-relation online-news park-motor-UPDRS park-total-UPDRS physico-protein pm25-beijing-dongsi pm25-beijing-dongsihuan pm25-beijing-nongzhanguan pm25-beijing-us-post pm25-chengdu-caotangsi pm25-chengdu-shahepu pm25-chengdu-us-post pm25-guangzhou-5th-middle-school pm25-guangzhou-city-station pm25-guangzhou-us-post pm25-shanghai-jingan pm25-shanghai-us-post pm25-shanghai-xuhui pm25-shenyang-taiyuanji pm25-shenyang-us-post pm25-shenyang-xiaoheyan video-transcode year-prediction
LE (7): large easy	bike-hour combined-cycle cond-turbine CT-slices KEGG-reaction UJ-lat UJ-long

Table 8: Groups of datasets according to its size (small/large) and complexity (easy/difficult).

3.1. Discussion by dataset group

Table 9 reports the 20 best regressors according to the Friedman ranking of R^2, RMSE and MAE for the datasets of **group SD**, which includes 20 **small difficult** datasets. For each regressor in the R^2 ranking, the percentage of datasets where it failed is also reported (column %Error). The last two columns report the regressors which achieved the best R^2 for some dataset and the percentage of datasets where this happened. First of all, `penalized` achieves the first positions in the three rankings, being the best R^2 for 25% of datasets. `ExtraTrees`, `rf` and `kelm` are the following in the

R^2 ranking, although the former achieves much lower positions in RMSE and MAE rankings. Specifically, `extraTrees` achieves the best R^2 in 40% of the datasets, although it fails in 5% of them, so it can be considered less regular as `penalized`. Other good positions in the R^2 ranking are for `qrf` and `bstTree`, followed by `avNNet`, `svr` and `svmRad`, Gaussian process (`gprRad` and `gprPol`), `bagEarth` and `cubist`, which achieves the best R^2 for 10% of datasets.

	R^2			RMSE		MAE		Best R^2	
Pos.	Model	Rank	%Error	Model	Rank	Model	Rank	Model	%Best
1	penalized	8.45	0.0	penalized	9.65	penalized	13.40	extraTrees	40.0
2	extraTrees	13.05	5.0	kelm	13.15	svmRad	13.90	penalized	25.0
3	rf	15.35	5.0	gprPol	14.45	svr	15.70	cubist	10.0
4	kelm	19.15	5.0	bagEarth	17.55	kelm	16.25	brnn	10.0
5	qrf	20.75	0.0	svmRad	18.00	bstTree	19.15	rbf	5.0
6	bstTree	21.00	0.0	cforest	18.65	gprPol	19.25	qrf	5.0
7	avNNet	21.25	5.0	bstTree	19.10	cubist	19.65	bagEarth	5.0
8	svr	22.20	10.0	svr	19.35	bagEarth	19.75	—	—
9	svmRad	23.15	5.0	enet	21.50	cforest	21.45	—	—
10	gprRad	23.20	0.0	BstLm	22.75	qrf	23.35	—	—
11	RRF	24.10	20.0	glmboost	23.80	avNNet	23.85	—	—
12	bagEarth	24.10	0.0	gbm	24.20	gbm	24.35	—	—
13	gprPol	24.35	0.0	foba	24.70	grnn	27.00	—	—
14	gbm	24.60	0.0	bMachine	25.30	gprRad	28.35	—	—
15	cubist	26.20	5.0	grnn	26.25	extraTrees	29.05	—	—
16	ridge	27.85	0.0	spls	27.25	rf	29.15	—	—
17	treebag	29.45	0.0	spikeslab	27.25	BstLm	29.45	—	—
18	foba	29.55	0.0	rf	27.90	treebag	29.50	—	—
19	spls	29.85	0.0	lars	28.35	rqlasso	29.75	—	—
20	lars	30.25	0.0	avNNet	28.90	glmboost	29.95	—	—

Table 9: List of the 20 best regressors according to the Friedman rank of R^2, RMSE and MAE for dataset group SD, with 20 **small difficult** datasets. The last two columns list the regressors which achieve the best R^2 for some dataset, sorted by decreasing number of datasets.

Since this group includes only small datasets, most regressors exhibit low error percentages (i.e., most regressors never or rarely fail on datasets of this group), although some regressors with errors achieve good positions, e.g. `svr` (10% of errors), `RRF` (20%), `extraTrees` and `cubist` (5% each one). Besides, `qrnn` and `nodeHarvest` are very slow and they were shutdown after 48 h. for the 20 datasets of this group. Considering memory errors, `rndGLM` is the regressor which requires more memory, overcoming the memory and time limits in 1 and 5 datasets of this group, respectively.

	R^2			RMSE		MAE		Best R^2	
Pos.	Model	Rank	%Error	Model	Rank	Model	Rank	Model	%Best
1	cubist	6.48	0.0	cubist	6.26	cubist	5.65	cubist	21.7
2	avNNet	10.13	4.3	avNNet	10.04	avNNet	10.96	avNNet	13.0
3	bstTree	12.57	0.0	bstTree	12.39	bstTree	13.70	extraTrees	13.0
4	gbm	12.87	0.0	gbm	12.74	ppr	13.91	gbm	8.7
5	bagEarth	14.57	0.0	bagEarth	14.30	gbm	13.96	penalized	8.7
6	ppr	14.65	0.0	ppr	14.52	bagEarth	15.96	bMachine	8.7
7	bMachine	14.96	4.3	bMachine	15.22	bMachine	16.70	kelm	4.3
8	extraTrees	17.13	8.7	earth	18.35	M5	16.83	M5	4.3
9	earth	18.57	0.0	kelm	18.65	qrf	17.43	rf	4.3
10	kelm	18.70	17.4	extraTrees	18.87	extraTrees	18.00	brnn	4.3
11	rf	19.26	4.3	rf	19.65	kelm	18.74	bagEarth	4.3
12	M5	20.30	0.0	M5	19.87	rf	19.70	bstTree	4.3
13	RRF	22.43	8.7	RRF	22.61	earth	21.00	—	—
14	qrf	22.48	0.0	qrf	23.43	brnn	22.48	—	—
15	brnn	23.65	21.7	brnn	23.61	RRF	22.83	—	—
16	gprPol	24.00	4.3	gprPol	24.22	pcaNNet	24.17	—	—
17	pcaNNet	25.04	0.0	pcaNNet	25.09	gprPol	25.78	—	—
18	dlkeras	27.35	0.0	dlkeras	27.61	rqlasso	26.52	—	—
19	Boruta	27.43	17.4	Boruta	27.78	cforest	27.13	—	—
20	enet	28.52	0.0	enet	28.09	Boruta	27.70	—	—

Table 10: List of the 20 best regressors according to the Friedman rank of R^2, RMSE and MAE over 23 datasets of group SE (**small easy**).

Considering **small easy** datasets (group **SE**, 23 datasets, table 10), the three rankings are even more coherent than for group SD, sharing the first three positions: `cubist`, which achieves the best R^2 for 21.7% of datasets, `avNNet` (the best R^2 for 13% of datasets) and `bstTree`. `Penalized` is not present in this list (although it is the best in 8.7% of datasets), but `gbm` and `bMachine` (which are the bests in 8.7% of datasets), `bagEarth`, `ppr`, `extraTrees` (the best in 13% of datasets), `earth` and `kelm` are in positions 4-10. Other regressors with good results are `rf`, `M5` (the best for 4.3% of datasets), `RRF` and `qrf`.

	R^2			RMSE		MAE		Best R^2	
Pos.	Model	Rank	%Error	Model	Rank	Model	Rank	Model	%Best
1	M5	9.48	0.0	M5	9.39	M5	9.55	extraTrees	48.5
2	cubist	12.39	15.2	gbm	12.61	kknn	12.55	bstTree	12.1
3	gbm	12.48	3.0	cubist	12.70	cubist	12.61	cubist	9.1
4	xgbTree	14.24	6.1	xgbTree	14.15	gbm	13.42	dlkeras	6.1
5	kknn	14.48	12.1	kknn	14.48	xgbTree	14.82	xgbTree	6.1
6	bstTree	15.12	12.1	bstTree	15.27	bstTree	16.00	ppr	3.0
7	blackboost	16.79	0.0	blackboost	17.27	grnn	17.33	kknn	3.0
8	dlkeras	18.36	15.2	pcaNNet	18.33	blackboost	17.70	M5	3.0
9	svr	18.58	27.3	svr	18.45	svr	18.76	rf	3.0
10	pcaNNet	18.76	0.0	dlkeras	18.67	pcaNNet	18.82	qrf	3.0
11	grnn	19.70	18.2	ppr	19.48	dlkeras	19.18	bMachine	3.0
12	ppr	19.73	3.0	grnn	19.52	ppr	19.58	—	—
13	qrf	21.33	27.3	qrf	21.27	qrf	20.30	—	—
14	svmRad	21.88	24.2	svmRad	21.79	svmRad	20.58	—	—
15	earth	22.52	0.0	earth	22.21	extraTrees	22.33	—	—
16	extraTrees	23.03	27.3	bag	22.91	bag	22.61	—	—
17	bag	23.03	15.2	avNNet	23.42	earth	22.88	—	—
18	avNNet	23.52	21.2	extraTrees	23.91	avNNet	23.64	—	—
19	bMachine	24.76	24.2	bMachine	24.64	bMachine	25.24	—	—
20	cforest	25.79	27.3	cforest	25.91	rpart	26.00	—	—

Table 11: List of the 20 best regressors according to the Friedman rank of R^2, RMSE and MAE for the 33 datasets of group LD (**large difficult**).

In **large difficult** datasets (group **LD**, 33 datasets, table 11), the `M5` achieves the first positions in the three rankings (although it achieves the best R^2 only in 3% of datasets), followed by `cubist` (which achieves the best R^2 and errors in 9.1% and 15.2% of datasets, respectively) and `gbm`. Other regressors with good performance are `xgbTree`, `knn`, `bstTree`, `blackboost`, `dlkeras` (15.2% of errors), `svr` (with errors in 27.3% of datasets) and `pcaNNet`. The high error frequency of several regressors (either by overcoming limits on memory or time) is due to the large size of datasets in this group. `ExtraTrees` also overcomes the maximum time in 27.3% of datasets and, as in groups SD and SE, it achieves the best R^2 for more datasets (48.5%). Specifically, it achieves the highest R^2 for 13 of the 16 datasets created from the original *PM2.5 Data 5 Chinese Cities* dataset in the UCI repository. In these datasets `svr` and `kelm` were run with a lower number of hyperparameter values ({ 0.125, 0.5, 1, 4, 16 } and { 0.00391 0.01562 0.125 1 4 } for C and γ, respectively), in order to avoid overcoming the maximum run time. Regressors `wkpls`, `gprPol`, `krlsRad`, `rvmRad`, `SBC` and `qrnn` failed for the 33 datasets of this group.

Pos.	Model	Rank	Pos.	Model	Rank	Pos.	Model	Rank	Pos.	Model	Rank
1	M5	8.1	6	pcaNNet	15.4	11	rpart	23.5	16	avNNet	26.6
2	gbm	9.8	7	earth	18.6	12	treebag	24.1	17	svmRad	26.8
3	blackboost	10.9	8	kknn	19.4	13	ctree2	25.1	18	enet	26.8
4	xgbTree	14.6	9	bstTree	19.8	14	elm	26.2	19	bag	26.9
5	ppr	14.8	10	cubist	20.4	15	svr	26.2	20	dlkeras	27.9

Table 12: Friedman rank of R^2 (first 20 regressors) of group LD (**large difficult** datasets) discarding PM2.5 Data Chinese Cities datasets.

The PM2.5 Data 5 Chinese Cities datasets represent almost the half of the 33 datasets in this group. Since this fact might bias the results, we calculated the R^2 Friedman rank discarding these 16 datasets (Table 12). In this case, the best regressor is `M5` again, `cubist` descends to the 10th position, replaced by `gbm` and followed by `blackboost`, `xgbTree` and `ppr`, while `extraTrees` leaves the top-20.

The rankings of group LE (**large easy**) is very similar to group LD: the `M5` achieves again the best position in the rankings of R^2, RMSE and MAE, followed by the same regressors as the previous group: `cubist` (which achieves the best R^2 in 42.9%, and errors in 14.3%, of the datasets), `gbm`, `bag`, `bstTree`, `blackboost` and `pcaNNet`. In this group, `extraTrees` only achieves the best R^2 in 1 dataset, which represents 14.3%, and achieves errors in 57.1% of datasets. Since the datasets are easy, the `lm` also achieves a good position

	R^2			RMSE		MAE		Best R^2	
Pos.	Model	Rank	%Error	Model	Rank	Model	Rank	Model	%Best
1	M5	6.57	0.0	M5	6.57	M5	5.14	cubist	42.9
2	cubist	10.43	14.3	cubist	10.43	cubist	10.29	extraTrees	14.3
3	gbm	11.43	0.0	gbm	11.43	gbm	12.43	rf	14.3
4	bag	14.57	0.0	bag	14.57	bag	13.43	brnn	14.3
5	bstTree	15.29	14.3	bstTree	15.29	blackboost	15.86	dlkeras	14.3
6	blackboost	15.43	0.0	blackboost	15.43	pcaNNet	17.00	—	—
7	pcaNNet	16.00	0.0	pcaNNet	15.71	bstTree	17.29	—	—
8	xgbTree	19.57	14.3	xgbTree	19.43	rlm	20.00	—	—
9	lm	21.43	0.0	earth	21.29	xgbTree	21.00	—	—
10	earth	21.86	0.0	kknn	22.14	kknn	21.57	—	—
11	bayesglm	21.86	0.0	lm	22.57	dlkeras	22.14	—	—
12	kknn	22.14	14.3	bayesglm	22.57	earth	23.00	—	—
13	avNNet	23.14	42.9	avNNet	23.14	avNNet	23.43	—	—
14	dlkeras	23.43	14.3	dlkeras	23.71	lm	23.71	—	—
15	svr	24.43	42.9	lasso	24.43	svr	25.29	—	—
16	lasso	24.71	0.0	svr	24.71	gam	25.43	—	—
17	spikeslab	25.29	0.0	enet	25.00	bayesglm	25.43	—	—
18	bagEarth	25.57	14.3	spikeslab	25.29	spikeslab	25.43	—	—
19	enet	26.14	14.3	bagEarth	25.43	lasso	25.71	—	—
20	gam	26.14	0.0	gam	25.86	lars	26.14	—	—

Table 13: List of the 20 best regressors according to the Friedman rank of R^2, RMSE and MAE for the 7 **large easy** datasets (group LE).

(9th). The regressors which fail in the 7 datasets of this group are `kelm`, `wkpls`, `gprPol`, `krlsRad`, `rvmRadial`, `SBC`, `nodeHarvest` and `qrnn`.

3.2. Global discussion

We also developed an analysis considering all the datasets together. Table 14 reports the 20 best regressors according to the Friedman rankings for R^2, RMSE and MAE over all the datasets, alongside with the percentage of datasets with errors for the 20 best regressors according to R^2 (column %Error) and the percentage of datasets where each regressor achieves the best R^2 (column %Best). The global results confirm the conclusions over the four dataset groups: `cubist` is globally the best regressor on the three rankings (although it achieves errors for 8.4% of datasets), followed by `gbm` and `bstTree`. The difference is higher in terms of MAE (ranks 12.18 and

	R^2			RMSE		MAE		Best R^2	
Pos.	Model	Rank	%Error	Model	Rank	Model	Rank	Model	%Best
1	cubist	13.92	8.4	cubist	14.96	cubist	12.18	extraTrees	33.7
2	gbm	15.42	1.2	gbm	15.34	gbm	16.12	cubist	15.7
3	bstTree	15.84	6.0	bstTree	15.40	bstTree	16.23	penalized	8.4
4	M5	18.20	0.0	M5	17.20	M5	16.36	bstTree	6.0
5	avNNet	19.23	14.5	avNNet	21.01	avNNet	20.16	brnn	4.8
6	extraTrees	19.61	19.3	bagEarth	22.46	qrf	21.11	avNNet	3.6
7	qrf	22.41	14.5	bMachine	22.48	svr	23.08	rf	3.6
8	pcaNNet	23.49	0.0	svr	23.54	extraTrees	23.41	bMachine	3.6
9	rf	23.82	24.1	earth	23.99	bagEarth	23.57	dlkeras	3.6
10	bMachine	23.83	15.7	blackboost	24.39	pcaNNet	24.29	gbm	2.4
11	bagEarth	24.14	7.2	extraTrees	24.71	bMachine	24.45	M5	2.4
12	svr	24.17	27.7	pcaNNet	24.83	ppr	24.76	qrf	2.4
13	ppr	24.57	4.8	ppr	26.06	kknn	25.07	bagEarth	2.4
14	earth	25.52	0.0	kknn	26.46	earth	25.40	xgbTree	2.4
15	blackboost	25.69	0.0	qrf	26.84	grnn	25.92	kelm	1.2
16	kknn	26.24	6.0	rf	27.01	svmRad	26.28	ppr	1.2
17	penalized	27.70	12.0	grnn	27.37	blackboost	26.92	kknn	1.2
18	dlkeras	28.07	7.2	enet	27.41	bag	27.27	rbf	1.2
19	svmRad	29.14	28.9	cforest	27.53	cforest	27.28	—	—
20	grnn	29.61	9.6	bag	27.64	rf	27.57	—	—

Table 14: List of the 20 best regressors according to the Friedman rank of R^2, RMSE and MAE over all the datasets.

16.12 for `cubist` and `gbm`, respectively) than in terms of R^2 or RMSE. `Cubist` is also the second regressor which achieves more often the best R^2 (in 15.7% of datasets) after `extraTrees` (33.7%), whose position is however much lower (6, 11 and 8 in the R^2, RMSE and MAE rankings, respectively), achieving errors for 19.3% of datasets. The `M5` achieves position 4 in the three rankings, but it never fails, so its difference with `cubist` is caused by lower performance in datasets where `cubist` does not fail. Globally, the best neural network is `avNNet` (position 5). Other regressors in the top-10 of some rankings are `qrf`, `pcaNNet`, `rf` (with 24.1% of errors), `bMachine`, `bagEarth`, `svr` (27.7% of errors), `earth` and `blackboost`. `Penalized`, which is the best regressor for 8.4% of datasets, achieves position 17 in the R^2 ranking, with 12% of errors. The `lm` falls outside this table (positions 33–34).

Pos.	Model	Rank	Pos.	Model	Rank	Pos.	Model	Rank	Pos.	Model	Rank
1	cubist	11.1	6	ppr	19.5	11	qrf	22.1	16	kknn	24.8
2	gbm	12.6	7	pcaNNet	20.4	12	bMachine	23.0	17	rf	26.0
3	M5	13.5	8	earth	20.6	13	bagEarth	23.6	18	bag	26.7
4	bstTree	14.1	9	blackboost	21.0	14	dlkeras	23.9	19	grnn	28.0
5	avNNet	18.8	10	extraTrees	21.1	15	svr	24.6	20	cforest	29.0

Table 15: List of the 20 best regressors according to the Friedman rank of R^2, RMSE and MAE over all the datasets excepting PM2.5 Data 5 Chinese Cities.

Despite its high number of errors, `extraTrees` achieves a good position because it achieves the best R^2 for the majority of the thirteen PM2.5 Data 5 Chinese Cities datasets. In order to confirm that this fact does not bias the global results, we created an alternative ranking discarding these datasets (see Table 15). This alternative rank is rather similar to the previous one, being `cubist`, `gbm`, `M5` and `bstTree` the first regressors, but `extraTrees` and `rf` move from positions 6 and 9 to 10 and 17, respectively.

We evaluated the statistical significance of the differences in R^2 among regressors with several tests. A Friedman test [74], implemented using the `stats` package, comparing all the regressors gives a p-value of $1.8{\cdot}10^{-45} < 0.05$, which means that the difference among them is statistically significant. Table 16 reports the results of several statistical tests [75] developed to compare the globally best regressor (`cubist`) and the 20 bests regressors in terms of R^2. We used: 1) the student T-test, with the Matlab `ttest` function: according to [75], since the number of datasets (83) is higher than 30, the requirement of normal distributions for the R^2 values is not necessary; 2) the two-sample student T-test, with the Matlab `ttest2` function; 3) the Wilcoxon rank sum test [76], with the Matlab `ranksum` function; 4) the sign test (Matlab `signtest` function); and 5) the Post-Hoc Friedman-Nemenyi

Pos.	Model	T-test	T-Test-2	Wilcoxon	SignTest	Post-Hoc
2	gbm	0.825	0.921	0.593	0.001*	0.152
3	bstTree	0.215	0.789	0.409	0.000*	0.971
4	M5	0.781	0.902	0.658	0.000*	0.005*
5	avNNet	0.000*	0.068	0.075	0.000*	0.001*
6	extraTrees	0.001*	0.083	0.124	0.909	0.550
7	qrf	0.034*	0.383	0.285	0.006*	0.000*
8	pcaNNet	0.034*	0.294	0.273	0.000*	0.050*
9	rf	0.001*	0.064	0.048*	0.002*	0.000*
10	bMachine	0.000*	0.109	0.051	0.000*	0.000*
11	bagEarth	0.000*	0.074	0.086	0.000*	0.000*
12	svr	0.000*	0.005*	0.002*	0.000*	0.142
13	ppr	0.004*	0.125	0.143	0.000*	0.000*
14	earth	0.013*	0.204	0.188	0.000*	0.000*
15	blackboost	0.045*	0.303	0.194	0.000*	0.000*
16	kknn	0.003*	0.452	0.097	0.000*	0.000*
17	penalized	0.000*	0.000*	0.001*	0.000*	0.000*
18	dlkeras	0.019*	0.343	0.133	0.000*	0.000*
19	svmRad	0.000*	0.000*	0.000*	0.000*	0.000*
20	grnn	0.000*	0.196	0.037*	0.000*	0.000*

Table 16: p-values achieved by the student-T, two-sample student-T, Wilcoxon ranksum, sign and Post-Hoc Friedman-Nemenyi tests comparing the R^2 of the globally best regressor (`cubist`) and the remaining regressors in the top-20. The asterisks label regressors where the comparison is statistically significant (p <0.05).

test (`PMCMR` [77] R package). The student T-test gives significant differences, labeled as an asterisk (*), except for the first three regressors, while the two-sample student T and Wilcoxon tests only label few regressors as statistically different, including `svr`, `penalized` and `svmRad`. The sign test, which counts the number of datasets where each regresor achieves the best R^2, labels all the regressors as statistically different to `cubist` excepting `extraTrees`. Finally, the Post-Hoc Friedman-Nemenyi test, which develops a comparison of multiple regressors, identifies as statistically significant the differences with all the regressors excepting `gbm`, `bstTree` `extraTrees` and `svr`.

Figure 2 plots R^2_{best} against R^2_{cubist} (left panel) and R^2_{M5} (right panel)

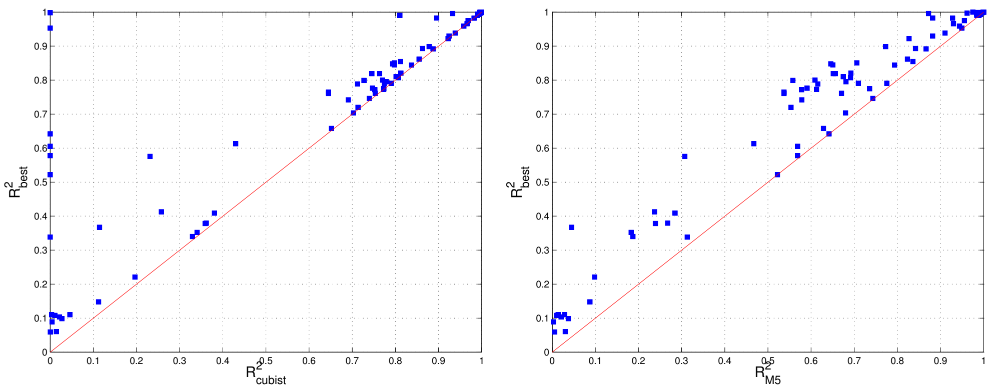

Figure 2: Value of R^2_{best} against R^2 achieved by `cubist` (left panel) and `M5` (right panel) for all the datasets.

for each dataset (`M5` is the first regressor in the ranking which never fails). `Cubist` is near the best R^2 for all the points above 0.6, but its R^2 is almost zero for more than 10 points, due mainly to errors, which are on the vertical axis. However, all the points are near the red line for `M5`, which never fails, whose R^2 is near zero only for those datasets whose best R^2 is already almost zero, so the probability that `M5` achieves R^2 near R^2_{best} is much higher.

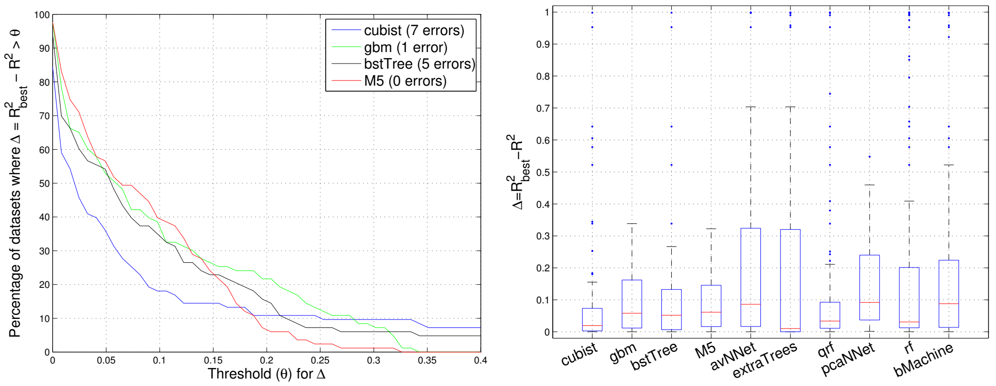

Figure 3: Left panel: percentage of datasets where the difference $\Delta = R^2_{best} - R^2$, with R^2 achieved by `cubist`, `gbm`, `bstTree` and `M5`, overcomes a given threshold θ. Right panel: boxplots of the differences $R^2_{best} - R^2$ for the 10 best regressors.

Figure 3 (left panel) plots the percentage of datasets where the difference $\Delta = R^2_{best} - R^2$ overcomes a threshold θ, where R^2 is the value achieved by the first 4 regressors in Table 14: `cubist`, `gbm`, `bstTree` and `M5`. The regressor is better when the line is lower, because the percentage of datasets where $\Delta > \theta$ is lower. For low θ values, the lines follow the order `cubist` $<$ `bstTree` $<$ `gbm` $<$ `M5`, but the high error frequency of `cubist` and `bstTree` (7 and 5, respectively) cause that blue and black lines fall to zero for $\theta > 0.4$ (outside the plot), while green and red lines (`gbm` and `M5`, respectively) fall to zero at 0.322 and 0.338. Note that `gbm` fails (achieving $R^2 = 0$) only for dataset *year-prediction*, for which by chance R^2_{best} is low (0.338), so $\Delta = 0.338$ for this dataset and the green curve falls to zero at $\theta = 0.338$. If the R^2_{best} were higher, the green line would continue to the right without falling to zero, similarly to blue and black lines. The right panel of Figure 3 shows the boxplots of the differences $R^2_{best} - R^2$ for the first 10 regressors in the global ranking. The blue boxes report the 25% and 75% quantiles, while the red line inside the box is the median, and the blue points outside the box are the outliers. Although `cubist`, `gbm`, `bstTree`, `extraTrees` and `rf` exhibit low medians, all the regressors have several outliers (caused by datasets where they fail) with high Δ values, excepting `M5`, the only one which guarantees low Δ values (below 0.322) for all the datasets.

Figure 4 (left panel) plots R^2_{best} and the R^2 achieved by `M5` and `gbm`. `M5` is near R^2_{best} more often tan `gbm`, and in several cases `gbm` is clearly below `M5`, but the former rarely outperforms the latter, and in these cases with lower difference. The right panel shows the histogram of the difference $R^2_{M5} - R^2_{gbm}$: its values are positive (i.e., `M5` outperforms `gbm`) for 52.4% of the datasets,

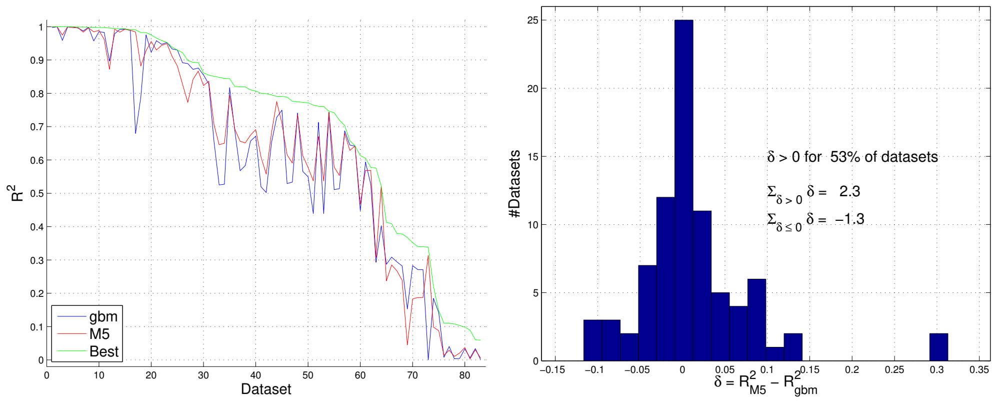

Figure 4: Left panel: R^2_{gbm} (blue), R^2_{M5} (red) and R^2_{best} (green) for each dataset, sorted by decreasing values of R^2_{gbm} values. Right panel: histogram of the difference $R^2_{M5} - R^2_{gbm}$.

and when they are positive, they are higher (in absolute value) than when they are negative, so the sum of positive Δ values (2.3) outperforms the sum of negative values (-1.3). This shows that overall `M5` outperforms `gbm`, although the latter is higher in the global ranking (Table 14). Remember that `cubist`, `gbm` and `bstTree` fail for some datasets, while `M5` never fails.

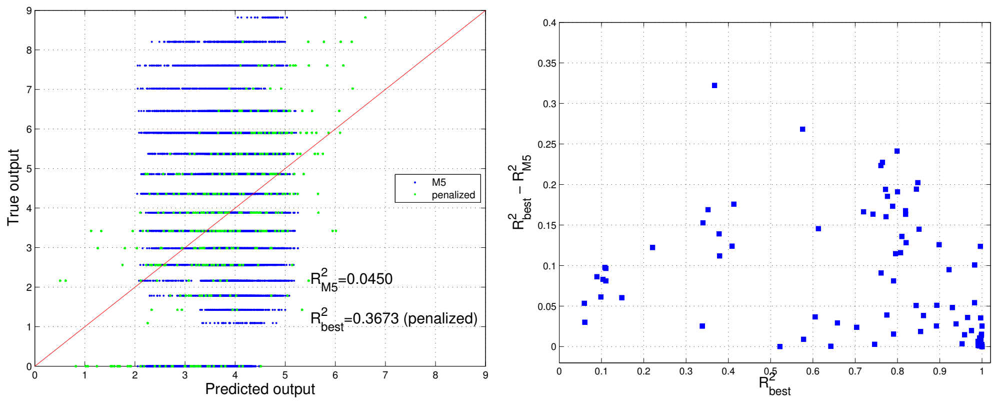

Figure 5: Left panel: true against predicted output for `M5` (blue points) and the best regressor (`penalized`) for dataset *student-mat*. Right panel: difference $R^2_{best} - R^2_{M5}$ against R^2_{best} for all the datasets.

In the left panel of Figure 4 above, the maximum difference $R^2_{best} - R^2_{M5}$ is 0.322 in dataset *student-mat*, whose output is discrete with more than 10 values (see the left panel of Figure 5), so the dataset was not excluded. The low $R^2_{best} = 0.3673$ for this dataset means that no regressor, and not only `M5`, fits accurately the true output, and both blue and green points fit equally bad the red line. The right panel of the same figure plots the difference $R^2_{best} - R^2_{M5}$ against R^2_{best}. This difference is low for all the datasets (note that the vertical scale is 0–0.4), being below 0.2 (resp. 0.1) for 92.8% (resp. 60.2%) of the datasets.

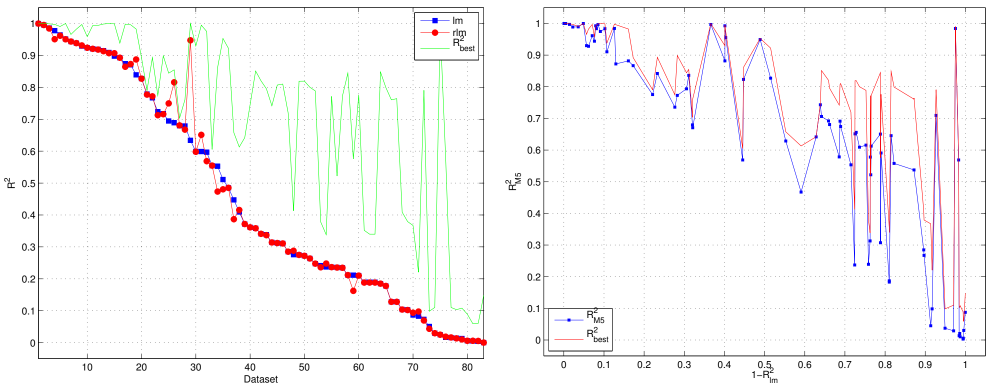

Figure 6: Left: values of R^2_{lm}, R^2_{rlm} and R^2_{best}, sorted by decreasing R^2_{lm}. Right: R^2_{M5} and R^2_{best} against $1 - R^2_{lm}$.

The left panel of Figure 6 compares R^2_{lm}, R^2_{rlm} and R^2_{best} in all datasets. Since both regressors only differ in the robustness against outliners, the difference between them identifies those datasets with outliers. This difference overcomes 0.05 only in 6 datasets and its highest value is 0.31, so that dataset outliers are few and not very relevant. In order to study the behavior of `M5` with the dataset complexity, the right panel shows R^2_{best} and R^2_{M5} against $1 - R^2_{lm}$, which measures the difficulty of the regression problem. The difference $R^2_{best} - R^2_{M5}$, instead of raising with $1 - R^2_{lm}$, achieves the highest values for $0.65 < 1 - R^2_{lm} < 0.9$. However, in the most difficult datasets, where $1 - R^2_{lm} > 0.9$, the R^2_{M5} follows very well R^2_{best}, so `M5` performs well even for hard datasets.

Small difficult		Small easy		Large difficult		Large easy	
Family-model	Pos.	Family-model	Pos.	Family-model	Pos.	Family-model	Pos.
PLM-penalized	1	RGR-cubist	1	RGR-M5	1	RGR-M5	1
RF-extraTrees	2	NET-avNNet	2	BST-gbm	3	BST-gbm	3
NET-kelm	4	BST-bstTree	3	NN-kknn	5	BAG-bag	4
BST-bstTree	6	BAG-bagEarth	5	DL-dlkeras	8	NET-pcaNNet	7
SVR-svr	8	PRJ-ppr	6	SVR-svr	9	LR-lm	9
SGP-gprRad	10	BYM-bMachine	7	NET-pcaNNet	10	AM-earth	10
		RF-extraTrees	8				
		AM-earth	9				

Table 17: Best regressor of each family within the 10 best positions in the R^2 Friedman ranking for each dataset group.

3.3. Discussion by family of regressors

It is interesting to analyze the behavior of the best regressor of each family. Table 17 reports the families with regressors in the top-10 of the R^2 ranking for each dataset group. The boosting family (BST, with regressors `bstTree` and `gbm`) and neural networks (NET, regressors `kelm`, `avNNet` and `pcaNNet`) families, are present in all the groups, while regression rules (RGR), with regressors `cubist` and `M5`, achieves the first position in three of four groups (small easy, large difficult and easy), and penalized linear regression (PLM) achieves the first position (`penalized`) in small difficult datasets. Bagging (BAG, with regressors `bag` and `bagEarth`) and support vector regression (SVR, `svr`) are present in two groups, while RF (`extraTrees`), projection methods (PRJ, `ppr`), Gaussian processes (SGP, `gprRad`), nearest neighbors

(NN, `kknn`), deep learning (DL, `dlkeras`) and linear regression (LR, `lm` and `rlm`) are only present in just one group.

Family	Best	Pos.	Family	Best	Pos.
Regression rules	cubist	1	Nearest neighbors	kknn	16
Boosting	gbm	2	Penalized linear models	penalized	17
Neural networks	avNNet	5	Deep learning	dlkeras	18
Random forests	extraTrees	6	Ridge	foba	27
Bayesian models	bMachine	10	Lasso	lars	28
Bagging	bagEarth	11	Linear regresion	lm	33
Support vector regression	svr	12	Regression trees	ctree2	37
Projection methods	ppr	13	Gaussian processes	gprPol	50
Generalized additive models	earth	14	Quantile regression	rqlasso	56

Table 18: Best regressor of each family and position in the global ranking.

Considering the global ranking, Table 18 reports the families, sorted by the position of their best regressors in Table 14. Only regression rules, boosting and neural networks are in the top-5, followed by random forests and Bayesian models with positions below 10. Most of the remaining families have best regressors which outperform `lm` (position 33), while regression trees, Gaussian processes and quantile regression achieve positions even higher.

3.4. Best result for each dataset

The green line of Figure 4 above shows R^2_{best} for the 83 datasets. For 50% of them $R^2_{best} > 0.8$, so that at least some regressor was able to predict correctly the output. However, $R^2_{best} < 0.6$ for 22 datasets, which represents 26.5%, while $R^2_{best} < 0.2$ for 9 datasets (10.8%). Some datasets are really hard, e.g. *stock-abs*, where $R^2_{best} = 0.059$. Figure 7 plots the best R^2, alongside with R^2 achieved by the best regressor and by `lm` for each dataset group. In group SD (upper left panel), the best regressor (`penalized`) is near to the best R^2 except for datasets *airfoil*, *gps-trajectory*, *slump-flow* and

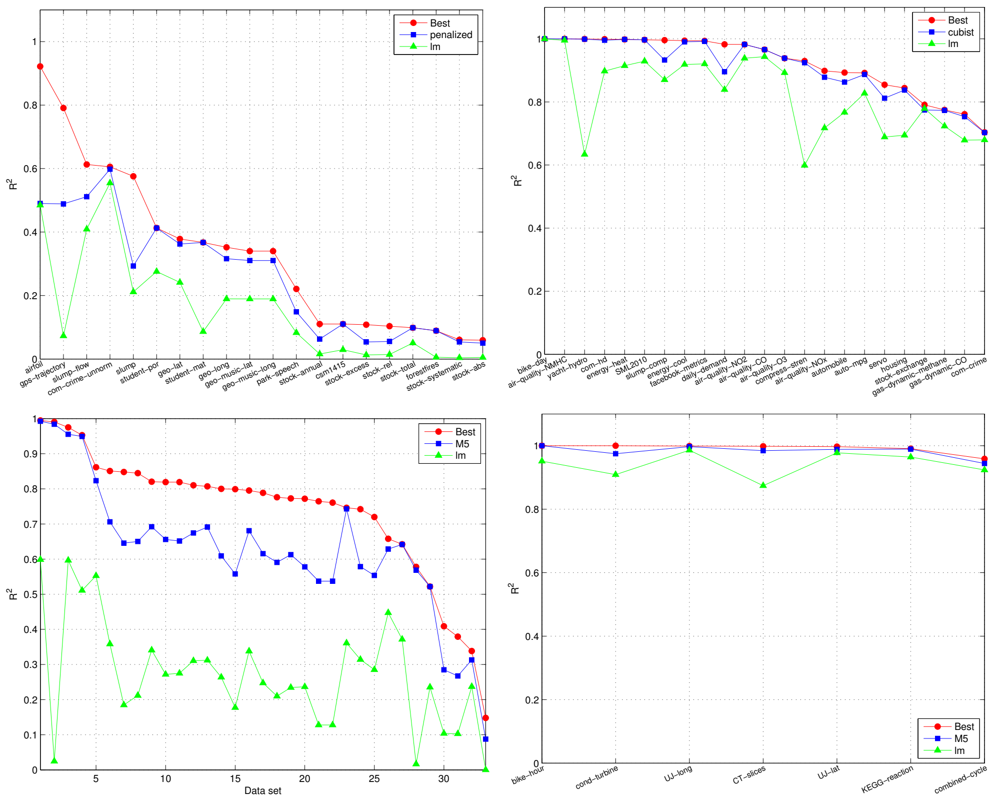

Figure 7: Best R^2 (in red), R^2 achieved by the regressor with the best R^2 Friedman rank (in blue), and R^2 achieved by `lm` (in green) for dataset groups SD (upper left panel), SE (upper right), LD (lower left) and LE (lower right).

slump. Since this group includes small difficult datasets, the R^2 of `lm` is always below 0.6, but for the first five datasets some regressor achieves higher R^2 values. The `penalized` is also better than `lm` for all datasets, although the difference is low for datasets after *geo-music-long*. For group small easy (SE, upper right panel), the R^2 values of `lm` are higher, but the best regressor (`cubist`) is always very near to the best R^2 with some difference with respect to `lm`. In the large difficult group (LD, lower left panel), the `lm` values are very low and the best regressor (`M5`) is far from `lm`, following the best R^2

very closely for 12 of 33 datasets with a margin of 0.2-0.4 for the remaining 21 datasets. Finally, in group LE (large easy datasets, lower right panel) the `lm` is already near the best R^2, although the best regressor (`M5` again) always achieves the best R^2.

3.5. Discussion by elapsed times and memory consumption

We studied the memory and time required by each regressor over all the datasets. Table 19 reports the information of the 20 best regressors according to the R^2 Friedman rank in each column: `%Best` reports the percentage of datasets where they achieve the best R^2; `%Error` reports the percentage of datasets where they failed; `%ME` reports the percentage of memory errors caused by overcoming the largest allowed memory (128 GB); `Datasets/mem` reports the number of datasets run on the memory queues with $\{2^i\}_{i=1}^6$ GB.

In order to measure the time required by each regressor, the time spent in hyper-parameter tuning is discarded to avoid biasing caused by differences among regressors in the number of hyper-parameters and hyper-parameter values. The column `%TE` reports the percentage of time errors, i.e., datasets where the regressor overcomes the maximum allowed time (48 h.). Although it may surprise that some regressors are not able to finish within 48 h., we must consider the size of some datasets (more than 2 millions of patterns, up to 640 inputs) and the high number of trials (500) for some datasets. Generally, high values in the %TE column happen with slow regressors, specially for large datasets. Since some regressors fail but do not overcome the allowed memory nor time, the sum of columns %ME and %TE is not always equal to column %Error, e.g. `nnls` has no memory nor time errors, but %Error is 4.8%. The column `Time` reports the time (in sec.) spent by the regressor

						#Datasets/mem(GB)		
Pos.	Model	Rank	%Best	%Error	%ME	2-4-8-16-32-64-128	%TE	Time
1	cubist	13.92	15.7	8.4		68-6-8-1-0-0-0	8.4	2.47
2	gbm	15.42	2.4	1.2		78-3-2-0-0-0-0	1.2	1.46
3	bstTree	15.84	6.0	6.0		74-5-3-1-0-0-0	6.0	3.84
4	M5	18.20	2.4			0-36-29-18-0-0-0		1.32
5	avNNet	19.23	3.6	14.5		77-3-2-0-0-1-0	14.5	3.20
6	extraTrees	19.61	33.7	19.3		72-9-2-0-0-0-0	20.5	3.62
7	qrf	22.41	2.4	14.5		62-12-4-1-3-1-0	14.5	3.99
8	pcaNNet	23.49				77-3-3-0-0-0-0		1.36
9	rf	23.82	3.6	24.1		69-6-2-0-5-1-0	24.1	3.05
10	bMachine	23.83	3.6	15.7		54-27-2-0-0-0-0	16.8	12.10
11	bagEarth	24.14	2.4	7.2		76-1-4-1-1-0-0	7.2	2.21
12	svr	24.17		27.7		78-5-0-0-0-0-0	27.7	172800
13	ppr	24.57	1.2	4.8		78-3-2-0-0-0-0	4.8	1.24
14	earth	25.52				77-4-2-0-0-0-0		1.46
15	blackboost	25.69				75-1-3-0-3-0-1		1.48
16	kknn	26.24	1.2	6.0		80-2-1-0-0-0-0	6.0	2.41
17	penalized	27.70	8.4	12.0		77-4-2-0-0-0-0	12.0	1.54
18	dlkeras	28.07	3.6	7.2		83-0-0-0-0-0-0	7.2	6.17
19	svmRad	29.14		28.9		77-2-3-0-0-1-0	28.9	172800
20	grnn	29.61		9.6	6.0	47-13-6-5-5-1-1	3.6	0.22
28	lars	33.16				79-2-2-0-0-0-0		0.03
77	qrnn	77.00		100.0		77-4-2-0-0-0-0	100.0	172800
63	rndGLM	51.80		51.8	44.6	0-0-0-12-22-10-2	7.2	2.54

Table 19: List of the 20 first regressors sorted by increasing R^2 Friedman rank, with the percentage of datasets where each regressor achieves the best R^2 (column %Best), percentage datasets with of errors (column %Error), percentage of memory errors (column %ME), number of datasets for each memory size (column #Datasets/mem), percentage of time errors (%TE) and training+test time (in sec.) spent for dataset *compress-stren* (column Time). Empty cells correspond to zero values.

for a training+testing trial on dataset *compress-stren*, whose size might be considered "standard": 1,030 patterns and 8 inputs. The time is set to the maximum allowed time for regressors with errors in this dataset.

Comparing `cubist`, `gbm`, `bstTree` and `M5` in terms of column %Best, `cubist` achieves often the best R^2 (15.7% of datasets) followed by `bstTree` (6%) while `gbm` and `M5` tie (2.4%). `Cubist` and `bstTree` fail in 8.4% and 6%

of datasets, respectively, while `gbm` fails less (the three overcome the allowed time) and `M5` never fails. None of them overcomes the memory limits, but `M5` requires more memory (4-16 GB), while the others require 2-8 GB. Finally, `M5` and `gbm` are faster (1.32 and 1.46 s./trial, respectively), while `bstTree` and `cubist` spend about 2-4 s. The `avNNet` and `extraTrees` spend about 3-4 s. but the former requires less memory (2 GB for 77 of 83 datasets)[8]. Among the remaining regressors, `pcaNNet` never fails, is very fast (1.36 s.) and requires few memory (2 GB for 77 datasets), while `bMachine` is slower (12.1 s.) and requires more memory (4 GB for 27 datasets). The `rf` is faster (3.05 s.) with memory very variable with the dataset size (69 datasets with 2 GB but 1 with 64 GB). Among the remaining regressors, `svr` and `svmRad` are are very slow with time errors in 28.9% of datasets, while `grnn`, `ppr`, `earth` and `blackboost` are fast (between 0.22 to 1.48 s.). However, `grnn` has time errors in 3.6% of datasets, requiring memory from 2 to 64 GB depending on the dataset with memory errors in 6% of datasets. Most regressors in positions 10–20 require few memory, and `dlkeras` requires the lowest memory (2 GB for all datasets), similar to `kknn`, although with time errors in 7.2% and 6% of datasets. To have time and memory references, the last three lines report the fastest and slowest regressors (`lars` and `qrnn`, respectively) in the *compress-stren* dataset, and the regressor which requires the largest memory (`rndGLM`). Considering times, `lars` spends 0.03 s. being 26 times faster than `M5` (the fastest regressor in the top-5), while `qrnn` is shutdown after 48 h. in all the datasets, being 130,910 times slower than `M5`. With respect to memory, `gbm`

[8]Both `extraTrees` and `bartMachine` use Java and by technical reasons their memory was limited to 8 GB.

and `bstTree` require only slightly more memory than `dlkeras` (2 GB for more than 74 datasets), while `rndGLM` always requires more than 16 GB with memory errors in 45.8% of the datasets.

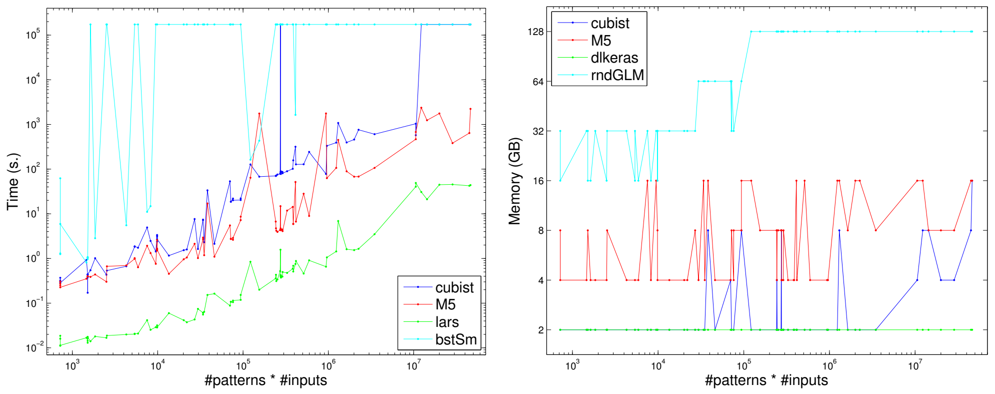

Figure 8: Left: times (in sec.) per trial spent by the best-performing (`cubist` and `M5`) and by the fastest and slowest regressors (`lars` and `bstSm`, respectively). Right: memory (in GB) required by `cubist` and `M5`, and by the regressors with least and most memory requirements (`dlkeras` and `rndGLM`, respectively). Both plotted against the product `#patterns`·`#inputs`.

Figure 8 (left panel) plots, in logarithmic scale, the times spent for each dataset by `cubist` and `M5`, alongside with `lars` and `bstSm`, which are fastest and slowest regressors, respectively, for comparative purposes (`qrnn` is even slower than `bstSm`, but the former overcomes the allowed time in all the datasets so it is replaced by `bstSm`). The times are plotted against the product `#patterns`·`#inputs` of the dataset, which measures its size. `Lars` is one order of magnitude below `M5` and `cubist`, which are similar for small datasets, but the difference grows with the dataset size. In largest datasets, `cubist` is almost two orders of magnitude slower than `M5`, overcoming the allowed time (48 h. or 172,800 s. $\sim 2 \cdot 10^5$ s.). Finally, `bstSm` is 2-3 orders slower than `lars` for small datasets, but it already overcomes the time limit for some small, most medium and all large datasets (overall, for the 78.5% of datasets). Other slow regressors are `xgbTree` and `xgbLinear`, `nodeHarvest`, `krlsRad` and `SBC`, with average times between 20,000 and 300,000 s. and time errors for 50-85% of datasets.

Considering memory, the right panel of Figure 8 plots `cubist` and `M5`, with `dlkeras` and `rndGLM`, which exhibit the lowest and highest memory requirements, against the product `#patterns`·`#inputs`. The `dlkeras` spends 2 GB for all the datasets, while `cubist` uses 2 GB excepting some medium and the 9 largest datasets. However, `M5` requires more memory: 4, 8 and 16 GB for 36, 29 and 18 datasets, respectively. Comparatively, `rndGLM` requires 16, 32, 64 and 128 GB in 12, 22, 10 and 1 datasets, respectively, overcoming 128 GB in 39 datasets (45.8%). Other regressors with high memory requirements are `gprLin`, `gprPol` and `gprRad`, `rvmRad`, `krlsRad`, `wkpls`, `kelm` and `grnn`, with memory errors in 6-10% of datasets.

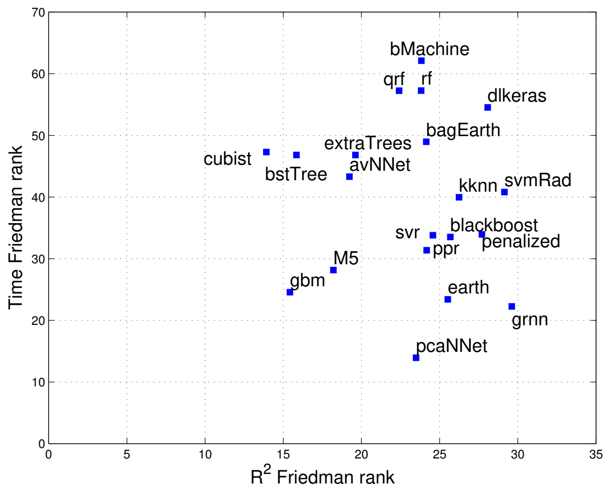

Figure 9: Friedman rank of the time (vertical axis) against the Friedman rank of R^2 (horizontal axis) for the 20 best regressors in Table 19.

Figure 9 plots the Friedman ranks of R^2 and time (horizontal and vertical axis, respectively) for the best 20 regressors. `Cubist` and `pcaNNet` achieve the lowest R^2 and time ranks, respectively, but the best trade-off between R^2 and time is achieved by `gbm` and `M5`. In fact, `cubist` is only slightly better than `gbm` according R^2, but it is much slower. Other regressors with good R^2 are `bstTree` (R^2 similar to `gbm`, but much slower), `avNNet` and `extraTrees`, but they are also slow. The following regressors according R^2 rank are `rf`, `qrf`, `bMachine` and `bagEarth`, whose R^2 rank is comparable to `pcaNNet` but they are much slower. According to time, the regressors after `pcaNNet` are `grnn` and `earth`, almost so fast as `gbm` but with much lower R^2.

4. Conclusion

The current work develops an exhaustive comparison of 77 regression methods, 73 implemented in R and other 4 in C++, Matlab and Python, over the whole collection of 83 regression datasets of the UCI machine learning repository, including large datasets up to 2 millions of patterns and 640 inputs. The collection of regressors, that belong to 19 different families, aims to be a representative sample of the most popular and well-known methods currently available for regression tasks. The results have been evaluated in terms of R^2, RMSE and MAE, being similar with the three measurements, and depending on the dataset properties (size and difficulty, measured by the performance achieved by the classical linear regression). For small and difficult datasets, the `penalized` linear regression achieves the best results, followed by random forest (`rf`) and extremely randomized regression trees (`extraTrees`). For small easy datasets, the M5 rule-based model with corrections based on nearest neighbors (`cubist`) achieves the best results, followed by the committee of back-propagation neural networks (`avNNet`) and the boosting ensemble of regression trees (`bstTree`). Finally, for both large difficult and easy datasets the `M5` regression tree is the best, followed the gradient boosted machine (`gbm`) and `cubist`. Considering globally all the datasets, `cubist`, `gbm`, `bstTree` and `M5` achieve the best positions, and the differences between them are related mainly with: 1) the number of cases where they overcome the memory and time limits (128 GB and 48 h., respectively): `cubist` and `bstTree` fail in 8% and 6% of datasets, respectively, `gbm` only for 1% and `M5` never fails; and 2) the speed (`gbm`, `M5` and `bstTree` are 70, 30 and 10 times faster than `cubist`). In terms of R^2, `gbm` and `M5` never decrease more than 0.35 below the best R^2 for any dataset, and $R^2_{best} - R^2_{M5} > 0.25$ only in 2.4% of datasets. Other regressors with good results are extremely randomized regression trees (`extraTrees`), which achieves the best R^2 in 33.7% of datasets, support vector regression (`svr`) and random forest (`rf`), but they are very slow, overcoming the maximum allowed time (48 h.) for more than 20% of the datasets. A post-hoc Friedman-Nemenyi test comparing `cubist` and the remaining regressors gives p <0.05 (i.e., difference statistically significant) excepting `gbm`, `bstTree` and `extraTrees`.

According to the position of their best regressors in the R^2 ranking, the best regressor families are regression rules (whose best regressors are `cubist` and `M5`), boosting ensembles (`gbm` and `bstTree`), neural networks (`avNNet`), random forests (`extraTrees` and `rf`), projection methods (projection pursuit, `ppr`) and support vector regression (`svr`). Other families with regressors included in the top-20 are bagging ensembles (bagging ensemble of MARS regressors, `bagEarth`), generalized additive models (MARS regressor, `earth`), nearest neighbors (`kknn`), generalized linear models (`penalized`) and deep learning (`dlkeras`). The remaining families exhibit poorer performances: ridge and LASSO, Bayesian models, linear regression, regression trees, Gaussian processes and quantile regression. The R^2_{best} overcomes 0.5625, considered the threshold for very good to excellent R^2 according the Colton scale [78], for 76.2% of the datasets. Considering the elapsed time, the fastest regressor is least angle regression (`lars`), while `M5` and `cubist` are 30 and 2,000 times slower, respectively. With respect to memory, the non-negative least squares regression (`nnls`) never requires more than 2 GB, while `cubist` and `M5` require in average about 3 and 8 GB, respectively, and the boosting ensemble of generalized linear models (`rndGLM`) requires about 78 GB, overcoming 128 GB in about half datasets. The future work includes to study the relations between the regression problem and the best regressors in order to predict the best regressor and its performance for a given dataset.

Acknowledgment

This work has received financial support from the Erasmus Mundus Euphrates programme [project number 2013-2540/001-001-EMA2], from the Xunta de Galicia (Centro singular de investigación de Galicia, accreditation 2016-2019) and the European Union (European Regional Development Fund - ERDF), Project MTM2016–76969–P (Spanish State Research Agency, AEI) co–funded by the European Regional Development Fund (ERDF) and IAP network from Belgian Science Policy.

Appendix A. Listing of regression methods

This appendix describes the regressor used in the current work, grouped by families, alongside with their software implementations and values of their tunable hyper-parameters. Default values are assumed for all the regressor parameters not cited explicitely.

I. *Linear regression (LR)*

1. **lm** is the linear model implemented by the `stats` package [11]. Collinear inputs exhibit undefined coefficients in the linear regression model returned by lm, being discarded by it and by other regressors in the list, as we told above.
2. **rlm** implements the robust linear model (`MASS` package), fitted using iteratively re-weighted least squares with maximum likelihood type estimation, which is robust to outliers in the output although not in inputs [13]. The only hyperparameter is the Ψ function, which can be `huber` (Huber function, which leads to a convex optimization problem), `hampel` and Tukey `bisquare`, both with local minima. In our experiments, these functions are selected as the best Ψ for 16%, 82% and 2%, respectively, of the datasets.

II. *Penalized linear regression (PLM)*

3. **penalized** is the penalized linear regression (`penalized` package), which fits generalized linear models with a combination of L1 and L2 penalties. The L1 penalty, also named LASSO, penalizes the sum of absolute values of the coefficients, thus reducing the coefficients of inputs which are not relevant, similarly to input selection. The L2 penalty (also

named ridge) penalizes the sum of squared coefficients, reducing the effects of input collinearity. The regression is regularized by weighting both penalties [15], whose weights are given by hyperparameters λ_1, tuned with values 1, 2, 4, 8 and 16, and λ_2, with values 1, 2, 4 and 8. In our experiments, $\lambda_1 = \lambda_2 = 1$ in the 87.9% of the datasets, and only in 10 of 83 datasets $\lambda_1 \neq 1$ or $\lambda_2 \neq 1$.

4. **enet** is the elastic-net regression model (`elasticnet` package), computed using the least angle regression - elasticnet (LARS-EN) algorithm [12]. Elastic-net provides a model for regularization and input selection, grouping together the inputs which are strongly correlated. This model is specially useful when the number of inputs is higher than the number of patterns, as opposed to LASSO models. There are two hyperparameters (5 values each one): the quadratic penalty, or regularization, hyperparameter (λ, with values 0, $\{10^{-i}\}_1^3$) and the fraction `s` of the L1 norm of the coefficient vector relative to the norm at the full least squares solution (the `fraction` mode is used in the `predict.enet` function, with values 0.05, 0.28, 0.52, 0.76, 1).

5. **glmnet** is the LASSO and elastic-net regularization for generalized linear models (GLM) implemented in the `glmnet` package [17]. The `glmnet` model uses penalized maximum likelihood to fit a GLM for the LASSO and elastic-net non-convex penalties. The mixing percentage α is tuned with 5 values from 0.1 to 1: the value α=1 (resp. < 1) corresponds to the LASSO (resp. elastic-net) penalty. The selected value for α during hyperparameter tuning was 0.1 in 41.7% of the datasets. The regularization hyperparameter λ is also tuned with values 0.00092,

0.0092 and 0.092.

6. **glmSAIC** is the generalized linear model with stepwise feature selection [19] using the Akaike information criterion and the `stepAIC` function in the `MASS` package (regressor `glmStepAIC` in the `caret` model list).

III. ***Additive models (AM)***

7. **gam** is the generalized additive model (GAM) using splines (`mgcv` package). This model [21] is a GLM whose linear predictor is a sum of smooth functions (penalized regression splines) of the covariates. The estimation of the spline parameters uses the generalized cross validation criterion. The only hyperparameter is `select`, a boolean flag that adds an extra penalty term to each function penalizing its wiggliness (waving).

8. **earth** is the multivariate adaptive regression spline (MARS) in the `earth` package. This method [23] is a hybrid of GAM and regression trees (see family XII) which uses a expansion of product spline functions to model non-linear data and interactions among inputs. The spline number and parameters are automatically determined from the data using recursive partitioning, and distinguishing between additive contributions of each input and interactions among them. The functions are added iteratively to reduce maximally the residual, until its change is too small or a number of iterations is reached. The maximum number of terms in the model (`nprune`) is tuned with 15 values (less for some datasets) between 2 and 24.

IV. ***Least squares (LS)***

9. **nnls** is the non-negative least squares regression (`nnls` package), which uses the Lawson-Hanson NNLS method [25] to solve for $\mathbf{x}$ the optimization problem $\min_{\mathbf{x}} |\mathbf{A}\mathbf{x} - \mathbf{b}|$ subject to $\mathbf{x} \geq 0$, where $\mathbf{A}$ is the input data matrix, $\mathbf{b}$ is the true output and $\mathbf{x}$ is the linear predictor.

10. **krlsRad** is the radial basis function kernel regularized least squares regression (`KRLS` package), which uses Gaussian radial basis functions to learn the best fitting function which minimizes the squared loss of a Tikhonov regularization problem [27]. The KRLS method, which corresponds to the `krlsRadial` in the `caret` model list, learns a closed form function which is so interpretable as ordinary regression models. The only hyperparameter is the kernel spread (σ), with 10 values in the set $\{10^i\}_{-7}^{2}$. By default, this method determines the trade-off between model fit and complexity, which is defined by the λ parameter, by minimizing the sum of squared leave-one-out errors. The `getModelInfo` function only lists one value for λ, despite being listed as a tunable hyperparameter in the `caret` model list.

V. *Projection methods (PRJ)*

11. **spls** is the sparse partial least squares regression (`spls` package). This method [29] uses sparse linear combinations of the inputs in the dimensionality reduction of PLS in order to avoid lack of consistency of PLS with high dimensional patterns. The hyperparameters are the number of latent components (`K`), with values 1, 2 and 3, and the threshold (η), with 7 values from 0.1 to 0.9.

12. **simpls** fits a PLS regression model with the simpls method [30], implemented by the `plsr` function in the `pls` package, with `method=simpls`.

The PLS method projects the inputs and the output to a new space and it searches the direction in the input space which explains the maximum output variance. Simpls is particularly useful when there are more inputs than patterns and inputs are collinear. It directly calculates the PLS factors as linear combinations of the inputs maximizing a covariance criterion with orthogonality and normalization constraints. The only hyperparameter is the number of components (`ncomp`) used by the `simpls` model, with values from 1 to `min(10,#inputs-1)`.

13. **kpls** is the PLS regression with `method=kernelpls` [31] in the same function and package as `simpls`, using the same hyperparameter setting as `simpls` with 6 values. This is the regressor named `kernelpls` in the `caret` model list.

14. **wkpls** uses `method=widekernelpls` [33] for PLS, tuning the number of components (`ncomp`) as `simpls` also with 10 values (regressor `widekernelpls` in the `caret` model list).

15. **enpls.fs** is an ensemble of sparse partial least squares (`spls`, see regressor #12) regressors implemented by the `enpls` package [35]. The `getModelInfo` function lists only one value for the number of components (`maxcomp`), while the `threshold` argument, specified as a hyperparameter by the `caret` model list, is missing in the `enpls.fit` function.

16. **plsRglm** is the partial least squares generalized linear model (`plsRglm` package) with `modele=pls-glm-gaussian` [37]. The hyperparameters are the number of extracted components (`nt`), tuned with values 1, 2, 3 and 4, and the input significance level (`alpha.pvals.expli`), with

values in the set $\{10^i\}_{-2}^{2}$.

17. **ppr** performs the projection pursuit regression (`stats` package), which models the output as a sum of averaging functions (mean, median, etc.) of linear combinations of the inputs [39]. The coefficients are iteratively calculated to minimize a projection pursuit (fitting criterion, given by the fraction of unexplained variance which is explained by each function) until it falls below a predefined threshold. The only tunable hyperparameter is the number of terms of the final model (`nterms`), with values from 1 to 10.
18. **pcr** develops principal component regression (`pls` package), which models the output using classical linear regression with coefficients estimated with principal component analysis (PCA), i.e., using the principal components as inputs [41]. It works in three stages: 1) performs PCA and selects a subset of the principal components; 2) uses ordinary least squares to model the output vector using linear regression on the selected components; 3) uses the eigenvectors corresponding to the selected components in order to calculate the final `pcr` estimator transforming the modeled output vector to the original space, and estimates the regression coefficients for the original outputs. The number of components (`ncomp`) is tuned with values from 1 to `min(10,#inputs-1)`.
19. **icr** is the independent component regression (`caret` package). The `icr` fits a linear regression model using independent component analysis (ICA), implemented by the `fastICA` package, instead of the original inputs [43]. The input data are considered a linear combination of a number of independent and non-Gaussian components (sources), so the

training set matrix is written as the product of the source matrix and a linear mixed matrix, which contains the coefficients of the linear combination. The ICA estimates a "separating" matrix, which multiplied by the original data, provides the sources. This matrix must maximize the non-gaussianity of the sources, measured by the neg-entropy. The only hyperparameter is the number of independent components `n.comp`, with values from 1 to `min(10,#inputs-1)`.

20. **superpc** is the supervised PCA (`superpc` package). This method [45] retains only a subset of the principal components which are correlated to the output. The tunable hyperparameters are the number of principal components (`n.components`), tuned with values 1, 2 and 3 (in all the datasets the value 1 is selected), and the `threshold` for retaining the input scores, with values 0.1 and 0.9.

VI. *Least absolute shrinkage and selection operator (LASSO)*

21. **lasso** performs LASSO regression, using the `enet` function in the `elasticnet` package with $\lambda = 0$ to obtain the LASSO solution.
22. **relaxo** develops relaxed LASSO (`relaxo` package), which generalizes the LASSO shrinkage method for linear regression [14]. This method is designed to overcome the trade-off between speed and convergence in the L2-loss function of the regular LASSO, specially for sparse high-dimensional patterns. It provides solutions sparser than LASSO with better prediction error. The relaxation hyperparameter (ϕ) is tuned with 7 values from 0.1 to 0.9, while the penalty hyperparameter (λ) is tuned with 3 data-dependent values.
23. **lars** is the least angle regression (`lars` package), a model selection

method [16] which is less greedy than the typical forward selection methods. It starts with zero coefficients for all the inputs and finds the input i most correlated with the output, increasing step-by-step its coefficient until another input j has high correlation with the current residual (i.e., the error, or difference between the true and predicted outputs). The coefficients of inputs i and j are increased in the equi-angular direction between inputs i and j until some other input k is so correlated with the residual as input j. Then, it proceeds in the equi-angular direction among i, j and k, which is the "least angle direction", and so on until all the coefficients are non-zero (i.e., all the inputs are in the model). The `lasso` and `fraction` options are specified for training and prediction respectively, and the `fraction` hyperparameter (ratio between the L1 norm of the coefficient vector and the norm at the full LS solution) is tuned with 10 values between 0.05 and 1 (for 46.7% of datasets the selected value of `fraction` was 1).

VII. *Ridge regression (RIDGE)*

24. **ridge** develops ridge regresion (`elasticnet` package), which introduces a regularization term, alongside with the squared difference between the desired and true outputs, in the function to optimize. This term, which evaluates the model complexity (e.g., the matrix norm for linear models), is weighted by the penalty or regularization hyperparameter (λ). We use the `enet` function in the `elasticnet` package, already used for regressor `enet`, tuning λ with 5 values between 0.01 and 0.1 (these two values are selected for 50% and 30% of the datasets, respectively).
25. **spikeslab** implements the spike and slab regression (`spikeslab` pack-

age), which computes weighted generalized ridge regression estimators using Bayesian spike and lab models [18]. The `spikeslab` method combines filtering for dimensionality reduction, model averaging using Bayesian model averaging, and variable selection using the `gnet` estimator. The only tunable hyperparameter is the number of selected inputs (`vars`), with the two values listed by the `getModelInfo` function: 2 and the number of inputs (both selected with similar frequencies).

26. **foba** is the ridge regression with forward, backward and sparse input selection [20], implemented in the `foba` package. We use the adaptive forward-backward greedy version of the method (with the default value `foba` for the `type` argument of the `foba` function), which does a backward step when the ridge penalized risk increases in less than the parameter ν (with value 0.5 by default) multiplied by the ridge penalized risk reduction in the previous forward step. The hyperparameters are regularization for ridge regression (λ), with 10 values between 10^{-5} and 0.1, and the number of selected inputs or sparsity (`k`) for the prediction, with two values: 2 and the number of inputs.

VIII. *Bayesian models (BYM)*

27. **bayesglm** is the Bayesian GLM, implemented by the `arm` package. It uses expectation maximization to update the β coefficients of the GLM at each iteration, using an augmented regression to represent the prior information [22]. The coefficients are calculated using a Student-t prior distribution.

28. **brnn** is the Bayesian regularized neural network (`brnn` package), a network with one hidden layer trained using Gauss-Newton optimization.

The training minimizes a combination of squared error and a regularization term which uses the squared network weights [24]. The Bayesian regularization [79] determines the weights of both terms based on inference techniques. This requires an iterative computation of the Hessian matrix (or its Gauss-Newton approximation) of the performance with respect to the weights and biases until a goal is met or a maximum number of iterations is reached. The weights are not normalized, and the number of hidden neurons (`neurons`) is a hyperparameter tuned with values between 1 and 15, selecting `neurons`=1 in 31.6% of the datasets.

29. **bMachine** is the Bayesian additive regression tree (`bartMachine` package), which consists of a sum of regression trees and a regularization process developed on the parameters of the tree set [26]. It corresponds to `bartMachine` in the `caret` model list. We use the default number of trees (`num_trees=50`, the unique value listed by the `getModelInfo` function), and the tunable hyperparameters are the prior boundary (`k`), with values 2, 3 and 4, and α (base value in tree prior to decide if a node is terminal or not), with 3 values between 0.9 and 0.99.

IX. *Space Gaussian processes (SGP, also known as kriging)*

30. **gprLin** implements Gaussian process regression (`gaussprLinear` in the `caret` model list), which interpolates values for the output using a sum of Gaussians, each specified by a mean and a covariance (or kernel) function that measures the similarity between inputs. This regressor uses linear (`vanilladot`) kernel in the `gausspr` function of the `kernlab` package.

31. **gprRad** (named `gaussprRadial` in the `caret` model list) uses the same function with Gaussian (`rbfdot`) kernel and automatically calculated kernel spread (default option `kpar=1`).
32. **gprPol** is the same method with polynomial (`polydot`) kernel (`gausspr Poly` in the `caret` model list), tuning the kernel hyperparameters `degree`, with values 1, 2 and 3, and `scale`, with values $\{10^{-i}\}_1^3$.

X. ***Quantile regression (QTR)***

33. **rqlasso** develops quantile regression with LASSO penalty, using the `rq.lasso.fit` function in the `rqPen` package. The quantile regression models optimize the so-called quantile regression error, which uses the tilted absolute value instead of the root mean squared error. This tilted function applies asymmetric weights to positive/negative errors, computing conditional quantiles of the predictive distribution. This method fits a quantile regression model with the LASSO penalty [32], tuning the regularization hyperparameter λ, with 10 values between 0.1 and 10^{-4} (for 76.7% of datasets the selected value was less than 0.01).
34. **rqnc** performs non-convex penalized quantile regression, with the `rq. nc.fit` function in the `rqPen` package. This regressor performs penalized quantile regression using local linear approximation [34] to maximize the penalized likelihood for non-convex penalties. The two hyperparameters are λ, with the same values as `rqlasso`, and `penalty`, which can be `MCP` (minimax concave penalty) or `SCAD` (smoothly clipped absolute deviation).
35. **qrnn** is the quantile regression neural network (`qrnn` package), a neural network which uses ramp transfer and quantile regression error

functions [36]. The hyperparameters are number of hidden neurons (`n.hidden`), with 7 values from 1 to 13, and the `penalty` for weight decay regularization, with values 0, 0.1 and 0.0001.

XI. ***Nearest neighbors (NN)***

36. **kknn** performs weighted k-nearest neighbors regression [38], implemented by the `kknn` package. The neighbors are weighted using a kernel function according to their distances to the test pattern. The only hyperparameter is the number of neighbors (`kmax`, with 10 odd values between 5 and 23).

XII. ***Regression trees (RGT)***

37. **rpart** is the classical regression tree trained using the recursive partitioning algorithm [40], implemented in the `rpart` package. Only the complexity hyperparameter (`cp`) is tuned (10 values).
38. **nodeHarvest** is a simple interpretable tree-based ensemble for high-dimensional regression with sparse results [42] implemented in the `nodeHarvest` package. A starting tree of few thousand nodes is randomly generated. For a test pattern assigned to a node, the output is the mean of its training outputs; when the test pattern is assigned to several nodes, the output is the weighted average of their means. The selection of the nodes and their weights requires to solve a quadratic programming problem with linear inequality constraints. Only few nodes with non-zero weights are selected, so the solution is sparse. The hyperparameters are the maximal interaction depth (`maxinter`, with 10 values between 1 and 10, the most selected were 6-8) and the `mode` (2 values), which can be `mean` (weighted group means) or `outbag` (zero values in

the smoothing matrix diagonal). This regressor is very slow, requiring huge times (more than 6 days) for high `maxinter` values and some datasets.

39. **ctree2** is the conditional inference tree (`party` package), which estimates the output using inference after a recursive partitioning of the input space [44]. The method tests the null hypothesis of statistical independence between any input and the output, and it stops when the hypothesis can not be rejected. Otherwise, it selects the input most related to the output, measured by the p-value of the partial test of independence between the output and that input. Then, it does a binary splitting of the selected input, and the two previous steps are recursively repeated. The hyperparameters are the threshold for $1 - p$ in order to do a split (`mincriterion`), with 4 linearly spaced values between 0.01 and 0.99, and the maximum tree depth (`maxdepth`), with integer values from 1 to 5, selecting `maxdepth`=5 for 68.3% of datasets.
40. **partDSA** develops partitioning using deletion, substitution, and addition, implemented in the `partDSA` package [46]. This method recursively partitions the space considering that multiple inputs jointly influence the output, predicting a piecewise constant estimation though a parsimonious model of `AND/OR` conjunctions. The only hyperparameter is the maximum number of terminal partitions (`cut.off.grow`), tuned with integer values between 1 and 10, although the value 1 is selected for all the datasets. The parameter `vfold` is set to 1 in order to reduce the computational cost for large datasets.
41. **evtree** is the tree model from genetic algorithms [47] which uses evo-

lutionary algorithms to learn globally optimal regression trees (`evtree` package). It chooses splits for the recursive partitioning in the forward stepwise search in order to optimize a global cost function. The only hyperparameter is the complexity (α) of the cost function, tuned with 10 linearly spaced values between 1 and 3, which weights negatively large tree sizes.

XIII. ***Regression rules (RGR)***

42. **M5** is the model tree/rules [48] implemented in the `RWeka` package, tuning the flags `pruned` and `smoothed` (values `yes/no` each one), and `rules/trees` (to create a tree or a rule set) of the Weka M5 implementation.

43. **cubist** learns a M5 rule-based model with corrections based on nearest neighbors in the training set [50], implemented by the `Cubist` package. A tree structure is created and translated to a collection of rules, which are pruned and combined, and each rule gives a regression model, applied to the patterns which accomplish that rule. Cubist extends M5 with boosting when the hyperparameter `committees` > 1, and using nearest neighbor based to correct the rule-based prediction. The tunable hyperparameters are the number of training `committees` (with 3 data-dependent odd values) and the number of `neighbors` (with values 0, 5 and 9) for prediction.

44. **SBC** is the substractive clustering and fuzzy C-means rules (`frbs` package), which uses substractive clustering to get the cluster centers of a fuzzy rule-based system for classification or regression [52]. Initially, each training pattern is weighted by a potential function which de-

creases with its distances to the remaining centers, and them it optimizes the centers using fuzzy C-means. The center with the highest potential is selected as a cluster center, and the potential of the remaining centers are updated. The only hyperparameter is the neighborhood radius (`r.a`), tuned with 7 linearly spaced values between 0 and 1 (this value is selected for nearly 50% of the 31 datasets where SBC does not fail). The selection of new cluster centers and potential updating is repeated until the potentials of the remaining patterns are below a pre-specified fraction of the potential of the first cluster center. Once all the centers are selected, they are optimized using fuzzy C-means. As we report in last rows of Table 5, we also tried the remaining 8 regression methods implemented in the `frbs` package and included in the `caret` model list (ANFIS, DENFIS, FIR.DM, GFS.FR.MOGUL, GFS.LT.RS, GFS.THRIFT, HYFIS and WM), but run-time errors happened for most or all the datasets.

XIV. *Random forests (RF)*

45. **rf** is the random forest ensemble of random regression trees implemented by the `randomForest` package [54]. The outputs of the base regressors are averaged to get the regressor output. Its only hyperparameter is the number of randomly selected inputs (`mtry`) with 10 linearly spaced values from 2 until the number of inputs, or less than 10 values when the number of dataset inputs is less than 11 (the lowest value `mtry`=2 was selected in 18% of the 64 datasets where `rf` does not fail).

46. **Boruta** combines RF with feature selection (`Boruta` package). An

input is removed when a statistical test proves that it is less relevant than a shadow random input, created by shuffling the original ones [55]. Conversely, inputs that are significantly better than shadowed ones are confirmed. The iterative search stops when only confirmed inputs are retained, or after a maximum number of iterations (`maxRuns=100` by default), in which case non-confirmed inputs remain unless the iterations or the test p-value (0.01 by default) are increased. The only hyperparameter is `mtry`, tuned as in `rf`.

47. **RRF** is the regularized random forest (`RRF` package), which uses regularization for input selection in `rf`, penalizing the selection of a new input for splitting when its Gini information gain is similar to the inputs included in the previous splits [57]. The hyperparameters are `mtry`, with 3 linearly spaced values between 2 and the number of inputs, and the regularization coefficient (`coefReg`), with values 0.01 and 1, both selected with similar frequencies.

48. **cforest** is a forest ensemble of conditional inference trees [54], each one fitting one bootstrap sample (`party` package [58]). The only hyperparameter is the number of selected inputs (`mtry`, with values 2 and the number of inputs) of the conditional trees.

49. **qrf** is the quantile regression forest (`quantregForest` package [59]), a tree-based ensemble which generalizes RF in order to estimate conditional quantile functions. This regressor grows several RFs, storing all the training patterns associated to each node in each tree. For each test pattern, the weight of each training pattern is the average of the weights of all the training patterns in the leaves activated by that

pattern in the different trees of the forest. Using these weights, the distribution function of each output value, and the conditional quantiles, are estimated. The only hyperparameter is `mtry` (tuned with 2 values as `cforest`). The quantile prediction threshold (argument `what` in the `predict.quantregForest` function) is set to 0.5.

50. **extraTrees** is the ensemble of extremely randomized regression trees [60] implemented by the `extraTrees` package. It randomizes the input and cut-point of each split (or node in the tree), using a parameter which tunes the randomization strength. The full training set is used instead of a bootstrap replica. It is expected that explicit randomization of input and cut-point splittings combined with ensemble averaging should reduce the variance more than other methods. Its hyperparameters are the number of inputs randomly selected at each node (`mtry`, tuned with 2 values as cforest) and the minimum sample size to split a node (`numRandomCuts`), tuned with integer values from 1 to 10 (the selected value was 1 for 48.3% of the datasets).

XV. *Bagging ensembles (BAG)*

51. **bag** [62] is the bagging ensemble of conditional inference regression trees (see regressor #39) implemented by the `caret` package. The output for a test pattern is the average of the outputs over the base regression trees.

52. **bagEarth** is the bagged MARS (`caret` package), a bagging ensemble of MARS base regressors implemented in the `earth` package (see regressor #9), which learns a MARS model with `degree=1` for each bootstrap sample. The only hyperparameter is the maximum number of terms

(`nprune`) in the pruned regression model (10 values).

53. **treebag** is the bagged CART, a bagging ensemble of `rpart` regression base trees (see regressor #37), implemented by the `ipredbagg` function in the `ipred` package [64].

XVI. ***Boosting ensembles (BST)***

54. **rndGLM** is a boosting ensemble of GLMs [65] implemented by the `randomGLM` package (also named `randomGLM` in the `caret` model list). It uses several bootstrap samples (`nBags=100` by default) of the training set, randomly selecting inputs and interaction terms among them depending on the `maxInteractionOrder` hyperparameter, tuned with values 1, 2 and 3 (selected with frequencies 53.3%, 40% and 6.7%, respectively). For each sample, inputs are ranked by its correlation with the output, and a predefined number of them are selected, using forward selection, to create a multivariate GLM. For a test pattern, the predicted value is the average of the GLM outputs. This regressor has very high memory requirements, overcoming the largest available memory (128GB) in 38 datasets, and requiring 128, 64, 32 and 16GB in 2, 10, 22 and 12 datasets, respectively.

55. **BstLm** is the gradient boosting machine with linear base regressors, implemented in the `bst` package. Gradient boosting optimizes arbitrary differentiable loss functions defining the fitting criteria [53]. Boosting combines weak base regressors into a strong ensemble by iteratively adding base regressors, and in each iteration the new regressor is learned to fit the error (residual) of the previous ensemble. Since the error can be viewed as the negative gradient of the squared error

loss function, boosting can be considered a gradient descent method. BstLm uses the `bst` function with linear base regressors (argument `learner=lm`) and Gaussian family, since squared error loss is used. The only hyperparameter is the number of boosting iterations (`mstop`), with 10 values from 50 to 500.

56. **bstSm** is the gradient boosting with smoothing spline base regressors (`learner=sm` in the `bst` function of the same package). The number of boosting iterations (`mstop`) is tuned with 10 values as `BstLm`.

57. **bstTree** is the gradient boosting with regression base trees (`learner=tree`, same function and package as `BstLm`). The hyperparameters are the number of boosting iterations (`mstop`, 4 values from 40 to 200) and the maximum depth of nodes in the final tree (`maxdepth` item in the list of the `control.tree` argument of the `bst` function), with integer values between 1 and 5 (this last value is selected in 55% of the datasets).

58. **glmboost** is the gradient boosting ensemble with GLM base regressors (`glmboost` function in the `mboost` package), tuning the number of boosting iterations (`mstop`, 10 values).

59. **gamboost** is the boosted generalized additive model (`mboost` package), a gradient boosting ensemble of GAM base regressors [69]. The ensemble minimizes a weighted sum of the loss function evaluated at the training patterns by computing its negative gradient. The base regressors are component-wise models (P-splines with a B-spline base, by default). The only hyperparameter is the number of initial boosting iterations (`mstop`), with 10 values from 50 to 500, selecting 500 as the best value for 56.7% of the datasets.

60. **gbm** is the generalized boosting regression model (`gbm` package [49]), named stochastic gradient boosting in the `caret` model list. The hyperparameters are the maximum depth of input interactions (`interaction.depth`), with integer values from 1 to 5 (the last value was selected in 48.3% of the datasets), and number of trees for prediction (`n.trees`), with values from 50 to 250 with step 50 (the value 250 was selected in 45% of the datasets). We use a Gaussian distribution and `shrinkage=0.1` (default values).

61. **blackboost** is the gradient boosting (`blackboost` function in the `mboost` package) with conditional inference regression base trees (`ctree` regressors in the `party` package, see regressor #40) and arbitrary loss functions [51]. The only hyperparameter is the maximum tree depth (`maxdepth` argument in the `party::ctree_control` function, used as `tree_controls` argument of the `blackboost` function), with integer values from 1 to 5, value selected in 79% of the datasets.

62. **xgbTree** is the extreme gradient boosting [53], using the `xgb.train` function in the `xgboost` package with `booster=gbtree`, root mean squared error as evaluation metric and linear regression as objective function. The hyperparameters are the maximum tree depth (`max_depth`), with values 1, 2 and 3 (`max_depth`=3 for 53.3% of the datasets); maximum number of boosting iterations (`nrounds`), with values 50, 100 and 150; and learning rate (η), with values 0.3 and 0.4.

63. **xgbLinear** is the extreme gradient boosting with `booster=gblinear` and linear regression as objective function (`xgboost` package). Its hyperparameters are the L2 (square loss) regularization term on weights

(λ, with values 0, 0.1 and 0.0001), bias (α, with values 0 and 0.1), and number of boosting iterations (`nrounds`, tuned as `xgbTree`).

XVII. ***Neural networks (NET)***

64. **mlpWD** is the classical multi-layer perceptron with one hidden layer and weight decay (named `mlpWeightDecay` in the `caret` model list). It uses the `mlp` function in the `RSNNS` package, with `learnFunc = BackpropWeightDecay`. The tunable hyperparameters are the `size` of the hidden layers (5 odd values between 1 and 5) and the weight `decay` (values 0, 0.1, 0.042, 0.01778 and 0.007498).
65. **mlpWDml** is the same network with three hidden layers (`RSNNS` package, named `mlpWeightDecayML` in the `caret` model list), tuning four hyperparameters: the sizes of the three hidden layers (`layer1`, `layer2` and `layer3`, tuned with values 1, 3 and 5 each one) and the weight `decay` (same values as `mlpWD`).
66. **avNNet** is the model averaged neural network (`caret` package). A committee of 5 (argument `repeats`) multi-layer perceptron neural networks of the same size trained using different random seeds, being averaged to give an output [80]. The boolean argument `linout` is set to have linear output neurons for regression, and `MaxNWts` is adjusted to allow the number of weights required by the dataset inputs. The hyperparameters are the network `size`, tuned with 7 odd values between 1 and 13, and the weight `decay` (with values 0, 0.1 and 0.0001).
67. **rbf** is the radial basis function network (`RSNNS` package) which does a linear combination of basis functions, each centered around a prototype [56]. The information is locally codified (opposed to globally

in the MLP), the training should be faster and the network is more interpretable, although the output might be undefined if a test pattern does not activate any prototype. The only hyperparameter is the `size` of the hidden layer (10 odd values from 1 to 19).

68. **grnn** is the generalized regression neural network [61], a special type of RBF network implemented by the Matlab neural network toolbox. After a clustering of the training set, the nodes of the hidden layer store the cluster centers, although the Matlab implementation uses so many clusters as training patterns. The output for a test pattern is a weighted sum of the Gaussian functions centered in the cluster centers, scaled by the cluster populations. During training, whenever a pattern is assigned to a cluster, the weight of the Gaussian function corresponding to that cluster is updated using the desired output. The Gaussian `spread` is the only hyperparameter (13 values between 0.01 and 2): large (resp. small) values lead to smooth (resp. close) approximations.
69. **elm** is the extreme learning machine [63] implemented by the `elmNN` package. The only hyperparameters are the number of hidden neurons (`nhid`), with 40 odd values between 1 and 79 (the last value was selected in 11.7% of the datasets), and the activation function (`actfun`), with 4 values: `sin`, `radbas`, `purelin` and `tansig`, selected with similar frequencies.
70. **kelm** is the ELM neural network with Gaussian kernel [63] using the publicly available Matlab code[9]. The hyperparameters are regularization C and kernel spread σ, tuned with values $\{2^i\}_{-5}^{14}$ and $\{2^i\}_{-16}^{8}$, with

[9]http://www.ntu.edu.sg/home/egbhuang/elm_kernel.html (visited March, 29, 2017).

20 and 25 values, respectively.

71. **pcaNNet** is a multi-layer perceptron neural network with one hidden layer trained on the PCA-mapped training patterns, implemented by the `caret` and `nnet` packages. The principal components which account for more than 95% of the data variance are used for training. Each test pattern is mapped to the principal component space and the trained `pcaNNet` model gives an output. The tunable hyperparameters are the `size` of the hidden layer, with 7 values between 1 and 13, and the weight `decay` of the network, with values 0, 0.1 and 0.0001.

72. **bdk** is the supervised bi-directional Kohonen network, implemented in the `kohonen` package [66]. The `bdk` combines Kohonen maps and counterpropagation networks using two maps, for inputs and output respectively. In each iteration, the direct (resp. inverse) pass updates only the weights of the input (resp. output) map, using a weighted similarity measurement (Euclidean distance for regression) which involves both maps, leading to a bi-directional updating. The test output is the weight of the winner node of the output map. The hyperparameters are the sizes of both maps (`xdim` and `ydim`, with 3 values from 3 to 17) and the initial weight (`xweight`) given to the input map during the distance calculation for the output map, and to the output map for updating the input map, tuned with values 0.5, 0.75 and 0.9).

XVIII. *Deep learning (DL)*

73. **dlkeras** is the deep learning neural network implemented by the `Keras` module [67] of the Python programming language, with three hidden layers tuned with 50 and 75 neurons for each layer (`nh1`, `nh2` and `nh3`,

with 8 combinations). The deep learning methods [81, 82] are very popular, specially for image classification, and they are included in this comparison for regression tasks.

74. **dnn** is the deep belief network implemented in R by the `DeepNet` package [68]. It uses three hidden layers, tuning their number of neurons using 3 values for each layer (27 combinations). The weights are initialized using stacked autoencoder (SAE), which in our experiments gave better results than deep belief network (DBN). Hidden and output neurons have hyperbolic tangent and linear activation functions, respectively.

XIX. *Support vector regression (SVR)*

75. **svr** is the ε-support vector regression with Gaussian, accesed via the C++ interface of the `LibSVM` library [9]. We tuned the regularization hyperparameter C and the kernel spread γ with values $\{2^i\}_{-5}^{14}$ and $\{2^i\}_{-16}^{8}$, with 20 and 25 values, respectively.

76. **svmRad** is another implementation of SVR (named `svmRadial` in the `caret` model list) with Gaussian kernel (`ksvm` function in the `kernlab` package [70] for regression (argument `type=eps-svr`). This implementation also uses `LibSVM`, and it tunes the regularization hyperparameter C, with 20 values in the set $\{2^i\}_{-4}^{15}$, and the kernel spread σ. Although we specify 25 values for σ, the `getModelInfo` function only lists 6 values in the set $\{2^{-i}\}_{5}^{7}$.

77. **rvmRad** is the relevance vector machine [71] with Gaussian kernel (`kernlab` package), named `rvmRadial` in the `caret` model list. The RVM has the same functional form as the SVM, but it uses a Bayesian

learning framework which reduces the number of basis functions, compared to the SVM, while keeping an accurate prediction. This regressor avoids the tunable regularization hyperparameter (C) of the SVM, but it uses a method similar to expectation-maximization which, unlike SMO, may fall in local minima. The value of the Gaussian spread σ is estimated by the `getModelInfo` function, which only lists one value.

References

[1] M. Fernández-Delgado, E. Cernadas, S. Barro, D. Amorim, Do we need hundreds of classifiers to solve real classification problems?, J. Mach. Learn. Res. 15 (2014) 3133–3181.

[2] D. H. Wolpert, The lack of a priori distinctions between learning algorithms, Neural Computation 9 (1996) 1341–1390.

[3] K. Bache, M. Lichman, UCI machine learning repository (2013).
URL `http://archive.ics.uci.edu/ml`

[4] G. E. P. Box, D. R. Cox, An analysis of transformations, J. Royal Stat. Soc. Series B (Methodological) 26 (2) (1964) 211–252.

[5] R Team, R: A language and environment for statistical computing, Vienna, Austria, ISBN 3-900051-07-0 (2008).
URL `https://www.R-project.org`

[6] M. Kuhn, Caret: classification and regression training, R package (2016).
URL `http://topepo.github.io/caret/train-models-by-tag.html`

[7] Theano Team, Theano: A Python framework for fast computation of mathematical expressions, arXiv e-prints.

[8] Python Software Foundation, Python Language (2017).
URL `https://www.python.org`

[9] C. Chang, C. Lin, LIBSVM: a library for support vector machines, ACM Trans. on Intel. Syst. and Technol. 2 (2011) 27:1–27:27.

[10] Matlab, version 9.2 (R2017a), Natick (MA) (2011).

[11] J. Chambers, Linear models, J. M. Chambers and T. J. Hastie, Wadsworth & Brooks/Cole, 1992, Ch. 4, pp. 96–138.

[12] H. Zou, T. Hastie, Regularization and variable selection via the elastic net, J. R. Stat. Soc. 67 (2005) 301–320.

[13] P. Huber, Robust statistics, Wiley, 1981.

[14] N. Meinshausen, Relaxed lasso, Comput. Stat. Data An. (2007) 374–393.

[15] J. Goeman, L-1 penalized estimation in the Cox proportional hazards model, Biometrical J. 52 (2010) 70–84.

[16] B. Efron, T. Hastie, I. Johnstone, R. Tibshirani, Least angle regression, Ann. Stat. 32 (2004) 407–499.

[17] N. Simon, J. Friedman, T. Hastie, R. Tibshirani, Regularization paths for Cox's proportional hazards model via coordinate descent, J. Stat. Softw. 39 (5) (2011) 1–13.

[18] H. Ishwaran, J. Rao, U. Kogalur, Spikeslab : prediction and variable selection using spike and slab regression, The R Journal 2 (2010) 68–73.

[19] B. Ripley, Modern applied statistics with S, Springer, 2002.

[20] T. Zhang, Adaptive forward-backward greedy algorithm for learning sparse representations, IEEE Trans. Inf. Theor. 57 (7) (2011) 4689–4708.

[21] S. Wood, Fast stable restricted maximum likelihood and marginal likelihood estimation of semiparametric generalized linear models, J. Royal Stat. Soc. 1 (73) (2011) 3–36.

[22] A. Gelman, A. Jakulin, M. Pittau, Y. Su, A weakly informative default prior distribution for logistic and other regression models, Ann. Appl. Stat. 2 (4) (2009) 1360–1383.

[23] J. Friedman, Multivariate adaptive regression splines, Ann. Stat. 19 (1) (1991) 1–141.

[24] F. Foresee, M. T. Hagan, Gauss-Newton approximation to Bayesian regularization, in: Intl. Joint Conf. on Neural Netw., 1997, pp. 1930–1935.

[25] C. Lawson, R. Hanson, Solving least squares problems, Vol. 15 of Classics in Appl. Math., Soc. Ind. Appl. Math. (SIAM), 1995.

[26] A. Kapelner, J. Bleich, BartMachine: machine learning with Bayesian additive regression trees, J. Stat. Softw. 70 (4) (2016) 1–40.

[27] J. Hainmueller, C. Hazlett, Kernel regularized least squares: reducing misspecification bias with a flexible and interpretable machine learning approach, Polit. Anal. 22 (2013) 143–168.

[28] C. Willians, D. Barber, Ibayesian classification with gaussian processes, IEEE Trans. Pat. Anal. Mach. Intel. 20 (12) (1998) 1342–1351.

[29] H. Chun, S. Keles, Sparse partial least squares for simultaneous dimension reduction and variable selection, J. of the Royal Stat. Soc. 72 (2010) 3–25.

[30] S. Jong, SIMPLS: an alternative approach to partial least squares regression, Chemometr. intel. lab. 18 (1993) 251–263.

[31] S. Jong, Comment on the PLS kernel algorithm, J. Chemometr. 8 (1994) 169–174.

[32] I. Mizera, R. Koenker, Convex optimization in R, J. Stat. Softw. 60 (5) (2014) 1–23.

[33] S. Rännar, F. Lindgren, P. Geladi, S. Wold, A PLS kernel algorithm for data sets with many variables and fewer objects. part 1: theory and algorithm, J. Chemometr. 8 (1994) 111–125.

[34] H. Zou, R. Li, One-step sparse estimates in nonconcave penalized likelihood models, Ann. Stat. 36 (4) (2008) 1509–1533.

[35] N. Xiao, D. Cao, M. Li, Q. Xu, Enpls: an R package for ensemble partial least squares regression, arXiv preprint.

[36] A. Cannon, Quantile regression neural networks: implementation in R and application to precipitation downscaling, Comput. & Geosci. 37 (2011) 1277–1284.

[37] F. Bertrand, M. Maumy-Bertrand, N. Meyer, Partial least squares regression for generalized linear models, r package version 1.1.1 (2014). URL `http://www-irma.u-strasbg.fr/~fbertran`

[38] K. Hechenbichler, K. Schliep, Weighted k-nearest-neighbor techniques and ordinal classification, Tech. rep., Ludwig-Maximilians University Munich (2004).

[39] J. Friedman, W. Stuetzle, Projection pursuit regression, J. Am. Stat. Assoc. 76 (1981) 817–823.

[40] L. Breiman, J. Friedman, R. Olshen, C. Stone, Classification and regression trees, Wadsworth and Brooks, 1984.

[41] B. Mevik, H. Cederkvist, Mean squared error of prediction (MSEP) estimates for principal component regression (PCR) and partial least squares regression (PLSR), J. Chemometr. 18 (9) (2004) 422–429.

[42] N. Meinshausen, Node harvest, Ann. Appl. Stat. 4 (4) (2010) 2049–2072.

[43] A. Hyvarinen, E. Oja, Independent component analysis: algorithms and applications, Neural networks 13 (2000) 411–430.

[44] T. Hothorn, K. Hornik, A. Zeileis, Unbiased recursive partitioning: a conditional inference framework, J. Comput. Graph. Stat. 15 (3) (2006) 651–674.

[45] E. Bair, R. Tibshirani, Semi-supervised methods to predict patient survival from gene expression data, PLoS Biol 2 (4) (2004) 511–522.

[46] A. Molinaro, K.Lostritto, M. van der Laan, PartDSA: deletion/substitution/ addition algorithm for partitioning the covariate space in prediction, Bioinformatics 26 (10) (2010) 1357–63.

[47] T. Grubinger, A. Zeileis, K. Pfeiffer, Evtree: evolutionary learning of globally optimal classification and regression trees in R, J. Stat. Softw. 61 (1) (2014) 1–29.

[48] R. Quinlan, Learning with continuous classes, in: 5th Australian J. Conf. on Artif. Intel., 1992, pp. 343–348.

[49] G. Ridgeway, Gbm package, https://cran.r-project.org/package=gbm (2017).

[50] R. Quinlan, Combining instance-based and model-based learning, in: Proc. Intl. Conf. on Mach. Learn., 1993, pp. 236–243.

[51] P. Buehlmann, T. Hothorn, Boosting algorithms: regularization, prediction and model fitting (with discussion), Stat. Sci. 22 (4) (2007) 477–505.

[52] S. Chiu, Method and software for extracting fuzzy classification rules by subtractive clustering, in: Fuzzy Inf. Proc. Soc., NAFIPS, 1996, pp. 461–465.

[53] J. Friedman, Greedy function approximation: a gradient boosting machine, Ann. Stat. 29 (2001) 1189–1232.

[54] L. Breiman, Random forests, Mach. Learn. 45 (2001) 5–32.

[55] M. Kursa, W. Rudnicki, Feature selection with the Boruta package, J. Stat. Softw. 36 (11) (2010) 1–13.

[56] A. Zell *et al.*, SNNS: Stuttgart Neural Network Simulator User Manual, Version 4.2, Tech. rep., IPVR, University of Stuttgart and WSI, University of Tbingen (1998).

[57] H. Deng, G. Runger, Gene selection with guided regularized random forest, Pat. Recog. 46 (12) (2013) 3483–3489.

[58] T. Hothorn, Party package, http://cran.r-project.org/package=party (2018).

[59] N. Meinshausen, Quantile regression forests, J. Mach. Learn. Res. 7 (2006) 983–999.

[60] P. Geurts, D. Ernst, L. Wehenkel, Extremely randomized trees, Mach. Learn. 63 (1) (2006) 3–42.

[61] D. Specht, A general regression neural network, IEEE T. Neural Netw. 2 (1991) 568–576.

[62] L. Breiman, Bagging predictors, Mach. Learn. 24 (1996) 123–140.

[63] G.-B. Huang, H. Zhou, X. Ding, R. Zhang, Extreme learning machine for regression and multiclass classification, IEEE Trans. Systs., Man, and Cybern.-Part B: Cybern. 42(2) (2012) 513–529.

[64] A. Peters, Ipred package (2015).
URL `http://cran.r-project.org/package=ipred`

[65] L. Song, P. Langfelder, S. Horvath, Random generalized linear model: a highly accurate and interpretable ensemble predictor, BMC Bioinformatics 14 (1) (2013) 1–22.

[66] W. Melssen, R. Wehrens, L. Buydens, Supervised Kohonen networks for classification problems, Chemom. Intell. Lab. Syst. 83 (2006) 99–113.

[67] F. Chollet, Keras: The Python Deep Learning library (2015).
URL `https://keras.io`

[68] X. Ron, Deepnet package (2015).
URL `https://cran.r-project.org/package=deepnet`

[69] P. Buehlmann, B. Yu, Boosting with the L2 loss: regression and classification, J. Am. Stat. Assoc. 98 (2003) 324–339.

[70] A. Karatzoglou, Kernlab package (2015).
URL `https://cran.r-project.org/package=kernlab`

[71] M. Tipping, Sparse Bayesian learning and the relevance vector machine, J. Mach. Learn. Res. 1 (2001) 211–244.

[72] T. Hothorn, F. Leish, A. Zeileis, K. Hornik, The design and analysis of benchmark experiments, J. Comput. and Graph. Stat. 13 (3) (2005) 675–699.

[73] S. García, A. Fernández, A. Benítez, F. Herrera, Statistical comparisons by means of non-parametric tests: a case study on genetic based machine

learning, in: Proc. II Congreso Español de Informática (CEDI 2007), 2007, pp. 95–104.

[74] M. Hollander, D. Wolfe, Nonparametric Statistical Methods, John Wiley & Sons, 1973.

[75] J. Demšar, Statistical comparisons of classifiers over multiple data sets, J. Mach. Learn. Res. 7 (2006) 1–30.

[76] J. D. Gibbons, S. Chakraborti, Nonparametric Statistical Inference, CRC Press, 2011.

[77] T. Pohlert, The pairwise multiple comparison of mean ranks package (PMCMR), r package (2014).
URL `http://CRAN.R-project.org/package=PMCMR`

[78] T. Colton, Statistical in medicine, Little Brown and Co., New York, NJ, 1974.

[79] D. MacKay, Bayesian interpolation, Neural Computation 4 (1992) 415–447.

[80] B. Ripley, Pattern recognition and neural networks, Cambridge Univ. Press, 1996.

[81] G. E. Hinton, S. Osindero, Y.-W. Teh, A fast learning algorithm for deep belief nets, Neural Comput. 18 (7) (2006) 1527–1554.

[82] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, F. Alsaadi, A survey of deep neural network architectures and their applications, Neurocomputing 234 (2017) 11–26.