Journal of Machine Learning Research 15 (2014) 3133-3181 Submitted 11/13; Revised 4/14; Published 10/14

Do we Need Hundreds of Classifiers to Solve Real World Classification Problems?

Manuel Fernández-Delgado MANUEL.FERNANDEZ.DELGADO@USC.ES
Eva Cernadas EVA.CERNADAS@USC.ES
Senén Barro SENEN.BARRO@USC.ES
CITIUS: Centro de Investigación en Tecnoloxías da Información da USC
University of Santiago de Compostela
Campus Vida, 15872, Santiago de Compostela, Spain

Dinani Amorim DINANIAMORIM@GMAIL.COM
Departamento de Tecnologia e Ciências Sociais- DTCS
Universidade do Estado da Bahia
Av. Edgard Chastinet S/N - São Geraldo - Juazeiro-BA, CEP: 48.305-680, Brasil

Editor: Russ Greiner

Abstract

We evaluate **179 classifiers** arising from **17 families** (discriminant analysis, Bayesian, neural networks, support vector machines, decision trees, rule-based classifiers, boosting, bagging, stacking, random forests and other ensembles, generalized linear models, nearest-neighbors, partial least squares and principal component regression, logistic and multinomial regression, multiple adaptive regression splines and other methods), implemented in Weka, R (with and without the caret package), C and Matlab, including all the relevant classifiers available today. We use **121 data sets**, which represent **the whole UCI** data base (excluding the large-scale problems) and other own real problems, in order to achieve significant conclusions about the classifier behavior, not dependent on the data set collection. **The classifiers most likely to be the bests are the random forest** (RF) versions, the best of which (implemented in R and accessed via caret) achieves 94.1% of the maximum accuracy overcoming 90% in the 84.3% of the data sets. However, the difference is not statistically significant with the second best, the SVM with Gaussian kernel implemented in C using LibSVM, which achieves 92.3% of the maximum accuracy. A few models are clearly better than the remaining ones: random forest, SVM with Gaussian and polynomial kernels, extreme learning machine with Gaussian kernel, C5.0 and avNNet (a committee of multi-layer perceptrons implemented in R with the caret package). The random forest is clearly the best family of classifiers (3 out of 5 bests classifiers are RF), followed by SVM (4 classifiers in the top-10), neural networks and boosting ensembles (5 and 3 members in the top-20, respectively).

Keywords: classification, UCI data base, random forest, support vector machine, neural networks, decision trees, ensembles, rule-based classifiers, discriminant analysis, Bayesian classifiers, generalized linear models, partial least squares and principal component regression, multiple adaptive regression splines, nearest-neighbors, logistic and multinomial regression

1. Introduction

When a researcher or data analyzer faces to the classification of a data set, he/she usually applies the classifier which he/she expects to be "the best one". This expectation is conditioned by the (often partial) researcher knowledge about the available classifiers. One reason is that they arise from different fields within computer science and mathematics, i.e., they belong to different "classifier families". For example, some classifiers (linear discriminant analysis or generalized linear models) come from statistics, while others come from symbolic artificial intelligence and data mining (rule-based classifiers or decision-trees), some others are connectionist approaches (neural networks), and others are ensembles, use regression or clustering approaches, etc. *A researcher may not be able to use classifiers arising from areas in which he/she is not an expert (for example, to develop parameter tuning), being often limited to use the methods within his/her domain of expertise.* However, there is no certainty that they work better, for a given data set, than other classifiers, which seem more "exotic" to him/her. The lack of available implementation for many classifiers is a major drawback, although it has been partially reduced due to the large amount of classifiers implemented in R[1] (mainly from Statistics), Weka[2] (from the data mining field) and, in a lesser extend, in Matlab using the Neural Network Toolbox[3]. Besides, the R package caret (Kuhn, 2008) provides a very easy interface for the execution of many classifiers, allowing automatic parameter tuning and reducing the requirements on the researcher's knowledge (about the tunable parameter values, among other issues). Of course, the researcher can review the literature to know about classifiers in families outside his/her domain of expertise and, if they work better, to use them instead of his/her preferred classifier. However, *usually the papers which propose a new classifier compare it only to classifiers within the same family, excluding families outside the author's area of expertise.* Thus, the researcher does not know whether these classifiers work better or not than the ones that he/she already knows. On the other hand, these comparisons are usually developed over a few, although expectedly relevant, data sets. Given that all the classifiers (even the "good" ones) show strong variations in their results among data sets, the average accuracy (over all the data sets) might be of limited significance if a reduced collection of data sets is used (Macià and Bernadó-Mansilla, 2014). Specifically, some classifiers with a good average performance over a reduced data set collection could achieve significantly worse results when the collection is extended, and conversely classifiers with sub-optimal performance on the reduced data collection could be not so bad when more data sets are included. There are useful guidelines (Hothorn et al., 2005; Eugster et al., 2014) to analyze and design benchmark exploratory and inferential experiments, giving also a very useful framework to inspect the relationship between data sets and classifiers.

Each time we find a new classifier or family of classifiers from areas outside our domain of expertise, we ask ourselves whether that classifier will work better than the ones that we use routinely. In order to have a clear idea of the capabilities of each classifier and family, it would be useful to develop a comparison of *a high number of classifiers* arising from *many different families and areas of knowledge* over a *large collection of data sets*. The objective

1. See `http://www.r-project.org`.
2. See `http://www.cs.waikato.ac.nz/ml/weka`.
3. See `http://www.mathworks.es/products/neural-network`.

is to select the classifier which more probably achieves the best performance for any data set. In the current paper we use a large collection of classifiers with publicly available implementations (in order to allow future comparisons), arising from a wide variety of classifier families, in order to achieve significant conclusions not conditioned by the number and variety of the classifiers considered. Using a *high number of classifiers* it is probable that some of them will achieve the "highest" possible performance for each data set, which can be used as reference (maximum accuracy) to evaluate the remaining classifiers. However, according to the No-Free-Lunch theorem (Wolpert, 1996), the best classifier will not be the same for all the data sets. Using *classifiers from many families*, we are not restricting the significance of our comparison to one specific family among many available methods. Using a *high number of data sets*, it is probable that each classifier will work well in some data sets and not so well in others, increasing the evaluation significance. Finally, considering the availability of several alternative implementations for the most popular classifiers, their comparison may also be interesting. The current work pursues: 1) to select the globally best classifier for the selected data set collection; 2) to rank each classifier and family according to its accuracy; 3) to determine, for each classifier, its probability of achieving the best accuracy, and the difference between its accuracy and the best one; 4) to evaluate the classifier behavior varying the data set properties (complexity, #patterns, #classes and #inputs).

Some recent papers have analyzed the comparison of classifiers over large collection of data sets. **OpenML** (Vanschoren et al., 2012), is a complete web interface[4] to anonymously access an experiment data base including 86 data sets from the UCI machine learning data base (Bache and Lichman, 2013) and 93 classifiers implemented in Weka. Although plug-ins for R, Knime and RapidMiner are under development, currently it only allows to use Weka classifiers. This environment allows to send queries about the classifier behavior with respect to tunable parameters, considering several common performance measures, feature selection techniques and bias-variance analysis. There is also an interesting analysis (Macià and Bernadó-Mansilla, 2014) about the use of the UCI repository launching several interesting criticisms about the usual practice in experimental comparisons. In the following, we synthesize these criticisms (the italicized sentences are literal cites) and describe how we tried to avoid them in our paper:

1. The criterion used to select the data set collection (which is usually reduced) may bias the comparison results. The same authors stated (Macià et al., 2013) that the superiority of a classifier may be restricted to a given domain characterized by some complexity measures, studying why and how the data set selection may change the results of classifier comparisons. Following these suggestions, we use all the data sets in the UCI classification repository, in order to avoid that a small data collection invalidate the conclusions of the comparison. This paper also emphasizes that the UCI repository was *not* designed to be a *complete, reliable framework composed of standardized real samples.*

2. The issue about *(1) whether the selection of learners is representative enough and (2) whether the selected learners are properly configured to work at their best performance*

4. See `http://expdb.cs.kuleuven.be/expdb`.

suggests that proposals of new classifiers usually design and tune them carefully, while the reference classifiers are run using a baseline configuration. This issue is also related to the lack of deep knowledge and experience about the details of all the classifiers with available implementations, so that the researchers *usually do not pay much attention about the selected reference algorithms, which may consequently bias the results in favour of the proposed algorithm*. With respect to this criticism, in the current paper we do not propose any new classifier nor changes on existing approaches, so we are not interested in favour any specific classifier, although we are more experienced with some classifier than others (for example, with respect to the tunable parameter values). We develop in this work a parameter tuning in the majority of the classifiers used (see below), selecting the best available configuration over a training set. Specifically, the classifiers implemented in R using caret automatically tune these parameters and, even more important, using pre-defined (and supposedly meaningful) values. This fact should compensate our lack of experience about some classifiers, and reduce its relevance on the results.

3. *It is still impossible to determine the maximum attainable accuracy for a data set, so that it is difficult to evaluate the true quality of each classifier*. In our paper, we use a large amount of classifiers (179) from many different families, so we hypothesize that the maximum accuracy achieved by some classifier is the maximum attainable accuracy for that data set: i.e., we suppose that if no classifier in our collection is able to reach higher accuracy, no one will reach. We can not test the validity of this hypothesis, but it seems reasonable that, when the number of classifiers increases, some of them will achieve the largest possible accuracy.

4. Since the data set complexity (measured somehow by the maximum attainable accuracy) is unknown, we do not know if the *classification error is caused by unfitted classifier design (learner's limitation) or by intrinsic difficulties of the problem (data limitation)*. In our work, since we consider that the attainable accuracy is the maximum accuracy achieved by some classifier in our collection, we can consider that low accuracies (with respect to this maximum accuracy) achieved by other classifiers are always caused by classifier limitations.

5. *The lack of standard data partitioning, defining training and testing data for cross-validation trials. Simply the use of different data partitionings will eventually bias the results, and make the comparison between experiments impossible*, something which is also emphasized by other researchers (Vanschoren et al., 2012). In the current paper, each data set uses the same partitioning for all the classifiers, so that this issue can not bias the results favouring any classifier. Besides, the partitions are publicly available (see Section 2.1), in order to make possible the experiment replication.

The paper is organized as follows: the Section 2 describes the collection of data sets and classifiers considered in this work; the Section 3 discusses the results of the experiments, and the Section 4 compiles the conclusions of the research developed.

2. Materials and Methods

In the following paragraphs we describe the materials (data sets) and methods (classifiers) used to develop this comparison.

Data set	#pat.	#inp.	#cl.	%Maj.	Data set	#pat.	#inp.	#cl.	%Maj.
abalone	4177	8	3	34.6	energy-y1	768	8	3	46.9
ac-inflam	120	6	2	50.8	energy-y2	768	8	3	49.9
acute-nephritis	120	6	2	58.3	fertility	100	9	2	88.0
adult	48842	14	2	75.9	flags	194	28	8	30.9
annealing	798	38	6	76.2	glass	214	9	6	35.5
arrhythmia	452	262	13	54.2	haberman-survival	306	3	2	73.5
audiology-std	226	59	18	26.3	hayes-roth	132	3	3	38.6
balance-scale	625	4	3	46.1	heart-cleveland	303	13	5	54.1
balloons	16	4	2	56.2	heart-hungarian	294	12	2	63.9
bank	45211	17	2	88.5	heart-switzerland	123	12	2	39.0
blood	748	4	2	76.2	heart-va	200	12	5	28.0
breast-cancer	286	9	2	70.3	hepatitis	155	19	2	79.3
bc-wisc	699	9	2	65.5	hill-valley	606	100	2	50.7
bc-wisc-diag	569	30	2	62.7	horse-colic	300	25	2	63.7
bc-wisc-prog	198	33	2	76.3	ilpd-indian-liver	583	9	2	71.4
breast-tissue	106	9	6	20.7	image-segmentation	210	19	7	14.3
car	1728	6	4	70.0	ionosphere	351	33	2	64.1
ctg-10classes	2126	21	10	27.2	iris	150	4	3	33.3
ctg-3classes	2126	21	3	77.8	led-display	1000	7	10	11.1
chess-krvk	28056	6	18	16.2	lenses	24	4	3	62.5
chess-krvkp	3196	36	2	52.2	letter	20000	16	26	4.1
congress-voting	435	16	2	61.4	libras	360	90	15	6.7
conn-bench-sonar	208	60	2	53.4	low-res-spect	531	100	9	51.9
conn-bench-vowel	528	11	11	9.1	lung-cancer	32	56	3	40.6
connect-4	67557	42	2	75.4	lymphography	148	18	4	54.7
contrac	1473	9	3	42.7	magic	19020	10	2	64.8
credit-approval	690	15	2	55.5	mammographic	961	5	2	53.7
cylinder-bands	512	35	2	60.9	miniboone	130064	50	2	71.9
dermatology	366	34	6	30.6	molec-biol-promoter	106	57	2	50.0
echocardiogram	131	10	2	67.2	molec-biol-splice	3190	60	3	51.9
ecoli	336	7	8	42.6	monks-1	124	6	2	50.0

Table 1: Collection of 121 data sets from the UCI data base and our real problems. It shows the number of patterns (#pat.), inputs (#inp.), classes (#cl.) and percentage of majority class (%Maj.) for each data set. Continued in Table 2. Some keys are: ac-inflam=acute-inflammation, bc=breast-cancer, congress-vot= congressional-voting, ctg=cardiotocography, conn-bench-sonar/vowel= connectionist-benchmark-sonar-mines-rocks/vowel-deterding, pb= pittsburg-bridges, st=statlog, vc=vertebral-column.

2.1 Data Sets

We use the whole UCI machine learning repository, the most widely used data base in the classification literature, to develop the classifier comparison. The UCI website[5] specifies a list of **165 data sets** which can be used for classification tasks (March, 2013). We **discarded 57 data sets** due to several reasons: 25 large-scale data sets (with very high #patterns and/or #inputs, for which our classifier implementations are not designed), 27 data sets which are not in the "common UCI format", and 5 data sets due to diverse reasons (just one input, classes without patterns, classes with only one pattern and sets not available). We also used **4 real-world data sets** (González-Rufino et al., 2013) not included in the UCI repository, about fecundity estimation for fisheries: they are denoted as oocMerl4D (2-class classification according to the presence/absence of oocyte nucleus), oocMerl2F (3-class classification according to the stage of development of the oocyte) for fish species *Merluccius*; and oocTris2F (nucleus) and oocTris5B (stages) for fish species *Trisopterus*. The inputs are texture features extracted from oocytes (cells) in histological images of fish gonads, and its calculation is described in the page 2400 (Table 4) of the cited paper.

Overall, we have 165 - 57 + 4 = **112 data sets**. However, some UCI data sets provide several "class" columns, so that actually they can be considered several classification problems. This is the case of data set *cardiotocography*, where the inputs can be classified into 3 or 10 classes, giving two classification problems (one additional data set); *energy*, where the classes can be given by columns y1 or y2 (one additional data set); *pittsburg-bridges*, where the classes can be material, rel-l, span, t-or-d and type (4 additional data sets); plant (whose complete UCI name is *One-hundred plant species*), with inputs margin, shape or texture (2 extra data sets); and *vertebral-column*, with 2 or 3 classes (1 extra data set). Therefore, we achieve a total of 112 + 1 + 1 + 4 + 2 + 1 = **121 data sets**[6], listed in the Tables 1 and 2 by alphabetic order (some data set names are reduced but significant versions of the UCI official names, which are often too long). OpenML (Vanschoren et al., 2012) includes only 86 data sets, of which seven do not belong to the UCI database: baseball, braziltourism, CoEPrA-2006_Classification_001/2/3, eucalyptus, labor, sick and solar-flare. In our work, the #patterns range from 10 (data set *trains*) to 130,064 (*miniboone*), with #inputs ranging from 3 (data set *hayes-roth*) to 262 (data set *arrhythmia*), and #classes between 2 and 100. We used even tiny data sets (such as *trains* or *balloons*), in order to assess that each classifier is able to learn these (expected to be "easy") data sets. In some data sets the classes with only two patterns were removed because they are not enough for training/test sets. The same data files were used for all the classifiers, excepting the ones provided by Weka, which require the ARFF format. We converted the nominal (or discrete) inputs to numeric values using a simple quantization: if an input x may take discrete values $\{v_1, \ldots, v_n\}$, when it takes the discrete value v_i it is converted to the numeric value $i \in \{1, \ldots, n\}$. We are conscious that *this change in the representation may have a high impact in the results of distance-based classifiers* (Macià and Bernadó-Mansilla, 2014), because contiguous discrete values (v_i and v_{i+1}) might not be nearer than non-contiguous values (v_1 and v_n). Each input

5. See `http://archive.ics.uci.edu/ml/datasets.html?task=cla`.
6. The whole data set and partitions are available from: `http://persoal.citius.usc.es/manuel.fernandez.delgado/papers/jmlr/data.tar.gz`.

Data set	#pat.	#inp.	#cl.	%Maj.	Data set	#pat.	#inp.	#cl.	%Maj.
monks-2	169	6	2	62.1	soybean	307	35	18	13.0
monks-3	3190	6	2	50.8	spambase	4601	57	2	60.6
mushroom	8124	21	2	51.8	spect	80	22	2	67.1
musk-1	476	166	2	56.5	spectf	80	44	2	50.0
musk-2	6598	166	2	84.6	st-australian-credit	690	14	2	67.8
nursery	12960	8	5	33.3	st-german-credit	1000	24	2	70.0
oocMerl2F	1022	25	3	67.0	st-heart	270	13	2	55.6
oocMerl4D	1022	41	2	68.7	st-image	2310	18	7	14.3
oocTris2F	912	25	2	57.8	st-landsat	4435	36	6	24.2
oocTris5B	912	32	3	57.6	st-shuttle	43500	9	7	78.4
optical	3823	62	10	10.2	st-vehicle	846	18	4	25.8
ozone	2536	72	2	97.1	steel-plates	1941	27	7	34.7
page-blocks	5473	10	5	89.8	synthetic-control	600	60	6	16.7
parkinsons	195	22	2	75.4	teaching	151	5	3	34.4
pendigits	7494	16	10	10.4	thyroid	3772	21	3	92.5
pima	768	8	2	65.1	tic-tac-toe	958	9	2	65.3
pb-MATERIAL	106	4	3	74.5	titanic	2201	3	2	67.7
pb-REL-L	103	4	3	51.5	trains	10	28	2	50.0
pb-SPAN	92	4	3	52.2	twonorm	7400	20	2	50.0
pb-T-OR-D	102	4	2	86.3	vc-2classes	310	6	2	67.7
pb-TYPE	105	4	6	41.9	vc-3classes	310	6	3	48.4
planning	182	12	2	71.4	wall-following	5456	24	4	40.4
plant-margin	1600	64	100	1.0	waveform	5000	21	3	33.9
plant-shape	1600	64	100	1.0	waveform-noise	5000	40	3	33.8
plant-texture	1600	64	100	1.0	wine	179	13	3	39.9
post-operative	90	8	3	71.1	wine-quality-red	1599	11	6	42.6
primary-tumor	330	17	15	25.4	wine-quality-white	4898	11	7	44.9
ringnorm	7400	20	2	50.5	yeast	1484	8	10	31.2
seeds	210	7	3	33.3	zoo	101	16	7	40.6
semeion	1593	256	10	10.2					

Table 2: Continuation of Table 1 (data set collection).

is pre-processed to have zero mean and standard deviation one, as is usual in the classifier literature. We do not use further pre-processing, data transformation or feature selection. The reasons are: 1) the impact of these transforms can be expected to be similar for all the classifiers; however, our objective is not to achieve the best possible performance for each data set (which eventually might require further pre-processing), but to compare classifiers on each set; 2) if pre-processing favours some classifier(s) with respect to others, this impact should be random, and therefore not statistically significant for the comparison; 3) in order to avoid comparison bias due to pre-processing, it seems advisable to use the original data; 4) in order to enhance the classification results, further pre-processing eventually should be specific to each data set, which would increase largely the present work; and 5) additional transformations would require a knowledge which is outside the scope of this paper, and should be explored in a different study. In those data sets with different training and test sets (annealing or audiology-std, among others), both files were not merged to follow the practice recommended by the data set creators, and to achieve "significant" accuracies on the right test data, using the right training data. In those data sets where the class attribute

must be defined grouping several values (in data set abalone) we follow the instructions in the data set description (file `data.names`). Given that our classifiers are not oriented to data with missing features, the missing inputs are treated as zero, which should not bias the comparison results. For each data set (abalone) two data files are created: `abalone_R.dat`, designed to be read by the R, C and Matlab classifiers, and `abalone.arff`, designed to be read by the Weka classifiers.

2.2 Classifiers

We use **179 classifiers** implemented in C/C++, Matlab, R and Weka. Excepting the Matlab classifiers, all of them are free software. We only developed own versions in C for the classifiers proposed by us (see below). Some of the R programs use directly the package that provides the classifier, but others use the classifier through the interface `train` provided by the caret[7] package. This function develops the parameter tuning, selecting the values which maximize the accuracy according to the validation selected (leave-one-out, k-fold, etc.). The caret package also allows to define the number of values used for each tunable parameter, although the specific values can not be selected. We used all the classifiers provided by Weka, running the command-line version of the java class for each classifier.

OpenML uses 93 Weka classifiers, from which we included 84. We could not include in our collection the remaining 9 classifiers: ADTree, alternating decision tree (Freund and Mason, 1999); AODE, aggregating one-dependence estimators (Webb et al., 2005); Id3 (Quinlan, 1986); LBR, lazy Bayesian rules (Zheng and Webb, 2000); M5Rules (Holmes et al., 1999); Prism (Cendrowska, 1987); ThresholdSelector; VotedPerceptron (Freund and Schapire, 1998) and Winnow (Littlestone, 1988). The reason is that they only accept nominal (not numerical) inputs, while we converted all the inputs to numeric values. Besides, we did not use classifiers ThresholdSelector, VotedPerceptron and Winnow, included in openML, because they accept only two-class problems. Note that classifiers LocallyWeightedLearning and RippleDownRuleLearner (Vanschoren et al., 2012) are included in our collection as LWL and Ridor respectively. Furthermore, we also included other 36 classifiers implemented in R, 48 classifiers in R using the caret package, as well as 6 classifiers implemented in C and other 5 in Matlab, summing up to 179 classifiers.

In the following, we briefly describe the 179 classifiers of the different families identified by acronyms (DA, BY, etc., see below), their names and implementations, coded as `name_implementation`, where `implementation` can be **C**, **m** (Matlab), **R**, **t** (in R using caret) and **w** (Weka), and their tunable parameter values (the notation A:B:C means from A to C step B). We found errors using several classifiers accessed via caret, but we used the corresponding R packages directly. This is the case of lvq, bdk, gaussprLinear, glmnet, kernelpls, widekernelpls, simpls, obliqueTree, spls, gpls, mars, multinom, lssvmRadial, partDSA, PenalizedLDA, qda, QdaCov, mda, rda, rpart, rrlda, sddaLDA, sddaQDA and sparseLDA. Some other classifiers as Linda, smda and xyf (not listed below) gave errors (both with and without caret) and could not be included in this work. In the R and caret implementations, we specify the function and, in typewriter font, the package which provide that classifier (the function name is absent when it is is equal to the classifier).

7. See `http://caret.r-forge.r-project.org`.

Discriminant analysis (DA): 20 classifiers.

1. **lda_R**, linear discriminant analysis, with the function lda in the `MASS` package.
2. **lda2_t**, from the `MASS` package, which develops LDA tuning the number of components to retain up to $\#classes - 1$.
3. **rrlda_R**, robust regularized LDA, from the `rrlda` package, tunes the parameters lambda (which controls the sparseness of the covariance matrix estimation) and alpha (robustness, it controls the number of outliers) with values {0.1, 0.01, 0.001} and {0.5, 0.75, 1.0} respectively.
4. **sda_t**, shrinkage discriminant analysis and CAT score variable selection (Ahdesmäki and Strimmer, 2010) from the `sda` package. It performs LDA or diagonal discriminant analysis (DDA) with variable selection using CAT (Correlation-Adjusted T) scores. The best classifier (LDA or DDA) is selected. The James-Stein method is used for shrinkage estimation.
5. **slda_t** with function slda from the `ipred` package, which develops LDA based on left-spherically distributed linear scores (Glimm et al., 1998).
6. **stepLDA_t** uses the function train in the `caret` package as interface to the function stepclass in the `klaR` package with method=lda. It develops classification by means of forward/backward feature selection, without upper bounds in the number of features.
7. **sddaLDA_R**, stepwise diagonal discriminant analysis, with function sdda in the `SDDA` package with method=lda. It creates a diagonal discriminant rule adding one input at a time using a forward stepwise strategy and LDA.
8. **PenalizedLDA_t** from the `penalizedLDA` package: it solves the high-dimensional discriminant problem using a diagonal covariance matrix and penalizing the discriminant vectors with lasso or fussed coefficients (Witten and Tibshirani, 2011). The lasso penalty parameter (lambda) is tuned with values $\{0.1, 0.0031, 10^{-4}\}$.
9. **sparseLDA_R**, with function sda in the `sparseLDA` package, minimizing the SDA criterion using an alternating method (Clemensen et al., 2011). The parameter lambda is tuned with values $0,\{10^{i}\}_{-1}^{4}$. The number of components is tuned from 2 to $\#classes - 1$.
10. **qda_t**, quadratic discriminant analysis (Venables and Ripley, 2002), with function qda in the `MASS` package.
11. **QdaCov_t** in the `rrcov` package, which develops Robust QDA (Todorov and Filzmoser, 2009).
12. **sddaQDA_R** uses the function sdda in the `SDDA` package with method=qda.
13. **stepQDA_t** uses function stepclass in the `klaR` package with method=qda, forward / backward variable selection (parameter direction=both) and without limit in the number of selected variables (maxvar=Inf).

14. **fda_R**, flexible discriminant analysis (Hastie et al., 1993), with function fda in the `mda` package and the default linear regression method.

15. **fda_t** is the same FDA, also with linear regression but tuning the parameter nprune with values 2:3:15 (5 values).

16. **mda_R**, mixture discriminant analysis (Hastie and Tibshirani, 1996), with function mda in the `mda` package.

17. **mda_t** uses the `caret` package as interface to function mda, tuning the parameter subclasses between 2 and 11.

18. **pda_t**, penalized discriminant analysis, uses the function gen.rigde in the `mda` package, which develops PDA tuning the shrinkage penalty coefficient lambda with values from 1 to 10.

19. **rda_R**, regularized discriminant analysis (Friedman, 1989), uses the function rda in the `klaR` package. This method uses regularized group covariance matrix to avoid the problems in LDA derived from collinearity in the data. The parameters lambda and gamma (used in the calculation of the robust covariance matrices) are tuned with values 0:0.25:1.

20. **hdda_R**, high-dimensional discriminant analysis (Bergé et al., 2012), assumes that each class lives in a different Gaussian subspace much smaller than the input space, calculating the subspace parameters in order to classify the test patterns. It uses the hdda function in the `HDclassif` package, selecting the best of the 14 available models.

Bayesian (BY) approaches: 6 classifiers.

21. **naiveBayes_R** uses the function NaiveBayes in R the `klaR` package, with Gaussian kernel, bandwidth 1 and Laplace correction 2.

22. **vbmpRadial_t**, variational Bayesian multinomial probit regression with Gaussian process priors (Girolami and Rogers, 2006), uses the function vbmp from the `vbmp` package, which fits a multinomial probit regression model with radial basis function kernel and covariance parameters estimated from the training patterns.

23. **NaiveBayes_w** (John and Langley, 1995) uses estimator precision values chosen from the analysis of the training data.

24. **NaiveBayesUpdateable_w** uses estimator precision values updated iteratively using the training patterns and starting from the scratch.

25. **BayesNet_w** is an ensemble of Bayes classifiers. It uses the K2 search method, which develops hill climbing restricted by the input order, using one parent and scores of type Bayes. It also uses the simpleEstimator method, which uses the training patterns to estimate the conditional probability tables in a Bayesian network once it has been learnt, which $\alpha = 0.5$ (initial count).

26. **NaiveBayesSimple_w** is a simple naive Bayes classifier (Duda et al., 2001) which uses a normal distribution to model numeric features.

Neural networks (NNET): 21 classifiers.

27. **rbf_m**, radial basis functions (RBF) neural network, uses the function newrb in the Matlab Neural Network Toolbox, tuning the spread of the Gaussian basis function with 19 values between 0.1 and 70. The network is created empty and new hidden neurons are added incrementally.

28. **rbf_t** uses `caret` as interface to the `RSNNS` package, tuning the size of the RBF network (number of hidden neurons) with values in the range 11:2:29.

29. **RBFNetwork_w** uses K-means to select the RBF centers and linear regression to learn the classification function, with symmetric multivariate Gaussians and normalized inputs. We use a number of clusters (or hidden neurons) equal to half the training patterns, ridge=10^{-8} for the linear regression and Gaussian minimum spread 0.1.

30. **rbfDDA_t** (Berthold and Diamond, 1995) creates incrementally from the scratch a RBF network with dynamic decay adjustment (DDA), using the `RSNNS` package and tuning the negativeThreshold parameter with values $\{10^{-i}\}_1^{10}$. The network grows incrementally adding new hidden neurons, avoiding the tuning of the network size.

31. **mlp_m**: multi-layer perceptron (MLP) implemented in Matlab (function newpr) tuning the number of hidden neurons with 11 values from 3 to 30.

32. **mlp_C**: MLP implemented in C using the fast artificial neural network (FANN) library[8], tuning the training algorithm (resilient, batch and incremental backpropagation, and quickprop), and the number of hidden neurons with 11 values between 3 and 30.

33. **mlp_t** uses the function mlp in the `RSNNS` package, tuning the network size with values 1:2:19.

34. **avNNet_t**, from the `caret` package, creates a committee of 5 MLPs (the number of MLPs is given by parameter repeat) trained with different random weight initializations and bag=false. The tunable parameters are the #hidden neurons (size) in {1, 3, 5} and the weight decay (values {0, 0.1, 10^{-4}}). This low number of hidden neurons is to reduce the computational cost of the ensemble.

35. **mlpWeightDecay_t** uses `caret` to access the `RSNNS` package tuning the parameters size and weight decay of the MLP network with values 1:2:9 and {0, 0.1, 0.01, 0.001, 0.0001} respectively.

36. **nnet_t** uses `caret` as interface to function nnet in the `nnet` package, training a MLP network with the same parameter tuning as in mlpWeightDecay_t.

37. **pcaNNet_t** trains the MLP using `caret` and the `nnet` package, but running principal component analysis (PCA) previously on the data set.

8. See `http://leenissen.dk/fann/wp`.

38. **MultilayerPerceptron_w** is a MLP network with sigmoid hidden neurons, unthresholded linear output neurons, learning rate 0.3, momentum 0.2, 500 training epochs, and #hidden neurons equal (#inputs and #classes)/2.

39. **pnn_m**: probabilistic neural network (Specht, 1990) in Matlab (function newpnn), tuning the Gaussian spread with 19 values in the range 0.01-10.

40. **elm_m**, extreme learning machine (Huang et al., 2012) implemented in Matlab using the code freely available[9]. We try 6 activation functions (sine, sign, sigmoid, hardlimit, triangular basis and radial basis) and 20 values for #hidden neurons between 3 and 200. As recommended, the inputs are scaled between [-1,1].

41. **elm_kernel_m** is the ELM with Gaussian kernel, which uses the code available from the previous site, tuning the regularization parameter and the kernel spread with values $2^{-5}..2^{14}$ and $2^{-16}..2^{8}$ respectively.

42. **cascor_C**, cascade correlation neural network (Fahlman, 1988) implemented in C using the FANN library (see classifier #32).

43. **lvq_R** is the learning vector quantization (Ripley, 1996) implemented using the function lvq in the `class` package, with codebook of size 50, and k=5 nearest neighbors. We selected the best results achieved using the functions lvq1, olvq2, lvq2 and lvq3.

44. **lvq_t** uses `caret` as interface to function lvq1 in the `class` package tuning the parameters size and k (the values are specific for each data set).

45. **bdk_R**, bi-directional Kohonen map (Melssen et al., 2006), with function bdk in the `kohonen` package, a kind of supervised Self Organized Map for classification, which maps high-dimensional patterns to 2D.

46. **dkp_C** (direct kernel perceptron) is a very simple and fast kernel-based classifier proposed by us (Fernández-Delgado et al., 2014) which achieves competitive results compared to SVM. The DKP requires the tuning of the kernel spread in the same range $2^{-16}..2^{8}$ as the SVM.

47. **dpp_C** (direct parallel perceptron) is a small and efficient Parallel Perceptron network proposed by us (Fernández-Delgado et al., 2011), based in the parallel-delta rule (Auer et al., 2008) with $n = 3$ perceptrons. The codes for DKP and DPP are freely available[10].

Support vector machines (SVM): 10 classifiers.

48. **svm_C** is the support vector machine, implemented in C using LibSVM (Chang and Lin, 2008) with Gaussian kernel. The regularization parameter C and kernel spread gamma are tuned in the ranges $2^{-5}..2^{14}$ and $2^{-16}..2^{8}$ respectively. LibSVM uses the one-vs.-one approach for multi-class data sets.

9. See `http://www.extreme-learning-machines.org`.
10. See `http://persoal.citius.usc.es/manuel.fernandez.delgado/papers/jmlr`.

49. **svmlight_C** (Joachims, 1999) is a very popular implementation of the SVM in C. It can only be used from the command-line and not as a library, so we could not use it so efficiently as LibSVM, and this fact leads us to errors for some large data sets (which are not taken into account in the calculation of the average accuracy). The parameters C and gamma (spread of the Gaussian kernel) are tuned with the same values as svm_C.

50. **LibSVM_w** uses the library LibSVM (Chang and Lin, 2008), calls from Weka for classification with Gaussian kernel, using the values of C and gamma selected for svm_C and tolerance=0.001.

51. **LibLINEAR_w** uses the library LibLinear (Fan et al., 2008) for large-scale linear high-dimensional classification, with L2-loss (dual) solver and parameters C=1, tolerance=0.01 and bias=1.

52. **svmRadial_t** is the SVM with Gaussian kernel (in the `kernlab` package), tuning C and kernel spread with values $2^{-2}..2^{2}$ and $10^{-2}..10^{2}$ respectively.

53. **svmRadialCost_t** (`kernlab` package) only tunes the cost C, while the spread of the Gaussian kernel is calculated automatically.

54. **svmLinear_t** uses the function ksvm (`kernlab` package) with linear kernel tuning C in the range $2^{-2}..2^{7}$.

55. **svmPoly_t** uses the `kernlab` package with linear, quadratic and cubic kernels $(s\mathbf{x}^T\mathbf{y}+o)^d$, using scale $s = \{0.001, 0.01, 0.1\}$, offset $o = 1$, degree $d = \{1, 2, 3\}$ and $C = \{0.25, 0.5, 1\}$.

56. **lssvmRadial_t** implements the least squares SVM (Suykens and Vandewalle, 1999), using the function lssvm in the `kernlab` package, with Gaussian kernel tuning the kernel spread with values $10^{-2}..10^{7}$.

57. **SMO_w** is a SVM trained using sequential minimal optimization (Platt, 1998) with one-against-one approach for multi-class classification, C=1, tolerance L=0.001, round-off error 10^{-12}, data normalization and quadratic kernel.

Decision trees (DT): 14 classifiers.

58. **rpart_R** uses the function rpart in the `rpart` package, which develops recursive partitioning (Breiman et al., 1984).

59. **rpart_t** uses the same function tuning the complexity parameter (threshold on the accuracy increasing achieved by a tentative split in order to be accepted) with 10 values from 0.18 to 0.01.

60. **rpart2_t** uses the function rpart tuning the tree depth with values up to 10.

61. **obliqueTree_R** uses the function obliqueTree in the `oblique.tree` package (Truong, 2009), with binary recursive partitioning, only oblique splits and linear combinations of the inputs.

62. **C5.0Tree_t** creates a single C5.0 decision tree (Quinlan, 1993) using the function C5.0 in the homonymous package without parameter tuning.

63. **ctree_t** uses the function ctree in the `party` package, which creates conditional inference trees by recursively making binary splittings on the variables with the highest association to the class (measured by a statistical test). The threshold in the association measure is given by the parameter mincriterion, tuned with the values 0.1:0.11:0.99 (10 values).

64. **ctree2_t** uses the function ctree tuning the maximum tree depth with values up to 10.

65. **J48_w** is a pruned C4.5 decision tree (Quinlan, 1993) with pruning confidence threshold C=0.25 and at least 2 training patterns per leaf.

66. **J48_t** uses the function J48 in the `RWeka` package, which learns pruned or unpruned C5.0 trees with C=0.25.

67. **RandomSubSpace_w** (Ho, 1998) trains multiple REPTrees classifiers selecting randomly subsets of inputs (random subspaces). Each REPTree is learnt using information gain/variance and error-based pruning with backfitting. Each subspace includes the 50% of the inputs. The minimum variance for splitting is 10^{-3}, with at least 2 pattern per leaf.

68. **NBTree_w** (Kohavi, 1996) is a decision tree with naive Bayes classifiers at the leafs.

69. **RandomTree_w** is a non-pruned tree where each leaf tests $\lfloor log_2(\#inputs + 1) \rfloor$ randomly chosen inputs, with at least 2 instances per leaf, unlimited tree depth, without backfitting and allowing unclassified patterns.

70. **REPTree_w** learns a pruned decision tree using information gain and reduced error pruning (REP). It uses at least 2 training patterns per leaf, 3 folds for reduced error pruning and unbounded tree depth. A split is executed when the class variance is more than 0.001 times the train variance.

71. **DecisionStump_w** is a one-node decision tree which develops classification or regression based on just one input using entropy.

Rule-based methods (RL): 12 classifiers.

72. **PART_w** builds a pruned partial C4.5 decision tree (Frank and Witten, 1999) in each iteration, converting the best leaf into a rule. It uses at least 2 objects per leaf, 3-fold REP (see classifier #70) and C=0.5.

73. **PART_t** uses the function PART in the `RWeka` package, which learns a pruned PART with C=0.25.

74. **C5.0Rules_t** uses the same function C5.0 (in the `C50` package) as classifiers C5.0Tree_t, but creating a collection of rules instead of a classification tree.

75. **JRip_t** uses the function JRip in the `RWeka` package, which learns a "repeated incremental pruning to produce error reduction" (RIPPER) classifier (Cohen, 1995), tuning the number of optimization runs (numOpt) from 1 to 5.

76. **JRip_w** learns a RIPPER classifier with 2 optimization runs and minimal weights of instances equal to 2.

77. **OneR_t** (Holte, 1993) uses function OneR in the `RWeka` package, which classifies using 1-rules applied on the input with the lowest error.

78. **OneR_w** creates a OneR classifier in Weka with at least 6 objects in a bucket.

79. **DTNB_w** learns a decision table/naive-Bayes hybrid classifier (Hall and Frank, 2008), using simultaneously both decision table and naive Bayes classifiers.

80. **Ridor_w** implements the ripple-down rule learner (Gaines and Compton, 1995) with at least 2 instance weights.

81. **ZeroR_w** predicts the mean class (i.e., the most populated class in the training data) for all the test patterns. Obviously, this classifier gives low accuracies, but it serves to give a lower limit on the accuracy.

82. **DecisionTable_w** (Kohavi, 1995) is a simple decision table majority classifier which uses BestFirst as search method.

83. **ConjunctiveRule_w** uses a single rule whose antecendent is the AND of several antecedents, and whose consequent is the distribution of available classes. It uses the antecedent information gain to classify each test pattern, and 3-fold REP (see classifier #70) to remove unnecessary rule antecedents.

Boosting (BST): 20 classifiers.

84. **adaboost_R** uses the function boosting in the `adabag` package (Alfaro et al., 2007), which implements the adaboost.M1 method (Freund and Schapire, 1996) to create an adaboost ensemble of classification trees.

85. **logitboost_R** is an ensemble of DecisionStump base classifiers (see classifier #71), using the function LogitBoost (Friedman et al., 1998) in the `caTools` package with 200 iterations.

86. **LogitBoost_w** uses additive logistic regressors (DecisionStump) base learners, the 100% of weight mass to base training on, without cross-validation, one run for internal cross-validation, threshold 1.79 on likelihood improvement, shrinkage parameter 1, and 10 iterations.

87. **RacedIncrementalLogitBoost_w** is a raced Logitboost committee (Frank et al., 2002) with incremental learning and DecisionStump base classifiers, chunks of size between 500 and 2000, validation set of size 1000 and log-likelihood pruning.

88. **AdaBoostM1_DecisionStump_w** implements the same Adaboost.M1 method with DecisionStump base classifiers.

89. **AdaBoostM1_J48_w** is an Adaboost.M1 ensemble which combines J48 base classifiers.

90. **C5.0_t** creates a Boosting ensemble of C5.0 decision trees and rule models (function C5.0 in the hononymous package), with and without winnow (feature selection), tuning the number of boosting trials in {1, 10, 20}.

91. **MultiBoostAB_DecisionStump_w** (Webb, 2000) is a MultiBoost ensemble, which combines Adaboost and Wagging using DecisionStump base classifiers, 3 sub-committees, 10 training iterations and 100% of the weight mass to base training on. The same options are used in the following MultiBoostAB ensembles.

92. **MultiBoostAB_DecisionTable_w** combines MultiBoost and DecisionTable, both with the same options as above.

93. **MultiBoostAB_IBk_w** uses MultiBoostAB with IBk base classifiers (see classifier #157).

94. **MultiBoostAB_J48_w** trains an ensemble of J48 decision trees, using pruning confidence C=0.25 and 2 training patterns per leaf.

95. **MultiBoostAB_LibSVM_w** uses LibSVM base classifiers with the optimal C and Gaussian kernel spread selected by the svm_C classifier (see classifier #48). We included it for comparison with previous papers (Vanschoren et al., 2012), although a strong classifier as LibSVM is in principle not recommended to use as base classifier.

96. **MultiBoostAB_Logistic_w** combines Logistic base classifiers (see classifier #86).

97. **MultiBoostAB_MultilayerPerceptron_w** uses MLP base classifiers with the same options as MultilayerPerceptron_w (which is another strong classifier).

98. **MultiBoostAB_NaiveBayes_w** uses NaiveBayes base classifiers.

99. **MultiBoostAB_OneR_w** uses OneR base classifiers.

100. **MultiBoostAB_PART_w** combines PART base classifiers.

101. **MultiBoostAB_RandomForest_w** combines RandomForest base classifiers. We tried this classifier for comparison with previous papers (Vanschoren et al., 2012), despite of RandomForest is itself an ensemble, so it seems not very useful to learn a MultiBoostAB ensemble of RandomForest ensembles.

102. **MultiBoostAB_RandomTree_w** uses RandomTrees with the same options as above.

103. **MultiBoostAB_REPTree_w** uses REPTree base classifiers.

Bagging (BAG): 24 classifiers.

104. **bagging_R** is a bagging (Breiman, 1996) ensemble of decision trees using the function bagging (in the `ipred` package).

105. **treebag_t** trains a bagging ensemble of classification trees using the `caret` interface to function bagging in the `ipred` package.

106. **ldaBag_R** creates a bagging ensemble of LDAs, using the function bag of the `caret` package (instead of the function train) with option bagControl=ldaBag.

107. **plsBag_R** is the previous one with bagControl=plsBag.

108. **nbBag_R** creates a bagging of naive Bayes classifiers using the previous bag function with bagControl=nbBag.

109. **ctreeBag_R** uses the same function bag with bagControl=ctreeBag (conditional inference tree base classifiers).

110. **svmBag_R** trains a bagging of SVMs, with bagControl=svmBag.

111. **nnetBag_R** learns a bagging of MLPs with bagControl=nnetBag.

112. **MetaCost_w** (Domingos, 1999) is based on bagging but using cost-sensitive ZeroR base classifiers and bags of the same size as the training set (the following bagging ensembles use the same configuration). The diagonal of the cost matrix is null and the remaining elements are one, so that each type of error is equally weighted.

113. **Bagging_DecisionStump_w** uses DecisionStump base classifiers with 10 bagging iterations.

114. **Bagging_DecisionTable_w** uses DecisionTable with BestFirst and forward search, leave-one-out validation and accuracy maximization for the input selection.

115. **Bagging_HyperPipes_w** with HyperPipes base classifiers.

116. **Bagging_IBk_w** uses IBk base classifiers, which develop KNN classification tuning K using cross-validation with linear neighbor search and Euclidean distance.

117. **Bagging_J48_w** with J48 base classifiers.

118. **Bagging_LibSVM_w**, with Gaussian kernel for LibSVM and the same options as the single LibSVM_w classifier.

119. **Bagging_Logistic_w**, with unlimited iterations and log-likelihood ridge 10^{-8} in the Logistic base classifier.

120. **Bagging_LWL_w** uses LocallyWeightedLearning base classifiers (see classifier #148) with linear weighted kernel shape and DecisionStump base classifiers.

121. **Bagging_MultilayerPerceptron_w** with the same configuration as the single MultilayerPerceptron_w.

122. **Bagging_NaiveBayes_w** with NaiveBayes classifiers.

123. **Bagging_OneR_w** uses OneR base classifiers with at least 6 objects per bucket.

124. **Bagging_PART_w** with at least 2 training patterns per leaf and pruning confidence C=0.25.

125. **Bagging_RandomForest_w** with forests of 500 trees, unlimited tree depth and $\lfloor log(\#inputs + 1) \rfloor$ inputs.

126. **Bagging_RandomTree_w** with RandomTree base classifiers without backfitting, investigating $\lfloor log_2(\#inputs) + 1 \rfloor$ random inputs, with unlimited tree depth and 2 training patterns per leaf.

127. **Bagging_REPTree_w** use REPTree with 2 patterns per leaf, minimum class variance 0.001, 3-fold for reduced error pruning and unlimited tree depth.

Stacking (STC): 2 classifiers.

128. **Stacking_w** is a stacking ensemble (Wolpert, 1992) using ZeroR as meta and base classifiers.

129. **StackingC_w** implements a more efficient stacking ensemble following (Seewald, 2002), with linear regression as meta-classifier.

Random Forests (RF): 8 classifiers.

130. **rforest_R** creates a random forest (Breiman, 2001) ensemble, using the R function randomForest in the `randomForest` package, with parameters ntree = 500 (number of trees in the forest) and mtry=$\sqrt{\#inputs}$.

131. **rf_t** creates a random forest using the `caret` interface to the function randomForest in the `randomForest` package, with ntree = 500 and tuning the parameter mtry with values 2:3:29.

132. **RRF_t** learns a regularized random forest (Deng and Runger, 2012) using `caret` as interface to the function RRF in the `RRF` package, with mtry=2 and tuning parameters coefReg={0.01, 0.5, 1} and coefImp={0, 0.5, 1}.

133. **cforest_t** is a random forest and bagging ensemble of conditional inference trees (ctrees) aggregated by averaging observation weights extracted from each ctree. The parameter mtry takes the values 2:2:8. It uses the `caret` package to access the `party` package.

134. **parRF_t** uses a parallel implementation of random forest using the `randomForest` package with mtry=2:2:8.

135. **RRFglobal_t** creates a RRF using the hononymous package with parameters mtry=2 and coefReg=0.01:0.12:1.

136. **RandomForest_w** implements a forest of RandomTree base classifiers with 500 trees, using $\lfloor log(\#inputs + 1) \rfloor$ inputs and unlimited depth trees.

137. **RotationForest_w** (Rodríguez et al., 2006) uses J48 as base classifier, principal component analysis filter, groups of 3 inputs, pruning confidence C=0.25 and 2 patterns per leaf.

Other ensembles (OEN): 11 classifiers.

138. **RandomCommittee_w** is an ensemble of RandomTrees (each one built using a different seed) whose output is the average of the base classifier outputs.

139. **OrdinalClassClassifier_w** is an ensemble method designed for ordinal classification problems (Frank and Hall, 2001) with J48 base classifiers, confidence threshold C=0.25 and 2 training patterns per leaf.

140. **MultiScheme_w** selects a classifier among several ZeroR classifiers using cross validation on the training set.

141. **MultiClassClassifier_w** solves multi-class problems with two-class Logistic_w base classifiers, combined with the One-Against-All approach, using multinomial logistic regression.

142. **CostSensitiveClassifier_w** combines ZeroR base classifiers on a training set where each pattern is weighted depending on the cost assigned to each error type. Similarly to MetaCost_w (see classifier #112), all the error types are equally weighted.

143. **Grading_w** is Grading ensemble (Seewald and Fuernkranz, 2001) with "graded" ZeroR base classifiers.

144. **END_w** is an Ensemble of Nested Dichotomies (Frank and Kramer, 2004) which classifies multi-class data sets with two-class J48 tree classifiers.

145. **Decorate_w** learns an ensemble of fifteen J48 tree classifiers with high diversity trained with specially constructed artificial training patterns (Melville and Mooney, 2004).

146. **Vote_w** (Kittler et al., 1998) trains an ensemble of ZeroR base classifiers combined using the average rule.

147. **Dagging_w** (Ting and Witten, 1997) is an ensemble of SMO_w (see classifier #57), with the same configuration as the single SMO classifier, trained on 4 different folds of the training data. The output is decided using the previous Vote_w meta-classifier.

148. **LWL_w**, Local Weighted Learning (Frank et al., 2003), is an ensemble of DecisionStump base classifiers. Each training pattern is weighted with a linear weighting kernel, using the Euclidean distance for a linear search of the nearest neighbor.

Generalized Linear Models (GLM): 5 classifiers.

149. **glm_R** (Dobson, 1990) uses the function glm in the `stats` package, with binomial and Poisson families for two-class and multi-class problems respectively.

150. **glmnet_R** trains a GLM via penalized maximum likelihood, with Lasso or elasticnet regularization parameter (Friedman et al., 2010) (function glmnet in the `glmnet` package). We use the binomial and multinomial distribution for two-class and multi-class problems respectively.

151. **mlm_R** (Multi-Log Linear Model) uses the function multinom in the `nnet` package, fitting the multi-log model with MLP neural networks.

152. **bayesglm_t**, Bayesian GLM (Gelman et al., 2009), with function bayesglm in the `arm` package. It creates a GLM using Bayesian functions, an approximated expectation-maximization method, and augmented regression to represent the prior probabilities.

153. **glmStepAIC_t** performs model selection by Akaike information criterion (Venables and Ripley, 2002) using the function stepAIC in the `MASS` package.

Nearest neighbor methods (NN): 5 classifiers.

154. **knn_R** uses the function knn in the `class` package, tuning the number of neighbors with values 1:2:37 (13 values).

155. **knn_t** uses function knn in the `caret` package with 10 number of neighbors in the range 5:2:23.

156. **NNge_w** is a NN classifier with non-nested generalized exemplars (Martin, 1995), using one folder for mutual information computation and 5 attempts for generalization.

157. **IBk_w** (Aha et al., 1991) is a KNN classifier which tunes K using cross-validation with linear neighbor search and Euclidean distance.

158. **IB1_w** is a simple 1-NN classifier.

Partial least squares and principal component regression (PLSR): 6 classifiers.

159. **pls_t** uses the function mvr in the `pls` package to fit a PLSR (Martens, 1989) model tuning the number of components from 1 to 10.

160. **gpls_R** trains a generalized PLS (Ding and Gentleman, 2005) model using the function gpls in the `gpls` package.

161. **spls_R** uses the function spls in the `spls` package to fit a sparse partial least squares (Chun and Keles, 2010) regression model tuning the parameters K and eta with values {1, 2, 3} and {0.1, 0.5, 0.9} respectively.

162. **simpls_R** fits a PLSR model using the SIMPLS (Jong, 1993) method, with the function plsr (in the `pls` package) and method=simpls.

163. **kernelpls_R** (Dayal and MacGregor, 1997) uses the same function plsr with method = kernelpls, with up to 8 principal components (always lower than $\#inputs - 1$). This method is faster when #patterns is much larger than #inputs.

164. **widekernelpls_R** fits a PLSR model with the function plsr and method = widekernelpls, faster when #inputs is larger than #patterns.

Logistic and multinomial regression (LMR): 3 classifiers.

165. **SimpleLogistic_w** learns linear logistic regression models (Landwehr et al., 2005) for classification. The logistic models are fitted using LogitBoost with simple regression functions as base classifiers.

166. **Logistic_w** learns a multinomial logistic regression model (Cessie and Houwelingen, 1992) with a ridge estimator, using ridge in the log-likelihood R=10^{-8}.

167. **multinom_t** uses the function multinom in the `nnet` package, which trains a MLP to learn a multinomial log-linear model. The parameter decay of the MLP is tuned with 10 values between 0 and 0.1.

Multivariate adaptive regression splines (MARS): 2 classifiers.

168. **mars_R** fits a MARS (Friedman, 1991) model using the function mars in the `mda` package.

169. **gcvEarth_t** uses the function earth in the `earth` package. It builds an additive MARS model without interaction terms using the fast MARS (Hastie et al., 2009) method.

Other Methods (OM): 10 classifiers.

170. **pam_t** (nearest shrunken centroids) uses the function pamr in the `pamr` package (Tibshirani et al., 2002).

171. **VFI_w** develops classification by voting feature intervals (Demiroz and Guvenir, 1997), with B=0.6 (exponential bias towards confident intervals).

172. **HyperPipes_w** classifies each test pattern to the class which most contains the pattern. Each class is defined by the bounds of each input in the patterns which belong to that class.

173. **FilteredClassifier_w** trains a J48 tree classifier on data filtered using the Discretize filter, which discretizes numerical into nominal attributes.

174. **CVParameterSelection_w** (Kohavi, 1995) selects the best parameters of classifier ZeroR using 10-fold cross-validation.

175. **ClassificationViaClustering_w** uses SimpleKmeans and EuclideanDistance to cluster the data. Following the Weka documentation, the number of clusters is set to #classes.

176. **AttributeSelectedClassifier_w** uses J48 trees to classify patterns reduced by attribute selection. The CfsSubsetEval method (Hall, 1998) selects the best group of attributes weighting their individual predictive ability and their degree of redundancy, preferring groups with high correlation within classes and low inter-class correlation. The BestFirst forward search method is used, stopping the search when five non-improving nodes are found.

177. **ClassificationViaRegression_w** (Frank et al., 1998) binarizes each class and learns its corresponding M5P tree/rule regression model (Quinlan, 1992), with at least 4 training patterns per leaf.

178. **KStar_w** (Cleary and Trigg, 1995) is an instance-based classifier which uses entropy-based similarity to assign a test pattern to the class of its nearest training patterns.

179. **gausspr Radial_t** uses the function gausspr in the `kernlab` package, which trains a Gaussian process-based classifier, with kernel= rbfdot and kernel spread (parameter sigma) tuned with values $\{10^i\}_{-2}^{7}$.

3. Results and Discussion

In the experimental work we evaluate 179 classifiers over 121 data sets, giving 21,659 combinations classifier-data set. We use Weka v. 3.6.8, R v. 2.15.3 with caret v. 5.16-04, Matlab v. 7.9.0 (R2009b) with Neural Network Toolbox v. 6.0.3, the C/C++ compiler v. gcc/g++ 4.7.2 and fast artificial neural networks (FANN) library v. 2.2.0 on a computer with Debian GNU/Linux v. 3.2.46-1 (64 bits). We found errors with some classifiers and data sets caused by a variety of reasons. Some classifiers (lda_R, qda_t, QdaCov_t, among others) give errors in some data sets due to collinearity of data, singular covariance matrices, and equal inputs for all the training patterns in some classes; rrlda_R requires that all the inputs must have different values in more than 50% of the training patterns; other errors are caused by discrete inputs, classes with low populations (specially in data sets with many classes), or too few classes (vbmpRadial requires 3 classes). Large data sets (miniboone and connect-4) give some lack of memory errors, and few small data sets (trains and balloons) give errors for some Weka classifiers requiring a minimum #patterns per class. Overall, we found 449 errors, which represent 2.1% of the 21,659 cases. These error cases are excluded from the average accuracy calculation for each classifier.

The validation methodology is the following. One training and one test set are generated randomly (each with 50% of the available patterns), but imposing that each class has the same number of training and test patterns (in order to have enough training and test patterns of every class). This couple of sets is used only for **parameter tuning** (in those classifiers which have tunable parameters), selecting the parameter values which provide the best accuracy on the test set. The indexes of the training and test patterns (i.e., the data partitioning) are given by the file `conxuntos.dat` for each data set, and are the same for all the classifiers. Then, using the selected values for the tunable parameters, a **4-fold cross validation** is developed using the whole available data. The indexes of the training and test patterns for each fold are the same for all the classifiers, and they are listed in the file `conxuntos_kfold.dat` for each data set. The test results is the average over the 4 test sets. However, for some data sets, which provide **separate data for training and testing** (data sets annealing and audiology-std, among others), the classifier (with the tuned parameter values) is trained and tested on the respective data sets. In this case, the test result is calculated on the test set. We used this methodology in order to keep low the computational cost of the experimental work. However, we are aware of that this methodology may lead to poor bias and variance, and that the classifier results for each data

Rank	Acc.	κ	Classifier	Rank	Acc.	κ	Classifier
32.9	82.0	63.5	parRF_t (RF)	67.3	77.7	55.6	pda_t (DA)
33.1	**82.3**	**63.6**	rf_t (RF)	67.6	78.7	55.2	elm_m (NNET)
36.8	81.8	62.2	svm_C (SVM)	67.6	77.8	54.2	SimpleLogistic_w (LMR)
38.0	81.2	60.1	svmPoly_t (SVM)	69.2	78.3	57.4	MAB_J48_w (BST)
39.4	81.9	62.5	rforest_R (RF)	69.8	78.8	56.7	BG_REPTree_w (BAG)
39.6	82.0	62.0	elm_kernel_m (NNET)	69.8	78.1	55.4	SMO_w (SVM)
40.3	81.4	61.1	svmRadialCost_t (SVM)	70.6	78.3	58.0	MLP_w (NNET)
42.5	81.0	60.0	svmRadial_t (SVM)	71.0	78.8	58.23	BG_RandomTree_w (BAG)
42.9	80.6	61.0	C5.0_t (BST)	71.0	77.1	55.1	mlm_R (GLM)
44.1	79.4	60.5	avNNet_t (NNET)	71.0	77.8	56.2	BG_J48_w (BAG)
45.5	79.5	61.0	nnet_t (NNET)	72.0	75.7	52.6	rbf_t (NNET)
47.0	78.7	59.4	pcaNNet_t (NNET)	72.1	77.1	54.8	fda_R (DA)
47.1	80.8	53.0	BG_LibSVM_w (BAG)	72.4	77.0	54.7	lda_R (DA)
47.3	80.3	62.0	mlp_t (NNET)	72.4	79.1	55.6	svmlight_C (NNET)
47.6	80.6	60.0	RotationForest_w (RF)	72.6	78.4	57.9	AdaBoostM1_J48_w (BST)
50.1	80.9	61.6	RRF_t (RF)	72.7	78.4	56.2	BG_IBk_w (BAG)
51.6	80.7	61.4	RRFglobal_t (RF)	72.9	77.1	54.6	ldaBag_R (BAG)
52.5	80.6	58.0	MAB_LibSVM_w (BST)	73.2	78.3	56.2	BG_LWL_w (BAG)
52.6	79.9	56.9	LibSVM_w (SVM)	73.7	77.9	56.0	MAB_REPTree_w (BST)
57.6	79.1	59.3	adaboost_R (BST)	74.0	77.4	52.6	RandomSubSpace_w (DT)
58.5	79.7	57.2	pnn_m (NNET)	74.4	76.9	54.2	lda2_t (DA)
58.9	78.5	54.7	cforest_t (RF)	74.6	74.1	51.8	svmBag_R (BAG)
59.9	79.7	42.6	dkp_C (NNET)	74.6	77.5	55.2	LibLINEAR_w (SVM)
60.4	80.1	55.8	gausssprRadial_R (OM)	75.9	77.2	55.6	rbfDDA_t (NNET)
60.5	80.0	57.4	RandomForest_w (RF)	76.5	76.9	53.8	sda_t (DA)
62.1	78.7	56.0	svmLinear_t (SVM)	76.6	78.1	56.5	END_w (OEN)
62.5	78.4	57.5	fda_t (DA)	76.6	77.3	54.8	LogitBoost_w (BST)
62.6	78.6	56.0	knn_t (NN)	76.6	78.2	57.3	MAB_RandomTree_w (BST)
62.8	78.5	58.1	mlp_C (NNET)	77.1	78.4	54.0	BG_RandomForest_w (BAG)
63.0	79.9	59.4	RandomCommittee_w (OEN)	78.5	76.5	53.7	Logistic_w (LMR)
63.4	78.7	58.4	Decorate_w (OEN)	78.7	76.6	50.5	ctreeBag_R (BAG)
63.6	76.9	56.0	mlpWeightDecay_t (NNET)	79.0	76.8	53.5	BG_Logistic_w (BAG)
63.8	78.7	56.7	rda_R (DA)	79.1	77.4	53.0	lvq_t (NNET)
64.0	79.0	58.6	MAB_MLP_w (BST)	79.1	74.4	50.7	pls_t (PLSR)
64.1	79.9	56.9	MAB_RandomForest_w (BST)	79.8	76.9	54.7	hdda_R (DA)
65.0	79.0	56.8	knn_R (NN)	80.6	75.9	53.3	MCC_w (OEN)
65.2	77.9	56.2	multinom_t (LMR)	80.9	76.9	54.5	mda_R (DA)
65.5	77.4	56.6	gcvEarth_t (MARS)	81.4	76.7	55.2	C5.0Rules_t (RL)
65.5	77.8	55.7	glmnet_R (GLM)	81.6	78.3	55.8	lssvmRadial_t (SVM)
65.6	78.6	58.4	MAB_PART_w (BST)	81.7	75.6	50.9	JRip_t (RL)
66.0	78.5	56.5	CVR_w (OM)	82.0	76.1	53.3	MAB_Logistic_w (BST)
66.4	79.2	58.9	treebag_t (BAG)	84.2	75.8	53.9	C5.0Tree_t (DT)
66.6	78.2	56.8	BG_PART_w (BAG)	84.6	75.7	50.8	BG_DecisionTable_w (BAG)
66.7	75.5	55.2	mda_t (DA)	84.9	76.5	53.4	NBTree_w (DT)

Table 3: Friedman ranking, average accuracy and Cohen κ (both in %) for each classifier, ordered by increasing Friedman ranking. Continued in the Table 4. BG = Bagging, MAB=MultiBoostAB.

Rank	Acc.	κ	Classifier	Rank	Acc.	κ	Classifier
86.4	76.3	52.6	ASC_w (OM)	110.4	71.6	46.5	BG_NaiveBayes_w (BAG)
87.2	77.1	54.2	KStar_w (OM)	111.3	62.5	38.4	widekernelpls_R (PLSR)
87.2	74.6	50.3	MAB_DecisionTable_w (BST)	111.9	63.3	43.7	mars_R (MARS)
87.6	76.4	51.3	J48_t (DT)	111.9	62.2	39.6	simpls_R (PLSR)
87.9	76.2	55.0	J48_w (DT)	112.6	70.1	38.0	sddaLDA_R (DA)
88.0	76.0	51.7	PART_t (DT)	113.1	61.0	38.2	kernelpls_R (PLSR)
89.0	76.1	52.4	DTNB_w (RL)	113.3	68.2	39.5	sparseLDA_R (DA)
89.5	75.8	54.8	PART_w (DT)	113.5	70.1	46.5	NBUpdateable_w (BY)
90.2	76.6	48.5	RBFNetwork_w (NNET)	113.5	70.7	39.9	stepLDA_t (DA)
90.5	67.5	45.8	bagging_R (BAG)	114.8	58.1	32.4	bayesglm_t (GLM)
91.2	74.0	50.9	rpart_t (DT)	115.8	70.6	46.4	QdaCov_t (DA)
91.5	74.0	48.9	ctree_t (DT)	116.0	69.5	39.6	stepQDA_t (DA)
91.7	76.6	54.1	NNge_w (NN)	118.3	67.5	34.3	sddaQDA_R (DA)
92.4	72.8	48.5	ctree2_t (DT)	118.9	72.0	45.9	NaiveBayesSimple_w (BY)
93.0	74.7	50.1	FilteredClassifier_w (OM)	120.1	55.3	33.3	gpls_R (PLSR)
93.1	74.8	51.4	JRip_w (RL)	120.8	57.6	32.5	glmStepAIC_t (GLM)
93.6	75.3	51.1	REPTree_w (DT)	122.2	63.5	35.1	AdaBoostM1_w (BST)
93.6	74.7	52.3	rpart2_t (DT)	122.7	68.3	39.4	LWL_w (OEN)
94.3	75.1	50.7	BayesNet_w (BY)	126.1	50.8	30.5	glm_R (GLM)
94.4	73.5	49.5	rpart_R (DT)	126.2	65.7	44.7	dpp_C (NNET)
94.5	76.4	54.5	IB1_w (NN)	129.6	62.3	31.8	MAB_w (BST)
94.6	76.5	51.6	Ridor_w (RL)	130.9	64.2	33.2	BG_OneR_w (BAG)
95.1	71.8	48.7	lvq_R (NNET)	130.9	62.1	29.6	MAB_IBk_w (BST)
95.3	76.0	53.9	IBk_w (NN)	132.1	63.3	36.2	OneR_t (RL)
95.3	73.9	45.8	Dagging_w (OEN)	133.2	64.2	34.3	MAB_OneR_w (BST)
96.0	74.4	50.7	qda_t (DA)	133.4	63.3	33.3	OneR_w (RL)
96.5	71.9	48.1	obliqueTree_R (DT)	133.7	61.8	28.3	BG_DecisionStump_w (BAG)
97.0	68.9	42.0	plsBag_R (BAG)	135.5	64.9	42.4	VFI_w (OM)
97.2	73.9	52.1	OCC_w (OEN)	136.6	60.4	27.7	ConjunctiveRule_w (RL)
99.5	71.3	44.9	mlp_m (NNET)	137.5	60.3	26.5	DecisionStump_w (DT)
99.6	74.4	51.6	cascor_C (NNET)	138.0	56.6	15.1	RILB_w (BST)
99.8	75.3	52.7	bdk_R (NNET)	138.6	60.3	26.1	BG_HyperPipes_w (BAG)
100.8	73.8	48.9	nbBag_R (BAG)	143.3	53.2	17.9	spls_R (PLSR)
101.6	73.6	49.3	naiveBayes_R (BY)	143.8	57.8	24.3	HyperPipes_w (OM)
103.2	72.2	44.5	slda_t (DA)	145.8	53.9	15.3	BG_MLP_w (BAG)
103.6	72.8	41.3	pam_t (OM)	154.0	49.3	3.2	Stacking_w (STC)
104.5	62.6	33.1	nnetBag_R (BAG)	154.0	49.3	3.2	Grading_w (OEN)
105.5	72.1	46.7	DecisionTable_w (RL)	154.0	49.3	3.2	CVPS_w (OM)
106.2	72.7	48.0	MAB_NaiveBayes_w (BST)	154.1	49.3	3.2	StackingC_w (STC)
106.6	59.3	71.7	logitboost_R (BST)	154.5	49.2	7.6	MetaCost_w (BAG)
106.8	68.1	41.5	PenalizedLDA_R (DA)	154.6	49.2	2.7	ZeroR_w (RL)
107.5	72.5	48.3	NaiveBayes_w (BY)	154.6	49.2	2.7	MultiScheme_w (OEN)
108.1	69.4	44.6	rbf_m (NNET)	154.6	49.2	5.6	CSC_w (OEN)
108.2	71.5	49.8	rrlda_R (DA)	154.6	49.2	2.7	Vote_w (OEN)
109.4	65.2	46.5	vbmpRadial_t (BY)	157.4	52.1	25.13	CVC_w (OM)
110.0	73.9	51.0	RandomTree_w (DT)				

Table 4: Continuation of Table 3. ASC = AttributeSelectedClassifier, BG = Bagging, CSC = CostSensitiveClassifier, CVPS = CVParameterSelection, CVC = ClassificationViaClustering, CVR = ClassificationViaRegression, MAB = MultiBoostAB, MCC = MultiClassClassifier, MLP = MultilayerPerceptron, NBUpdeatable = NaiveBayesUpdateable, OCC = OrdinalClassClassifier, RILB = RacedIncrementalLogitBoost.

set may vary with respect to previous papers in the literature due to resampling differences. Although a leave-one-out validation might be more adequate (because it does not depend

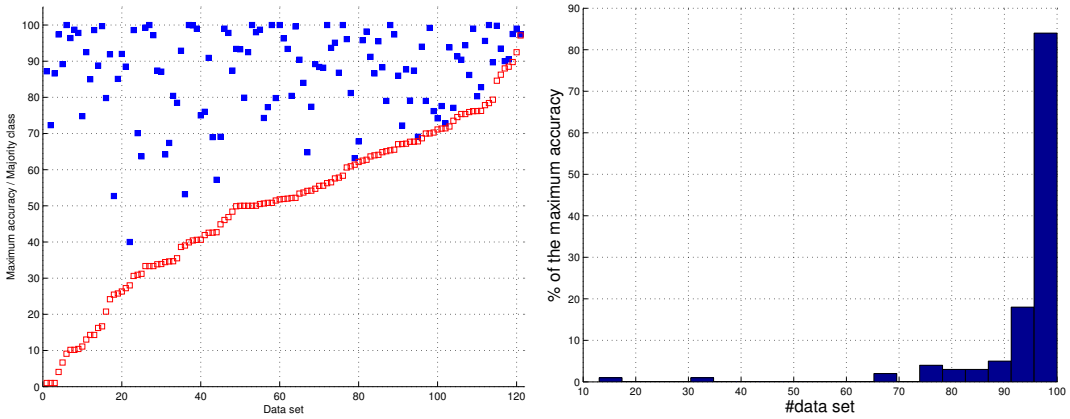

Figure 1: **Left**: Maximum accuracy (blue) and majority class (red), both in % ordered by increasing %Maj. for each data set. **Right**: Histogram of the accuracy achieved by parRF_t (measured as percentage of the best accuracy for each data set).

on the data partitioning), specially for the small data sets, it would not be feasible for some other larger data sets included in this study.

3.1 Average Accuracy and Friedman Ranking

Given its huge size (21,659 entries), the table with the complete results[11] is not included in the paper. Taking into account all the trials developed for parameter tuning in many classifiers (number of tunable parameters and number of values used for tuning), the total number of experiments is 241,637. The average accuracy for each classifier is calculated excluding the data sets in which that classifier found errors (denoted as -- in the complete table). The Figure 1 (left panel) plots, for each data set, the percentage of majority class (see columns %Maj. in Tables 1 and 2) and the maximum accuracy achieved by some classifier, ordered by increasing %Maj. Except for very few unbalanced data sets (with very populated majority classes), the best accuracy is much higher than the %Maj. (which is the accuracy achieved by classifier ZeroR_w). The Friedman ranking (Sheskin, 2006) was also computed to statistically sort the classifiers (this rank is increasing with the classifier error) taking into account the whole data set collection. Given that this test requires the same number of accuracy values for all the classifiers, in the error cases we use (only for this test) the average accuracy for that data set over all the classifiers.

The Tables 3 and 4 report the Friedman ranking, the average accuracy and the Cohen κ (Carletta, 1996), which excludes the probability of classifier success by chance, for the 179 classifiers, ordered following the Friedman ranking. The **best classifier is parRF_t (parallel random forest implemented in R using the randomForest and caret packages)**, with rank 32.9, average accuracy 82.0%(±16.3) and κ=63.5%(±30.6), **followed by rf_t (random forest using the randomForest package and tuned with caret)**, with rank 33.1 and the highest accuracy 82.3%(±15.3) and κ=63.6(±30.0). This result is

11. See `http://persoal.citius.usc.es/manuel.fernandez.delgado/papers/jmlr/results.txt`.

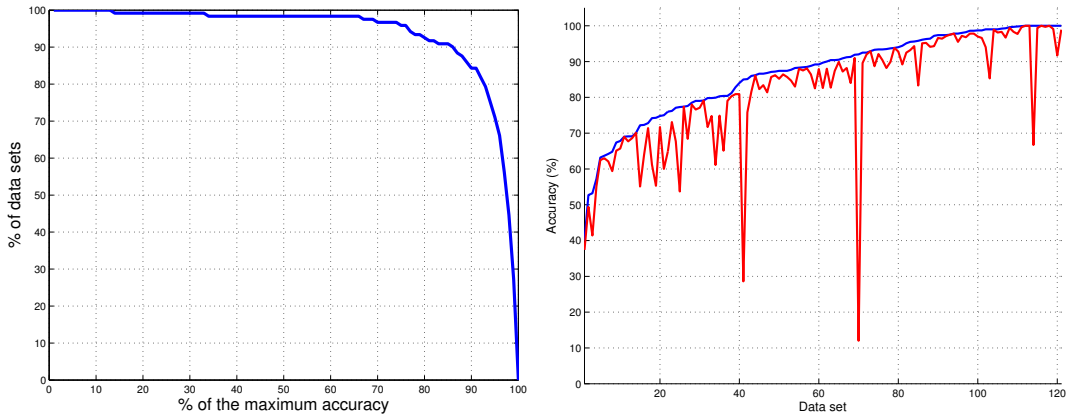

Figure 2: **Left**: for each % of the maximum accuracy in the horizontal axis, the vertical axis shows the percentage of data sets for which parRF_t overcomes that % of the maximum accuracy. **Right**: Accuracy (in %) achieved by parRF_t (in red) and maximum accuracy (in blue) for each data set (ordered by increasing maximum accuracies).

somehow surprising, because Random Forest is an old method, but it works better than other newer classifiers. The high deviations in accuracies and κ are expected, due to the large amount and variability of data sets. Since parRF_t is a parallel version of rf_t, using different random seeds, the difference between both can be considered not significant: parRF_t achieves better Friedman ranking, while rf_t achieves better accuracy and κ. Similar situations arise with other couples of classifiers within the same family, which are slightly different versions of the same classifier or versions with/without parameter tuning (svmRadial_t and svmRadialCost_t, lda_R and lda2_t, among others), with similar results, being the difference between them caused by noise, random initializations, etc. The parRF_t is the best classifier in 12 out of 121 data sets, and its average accuracy is 4.9% below the maximum average accuracy (i.e., the maximum accuracy over all the classifiers for each data set, averaged over all the data sets), which is 86.9%. It is very significant (and it can not be casual) that, among so many classifiers (179), the two bests ones (parRF_t and rf_t, according both to average accuracy and Friedman rank) are random forests implemented with the randomForest package and tuned with caret: this fact shows a clear superiority with respect to the remaining classifiers. It is also interesting that an "old" classifier as RF works better than many other, more recent, approaches. The Figure 1 (right panel) shows that for the majority of the data sets (specifically for 102 out of 121, which represents the 84.3%), the parRF_t achieves more than 90% of the maximum accuracy, being very near to the best accuracy for almost all the data sets. The Figure 2 (left panel) plots, for each % of the maximum accuracy in the horizontal axis, the % of data sets for which parRF overcomes that percentage: for the 93% (resp. for 84.3%) of the data sets parRF achieves more than 80% (resp. 90%) of the maximum accuracy. In this figure, the area under curve (AUC) of the three bests classifiers (parRF_t, rf_t and svm_C) are 0.9349, 0.9382 and 0.9312 respectively, being rf_t slightly better than parRF_t (as the accuracy in Table 3) and

svm_C slightly worse. As we commented in the introduction, given the large number of classifiers used in this work, it is reasonable to estimate the maximum attainable accuracy for a data set as the maximum accuracy achieved by some classifier. Therefore, although the No-Free-Lunch theorem states that no classifier can be always the best, in the practice, parRF_t is very near to the best attainable accuracy for almost all the data sets. Specifically, the Figure 2 (right panel) shows that parRF is very near to the maximum accuracy for almost all the data sets, excepting three data sets: #41 (image-segmentation, 33.6%), #70 (audiology-std, 13.0%) and #114 (balloons, 66.7%).

The third best classifier is svm_C (LibSVM with Gaussian kernel), with rank (36.8, two points above parRT_t) and average accuracy 81.8%($\pm$16.2). The following classifiers are: svmPoly_t (SVM with polynomial kernel, rank 38.0), rforest_R (random forest without mtry tuning, rank 39.4), elm_kernel_m (extreme learning machine, rank 39.6), svmRadialCost_t (40.3), svmRadial_t (42.5), C5.0_t (42.9) and avNNet_t (44.1). It may not be a casuality the presence of three RF and two SVM among the five best classifiers, identifying both classifier families as the best ones. Besides, there are also two neural networks and one boosting ensemble (C5.0_t) among the top-10. The Figure 3 shows the 25 classifiers with the lowest Friedman ranks (upper panel) and the classifiers with the highest average accuracies (lower panel): parRF_t and rf_t have ranks clearly lower than svm_C and the following classifiers. In fact, the highest increment (3.7) between two classifier ranks is between rf_t and svm_C, which shows that parRF_t and rf_t are clearly better than the remaining classifiers in the plot. Besides, rf_t, parRF, svm_C, rforest_R and elm_kernel_m have higher accuracies than the others (the largest accuracy reduction, 0.37, is between svm_C and svmRadialCost_t). Our proposal dkp_C is in the 23th (resp. 21th) position according to the Friedman ranking (resp. to the accuracy, 79.7%), but this apparently good result is somehow obscured by the low value of κ (42.6%). It is caused by some data sets where dkp_C assigns all the patterns to the most populated class: for these data sets, $\kappa = 0$, which reduces the average κ over all the data sets.

We developed **paired T-tests** comparing the accuracies of parRF_t and the following 9 classifiers in Table 3 (the null hypothesis is that the two accuracies compared are not significantly different, so that, within a tolerance $\alpha = 0.05$, when $p < 0.05$ parRF_t is significantly better than the other classifier. The Figure 4 (left panel) plots the T-statistic, 95%-limits and p-values, showing that parRF_t is only significantly better (high T-statistic, $p < 0.05$) than with C5.0_t and avNNet_t. Although parRF_t is better than svm_C in 56 of 121 data sets, worse than svm_C in 55 sets, and equal in 10 sets, the Figure 4 (right panel) compares their percentages of the maximum accuracy for each data set (ordered by increasing percentages): for the majority of the data sets they are almost 100% (i.e., parRF_t and svm_C are near to the maximum accuracy). Besides, svm_C is never much better than parRF_t: when svm_C outperforms parRF_t, the difference is small, but when parRF_t outperforms svm_C, the difference is higher (data sets 1-20). In fact, calculating for each data set the difference between the accuracies of parRF_t and svm_C, the sum of positive differences (parRF is better) is 193.8, while the negative ones (svm_C better) sum 139.8.

All the classifiers of the random forest and SVM families are included among the 25 best classifiers, with accuracies above 79% (while the best is 82.3%), which identify both families as the best ones. Other classifiers included among the top-20, not belonging to RF

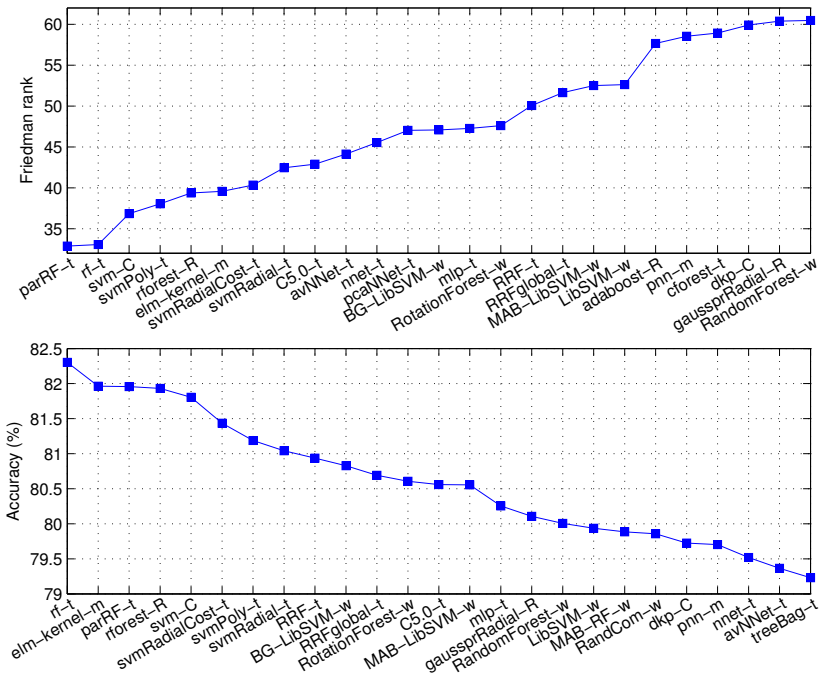

Figure 3: Friedman rank (upper panel, increasing order) and average accuracies (lower panel, decreasing order) for the 25 best classifiers.

and SVM families, are nnet_t (MLP network, rank 45.5), pcaNNet_t (MLP + PCA network, rank 47.0), Bagging_LibSVM_w (ensemble of Gaussian LibSVMs, rank 47.1), mlp_t (RSNNS MLP with tunable network size, rank 38.0), MultiBoostAB_LibSVM_w (MultiBoostAB ensemble of Gaussian LibSVMs, rank 52.5) and adaboost_R (Adaboost.M1 ensemble of decision trees, rank 57.6). Beyond the 20th position are pnn_m (Probabilistic Neural Network with tunable Gaussian spread, rank 58.5), and our proposal dkp_C (rank 59.9). Besides, note that 12 classifiers in the top-20 use caret, which might be due to the automatic parameter tuning (only rforest_R and adaboost_R have no tunable parameter). We must emphasize that, since parameter tuning and testing use different data sets, the final result can not be biased by parameter optimization, because the set of parameter values selected in the tuning stage is not necessarily the best on the test set. In some cases, the tuning is not relevant: for C5.0_t the differences among the performances using different parameter values are low, so it would work similarly without parameter tuning.

OpenML Vanschoren et al. (2012) uses only 86 data sets and 93 classifiers, while our work is much wider (121 and 179, respectively), including Weka classifiers in a later version (3.6.9), for example openML uses Bagging 1.31.2.2, while we use Bagging version 6502. Besides, as we commented above, we do not use 9 of 93 classifiers included in the previous reference. The results in Figure 17 of that paper rank Bagging-NBayesTree as the best classifier, followed by Bagging-PART, SVM-Polynomial, MultilayerPerceptron, Boosting-NBayesTree, RandomForest, Boosting-PART, Bagging-C45, Boosting-C45 and SVM-RBF. However, in our results the best Weka classifiers (in the top-20) are Baggging_LibSVM_w, RotationFor-

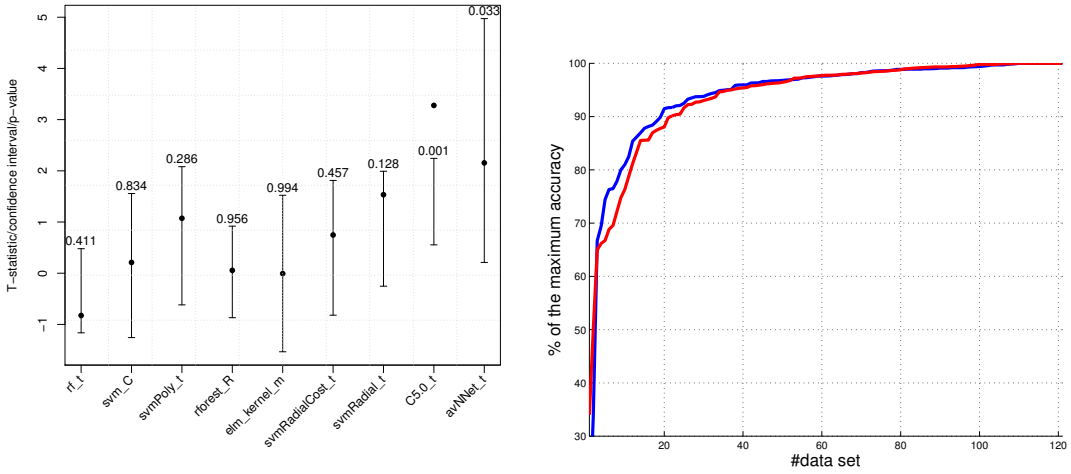

Figure 4: **Left panel**: T-statistics (point), confidence intervals and p-values (above upper interval limits) of the T-tests comparing parRF_T and the remaining 9 best classifiers. **Right panel**: Percentage of the maximum accuracy achieved by parRF_t (blue) and svm_C (red) for the 121 data sets (ordered by increasing percentage)

est_w, MultiBoostAB_LibSVM_w and LibSVM_w, i.e., a Random Forest, a SVM and two ensembles of SVMs. This is expected, because it is known that ensembles of strong classifiers do not work better than the single classifier. Therefore, Bagging and MultiBoostAB of LibSVM do not work better than LibSVM_w, although the three are worse than svm_C (the same Gaussian LibSVM in C) and the caret SVM versions (svmPoly_t, svmRadialCost_t and svmRadial_t). Besides, similarly to openML, in our work svmPoly_t (polynomial kernel) is near to svm_C (Gaussian kernel). However, in our results Bagging_NaiveBayes_w works very bad (rank 110.4, Table 4), while other Baggging ensembles are better: Bagging_PART_w (66.6), MultilayerPerceptron_w (70.6), MultiBoostAB_NaiveBayes_w (equivalent to Boosting-NaiveBayes in openML, rank 106.2) and MultiBoostAB_PART_w (65.6). Therefore, in our experiments the bagging and multiBoostAB ensembles (except of LibSVM_w) do not work well. We use the same configurations for bagging (10 bagging iterations, 100% of the training set for bag size, changing only the base learner) and MultiBoostAB (3 sub-committees, 10 boost iterations and 100% of build mass used to build classifiers) as openML, so these bad results can not be caused by improper configuration (or parameter tuning) of the ensemble or base classifier. Therefore, they might be caused by the larger number of data sets, or by the inclusion in our collection of other classifiers and implementations (in R, caret, C and Matlab), with better accuracies, not considered by OpenML.

No.	Classifier	PAMA	No.	Classifier	PAMA
1	elm_kernel_m	13.2	11	mlp_t	5.0
2	svm_C	10.7	12	pnn_m	5.0
3	parRF_t	9.9	13	dkp_C	5.0
4	C5.0_t	9.1	14	LibSVM_w	5.0
5	adaboost_R	9.1	15	svmPoly_t	5.0
6	rforest_R	8.3	16	treebag_t	5.0
7	nnet_t	6.6	17	RRFglobal_t	5.0
8	svmRadialCost_t	6.6	18	svmlight_C	5.0
9	rf_t	5.8	19	Bagging_RandomForest_w	4.1
10	RRF_t	5.8	20	mda_t	4.1

No.	Classifier	P95	No.	Classifier	P95
1	parRF_t	71.1	11	elm_kernel_m	60.3
2	svm_C	70.2	12	MAB-LibSVM_w	60.3
3	rf_t	68.6	13	RandomForest_w	57.0
4	rforest_R	65.3	14	RRF_t	56.2
5	Bagging-LibSVM_w	63.6	15	pcaNNet_t	55.4
6	svmRadialCost_t	63.6	16	RotationForest_w	54.5
7	svmRadial_t	62.8	17	avNNet_t	53.7
8	svmPoly_t	62.8	18	nnet_t	53.7
9	LibSVM_w	62.0	19	RRFglobal_t	53.7
10	C5.0_t	61.2	20	mlp_t	52.1

No.	Classifier	PMA	No.	Classifier	PMA
1	parRF_t	94.1	11	RandomCommittee_w	91.4
2	rf_t	93.6	12	nnet_t	91.3
3	rforest_R	93.3	13	avNNet_t	91.1
4	C5.0_t	92.5	14	RRFglobal_t	91.0
5	RotationForest_w	92.5	15	knn_R	90.5
6	svm_C	92.3	16	Bagging-LibSVM_w	90.5
7	mlp_t	92.1	17	Bagging_REPTree_w	90.4
8	LibSVM_w	91.7	18	MAB_MLP_w	90.4
9	RRF_t	91.4	19	elm_m	90.3
10	dkp_C	91.4	20	rda_R	90.3

Table 5: **Up**: list of the 20 classifiers with the highest Probabilities of Achieving the Maximum Accuracies (PAMA, in %). **Middle**: List of the 20 classifiers with the highest probabilities of achieving 95% (P95) of the maximum accuracy over all the data sets. **Down**: Classifiers sorted by its Percentage of the Maximum Accuracy (PMA) for each data set, averaged over all the data sets. MAB means MultiBoostAB.

3.2 Probability of Achieving the Best Accuracy

One of the objectives of this paper (Section 1) is to estimate, for each classifier, the **Probability of Achieving the Maximum Accuracy** (**PAMA**) for a given data set, as the number of data sets for which it achieves the highest accuracy, divided by the number of data sets. The Table 5 (upper part) shows the 20 classifiers with the highest values for these probabilities (in %), being elm_kernel_m the best (for 13.2% of the data sets) followed by svm_C (10.7%) and parRF (9.9%). These values are very far from 100%, which confirms

that no classifier is the best for most data sets (following the No-Free-Lunch theorem). The C5.0_t and adaboost_R have about 9%. The remaining classifiers are about 4-8%, so that many classifiers are the best for only few data sets. There are 5 classifiers of family RF, 5 SVM, 5 NNET and 4 ensembles among the 20 classifiers with the highest probabilities of being the best. Our proposal dkp_C achieves the 13th position.

The PAMA does not take into account that a classifier may be very near from the best accuracy without being the best one. Therefore, an alternative, more significant, measure is the **probability of achieving more than 95% of the maximum accuracy** (**P95**) (middle part of Table 5 for the best 20 values). This probability (in %), for a given classifier, is estimated dividing the number of data sets in which it achieves 95% or more of the maximum accuracy (achieved by any other classifier on that data set), by the number of data sets. The ten classifiers with the highest P95 are almost the same as in the Friedman rank, with a different order. In this table, ParRF_t achieves more than 95% of the maximum accuracy for 71.1% of the data sets (again far from the 100%), followed by svm_C (70.2%) and rf_t (68.6). The other classifiers have P95 below 65%. The low P95 of elm_kernel_w (60.3%, 11th position), being the best for a highest number of data sets, shows a behavior less stable than rf_t, parRF_t and svm_C, because its accuracy in the other data sets is lower in average.

Another interesting measurement is the **Percentage of the Maximum Accuracy** (**PMA**) achieved by each classifier, averaged over the whole collection of data sets (the 20 first are shown in the lower part of the Table 5). Again, parRF_t is the best achieving 94.1%($\pm$11.3) of the maximum accuracy, followed by other two Random Forests: rf_t and rforest_R (93.6% and 93.3% respectively). The svm_C is in the 6th position, with PMA 92.3%($\pm$15.9). Note that six out of eight Random Forest classifiers are in the top-20. The PMA values are high, very near to, but below, the threshold of 95% used in the middle part of Table 5. This explains the low values of P95: the bests classifiers have PMA about 94%, so their probability of achieving 95% or more of the maximum accuracy is low (about 70%). Setting the threshold in 90% of the maximum accuracies, the corresponding probabilities would be much higher. The elm_kernel_m is not included in this table: this confirms its unstable behavior, because in average it does not achieve PAM above 90.3% (even elm_m, without kernels, has better PMA). The mlp_t (92.2%) has also a good value. The dkp_C is the 10th position, achieving in average the 91.4%, only 2.7 below the best. The 20 classifiers are in a narrow margin between 90%-94% of the maximum accuracy, so that there are many classifiers which a high percentage of the maximum accuracy. The Figure 5 shows that the three Random Forests (parRT_t, rf_t and rforest_R) achieve PMAs clearly higher than the remaining classifiers (including svm_C), being the greatest gap (0.8) between rforest_R and C5.0_t.

3.3 Discussion by Classifier Family

The Figure 6 compares the classifier families showing in the upper panel the error bars with the mean (blue square), minimum and maximum values of the Friedman ranks for each family. The lower panel shows the minimum rank (corresponding to the best classifier) for each family, by ascending order. The family RF has the lowest minimum rank (32.9) and mean (46.7), and also a narrow interval (up to 60.5), which means that all the RF classifiers

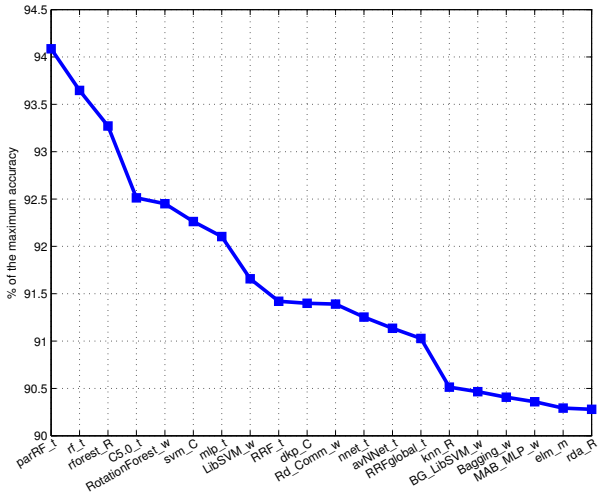

Figure 5: Twenty classifiers with the highest percentages of the maximum accuracy. MAB means MultiBoostAB, BG means Bagging.

work very well. The SVM has the following minimum (36.8), but the mean is much higher (55.4), and the interval is also much wider (up to 81.6). The third best type is NNET, whose minimum and mean rank are 39.6 (elm_kernel_m) and 73.8 respectively. The DTs have the following minimum (42.9), followed by BAG (47.1), BST (52.5), OM (other methods, specifically gaussprRadial, 60.4), DA (62.5), NN (62.6), OEN (other ensembles, specifically RandomCommittee_w, 63.0), LMR (65.2), MARS and GLM (65.5), PLSR (79.1), RL (81.4), BY (94.3) and STC (154.0). We can make three family groups in the lower panel of Figure 6: a) the best ones (RF, SVM, NNET, DT, BAG and BST), with the lowest ranks (about 30-50); b) the intermediate families (OM, DA, NN, OEN, LMR, MARS and GLM), about 60-70; and c) the worst families (PLSR, RL, BY and STC), with ranks above 80.

Now, we discuss the results for each classifier family (see Tables 6 and 7). The **discriminant analysis** (**DA**) classifiers work relatively well, being fda_t the best one, followed by rda_R, mda_t and pda_t. The lda_R works better than the caret version lda2_t (74.4), which however tunes of the number of retained components. In other DA classifiers (fda and mda) the parameter tuning developed in the caret versions allows to achieve better accuracies than their R counterparts (without tuning). It is surprising that sophisticated versions of LDA are worse: slda_t, PenalizedLDA_t, rrlda_R, sddaLDA_R, sparseLDA_R and stepLDA_t. Finally, the QDA classifiers are very bad, achieving again the classical qda_t the best results compared to more advanced versions (QdaCov_t, stepQDA_t and sddaQDA_R). The **Bayesian methods** (**BY**) are clearly worse than DA, and they are not competitive at all to the globally best classifiers, achieving the best (BayesNet_w) a high rank (94.3).

Among the **neural networks** (**NNET**), the elm_kernel_m is the best one, followed by several caret MLP implementations (avNNet_t, nnet_t, pcaNNet_t and mlp_t), included in the top-20, better than other MLP implementations: mlp_C (LibFANN), MultilayerPerceptron_w (Weka) and mlp_m (Matlab). The good result of avNNet (an ensemble of 4 small MLPs with up to 9 hidden neurons whose weights are randomly initialized), compared to

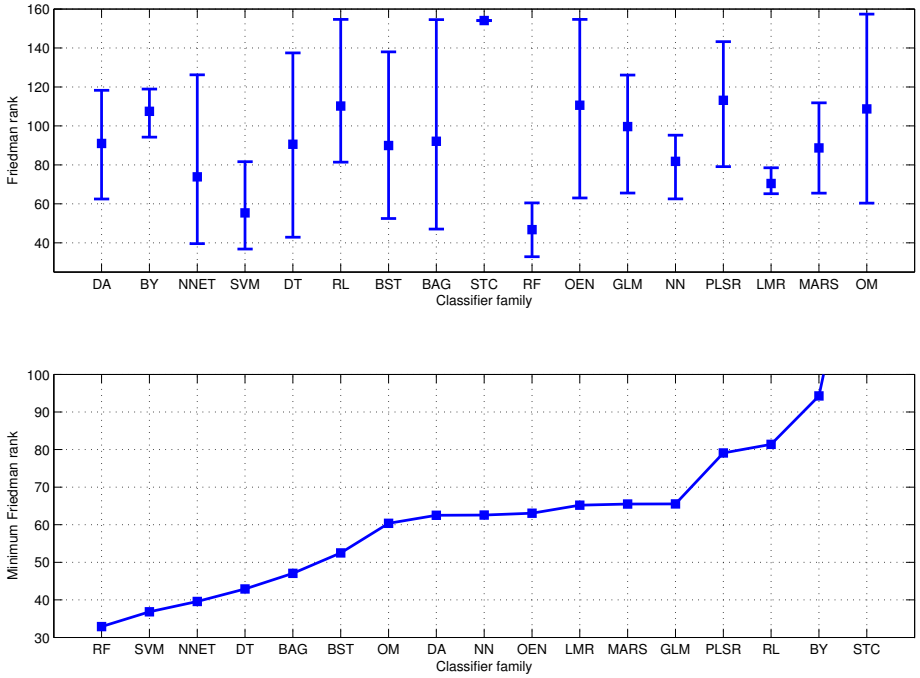

Figure 6: Friedman rank interval for the classifiers of each family (upper panel) and minimum rank (by ascending order) for each family (lower panel).

greater MLPs, as mlp_C and mlp_m (up to 30 hidden neurons), is due to its ensemble nature, because mlpWeightDecay_t also has up to 9 hidden neurons, with worse results. The rule used for size selection by MultilayerPerceptron_w (#inputs + #classes)/2 does not achieve good results. The pnn_m (probabilistic neural network) and our proposal dkp_C (direct kernel perceptron) are very near to the top-20. The bad results of elm_m (67.6) are surprising taking into account the good behavior of the Gaussian elm_kernel_w. Similarly, the LVQ versions are not good: lvq_t, which tunes the size and k, works much better than lvq_R, being bdk_R (99.8) the worst one. The cascor_C (cascade correlation), which uses LibFANN, is also worse (99.6) than the best MLP version (avNNet_t). Finally, the RBF networks are also bad, although the caret versions outperform the Weka and Matlab versions. The dpp_C is not competitive at all with the other networks.

The svm_C, with Gaussian kernel using LibSVM is the best **Support vector machine** (**SVM**), followed by the caret versions svmPoly_t (polynomial kernel), svmRadialCost_t and svmRadial_t (Gaussian kernel), better than the Weka versions LibSVM_w and SMO_w and that svmlight_C. The linear kernel versions (svmLinear_t and LibLINEAR_w) are clearly worse, and lssvmRadial_t is the worst one. Overall, the ten SVM classifiers achieve very good results, with ranks in the (relatively narrow) interval 36.8—72.4 (excluding linear kernels).

RandomSubSpace_w is the best **decision tree** (**DT**), with a bad rank (74.0); both J48_t and J48_w achieve similar results (the former runs the latter in the RWeka package tuned with caret). The best **Rule-based** (**RL**) classifiers are C5.0Rules_t and JRip_t,

Discriminant analysis (DA)					
1	fda_t	62.5	11	qda_t	96.0
2	rda_R	63.8	12	slda_t	103.2
3	mda_t	66.7	13	PenalizedLDA_t	106.8
4	pda_t	67.3	14	rrlda_R	108.2
5	fda_R	72.1	15	sddaLDA_R	112.6
6	lda_R	72.4	16	sparseLDA_R	113.3
7	lda2_t	74.4	17	stepLDA_t	113.5
8	sda_t	76.5	18	QdaCov_t	115.8
9	hdda_R	79.8	19	stepQDA_t	116.0
10	mda_R	80.9	20	sddaQDA_R	118.3
Bayesian methods (BY)					
1	BayesNet_w	94.3	4	vbmpRadial_t	109.4
2	naiveBayes_R	101.6	5	NBUpdateable_w	113.5
3	NaiveBayes_w	107.5			
Neural networks (NNET)					
1	elm_kernel_m	39.6	12	rbf_t	72.0
2	avNNet_t	44.1	13	rbfDDA_t	75.9
3	nnet_t	45.5	14	lvq_t	79.1
4	pcaNNet_t	47.0	15	RBFNetwork_w	90.2
5	mlp_t	47.3	16	lvq_R	95.1
6	pnn_m	58.5	17	mlp_m	99.5
7	dkp_C	59.9	18	cascor_C	99.6
8	mlp_C	62.8	19	bdk_R	99.8
9	mlpWeightDecay	63.6	20	rbf_m	108.1
10	elm_m	67.6	21	dpp_C	126.2
11	MultilayerPerceptron_w	70.6			
Support vector machines (SVM)					
1	svm_C	36.8	6	svmLinear_t	62.1
2	svmPoly_t	38.0	7	SMO_w	62.8
3	svmRadialCost_t	40.3	8	svmlight_C	72.4
4	svmRadial_t	42.5	9	LibLINEAR_w	74.6
5	LibSVM_w	52.6	19	lssvmRadial_t	81.6

Table 6: Friedman ranks of the classifiers in each family (continued in Table 7).

slightly worse than the best DT. The difference between JRip_t and JRip_w suggests that the tuning of the number of optimization runs developed in the caret version, but not with Weka, is important. ZeroR_w is among the worst ones, because it only predicts the same mean class for every test pattern: we included it to define the "zero-level" for the accuracy (49.2%, there is no classifier with lower accuracy).

Among the **boosting** (**BST**) ensembles, C5.0_t is the best (position 9), followed by MultiBoostAB_LibSVM (position 18), adaboost_R (position 19) and other MultiBoostAB ensembles with strong base classifiers (MultilayerPerceptron, RandomForest, PART and J48), while the ones with weak classifiers (OneR, IBk, DecisionStump, NaiveBayes, among others) are worse. The Figure 7 (upper panel, only Weka ensembles and base classifiers are plotted) shows that MultiBoostAB ensembles achieves much lower ranks than their corresponding base classifiers, excepting LibSVM, RandomForest and Logistic, where both are similar, and IBk, where the base classifier works much better. The same happens with

Decision trees (DT)					
1	RandomSubSpace_w	74.0	9	ctree2_t	92.4
2	C5.0Tree_t	84.2	10	REPTree_w	93.6
3			11	rpart2_t	93.6
4	NBTree_w	84.6	12	rpart_R	94.4
5	J48_t	87.6	13	obliqueTree_R	96.5
6	J48_w	87.9	14	RandomTree_w	110.0
7	rpart_t	91.2	15	DecisionStump_w	137.5
8	ctree_t	91.5			
Rule-based classifiers (RL)					
1	C5.0Rules_t	81.4	7	Ridor_w	94.6
2	JRip_t	81.7	8	DecisionTable_w	105.5
3	PART_t	88.0	9	OneR_t	132.1
4	DTNB_w	89.0	10	OneR_w	133.4
5	PART_w	89.5	11	ConjunctiveRule_w	136.6
6	JRip_w	93.1	12	ZeroR_w	154.6
Boosting (BST)					
1	C5.0_t	42.9	11	MAB_RandomTree_w	76.6
2	MAB_LibSVM_w	52.5	12	MAB_Logistic_w	82.0
3	adaboost_R	57.9	13	MAB_DecisionTable_w	87.2
4	MAB_MultilayerPerceptron_w	64.0	14	MAB_NaiveBayes_w	106.2
5	MAB_RandomForest_w	64.1	15	logitboost_R	106.6
6	MAB_PART_w	65.6	16	AdaBoostM1_DecisionStump_w	122.2
7	MAB_J48_w	69.2	17	MAB_DecisionStump_w	129.6
8	AdaBoostM1_J48_w	72.6	18	MAB_IBk_w	130.9
9	MAB_REPTree_w	73.7	19	MAB_OneR_w	133.2
10	LogitBoost_w	76.6	20	RILB_w	138.0
Bagging (BAG)					
1	BG_LibSVM_w	47.1	13	BG_Logistic_w	79.0
2	treebag_t	66.4	14	BG_DecisionTable_w	84.6
3	BG_PART_w	66.6	15	bagging_R	90.5
4	Bagging_REPTree_w	69.8	16	plsBag_R	97.0
5	BG_RandomTree_w	71.0	17	nbBag_R	100.8
6	BG_J48_w	71.0	18	nnetBag_R	104.5
7	BG_IBk_w	72.7	19	BG_NaiveBayes_w	110.4
8	ldaBag_R	72.9	20	BG_OneR_w	130.9
9	BG_LWL_w	73.2	21	BG_DecisionStump_w	133.7
10	svmBag_R	74.6	22	BG_HyperPipes_w	143.8
11	BG_RandomForest_w	77.1	23	BG_MLP_w	145.8
12	ctreeBag_R	78.7	24	MetaCost_w	154.5

Table 7: Continuation of Table 6. MAB means MultiBoostAB. RILB means RacedIncrementalLogitBoost. BG means Bagging. Continued in Table 8.

AdaBoostM1 (J48 much better than DecisionStump). The adaboost_R (AdaboostM1 with classification trees) works very well (included in the top-20), while AdaBoostM1_J48_w and AdaBoostM1_DecisionStump_w work much worse: this big difference might be in the AdaboostM1 implementation or in the base classifiers. There is also difference between LogitBoost_w and logitboost_R, despite of using the same base classifier (DecisionStump):

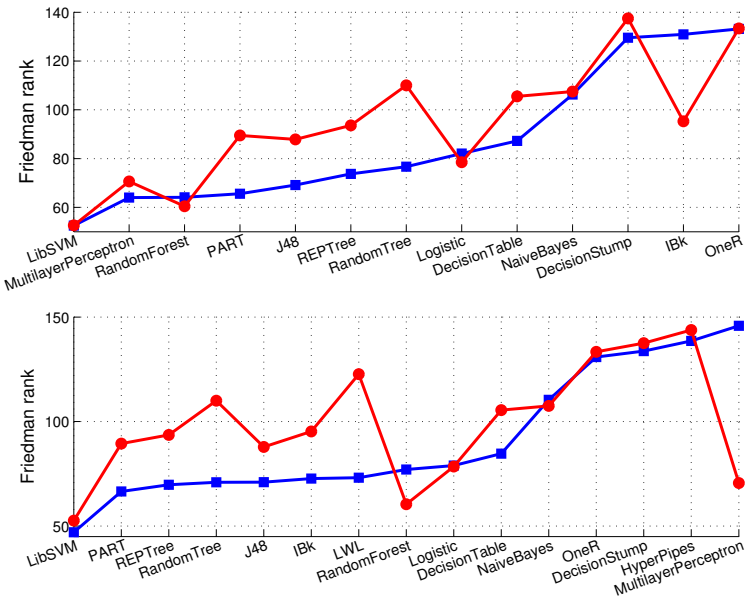

Figure 7: Upper panel: Friedman rank (ordered increasingly) of each Weka MultiBoostAB ensemble (blue squares) and its corresponding Weka base classifier (red circles). Lower panel: the same for Weka bagging ensembles (blue squares) and base classifiers (red circles).

the Weka implementation is clearly better. The RacedIncrementalLogitBoost_w is the worst one, despite of being a committee of LogitBoost.

The best **bagging (BG)** ensemble is also the Baggging_LibSVM_w (included in the top-20), although the svmBag_R is not so good, revealing big differences between implementations. The Figure 7 (lower panel) compares 15 Bagging ensembles to their respective base classifiers (both implemented in Weka), being the ensembles better except for RandomForest, NaiveBayes and MultilayerPerceptron. This means that RandomForest works better than in MultiBoostAB and Baggging ensembles. The remaining Bagging classifiers are not good: The ldaBag_R, ctreeBag_R, nbBag_R and nnetBag_R work also bad, similarly to their Weka correspondents (Bagging_NaiveBayes_w and Baggging_MultilayerPerceptron_w). Both **stacking** classifiers Stacking_w and StackingC_w work equally bad. The eight **random forest** classifiers are included among the 25 best classifiers having all of them low ranks, so this is clearly the best family of classifiers. Although one could think that there is a redundancy in RF models that might over-emphasize some results (parRF_t and rf_t are very similar classifiers), we must note that RRF_t (Regularized RF), RRFglobal_t (for which the caret documentation does not give differences with RRF_t, except in the tunable parameters) and cforest_t are different classifiers. Besides, the Weka RF implementations (RandomForest_w and RotationForest_w) are also among the 25 best classifiers, confirming that good positions of RF classifiers are not due to redundancy. Finally, none of the **other ensembles** (**OEN**) achieves good results, being RandomCommittee_w and Decorate_w the bests, but many of them are at the end of the list (rank 154.0).

Stacking (STC)					
1	Stacking_w	154.0	2	StackingC_w	154.1
Random forests (RF)					
1	parRF_t	32.9	5	RRF_t	50.1
2	rf_t	33.1	6	RRFglobal_t	51.6
3	rforest_R	39.4	7	cforest_t	58.9
4	RotationForest_w	47.6	8	RandomForest_w	60.5
Other ensembles (OEN)					
1	RandomCommittee_w	63.0	7	LWL_w	122.7
2	Decorate_w	63.4	8	Grading_w	154.0
3	END_w	76.6	9	MultiScheme_w	154.6
4	MultiClassClassifier_w	80.6	10	CostSensitiveClassifier_w	154.6
5	Dagging_w	95.3	11	Vote_w	154.6
6	OrdinalClassClassifier_w	97.2			
Generalized linear models (GLM)					
1	gmlnet_R	65.5	4	glmStepAIC_t	120.8
2	mlm_R	71.0	5	glm_R	126.1
3	bayesglm_t	114.8			
Nearest neighbors (NN)					
1	knn_t	62.6	4	IBk_w	94.5
2	knn_R	65.0	5	IB1_w	95.3
3	NNge_w	91.7			
Partial least squares and principal component regression (PLSR)					
1	pls_t	79.1	4	kernelpls_R	113.1
2	widekernelpls_R	111.3	5	gpls_R	120.1
3	simpls_R	111.9	6	spls_R	143.3
Logistic and multinomial regression(LMR)					
1	multinom_t	65.2	3	Logistic_w	78.5
2	SimpleLogistic_w	67.6			
Multivariate adaptive regression splines (MARS)					
1	gcvEarth_t	65.5	2	mars_R	111.9
Other methods (OM)					
1	gaussprRadial	60.4	6	pam_t	103.6
2	ClassificationViaRegression_w	66.0	7	VFI_w	135.5
3	AttributeSelectedClassifier_w	86.4	8	HyperPipes_w	143.8
4	KStar_w	87.2	9	CVParameterSelection	154.0
5	FilteredClassifier_w	93.0	10	ClassificationViaClustering_w	157.4

Table 8: Continuation of Tables 6 and 7.

The **GLM** classifiers are divided in two groups: gmlnet_R and mlm_R, with relatively good ranks (60-70), and the others, with much worse results. Something similar happens with **NN**, where the R and caret versions (knn_t and knn_R) are about 70, while the Weka variants NNge_w, IBk_w and IB1_w are much worse (about 90). With respect to the **PLSR** classifiers, the simplest one (pls_t) is the best, while the remaining, more sophisticated, versions are much worse. The three **LMR** classifiers achieve ranks about 65-75, being multinom_t the best one. The original **MARS** classifier (mars_R) is very bad, while the fast MARS version (gcvEarth_t) works much better. Finally, only the gaussprRadial_R and

ClassificationViaRegression_w achieve good results among the **Other methods**, while the remaining ones have ranks about 90 (AttributeSelectedClassifier_w, KStar_w and FilteredClassifier_w), and more, being some of the worse classifiers in the collection (ClassificationViaClustering_w).

Rank	Classifier	Acc. (%)	Rank	Classifier	Acc (%)
36.2	avNNet_t	83.0	50.0	mlp_t	82.2
39.9	svmPoly_t	79.9	51.4	elm_kernel_m	77.5
41.0	pcaNNet_t	82.9	54.1	RotationForest_w	82.0
42.2	svmRadialCost_t	80.0	54.9	rforest_R	80.9
44.2	parRF_t	82.6	57.6	mlpWeightDecay_t	79.7
44.7	rf_t	81.2	57.7	svmBag_R	78.8
47.1	C5.0_t	82.0	59.7	fda_t	81.0
47.2	svm_C	79.0	60.8	cforest_t	74.7
47.5	nnet_t	82.1	61.5	Bagging_LibSVM_w	77.9
48.0	svmRadial_t	79.4	62.9	knn_t	80.4

No.	Classifier	P95	No.	Classifier	P95
1	svmRadialCost_t	78.2	11	MultiBoostAB_LibSVM_w	65.5
2	svm_C	74.5	12	pcaNNet_t	63.6
3	svmPoly_t	74.5	13	svmBag_R	63.6
4	svmRadial_t	72.7	14	elm_kernel_m	61.8
5	Bagging_LibSVM_w	70.9	15	nnet_t	61.8
6	avNNet_t	69.1	16	RotationForest_w	61.8
7	parRF_t	69.1	17	fda_t	60.0
8	LibSVM_w	67.3	18	mlp_t	60.0
9	C5.0_t	67.3	19	MultiBoostAB_REPTree_w	58.2
10	rf_t	67.3	20	RandomForest_w	58.2

No.	Classifier	PMA	No.	Classifier	PMA
1	avNNet_t	95.0	11	pda_t	92.8
2	pcaNNet_t	94.9	12	mlm_R	92.7
3	parRF_t	94.3	13	fda_t	92.7
4	nnet_t	94.1	14	MAB_MLP_w	92.7
5	mlp_t	94.1	15	bayesglm_t	92.6
6	C5.0_t	93.8	16	simpls_R	92.5
7	RotationForest_w	93.7	17	rforest_R	92.5
8	glmnet_R	93.5	18	MultiBoostAB_PART_w	92.5
9	rda_R	93.2	19	fda_R	92.3
10	rf_t	92.8	20	nnetBag_R	92.2

Table 9: Results for **two class** data sets. **Up**: Friedman rank and average accuracies for the 20 best classifiers. RF_w = RotationForest_w. MWD_t = mlpWeightDecay_t. **Middle**: Probability (in %) of achieving 95% or more of the maximum accuracy. **Down**: 20 classifiers with the highest average Percentage of the Maximum Accuracy (PMA) over the two-class data sets. MAB_MLP_w means MultiBoostAB_MultilayerPerceptron_w.

3.4 Two-Class Data Sets

Since 45.4% of the data sets (55 out of 121) have only two classes, it is interesting to see what happens when only 2-class data sets are considered. We repeated our analysis of the Subsections 3.1 and 3.2, calculating the Friedman rank and the average accuracy, alongside with the P95 and PMA, for all the classifiers and two-class data sets. Although it should be recommendable, we did not use the area under ROC curve as quality measure, nor develop cutoff tuning (Kuhn and Johnson, 2013), because some classifiers do not give probabilistic output. The Table 9 reports the results:

- The **upper part** shows the 20 classifiers with the best Friedman rank (calculated using only 2-class data sets), alongside with their average accuracies. The classifiers in this new list are approximately the same as in the top-20 of Table 3, but the order is different: avNNet_t (rank 36.2) is now the best, while the parRF_t, rf_t and svm_C (the three bests ones in Table 3) are now the 5th, 6th and 8th respectively. Besides, the best average accuracy (83.0%) is almost the same as in Table 3 (82.3%), so the classification results are not globally better for two class problems. Except the C5.0_t, all the classifiers in the top-10 are NMP neural networks, SVMs and Random Forests. As well, these families occupy 6 places in positions 11-20. Besides, 14 of 20 classifiers use caret. The elm_kernel_m is worse than in Table 3.

- The **middle part** reports the probabilities (in %) of achieving 95% or more of the maximum accuracy (P95). The best one is 78.2% (svmRadialCost), higher than in Table 3 (71.1%, parRF_t). The first four classifiers are SVMs, while parRF_t and rf_t are in 7th and 10th positions. The avNNet_t, Baggging_LibSVM_w, LibSVM_w and C5.0_t also are in the top-10. In positions 11-20 there are two MultiBoostAB ensembles (LibSVM and REPTree), svmBag_R and fda_t, alongside with several neural networks (pcaNNet_t, elm_kernel_m, nnet_t and mlp_t) and Random Forests (RotationForest_w and RandomForest_w).

- The **lower part** shows the 20 classifiers with the highest average Percentage of the Maximum Accuracy (PMA). The maximum value (95.0%, avNNet_t) is similar to the multi-class value (94.1%, lower part of Table 5), being parRF_t in the 3th position (94.3%). Other NNET classifiers also achieve good PMAs: pcaNNet_t, nnet_t and mlp_t. The C5.0_t keeps its good results, while rf_t falls to the 10th position. The table also includes some classifiers with bad multi-class results: glmnet_R, rda_R, pda_t, fda_t, bayesglm_t and simpls_R (both with bad multi-class rank), belonging to families GLM, DA and PLSR, which behave well for two-class problems. The best ensembles, apart from Random Forests, are MultiBoostAB_MultilayerPerceptron_w, MultiBoostAB_PART_w and nnetBag_R. Overall, the 20 classifiers are in a narrow range between 92.2%-95% of accuracy.

3.5 Discussion by Data Set Properties

In this section we study the classifier behavior in function of five data set properties: its "complexity", increasing and decreasing #patterns, #inputs and #classes. This study will be developed by calculating a modified average accuracy μ_j (in %) for each classifier j, in

which each data set is "weighted" according to each property as $\mu_j = \frac{1}{N_d}\sum_{i=1}^{N_d} w_i A_{ij}, j = 1,\ldots,N_c$, being w_i is the weight measuring the property for data set i ($0 \leq w_i \leq N_d$), defined in the following subsections; $N_d = 121$ is the number of data sets; A_{ij} is the accuracy (in %) achieved by classifier j in data set i; and $N_c = 179$ is the number of classifiers. The **classifier behavior with the data complexity** is difficult to evaluate, because the own data set complexity is hard to define (Ho and Basu, 2002), and it may be relative to the classifier used. In our case, since we are trying a large number of classifiers, we can suppose that some of them achieves the highest possible accuracy for each data set. Since this maximum accuracy is higher for some data sets than for others, we can believe that some data sets are harder, independently of the classifier used. Therefore, we can calculate the weighted average accuracy μ_j^C (the C superscript denotes "complexity") of classifier j using the weights w_i^C (which evaluate the complexity of data set i) defined as $w_i^C = \frac{N_d(1-M_i)}{N_d-\sum_{k=1}^{N_d} M_k}, i = 1,\ldots,N_d$, being $M_i = \max_{j=1,\ldots,N_c}\{A_{ij}/100\}$, the maximum accuracy for data set i divided by 100. Note that $\sum_{i=1}^{N_d} w_i^C = N_d$. The weighted accuracy μ_j^C (see below) with w_i^C defined above weights more the data sets i with maximum accuracy M_i low, which are expected to be more complex. The Table 10 (upper panel) shows the 20 classifiers with the highest μ^C, which exhibit the best behavior when the hardest data sets have stronger weight (data sets with maximum accuracy M_i low). The parRF_t is the best one, and the three best classifiers (5 in the top-10) belong to the family RF. Other two classifiers are neural networks (mlp_t and avNNet_t), C5.0_t is the 4th, and two SVMs (svm_C and LibSVM_w) are 6th and 9th respectively. Our proposal dkp_C exhibits a good behavior (12th position), while other classifiers in the top-20 of Table 3 as nnet_t, Bagging_LibSVM_w and RRFglobal_t are also included. The 20 classifiers are in a narrow range between 70.0% and 66.9% (3.1 points), so the differences among them are not too high. In order to study the **classifier behavior increasing #patterns**, the weighted accuracy μ^P uses the following weights $w_i^P = \frac{N_d N_i}{\sum_{k=1}^{N_d} N_k}, i = 1,\ldots,N_d$, where N_i is the #patterns (population) of data set i. The middle part of the Table 10 shows the weighted accuracy μ^P (the two largest data sets, connect-4 and miniboone, give errors for some classifiers which disturb this measure, so that they are excluded). Although the range is narrow (89.4%-91.1%), again the rf_t and parRF_t are the bests, and svm_C is the 3rd. There are six random forests in the top-10. The and treebag_t are also in the top-10. The positions 11-20 are completely filled by ensembles: Bagging, MultiBoostAB and AdaboostM1.

The **classifier behavior decreasing #patterns** in the data set can be analyzed calculating the weighted accuracy using weights w_i^D decreasing with the #patterns (N_m is the maximum #patterns for all the data sets) $w_i^D = \frac{N_d(N_m-N_i)}{N_m-\sum_{k=1}^{N_d} N_k}, i = 1,\ldots,N_d; N_m = \max_{j=1,\ldots,N_d}\{N_j\}, i = 1,\ldots,N_d$. The lower part of Table 10 shows the accuracies μ^D weighting each data set decreasingly with the #patterns. The rf_t is the best, followed by rforest_R, svm_C and parRF_t, which are only slightly worse than rf_t. Again, there are 6 random forests in the top-10. The positions 11-20 include dkp_C, elm_kernel_m, and MultiBoostAB ensembles of LibSVM and MultilayerPerceptron. The **dependence of the results with the #classes** N_i^c of the data set i can be analyzed calculating the weighted accuracy μ^L with data set weights w_i^L given by $w_i^L = \frac{N_d N_i^c}{\sum_{k=1}^{N_d} N_k^c}, i = 1,\ldots,N_d$. The Table 11

No.	Classifier	μ^C	No.	Classifier	μ^C
1	parRF_t	69.9	11	nnet_t	67.7
2	rf_t	69.6	12	dkp_C	67.6
3	rforest_R	69.3	13	RRFglobal_t	67.4
4	C5.0_t	69.0	14	Bagging_LibSVM_w	67.3
5	RotationForest_w	68.6	15	Decorate_w	67.1
6	svm_C	68.4	16	knn_t	67.1
7	mlp_t	68.4	17	Bagging_REPTree_w	67.0
8	RRF_t	68.1	18	elm_m	67.0
9	LibSVM_w	67.8	19	pda_t	67.0
10	avNNet_t	67.8	20	RandomCommittee_w	66.9
No.	Classifier	μ^P	No.	Classifier	μ^P
1	rf_t	91.1	11	Bagging_LibSVM_w	89.9
2	parRF_t	91.1	12	RandomCommittee_w	89.9
3	svm_C	90.7	13	Bagging_RandomTree_w	89.8
4	RRF_t	90.6	14	MultiBoostAB_RandomTree_w	89.8
5	RRFglobal_t	90.6	15	MultiBoostAB_LibSVM_w	89.8
6	LibSVM_w	90.6	16	MultiBoostAB_PART_w	89.7
7	RotationForest_w	90.5	17	Bagging_PART_w	89.7
8	C5.0_t	90.5	18	AdaBoostM1_J48_w	89.5
9	rforest_R	90.3	19	Bagging_REPTree_w	89.5
10	treebag_t	90.2	20	MultiBoostAB_J48_w	89.4
No.	Classifier	μ^D	No.	Classifier	μ^D
1	rf_t	82.1	11	MultiBoostAB_LibSVM_w	79.7
2	rforest_R	81.8	12	LibSVM_w	79.6
3	svm_C	81.6	13	RandomCommittee_w	79.5
4	parRF_t	81.6	14	dkp_C	79.5
5	RRF_t	80.8	15	nnet_t	79.3
6	RotationForest_w	80.3	16	elm_kernel_m	79.2
7	C5.0_t	80.2	17	avNNet_t	79.2
8	mlp_t	80.0	18	treebag_t	79.0
9	Bagging_LibSVM_w	80.0	19	MAB_MLP_w	78.8
10	RRFglobal_t	79.8	20	knn_R	78.7

Table 10: Twenty best classifiers depending on the data set complexity and population. **Up**: average accuracy μ^C (in %) weighting each data set decreasingly with its complexity. **Middle**: accuracy μ^P weighting the data sets increasingly with #patterns. **Down**: average accuracy μ^D weighted decreasingly with #patterns.

(upper part) shows the accuracy μ^L for the 20 best classifiers. The best classifiers are svm_C and rf_t (with the same accuracy), followed by rforest_t, Bagging_LibSVM_w, parRF_t and others, only 1% below the bests. There are 4 Random Forests and 2 SVMs in the top-10. The Bagging_LibSVM_w, MultiBoostAB_LibSVM_w and MultiBoostAB_Multilayer Perceptron_w ensembles are also included in the top-10. The best neural networks are dkp_C (9th position), MultilayerPerceptron_w and elm_m. Two DA classifiers (rda_R and hdda_R) and two NN classifiers (knn_R and IBk_w) are included. With respect to the **number of inputs**, the weighted average accuracy μ^I according to the #inputs N_i^I can be calculated

No.	Classifier	μ^L	No.	Classifier	μ^L
1	svm_C	80.5	11	RotationForest_w	76.6
2	rf_t	80.5	12	RRFglobal_t	76.1
3	rforest_R	79.8	13	MultilayerPerceptron_w	76.1
4	Bagging_LibSVM_w	79.7	14	rda_R	76.0
5	parRF_t	79.5	15	knn_R	75.9
6	MultiBoostAB_LibSVM_w	79.5	16	SMO_w	75.6
7	LibSVM_w	79.5	17	hdda_R	75.4
8	RRF_t	77.9	18	KStar_w	75.3
9	dkp_C	77.7	19	elm_m	75.1
10	MAB_MLP_w	76.9	20	RandomCommittee_w	75.1

No.	Classifier	μ^I	No.	Classifier	μ^I
1	parRF_t	84.0	11	mlp_t	81.5
2	rf_t	83.3	12	SMO_w	81.3
3	rforest_R	82.9	13	Bagging_RandomTree_w	81.3
4	RotationForest_w	82.8	14	elm_kernel_m	81.1
5	MAB_MLP_w	82.5	15	mlp_C	81.0
6	LibSVM_w	82.4	16	dkp_C	80.8
7	MultilayerPerceptron_w	82.0	17	fda_t	80.8
8	svm_C	82.0	18	rda_R	80.8
9	RandomCommittee_w	81.8	19	SimpleLogistic_w	80.7
10	C5.0_t	81.6	20	RRF_t	80.4

Table 11: **Up**: average accuracy μ^L weighted using the #classes w^L (only 20 first classifiers). **Down**: average accuracy μ^I weighted with the #inputs w^I.

defining the weights w^I as $w_i^I = \frac{N_d N_i^I}{\sum_{k=1}^{N_d} N_k^I}, i = 1, \ldots, N_d$. The lower part of the Table 11 shows μ^I for the 20 best classifiers: parRF_t and rf_t are the bests, with 4 random forests among the top-5 (the other is MultiBoostAB_Multilayer Perceptron_w), while the svm_C falls to the 8th position, below LibSVM_w (6th). The MultilayerPerceptron_w and mlp_t are also included in the top-10. The dkp_C is again in the top-20. **Considering jointly the four dependencies** (complexity, population, #classes and #inputs), parRF_t and rf_t are always in the first positions, while the svm_C is not so regular: good behavior with #classes and #patterns, but not so good with complexity and #inputs (6th and 8th positions). The svm_C and parRF_t are worse than rf_t with decreasing #patterns. Besides, the averages of $\mu^C, \mu^P, \mu^D, \mu^L, \mu^I$ are 81.3%, 81.2% and 80.8% for rf_t, parRF_t and svm_C respectively, which shows the similarity between rf_t and parRF_t, and their difference to svm_C. Most of the random forest versions (rforest_R, RotationForest_w, RRF_t and RRFglobal_t), and LibSVM_w, are in the five tables. Apart from the RF and SVM classifiers, which fill most of the 10 best positions in the five tables, it is remarkable the good behavior of C5.0_t (family DT), included in the four tables and three times in the top-10. Among the *neural networks*, the dkp_C appears more often (in four of five tables): in fact, the μ^P table does not include any neural network, showing a bad behavior for populated data sets. The Bagging_LibSVM_w is also the first **bagging** classifier in four tables, while MultiBoostAB of LibSVM or MLP is the best **boosting** classifier, appearing in four tables. The Random-

Committee_w (the best classifier of family **OEN**) is also included in five tables, and in the top-10 for μ^I. On the other hand, three of five tables include a classifier of family **NN** (knn_t or knn_R). The **DA** classifiers show bad behavior with population, being included only pda_t in μ^C; rda_R and hdda_R in μ^L; fda_t and rda_R in μ^I.

4. Conclusion

This paper presents an exhaustive evaluation of 179 classifiers belonging to a wide collection of 17 families over the whole UCI machine learning classification database, discarding the large-scale data sets due to technical reasons, plus 4 own real sets, summing up to 121 data sets from 10 to 130,064 patterns, from 3 to 262 inputs and from 2 to 100 classes. **The best results are achieved by the parallel random forest** (parRF_t), implemented in R with caret, tuning the parameter mtry. The parRF_t achieves in average 94.1% of the maximum accuracy over all the data sets (Table 5, lower part), and overcomes the 90% of the maximum accuracy in 102 out of 121 data sets. Its average accuracy over all the data sets is 82.0%, while the maximum average accuracy (achieved by the best classifier for each data set) is 86.9%. The random forest in R and tuned with caret (rf_t) is slightly worse (93.6% of the maximum accuracy), although it achieves slightly better average accuracy (82.3%) than parRF_t. The LibSVM implementation of SVM in C with Gaussian kernel (svm_C), tuning the regularization and kernel spread, achieves 92.3% of the maximum accuracy. Six RFs and five SVMs are included among the 20 best classifiers, which are the bests families. The parRF_t may be considered as a reference ("gold-standard") to compare with new classifier proposals in order to assess their performance for general classification in general (not requiring special features as large-scale, on-line learning, non-stationary data, etc.). Other classifiers with good results are the extreme learning machine with Gaussian kernel, the C5.0 decision tree and the multi-layer perceptron (avNNet_t, a committee of 5 multi-layer perceptrons randomly initialized tuning the size and decay rate). The best boosting and bagging ensembles use LibSVM as base classifiers (in Weka), being slightly better than the single LibSVM classifier, and adaboost_R (ensemble of decision trees trained using Adaboost.M1). For **two-class data sets**, avNNet_t is the best (95% of the maximum accuracy), being the parRF_t also very good (94.3%). It is also the best when the complexity, #patterns and #inputs of the data set increase, being also good when #patterns decrease (rf_t is the best) and #classes increase (svm_C is the best). The probabilistic neural network in Matlab, tuning the Gaussian kernel spread (pnn_m), and the direct kernel perceptron in C (dkp_C), a very simple and fast neural network proposed by us (Fernández-Delgado et al., 2014), are also very near to the top-20. The remaining families of classifiers, including other neural networks (radial basis functions, learning vector quantization and cascade correlation), discriminant analysis, decision trees other than C5.0, rule-based classifiers, other bagging and boosting ensembles, nearest neighbors, Bayesian, GLM, PLSR, MARS, etc., are not competitive at all. Most of the best classifiers are implemented in R and tuned using caret, which seems the best alternative to select a classifier implementation.

Acknowledgments

We would like to acknowledge support from the Spanish Ministry of Science and Innovation (MICINN), which supported this work under projects TIN2011-22935 and TIN2012-32262.

References

David W. Aha, Dennis Kibler, and Marc K. Albert. Instance-based learning algorithms. *Machine Learning*, 6:37–66, 1991.

Miika Ahdesmäki and Korbinian Strimmer. Feature selection in omics prediction problems using cat scores and false non-discovery rate control. *Annals of Applied Stat.*, 4:503–519, 2010.

Esteban Alfaro, Matías Gámez, and Noelia García. Multiclass corporate failure prediction by Adaboost.M1. *Int. Advances in Economic Research*, 13:301–312, 2007.

Peter Auer, Harald Burgsteiner, and Wolfang Maass. A learning rule for very simple universal approximators consisting of a single layer of perceptrons. *Neural Networks*, 1(21): 786–795, 2008.

Kevin Bache and Moshe Lichman. UCI machine learning repository, 2013. URL `http://archive.ics.uci.edu/ml`.

Laurent Bergé, Charles Bouveyron, and Stéphane Girard. HDclassif: an R package for model-based clustering and discriminant analysis of high-dimensional data. *J. Stat. Softw.*, 46(6):1–29, 2012.

Michael R. Berthold and Jay Diamond. Boosting the performance of RBF networks with dynamic decay adjustment. In *Advances in Neural Information Processing Systems*, pages 521–528. MIT Press, 1995.

Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

Leo Breiman, Jerome Friedman, R.A. Olshen, and Charles J. Stone. *Classification and Regression Trees*. Wadsworth and Brooks, 1984.

Jean Carletta. Assessing agreement on classification tasks: The kappa statistic. *Computational Linguistics*, 22(2):249–254, 1996.

Jadzia Cendrowska. PRISM: An algorithm for inducing modular rules. *Int. J. of Man-Machine Studies*, 27(4):349–370, 1987.

S. Le Cessie and J.C. Van Houwelingen. Ridge estimators in logistic regression. *Applied Stat.*, 41(1):191–201, 1992.

Chih-Chung Chang and Chih-Jen. Lin. Libsvm: a library for support vector machines, 2008. URL `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

Hyonho Chun and Sunduz Keles. Sparse partial least squares for simultaneous dimension reduction and variable selection. *J. of the Royal Stat. Soc. - Series B*, 72:3–25, 2010.

John G. Cleary and Leonard E. Trigg. K*: an instance-based learner using an entropic distance measure. In *Int. Conf. on Machine Learning*, pages 108–114, 1995.

Line H. Clemensen, Trevor Hastie, Daniela Witten, and Bjarne Ersboll. Sparse discriminant analysis. *Technometrics*, 53(4):406–413, 2011.

William W. Cohen. Fast effective rule induction. In *Int. Conf. on Machine Learning*, pages 115–123, 1995.

Bhupinder S. Dayal and John F. MacGregor. Improved PLS algorithms. *J. of Chemometrics*, 11:73–85, 1997.

Gülsen Demiroz and H. Altay Guvenir. Classification by voting feature intervals. In *European Conf. on Machine Learning*, pages 85–92. Springer, 1997.

Houtao Deng and George Runger. Feature selection via regularized trees. In *Int. Joint Conf. on Neural Networks*, pages 1–8, 2012.

Beijing Ding and Robert Gentleman. Classification using generalized partial least squares. *J. of Computational and Graphical Stat.*, 14(2):280–298, 2005.

Annette J. Dobson. *An Introduction to Generalized Linear Models*. Chapman and Hall, 1990.

Pedro Domingos. Metacost: A general method for making classifiers cost-sensitive. In *Int. Conf. on Knowledge Discovery and Data Mining*, pages 155–164, 1999.

Richard Duda, Peter Hart, and David Stork. *Pattern Classification*. Wiley, 2001.

Manuel J.A. Eugster, Torsten Hothorn, and Friedrich Leisch. Domain-based benchmark experiments: exploratory and inferential analysis. *Austrian J. of Stat.*, 41:5–26, 2014.

Scott E. Fahlman. Faster-learning variations on back-propagation: an empirical study. In *1988 Connectionist Models Summer School*, pages 38–50. Morgan-Kaufmann, 1988.

Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. LIBLINEAR: a library for large linear classification. *J. Mach. Learn. Res.*, 9:1871–1874, 2008.

Manuel Fernández-Delgado, Jorge Ribeiro, Eva Cernadas, and Senén Barro. Direct parallel perceptrons (DPPs): fast analytical calculation of the parallel perceptrons weights with margin control for classification tasks. *IEEE Trans. on Neural Networks*, 22:1837–1848, 2011.

Manuel Fernández-Delgado, Eva Cernadas, Senén Barro, Jorge Ribeiro, and José Neves. Direct kernel perceptron (DKP): ultra-fast kernel ELM-based classification with non-iterative closed-form weight calculation. *Neural Networks*, 50:60–71, 2014.

Eibe Frank and Mark Hall. A simple approach to ordinal classification. In *European Conf. on Machine Learning*, pages 145–156, 2001.

Eibe Frank and Stefan Kramer. Ensembles of nested dichotomies for multi-class problems. In *Int. Conf. on Machine Learning*, pages 305–312. ACM, 2004.

Eibe Frank and Ian H. Witten. Generating accurate rule sets without global optimization. In *Int. Conf. on Machine Learning*, pages 144–151, 1999.

Eibe Frank, Yong Wang, Stuart Inglis, Geoffrey Holmes, and Ian H. Witten. Using model trees for classification. *Machine Learning*, 32(1):63–76, 1998.

Eibe Frank, Geoffrey Holmes, Richard Kirkby, and Mark Hall. Racing committees for large datasets. In *Int. Conf. on Discovery Science*, pages 153–164, 2002.

Eibe Frank, Mark Hall, and Bernhard Pfahringer. Locally weighted naive Bayes. In *Conf. on Uncertainty in Artificial Intelligence*, pages 249–256, 2003.

Yoav Freund and Llew Mason. The alternating decision tree learning algorithm. In *Int. Conf. on Machine Learning*, pages 124–133, 1999.

Yoav Freund and Robert E. Schapire. Experiments with a new boosting algorithm. In *Int. Conf. on Machine Learning*, pages 148–156. Morgan Kaufmann, 1996.

Yoav Freund and Robert E. Schapire. Large margin classification using the perceptron algorithm. In *Conf. on Computational Learning Theory*, pages 209–217, 1998.

Jerome Friedman. Regularized discriminant analysis. *J. of the American Stat. Assoc.*, 84: 165–175, 1989.

Jerome Friedman. Multivariate adaptive regression splines. *Annals of Stat.*, 19(1):1–141, 1991.

Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Additive logistic regression: a statistical view of boosting. *Annals of Stat.*, 28:2000, 1998.

Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Regularization paths for generalized linear models via coordinate descent. *J. of Stat. Softw.*, 33(1):1–22, 2010.

Brian R. Gaines and Paul Compton. Induction of ripple-down rules applied to modeling large databases. *J. Intell. Inf. Syst.*, 5(3):211–228, 1995.

Andrew Gelman, Aleks Jakulin, Maria G. Pittau, and Yu-Sung Su. A weakly informative default prior distribution for logistic and other regression models. *The Annals of Applied Stat.*, 2(4):1360–1383, 2009.

Mark Girolami and Simon Rogers. Variational bayesian multinomial probit regression with Gaussian process priors. *Neural Computation*, 18:1790–1817, 2006.

Ekkehard Glimm, Siegfried Kropf, and Jürgen Läuter. Multivariate tests based on left-spherically distributed linear scores. *The Annals of Stat.*, 26(5):1972–1988, 1998.

Encarnación González-Rufino, Pilar Carrión, Eva Cernadas, Manuel Fernández-Delgado, and Rosario Domínguez-Petit. Exhaustive comparison of colour texture features and classification methods to discriminate cells categories in histological images of fish ovary. *Pattern Recognition*, 46:2391–2407, 2013.

Mark Hall. *Correlation-Based Feature Subset Selection for Machine Learning*. PhD thesis, University of Waikato, 1998.

Mark Hall and Eibe Frank. Combining naive Bayes and decision tables. In *Florida Artificial Intel. Soc. Conf.*, pages 318–319. AAAI press, 2008.

Trevor Hastie and Robert Tibshirani. Discriminant analysis by Gaussian mixtures. *J. of the Royal Stat. Soc. series B*, 58:158–176, 1996.

Trevor Hastie, Robert Tibshirani, and Andreas Buja. Flexible discriminant analysis by optimal scoring. *J. of the American Stat. Assoc.*, 89:1255–1270, 1993.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer, 2009.

Tin Kam Ho. The random subspace method for constructing decision forests. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 20(8):832–844, 1998.

Tin Kam Ho and Mitra Basu. Complexity measures of supervised classification problems. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 24(3):289–300, 2002.

Geoffrey Holmes, Mark Hall, and Eibe Frank. Generating rule sets from model trees. In *Australian Joint Conf. on Artificial Intelligence*, pages 1–12, 1999.

Robert C. Holte. Very simple classification rules perform well on most commonly used datasets. *Machine Learning*, 11:63–91, 1993.

Torsten Hothorn, Friedrich Leisch, Achim Zeileis, and Kurt Hornik. The design and analysis of benchmark experiments. *J. Computational and Graphical Stat.*, 14:675–699, 2005.

Guang-Bin Huang, Hongming Zhou, Xiaojian Ding, and Rui Zhang. Extreme learning machine for regression and multiclass classification. *IEEE Trans. Syst. Man Cybern. - Part B: Cybernetics*, 42:513–529, 2012.

Torsten Joachims. Making Large-Scale Support Vector Machine Learning Practical. In Bernhard Scholköpf, Cristopher J.C. Burges, and Alexander Smola, editors, *Advances in Kernel Methods - Support Vector Learning*, pages 169–184. MIT-Press, 1999.

George H. John and Pat Langley. Estimating continuous distributions in Bayesian classifiers. In *Conf. on Uncertainty in Artificial Intelligence*, pages 338–345, 1995.

Sijmen De Jong. SIMPLS: an alternative approach to partial least squares regression. *Chemometrics and Intelligent Laboratory Systems*, 18:251–263, 1993.

Josef Kittler, Mohammad Hatef, Robert P.W. Duin, and Jiri Matas. On combining classifiers. *IEEE Trans. on Pat. Anal. and Machine Intel.*, 20:226–239, 1998.

Ron Kohavi. The power of decision tables. In *European Conf. on Machine Learning*, pages 174–189. Springer, 1995.

Ron Kohavi. Scaling up the accuracy of naive-Bayes classifiers: a decision-tree hybrid. In *Int. Conf. on Knoledge Discovery and Data Mining*, pages 202–207, 1996.

Max Kuhn. Building predictive models in R using the caret package. *J. Stat. Softw.*, 28(5): 1–26, 2008.

Max Kuhn and Kjell Johnson. *Applied Predictive Modeling*. Springer, New York, 2013.

Niels Landwehr, Mark Hall, and Eibe Frank. Logistic model trees. *Machine Learning*, 95 (1-2):161–205, 2005.

Nick Littlestone. Learning quickly when irrelevant attributes are abound: a new linear threshold algorithm. *Machine Learning*, 2:285–318, 1988.

Nuria Macià and Ester Bernadó-Mansilla. Towards UCI+: a mindful repository design. *Information Sciences*, 261(10):237–262, 2014.

Nuria Macià, Ester Bernadó-Mansilla, Albert Orriols-Puig, and Tin Kam Ho. Learner excellence biased by data set selection: a case for data characterisation and artificial data sets. *Pattern Recognition*, 46:1054–1066, 2013.

Harald Martens. *Multivariate Calibration*. Wiley, 1989.

Brent Martin. *Instance-Based Learning: Nearest Neighbor with Generalization*. PhD thesis, Univ. of Waikato, Hamilton, New Zealand, 1995.

Willem Melssen, Ron Wehrens, and Lutgarde Buydens. Supervised Kohonen networks for classification problems. *Chemom. Intell. Lab. Syst.*, 83:99–113, 2006.

Prem Melville and Raymond J. Mooney. Creating diversity in ensembles using artificial data. *Information Fusion: Special Issue on Diversity in Multiclassifier Systems*, 6(1): 99–111, 2004.

John C. Platt. Fast training of support vector machines using sequential minimal optimization. In Bernhard Scholköpf, Cristopher J.C. Burges, and Alexander Smola, editors, *Advances in Kernel Methods - Support Vector Learning*, pages 185–208. MIT Press, 1998.

Ross Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

Ross Quinlan. Learning with continuous classes. In *Australian Joint Conf. on Artificial Intelligence*, pages 343–348, 1992.

Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, 1993.

Brian D. Ripley. *Pattern Recognition and Neural Networks*. Cambridge Univ. Press, 1996.

Juan J. Rodríguez, Ludmila I. Kuncheva, and Carlos J. Alonso. Rotation forest: a new classifier ensemble method. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 28(10):1619–1630, 2006.

Alexander K. Seewald. How to make stacking better and faster while also taking care of an unknown weakness. In *Int. Conf. on Machine Learning*, pages 554–561. Morgan Kaufmann Publishers, 2002.

Alexander K. Seewald and Johannes Fuernkranz. An evaluation of grading classifiers. In *Int. Conf. on Advances in Intelligent Data Analysis*, pages 115–124, 2001.

David J. Sheskin. *Handbook of Parametric and Nonparametric Statistical Procedures*. CRC Press, 2006.

Donald F. Specht. Probabilistic neural networks. *Neural Networks*, 3(1):109–118, 1990.

Johan A.K. Suykens and Joos Vandewalle. Least squares support vector machine classifiers. *Neural Processing Letters*, 9(3):293–300, 1999.

Robert Tibshirani, Trevor Hastie, Balasubramanian Narasimhan, and Gilbert Chu. Diagnosis of multiple cancer types by shrunken centroids of gene expression. *Proc. of the National Academy of Sciences*, 99(10):6567–6572, 2002.

Kai M. Ting and Ian H. Witten. Stacking bagged and dagged models. In *Int. Conf. on Machine Learning*, pages 367–375, 1997.

Valentin Todorov and Peter Filzmoser. An object oriented framework for robust multivariate analysis. *J. Stat. Softw.*, 32(3):1–47, 2009.

Alfred Truong. *Fast Growing and Interpretable Oblique Trees via Probabilistic Models*. PhD thesis, Univ. Oxford, 2009.

Joaquin Vanschoren, Hendrik Blockeel, Bernhard. Pfahringer, and Geoffrey Holmes. Experiment databases. A new way to share, organize and learn from experiments. *Machine Learning*, 87(2):127–158, 2012.

William N. Venables and Brian D. Ripley. *Modern Applied Statistics with S*. Springer, 2002.

Geoffrey Webb, Janice Boughton, and Zhihai Wang. Not so naive Bayes: aggregating one-dependence estimators. *Machine Learning*, 58(1):5–24, 2005.

Geoffrey I. Webb. Multiboosting: a technique for combining boosting and wagging. *Machine Learning*, 40(2):159–196, 2000.

Daniela M. Witten and Robert Tibshirani. Penalized classification using Fisher's linear discriminant. *J. of the Royal Stat. Soc. Series B*, 73(5):753–772, 2011.

David H. Wolpert. Stacked generalization. *Neural Networks*, 5:241–259, 1992.

David H. Wolpert. The lack of a priori distinctions between learning algorithms. *Neural Computation*, 9:1341–1390, 1996.

Zijian Zheng and Goeffrey I. Webb. Lazy learning of Bayesian rules. *Machine Learning*, 4 (1):53–84, 2000.