Benchmarking Classification Algorithms on High-Performance Computing Clusters

Bernd Bischl, Julia Schiffner, and Claus Weihs

Chair of Computational Statistics, Department of Statistics, TU Dortmund, Germany, {bischl,schiffner,weihs}@statistik.tu-dortmund.de

Abstract. Comparing and benchmarking classification algorithms is an important topic in applied data analysis. Extensive and thorough studies of such a kind will produce a considerable computational burden and are therefore best delegated to high-performance computing clusters. We build upon our recently developed R packages `BatchJobs` (Map, Reduce and Filter operations from functional programming for clusters) and `BatchExperiments` (Parallelization and management of statistical experiments). Using these two packages, such experiments can now effectively and reproducibly be performed with minimal effort for the researcher. We present benchmarking results for standard classification algorithms and study the influence of pre-processing steps on their performance.

1 Introduction

Assessing the performance of (supervised) classification methods by means of benchmark experiments is common practice. For example, a well-known study of such a kind was conducted in the StatLog project (King et al. (1995)). Benchmark studies often require large computational resources and are therefore best executed on high-performance computing clusters. Bischl et al. (2012) have recently developed two R packages `BatchJobs` and `BatchExperiments` that allow to comfortably control a batch cluster within R. An interesting problem that can be investigated by means of a benchmark study is the impact of data pre-processing operations on the performance of classification methods. Questions of interest are for example: 'How often does pre-processing lead to a considerably increased/decreased performance?' or 'Are there pre-processing steps that work well with certain classification methods?'. There are many case studies available which report that certain pre-processing options work well for the classification problem at hand. We have found also some studies that compare several pre-processing options (e.g. Pechenizkiy et al. (2004)), but many of them consider only very few classifiers and/or classification problems. To our knowledge there are no larger studies that systematically investigate the usefulness of several pre-processing options

and their combinations for several classification methods and a larger number of data sets (cp. e.g. Crone et al. (2006)).

We investigate the effect of three common steps, *outlier removal*, *principal component analysis* and *variable selection* and their combinations on the performance of eight standard classification methods based on 36 benchmark data sets. Data pre-processing is discussed in Section 2. In Section 3 the design of the benchmark study is described. Section 4 addresses some technical details concerning the execution of the study by means of the R packages `BatchJobs` and `BatchExperiments`. The results are given in Section 5. Section 6 summarizes our findings and provides an outlook to future research.

2 Data Pre-processing

In supervised classification we are given a training data set $\{(\boldsymbol{x}_i, y_i), i = 1, \ldots, n\}$, where $\boldsymbol{x}_i \in \mathbb{R}^p$, $i = 1, \ldots, n$, are realizations of p random variables. We suppose that there are p_{num} numerical and p_{cat} categorical variables ($p_{\text{num}} + p_{\text{cat}} = p$), and write $\boldsymbol{x}_i = (\boldsymbol{x}_{i,\text{num}}, \boldsymbol{x}_{i,\text{cat}})$. Each observation has a class label $y_i = k \in \{1, \ldots, K\}$. The number of training observations from class k is denoted by n_k.

In the following we describe the three pre-processing steps, *outlier removal*, *principal component analysis* and *variable selection*, investigated in our study.

It is well known that if classical classification methods are applied to data containing outliers their performance can be negatively affected. A universal way to deal with this problem is to remove the outliers in an initial preprocessing step. In our study the identification of outliers is based on the numerical variables only and every class is considered separately. We use a common approach based on a robust version of the Mahalanobis distance. For each class k we calculate

$$RM_i^k = \sqrt{(\boldsymbol{x}_{i,\text{num}} - \hat{\boldsymbol{\mu}}_{k,\text{MCD}})' \hat{\boldsymbol{\Sigma}}_{k,\text{MCD}}^{-1} (\boldsymbol{x}_{i,\text{num}} - \hat{\boldsymbol{\mu}}_{k,\text{MCD}})} \tag{1}$$

for all i with $y_i = k$. $\hat{\boldsymbol{\mu}}_{k,\text{MCD}} \in \mathbb{R}^{p_{\text{num}}}$ and $\hat{\boldsymbol{\Sigma}}_{k,\text{MCD}} \in \mathbb{R}^{p_{\text{num}} \times p_{\text{num}}}$ are the class-specific *minimum covariance determinant* (MCD) estimates of location and scatter of the numerical variables computed by the Fast MCD algorithm (Rousseeuw and van Driessen (1999)). The i-th observation is regarded as outlier and $(\boldsymbol{x}_i, y_i)$ is removed from the training set if $RM_i^k > \chi^2_{p_{\text{num}},0.975}$ (the 0.975-quantile of the χ^2-distribution with p_{num} degrees of freedom). For MCD estimation only the $h_k < n_k$ observations whose covariance matrix has the lowest determinant are used. h_k is calculated by

$$h_k = \begin{cases} m_k & \text{if } \alpha = 0.5 \\ \lfloor 2m_k - n_k + 2(n_k - m_k)\alpha \rfloor & \text{if } 0.5 < \alpha < 1 \end{cases} \tag{2}$$

with $m_k = \lfloor (n_k + p_{\text{num}} + 1)/2 \rfloor$ (Rousseeuw et al. (2012)). In our study, for each class the same α-value is used and α is tuned as described in Section 3.

Table 1. Misclassification rates of CART on the threenorm problem obtained by using the p original variables and all p principal components

p	2	4	6	8	10	12	14	16	18	20
original variables	0.04	0.07	0.11	0.12	0.16	0.14	0.19	0.20	0.22	0.21
principal components	0.03	0.02	0.02	0.01	0.02	0.02	0.02	0.02	0.02	0.01

Principal Component Analysis (PCA) converts a set of variables via an orthogonal transformation into a set of uncorrelated variables that are called principal components. In our study a PCA is conducted for the numerical variables, which are scaled to zero mean and unit variance first, based on all training observations. The original observations $\boldsymbol{x}_{i,\text{num}}$ can be replaced by the PCA scores $\boldsymbol{z}_i \in \mathbb{R}^{p_{\text{num}}}$ resulting in a training set with elements $((\boldsymbol{z}_i, \boldsymbol{x}_i^{\text{cat}}), y_i)$. Usually, PCA is used for dimension reduction and just the first few principal components that explain a fixed large percentage of the total variance are selected. Alternatively, the most important principal components can be chosen via some variable selection method. But even if no dimension reduction is done, the rotation of the original data may have a positive effect on the classification performance. As an illustration we consider the *threenorm* problem of Breiman (1996), an artificial binary classification problem. The data for the first class are drawn with equal probability from two p-dimensional standard normal distributions with mean $(a, a, \ldots, a)' \in \mathbb{R}^p$ and $(-a, -a, \ldots, -a)'$ respectively. The second class is drawn from a multivariate normal with mean $(a, -a, a, -a, \ldots)'$ where $a = 2/\sqrt{p}$. The dimension p was varied from 2 to 20 and for each p we generated a training and a test data set of size 1000. A classification tree (CART) was fitted to the training data and predicted on the test data. As Table 1 shows, the misclassification rate of CART obtained on the original variables increases with the dimension p. Moreover, a PCA was conducted on the training data and all p principal components were used in place of the original variables. In this case the error rate of CART is nearly zero, even for large values of p. However, since PCA does not take class labels into account, it is not guaranteed that the principal components are helpful for discriminating the classes. Moreover, PCA captures only linear relationships in the data. For this reason kernel PCA or (kernel) Fisher Discriminant Analysis are also in use. But since we found that PCA is regularly applied and that, to our knowledge, there are only few studies that assess the impact of PCA pre-processing in classification (e.g. Pechenizkiy et al. (2004)), PCA is investigated in our study. As described above we conduct a PCA for the numerical variables based on all training observations. We either use all components in place of the original variables or choose only some of them by applying a variable selection method in the next step.

For variable selection we consider a filter approach. Filter methods rank the variables according to some importance measure. In order to reduce the number of variables based on this ranking, it is common to select either all

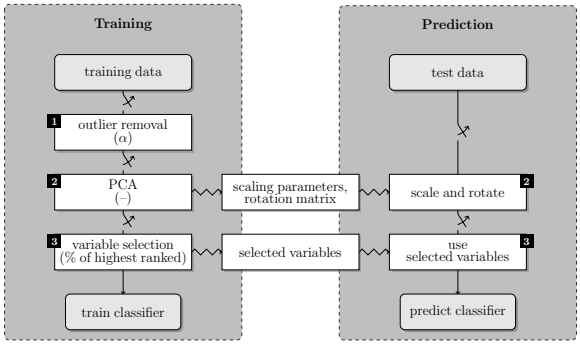

Fig. 1. Pre-processing steps done before training and predicting a classifier

variables with an importance value larger than a fixed threshold or a certain percentage of highest ranked variables (cp. Guyon and Elisseeff (2003)). We employ the latter approach and use the mean decrease in accuracy in random forests as importance criterion. In contrast to the first two pre-processing steps, numerical as well as categorical variables are taken into account. The percentage of selected variables is tuned as described in Section 3.

Fig. 1 summarizes the pre-processing steps conducted every time when training a classifier (left-hand side) and when making predictions (right-hand side). The three pre-processing operations are applied in the displayed order from top to bottom. Outlier removal, PCA and variable selection are conducted solely on the data used for training. The scaling parameters and the rotation matrix determined by PCA, as well as the names of the selected variables, are stored and the test data are transformed accordingly when making predictions. The switch symbols between the individual pre-processing steps in Fig. 1 indicate that every single step can be activated or deactivated. Thus, there exist $2^3 = 8$ possible pre-processing variants (including the case where no pre-processing is done at all).

3 Study Design

In order to assess the impact of the pre-processing operations described in Section 2 on the performance of classification methods we have used the following experimental setup: For every classifier we considered all 8 possible pre-processing variants. Prediction performance was evaluated by using nested resampling and measuring the misclassification error rate. In the outer loop, the complete data set was subsampled 30 times, producing 30 (outer) training data sets of size 80% and 30 (outer) test sets of size 20%. If a classifier (with pre-processing) had $q > 0$ associated hyperparameters, these were tuned on the training data set by measuring the misclassification rate via 3 fold cross-validation. Tuning was performed by choosing an effective sequen-

Table 2. Classification methods and pre-processing steps under consideration

method		hyper-parameters	box constraints	R package
ro	outlier removal	α	$[0.5, 1]$	`robustbase`
pca	PCA	–	–	`stats`
fil	filter	percentage	$[0.7, 1]$	`FSelector`
lda	Linear Discriminant Analysis	–	–	`MASS`
multinom	Multinomial Regression	–	–	`nnet`
qda	Quadratic Discriminant Analysis	–	–	`MASS`
naiveBayes	Naive Bayes	–	–	
rbfsvm	Support Vector Machine	`C`	$[2^{-10}, 2^{10}]$	`kernlab`
	with RBF kernel	`sigma`	$[2^{-10}, 2^{10}]$	
nnet	Neural Networks	`decay`	$[0.001, 0.1]$	`nnet`
rpart	CART Decision Tree	`cp`	$[0.001, 0.1]$	`rpart`
		`minsplit`	$\{5, \ldots, 50\}$	
randomForest	Random Forest	`ntree`	$\{100, \ldots, 2000\}$	`randomForest`

Table 3. Data sets taken from the UCI repository. Displayed are the number of observations and the number of numerical and categorical variables

data	obs	num	cat	data	obs	num	cat
BalanceScale	625	4	0	LiverDisorders	345	6	0
BloodTransfusion	748	4	0	MolecularBiologyPromoters	106	0	57
BreastCancer	699	0	9	Monks3	122	6	0
BreastCancerUCI	286	0	9	Parkinsons	195	22	0
BreastTissue	106	9	0	Phoneme	4509	256	0
Cmc	1473	2	7	PimaIndiansDiabetes	768	8	0
CoronaryHeartSurgery	1163	2	20	SAheart	462	8	1
Crabs	200	5	1	Segment	2310	19	0
Dermatology	366	1	33	Shuttle	256	0	6
Glass2	163	9	0	Sonar	208	60	0
GermanCredit	1000	7	13	Spambase	4601	57	0
Haberman	306	2	1	Spect	80	0	22
HayesRoth	132	4	0	Spectf	80	44	0
HeartCleveland	303	6	7	Splice	3190	0	60
HeartStatlog	270	13	0	TicTacToe	958	0	9
Ionosphere	351	32	1	Vehicle	846	18	0
Iris	150	4	0	VertebralColumn	310	6	0
KingRook_vs_KingPawn	3196	0	36	Waveform5000	5000	40	0

tial model-based optimization approach, in which the true relation between the parameters and the performance is approximated by a kriging regression model in each iteration (Koch et al. (2012)). In every iteration the so called expected improvement was maximized to generate a new promising design point to visit subsequently. The budget for the optimization process was $10q$ evaluations for an initial latin hypercube design and $40q$ evaluations for sequential improvements. After tuning, the best parameter combination was selected, the model was trained on the complete (if necessary pre-processed) outer training data set and the (pre-processed) outer test set was predicted. Table 2 shows the pre-processing steps and classification methods under consideration and displays the box constraints for the optimized hyperparameters.

We used 36 data sets from the UCI Machine Learning Repository. Table 3 provides a survey of basic properties of the data sets.

4 BatchExperiments and Parallelization Scheme

Bischl et al. (2012) have recently published two R packages `BatchJobs` and `BatchExperiments` for parallelizing arbitrary R code on high-performance

batch computing clusters. The former enables the basic parallelization of Map and Reduce operations from functional programming on batch systems. In this study we have used `BatchExperiments`, as it is especially constructed for evaluating arbitrary algorithms on arbitrary problem instances. A problem instance in our case is a classification data set, while an algorithm application is one run of tuning, model-fitting and test set prediction for one classifier with pre-processing operations. This leads to 36 datasets $\times$ 30 iterations of outer subsampling $\times$ 8 classifiers $\times$ 8 preprocessing variants = 69120 jobs. It should be noted that one computational job is already quite complicated as it contains up to 200 iterations of tuning via sequential model-based optimization. Due to space limitations we cannot go into more technical details how the code is structured, but refer the reader to Bischl et al. (2012), who demonstrate the parallelization of a simple classification experiment for the well-known iris data set. Job runtimes were quite diverse and ranged from a few seconds to more than 18 hours, depending on the classifier and data set, summing up to more than 750 days of sequential computation time.

5 Results

In order to analyze the results we have used the non-parametric Friedman test as a means of comparing the locations of the 30 misclassification rates per classifier (0.05 level of significance). Significant differences were detected by post-hoc analysis on all pairs of considered classifiers. We controlled the family-wise error rate through the usual procedure for multiple comparisons for this test as outlined in Hollander and Wolfe (1999). We have performed comparisons in two different ways: First, we compared in the group of basic classifiers without any pre-processing (to figure out which classifiers worked best in their basic form), then we compared in 8 groups of the 8 pre-processing variants of each classifier (to figure out which pre-processing operations worked best for which classifiers). In Table 4 the main aggregated results of this study are presented: The first row displays how often each basic classifier was among the best basic methods for each of the 36 considered data sets. "Among the best" here means that it was not significantly outperformed by another basic method. The rows labeled "ro" (outlier removal), "pca" (PCA) and "fil" (variable selection) count how often a classifier was significantly improved by adding only the respective pre-processing operation. The number in parentheses indicates how often the classifier was significantly worsened by doing this. The last line counts how often a classifier was significantly improved by comparing it to the best of the 7 pre-processing variants of itself.

It is in line with theoretical considerations of the 8 basic classifiers that a) non-robust methods like lda and naiveBayes benefit from outlier removal, b) a method like naiveBayes, which assumes independent variables given the class, benefits from decorrelating the variables by PCA and c) that the performance of methods like naiveBayes and qda can deteriorate with an increasing

Table 4. Main aggregated results, for details see text in this section

	rbfsvm	lda	multinom	naiveBayes	nnet	qda	randomForest	rpart
basic	32 (-)	14 (-)	17 (-)	13 (-)	13 (-)	5 (-)	25 (-)	13 (-)
ro	0 (4)	3 (4)	2 (7)	3 (4)	0 (2)	1 (8)	0 (4)	0 (3)
pca	0 (3)	1 (2)	0 (0)	7 (7)	6 (0)	1 (0)	2 (8)	3 (8)
fil	2 (0)	1 (0)	1 (0)	5 (0)	3 (0)	5 (0)	1 (0)	0 (0)
any	2 (-)	6 (-)	4 (-)	14 (-)	9 (-)	10 (-)	5 (-)	3 (-)

Table 5. Some selected results where strong improvements occurred

data	learner	pre-processing	error reduction
BreastTissue	nnet	pca	0.226
Crabs	naiveBayes	pca	0.361
Crabs	randomForest	ro+pca+fil	0.083
Haberman	qda	pca+fil	0.125
HayesRoth	lda	ro	0.122
HayesRoth	multinom	ro+fil	0.086
Segment	nnet	pca+fil	0.221
Spectf	lda	ro+fil	0.121
Spectf	nnet	pca+fil	0.081

number of variables and therefore a filter method might be helpful. Table 5 displays some additional selected results, where extremely large absolute error reductions were observed.

Unfortunately not every (tuned) pre-processing operation will always either improve the model or result in comparable performance. The filtering operation is an exception here (see Table 4). This is due to the fact that setting the percentage parameter of the filter operator to 1 results in the basic classifier with all variables, and our tuning process is apparently able to detect this for the data sets where this is appropriate. Actually, this should be the preferable behavior of the operator for outlier removal as well: When it is best to remove no data point, tuning should detect this and fall back to the basic model. The reason that this does not perfectly work in all of our experiments, points to the fact that the quantile value of the χ^2- distribution for outlier removal should have been included in tuning as well. In summary: If one is interested in the absolute best model for a given data set, we recommend to tune across the whole model space of all reasonable pre-processing variants. This is time-consuming, but can again be sped up by parallelization (and the use of our packages).

6 Summary and Outlook

In this article we have applied the R packages of Bischl et al. (2012) to perform a large scale experimental analysis of classification algorithms on a high-performance batch computing cluster. In this study, our goal was to analyze the influence of various pre-processing operations on 8 different classifiers. It appears that it is possible to considerably improve the performance by data pre-processing in some cases. However, for the majority of the investigated classification problems pre-processing did not result in improvements. We can

also see that for different classifiers different pre-processing options are beneficial and that some classifiers profit much more from the pre-processing steps in this investigation than others. It was especially hard to improve upon the best performing basic method per data set. Here, sometimes improvements around 1-2 % could be observed but as none of these were significant we were reluctant to report these. We also think that it would be useful for the community as a whole, if a digital repository would exist, where results and descriptions of experiments, such as the ones conducted in this paper, are stored in digital form.

References

BREIMAN, L. (1996): *Bias, variance, and arcing classifiers*, Technical Report 460, Statistics Department, University of California at Berkeley, Berkeley, CA.

BISCHL, B., LANG, M., MERSMANN, O., RAHNFÜHRER, J. and WEIHS, C. (2012): BatchJobs and BatchExperiments: Abstraction Mechanisms for Using R in Batch Environments. *Journal of Statistical Software*, submitted.

CRONE, S. F., LESSMANN, S. and STAHLBOCK, R. (2006): The Impact of Preprocessing on Data Mining: An Evaluation of Classifier Sensitivity in Direct Marketing. *European Journal of Operational Research, 173, 781–800.*

GUYON, I. and ELISSEEFF, A. (2003): An Introduction to Variable and Feature Selection. *Journal of Machine Learning Research, 3, 1157–1182.*

HOLLANDER, M. and WOLFE, D. A. (1999): *Nonparametric Statistical Methods.* 2nd Edition, Wiley, New York.

JONES, D. R., SCHONLAU, M. and WELCH, W. J. (1998): Efficient Global Optimization of Expensive Black-Box Functions. *Journal of Global Optimization, 13(4), 455–492.*

KING, R. D. and FENG, C. and SUTHERLAND, A. (1995): StatLog: Comparison of Classification Algorithms on Large Real-World Problems. *Applied Artificial Intelligence, 9(3), 289–333.*

KOCH, P., BISCHL, B., FLASCH, O., BARTZ-BEIELSTEIN, T., WEIHS, C. and KONEN, W. (2012): Tuning and Evolution of Support Vector Kernels. *Evolutionary Intelligence, 5(3), 153–170.*

PECHENIZKIY, M., TSYMBAL, A. and PUURONEN, S. (2004): PCA-Based Feature Transformation for Classification: Issues in Medical Diagnostics. In: *Proceedings of the 17th IEEE Symposium on Computer-Based Medical Systems.*

ROUSSEEUW, P., CROUX, C., TODOROV, V., RUCKSTUHL, A., SALIBIAN-BARRERA, M., VERBEKE, T., KOLLER, M. and MAECHLER, M. (2012): *robustbase: Basic Robust Statistics. R package version 0.9-2.* URL `http://CRAN.R-project.org/package=robustbase`.

ROUSSEEUW, P. J. and VAN DRIESSEN, K. (1999): A Fast Algorithm for the Minimum Covariance Determinant Estimator. *Technometrics, 41(3), 212–232.*