Reducing Rankings of Classifiers by Eliminating Redundant Classifiers

Pavel Brazdil, Carlos Soares, and Rui Pereira

LIACC / FEP, University of Porto, Rua do Campo Alegre, 823, 4150-180 Porto
{pbrazdil,csoares}@liacc.up.pt

Abstract. Several methods have been proposed to generate rankings of supervised classification algorithms based on their previous performance on other datasets [8, 4]. Like any other prediction method, ranking methods will sometimes err, for instance, they may not rank the best algorithm in the first position. Often the user is willing to try more than one algorithm to increase the possibility of identifying the best one. The information provided in the ranking methods mentioned is not quite adequate for this purpose. That is, they do not identify those algorithms in the ranking that have reasonable possibility of performing best. In this paper, we describe a method for that purpose. We compare our method to the strategy of executing all algorithms and to a very simple reduction method, consisting of running the top three algorithms. In all this work we take time as well as accuracy into account. As expected, out method performs better than the simple reduction method and shows a more stable behavior than running all algorithms. **Keywords**: meta-learning, ranking, algorithm selection

1 Introduction

There is a growing interest in providing methods that would assist the user in selecting appropriate classification (or other data analysis) algorithms. Some meta-learning approaches attempt to suggest one algorithm while others provide a ranking of the candidate algorithms [8, 4]. The raking is normally used to express certain expectation concerning performance (e.g. accuracy). The algorithm that appears in a certain position in the ranking is expected to be in some way better (or at least not worse) than the ones that appear afterwards. So, if the user is looking for the best performing algorithm, he can just follow the order defined by the ranking and try the corresponding algorithms out. In many cases the user may simply use the algorithm at the top of the ranking. However, the best performing algorithm may appear further on in the ranking. So, if the user is interested to invest more time in this, he can run few more algorithms and identify the algorithm that is best. Therefore our aim is (a) to come up with a ranking that includes only some of the given candidate algorithms, (b) not to decrease the chance that the best performing algorithm is excluded from the ranking. In this paper we describe an extension to the basic ranking method that has been designed for this aim.

In the next section we briefly review the basic ranking method. Section 3 describes the method used to exclude items from a given ranking. Section 4 presents the evaluation method adopted and while the following section describes the experimental setting and results. In Section 6 some alternative evaluation methods are briefly outlined and the last section presents the conclusions.

2 Constructing Rankings of Classifiers on Past Experience

Earlier we have proposed a method [8] that divides the problem of generating a ranking into two distinct phases. In the first one we employ a k-Nearest Neighbor method to identify datasets similar to the one at hand. This is done on the basis of different data characterization measures. In the second phase we consider the performance of the given classification algorithms on the datasets identified. This method uses performance information on both accuracy and time. These are combined into an Adjusted Ratio of Ratios (ARR) measure as follows:

$$ARR^{di}_{ap,aq} = (SR^{di}_{ap}/SR^{di}_{aq})/(1 + \log(TR^{di}_{ap}/TR^{di}_{aq})/K_T) \quad (1)$$

where $SR^{di}_{ap}/SR^{di}_{aq}$ represents the ratio of success rates of algorithm ap and aq on dataset di and similarly $TR^{di}_{ap}/TR^{di}_{aq}$ represents the ratio of times. As time ratios may vary substantially when compared to the ratio of success rates, this ratio is scaled down by taking a log, as shown. This will prevent the time ratio from dominating the ARR measure. K_t is a user-defined constant that determines the relative importance of time. The value $K_T = 10$ is equivalent to a compromise determining that 10x speed up is equivalent to a 10% drop in accuracy. That is, time is considered quite important. The value $K_T = 100$ is equivalent to a compromise between 10x speed up and 1% drop in accuracy, that is, time is considered less important than in the previous setting. Finally, the value $K_T = 1000$ is equivalent to a compromise between 10x speed up and a 0.1% drop in accuracy. Here improving on time will hardly bring any benefits at all. More details concerning this can be found in [8].

The individual ARR measures are used as a basis for generating a ranking. Basically, the method aggregates the information for different datasets by calculating overall means of individual algorithms:

$$ARR_{ap} = \sum_{aq}(\sum_{di} ARR^{di}_{ap,aq}/n)/(m-1) \quad (2)$$

where ARR_{ap} represents the mean of individual ARR measures for algorithm ap, n represents the number of datasets and m the number of algorithms. The values of ARR_{ap} are then examined and the algorithms ordered according to these values.

As it has been shown this method provides quite good rankings overall and is quite competitive when compared to other approaches, such as DEA [4]. One disadvantage of this method is that it does not provide any help as to how many

algorithms the user should actually try out. This problem is addressed in the next section.

3 Reducing Rankings of Classifiers by Eliminating Redundant Classifiers

There are really two reasons why we would want to eliminate algorithms from a given ranking. The first one is that the ranking recommended for a given dataset may include one or more algorithms that have proven to be significantly worse than others in similar datasets in the past and, thus, have virtually no chance of achieving better results than its competitors on the current dataset. These algorithms can thus be dropped. The second one is that the given ranking may include one or more clones of a given algorithm, that are virtually indistinguishable from it. These algorithms can again be dropped.

We may ask why we would want to include clones in the candidate set in the first place. The answer is that recognizing clones is not an easy matter. Different people may develop different algorithms and give them different names. These, in some cases, may turn out to be quite similar in the end. There is another reason for that. We may on purpose decide to include use the same algorithm several times with different parameter settings. We must thus have a way to distinguish useful variants from rather useless clones. Our objective here is to describe a method that can be used to do that.

Suppose a given ranking R consists of n classifiers, $C_1 ... C_n$. Our aim is to generate a reduced ranking R', which contains some of the items in R. The method considers all algorithms in the ranking, following the order in which they appear. Suppose, we have come to algorithm C_i. Then the sequence of subsequent algorithms C_j (where $j > i$) is then examined one by one. The aim is to determine whether each algorithm C_j should be left in the ranking, or whether it should be dropped. This decision is done on the basis of past results on datasets that are similar to the one at hand, i.e. datasets where the performance of the algorithms is similar to the performance that is expected on the new dataset. If the algorithm C_j has achieved significantly better performance than C_i on some datasets (shortly $C_j \gg C_i$) the algorithm is maintained. If it did not, then two possibilities arise. The first one is that C_j is significantly worse than C_i ($C_j \ll C_i$) on some datasets, and comparable to C_i on others. In this case C_j can be dropped. The second possibility is that neither algorithm has significant advantage over the other ($C_j \approx C_i$), indicating that they are probably clones. In this case too, C_j can be dropped. This process is then repeated until all combinations have been explored.

Here performance is judged according to measure ARR, which combines both accuracy and time. The historical data we use contains information on different folds of cross validation procedure. It is thus relatively easy to conduct a statistical test determining whether $ARR^{di}_{Cj,Ci}$ is consistently greater than 1 on different folds, indicating that C_j is significantly better than C_i on dataset di.

After the reduction process has terminated, the resulting ranking will contain the first item that was in R, plus a few more items from this ranking. Note that this need not be a sequence of several consecutive items. Should the best performing algorithm have a clone under a different name, this clone will be left out.

Reducing a subsequence of algorithms is beneficial in general, as it reduces the time the user needs to spend experimenting. However, this process involves certain risks. If the sequence is reduced too much, the optimal classifier may be missed out. If on the other hand the sequence is longer than required, the total time required to determine which algorithms is best increases. It is therefore necessary to have a way of evaluating different alternatives and comparing them. This topic is described in the next section.

4 Comparing Different Rankings of Uneven Length

Let us now consider the next objective - how to compare two rankings and determine which one is better. In our previous work [1, 8, 9] we have used a method that involves calculating a rank correlation coefficient to a ranking, constructed simply by running the algorithms on the dataset at hand.

This method has one disadvantage - it does not enable us to compare rankings of different sizes. We have therefore decided to use another method here. We evaluate a reduced ranking as if it was a single algorithm with the accuracy of the most accurate algorithm included in the ranking and with total time equal to the sum of the times of all algorithms included. Given that ARR was the performance measure used to construct and reduce the rankings, it makes sense to use the same scheme in the evaluation.

To be able to determine which reduced ranking is better, we need to consider both criteria used here, that is accuracy and time. One way of doing this is by combining accuracy and time into one measure, using the ARR measure described earlier.

5 Experimental Evaluation

The meta-data used was obtained from the METAL project (`http://www.metal-kdd.org`). It contains results of 10 algorithms on 53 datasets. We have used three decision tree classifiers, C5.0 rules, C5.0 with boosting [7] and Ltree, which is a decision tree which can introduce oblique decision surfaces [3]. We have also used a linear discriminant [6], two neural networks from the SPSS Clementine package (Multilayer Perceptron and Radial Basis Function Network) and two rule-based systems, C5.0 rules and RIPPER [2]. Finally, we used the instance-based learner and naive bayes implementations from the MLC++ library [5].

As for the datasets, we have used all datasets from the UCI repository with more than 1000 cases, plus the Sisyphus data and a few applications provided by DaimlerChrysler. The algorithms were executed with default parameters and the error rate and time were estimated using 10-fold cross-validation. Not all of

	K_T		
methods	10	100	1000
reduced	1.1225	1.0048	1.0037
top 3	1.1207	0.9999	0.9889
all	0.8606	0.9975	1.0093

Table 1. ARR scores for different values of K_T

the algorithms were executed on the same machine and so we have employed a time normalization factor to minimize the differences.

The rankings used to test the reduction algorithm were generated with the ranking method reviewed briefly in Section 2. Three settings for the compromise between accuracy were tested, with $K_T \in \{10, 100, 1000\}$ respectively.

The parameters used in the ranking reduction method were the following:

- Performance information for the *ten* nearest neighbors (datasets) was used to determine the redundancy of an algorithm.
- An algorithm is dropped if it is not significantly better than another algorithm retained in the ranking on at least *10%* of datasets selected (i.e. one dataset, since ten neighbors are used).
- The significance of differences between two algorithms has been checked with the paired t test with a confidence level of *95%*.

We have compared our method with the full ranking and with a reduced ranking obtained by selecting the top three algorithms in the full ranking.

As can be seen in Table 1 and Figure 1 the method proposed always performs better than the simple reduction method that uses the top three algorithms in the full ranking.

As for the choice of executing all the algorithms, when time is given great importance, this option is severely penalized in terms of the ARR score. When accuracy is the dominant criterion, executing all the algorithms represents the option. This result is of course to be expected, since this strategy consists of executing all algorithms with the aim of identifying the best option. So, our method is suitable if time matters at least a bit.

We were interested to analyze some of the reduced ranking to see how many algorithms were actually dropped and which ones. The following table presents the recommendation that was generated for the `segment` dataset.

When time is considered very important ($K_T = 10$, equivalent to a compromise between 10x speed up and a 10% drop in accuracy), only two algorithms were retained in the ranking: c50tree and lindiscr. All the others marked with * have been dropped. This result corresponds quite well to our intuitions. Both c50tree and lindiscr are known to be relatively fast algorithms. It is conceivable that this does not vary from dataset to dataset and hence variations in accuracy on different datasets did not have affect things much.

The situation is quite different if we give more importance to accuracy. Let us consider, for instance the case when $K_T = 10$, equivalent to a compromise

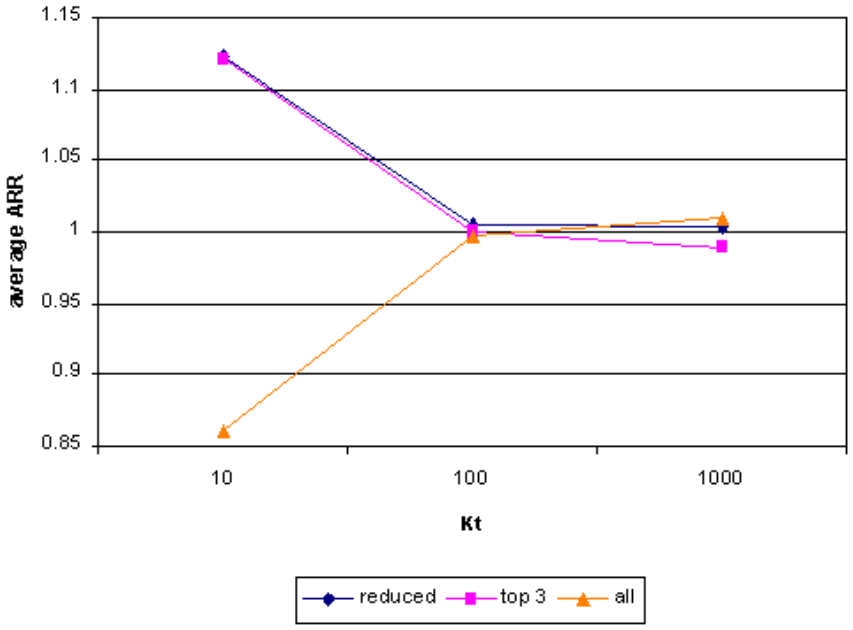

Fig. 1. Average ARR scores over 53 dataset for different values of K_T.

	K_T			
rank	10		100	
1	c50tree		c50boost	
2	lindiscr		c50rules	
3	c50rules	*	c50tree	
4	c50boost	*	ltree	
5	ltree	*	mlcib1	
6	mlcib1	*	lindiscr	
7	mlcnb	*	mlcnb	*
8	ripper	*	ripper	*
9	clemMLP	*	clemMLP	
10	clemRBFN	*	clemRBFN	*

Table 2. Complete and reduced rankings for the `segment`dataset for two different values of K_T. The algorithms eliminated in the reduced ranking are indicated by a '*'.

between 10x speed up and a 1% drop in accuracy. Here 7 out of 10 algorithms were retained, including ClemMLP which appeared in the 9th position in the original ranking. This is justified by the historical evidence that the method takes into account when giving this recommendation. Each of the algorithms retained must have done well at least on one dataset. It appears that mlcnb, ripper and clemRBFN do not satisfy this condition and hence were left out[1].

6 Discussion

The evaluation measure presented can be improved. We can then try to determine whether the ARR measure associated with one reduced ranking is higher than (or equal, or lower than) the ARR measure associated with another plan. In addition, it is possible to carry out statistical tests with the aim of determining whether these differences are significant. This analysis can be carried out for different setting of the constant K_T determining the relative importance of time. This procedure can thus be used to identify the algorithms lying on, or near the, so called, efficiency frontier [4].

Furthermore, note that this method can be used to compare reduced rankings of different lengths. However, if we were interested to evaluate not only the final outcome, but also some intermediate results, this could be done as follows. In this setting, a reduced ranking can be regarded as a *plan* to be executed sequentially. Suppose a given plan P_i consists of n_1 items (classifiers) and plan P_j of n_2 items, where, say, $n2 < n1$. The first n_2 comparisons can be carried out as described earlier, using the ARR measure. After that we simply complete the shorter plan with $n_1 - n_2$ dummy entries. Each entry just replicates the accuracy characterizing the plan. It is assumed that no time is consumed in this process.

7 Conclusions

We have presented a method to eliminate the algorithms in a ranking that are expected to be redundant. The method exploits past performance information for those algorithms. The method presented addresses one shortcoming of existing ranking methods [8, 4], that don't indicate, from the ranking of candidate algorithms, which ones are really worth trying. As expected, the experiments carried out show that our method performs better than the simple strategy of selecting the top three algorithms. Also, when time is very important, our method represents a significant improvement compared to running all algorithms. It is interesting to note that when accuracy is very important, although running all algorithms is the best strategy, the difference to our method is small.

[1] All results were obtained with default parameter settings. However, the performance of some algorithms is highly dependent on the parameter setting used. If parameter tuning was performed, some of the algorithms that were left out could be competitive in relation to others. This issue was ignored in this study.

Acknowledgments We would like to thank all the METAL partners for a fruitful working atmosphere, in particular to Johann Petrak for providing the scripts to obtain the meta-data and to Joerg Keller and Iain Patterson for useful exchange of ideas. The financial support from ESPRIT project METAL, project ECO under PRAXIS XXI, FEDER, FSE, Programa de Financiamento Plurianual de Unidades de I&D and Faculty of Economics is gratefully acknowledged.

References

1. P. Brazdil and C. Soares. A comparison of ranking methods for classification algorithm selection. In R.L. de Mántaras and E. Plaza, editors, *Machine Learning: Proceedings of the 11th European Conference on Machine Learning ECML2000*, pages 63–74. Springer, 2000.
2. W.W. Cohen. Fast effective rule induction. In A. Prieditis and S. Russell, editors, *Proceedings of the 11th International Conference on Machine Learning*, pages 115–123. Morgan Kaufmann, 1995.
3. J. Gama. Probabilistic linear tree. In D. Fisher, editor, *Proceedings of the 14th International Machine Learning Conference (ICML97)*, pages 134–142. Morgan Kaufmann, 1997.
4. J. Keller, I. Paterson, and H. Berrer. An integrated concept for multi-criteria ranking of data-mining algorithms. In J. Keller and C. Giraud-Carrier, editors, *Meta-Learning: Building Automatic Advice Strategies for Model Selection and Method Combination*, 2000.
5. R. Kohavi, G. John, R. Long, D. Mangley, and K. Pfleger. MLC++: A machine learning library in c++. Technical report, Stanford University, 1994.
6. D. Michie, D.J. Spiegelhalter, and C.C. Taylor. *Machine Learning, Neural and Statistical Classification*. Ellis Horwood, 1994.
7. R. Quinlan. *C5.0: An Informal Tutorial*. RuleQuest, 1998. http://www.rulequest.com/see5-unix.html.
8. C. Soares and P. Brazdil. Zoomed ranking: Selection of classification algorithms based on relevant performance information. In D.A. Zighed, J. Komorowski, and J. Zytkow, editors, *Proceedings of the Fourth European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD2000)*, pages 126–135. Springer, 2000.
9. C. Soares, P. Brazdil, and J. Costa. Measures to compare rankings of classification algorithms. In H.A.L. Kiers, J.-P. Rasson, P.J.F. Groenen, and M. Schader, editors, *Data Analysis, Classification and Related Methods, Proceedings of the Seventh Conference of the International Federation of Classification Societies IFCS*, pages 119–124. Springer, 2000.