Pattern Recognition 45 (2012) 1772–1781

Contents lists available at SciVerse ScienceDirect

Pattern Recognition

journal homepage: www.elsevier.com/locate/pr

Cost-conscious comparison of supervised learning algorithms over multiple data sets

Aydın Ulaş [a],*, Olcay Taner Yıldız [b], Ethem Alpaydın [a]

[a] Department of Computer Engineering, Boğaziçi University, TR-34342 İstanbul, Turkey
[b] Department of Computer Engineering, Işık University, TR-34398 İstanbul, Turkey

ARTICLE INFO

Article history:
Received 12 October 2010
Received in revised form
25 September 2011
Accepted 7 October 2011
Available online 15 October 2011

Keywords:
Machine learning
Statistical tests
Classifier comparison
Model selection
Model complexity

ABSTRACT

In the literature, there exist statistical tests to compare supervised learning algorithms on multiple data sets in terms of accuracy but they do not always generate an ordering. We propose Multi^{2}Test, a generalization of our previous work, for ordering multiple learning algorithms on multiple data sets from "best" to "worst" where our goodness measure is composed of a prior cost term additional to generalization error. Our simulations show that Multi^{2}Test generates orderings using pairwise tests on error and different types of cost using time and space complexity of the learning algorithms.

1. Introduction

In choosing among multiple algorithms, one can either select according to past experience, choose the one that is currently the most popular, or resort to some kind of objective measure. In classification, there is no single algorithm which is always the most accurate and the user is faced with the question of which one to favor. We also note that generalization error, though the most important, is rarely the sole criterion in choosing among algorithms and other criteria, such as training and/or testing time and/or space complexity, interpretability of results, ease of programming, etc. may also play an important role.

When a researcher proposes a new learning algorithm or a variant, he/she compares its performance with a number of existing algorithms on a number of data sets. These data sets may come from a variety of applications (such as those in the UCI repository [1]) or may be from some particular domain (for example, a set of face recognition data sets). In either case, the aim is to see how this new algorithm/variant ranks with respect to the existing algorithms either in general, or for the particular domain at hand, and this is where a method to compare algorithms on multiple data sets will be useful. Especially in data mining applications where users are not necessarily experts in machine learning, a methodology is needed to compare multiple algorithms over multiple data sets automatically without any user intervention.

To compare the generalization error of learning algorithms, statistical tests have been proposed [2,3]. In choosing between two, a pairwise test can be used to compare their generalization error and select the one that has lower error. Typically, cross-validation is used to generate a set of training, validation folds, and we compare the expected error on the validation folds. Examples of such tests are parametric tests (such as k-fold paired t test, 5×2 cv t test [2], 5×2 cv F test [4]) or nonparametric tests (such as the Sign test and Friedman's test [5]), or range tests (such as Wilcoxon's signed rank test [6,7]) on error, or on other performance measures such as the Area Under the Curve (AUC) [8,9]. Bouckeart [10] showed that the widely used t test showed superior performance compared to the Sign test in terms of replicability. On the other hand, he found the 5×2 cv t test dissatisfactory and suggested the corrected resampled t test. Resampling still has the problem of high Type I error and this issue has been theoretically investigated by Nadeau and Bengio [11]. They propose variance correction to take into account not only the variability due to test sets, but also the variability due to training examples.

Although such tests are for comparing the means of two populations (that is, the expected error rate of two algorithms), they cannot be used to compare multiple populations (algorithms). In our previous work [12], we proposed the MultiTest method to order multiple algorithms in terms of "goodness"

* Corresponding author. The author is currently with the Department of Computer Science, University of Verona.
E-mail address: ulasmehm@boun.edu.tr (A. Ulaş).

0031-3203/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.patcog.2011.10.005

where goodness takes into account both the generalization error and a prior term of cost. This cost term accounts for what we try to minimize additional to error and allows us to choose between algorithms when they have equal expected error; i.e. their expected errors are not pairwise significantly different from each other.

A further need is to be able to compare algorithms over not a single data set but over multiple data sets. Demsar [3] examines various methods, such as the Sign test and Friedman's test together with its post hoc Nemenyi's test, for comparing multiple algorithms over multiple data sets. These methods can make pairwise comparisons, or find subsets of equal error, but lack a mechanism of ordering and therefore, for example, cannot always tell which algorithm is the best.

In this paper, we generalize the MultiTest method so that it can work on multiple data sets and hence is able to choose the best, or in the general case, order an arbitrary number of learning algorithms from best to worst on an arbitrary number of data sets. Our simulation results using eight classification algorithms on 38 data sets indicate the utility of this novel Multi^{2}Test method. We also show the effect of different cost terms on the final ordering.

This paper is organized as follows: In Section 2, we review the statistical tests for comparing multiple algorithms. We propose the Multi^{2}Test method in Section 3. Our experimental results are given in Section 4 and Section 5 concludes.

2. Comparing multiple algorithms over multiple data sets

When we compare two or more algorithms on multiple data sets, because these data sets may have different properties, we cannot make any parametric assumptions about the distribution of errors and we cannot use a parametric test, for example, we cannot use the average accuracy over multiple data sets. We need to use nonparametric tests [3] which compare errors and use the rank information.

2.1. The sign test

Given S data sets, we compare two algorithms by using a pairwise test (over the validation folds) and we let the number of wins of one algorithm over the other be w, and we let the number of losses be l where $S = w + l$ (if there are ties, they are split equally between w and l). The Sign test assumes that the wins/losses are binomially distributed and tests the null hypothesis that $w = l$. We calculate $p = B(w,S)$ of the binomial distribution and if $p > \alpha$, we fail to reject the hypothesis that the two have equal error with significance α. Otherwise, we say that the first one is more accurate if $w > l$, and the second one is more accurate if $w < l$. For large values of S, we can use an approximation for $z = (w - S/2)/\sqrt{S/4}$; we fail to reject the test if $z \in (-z_{\alpha/2}, z_{\alpha/2})$, where $z_{\alpha/2}$ is the value such that $(\alpha/2)100$ percent of the standard normal distribution (Z) lies after $z_{\alpha/2}$ (or before $-z_{\alpha/2}$); in other words, it is c such that $P(Z > c) = P(Z < -c) = \alpha/2$.

Note that the Sign test results cannot be used to find an ordering: For three algorithms A,B,C, if A is more accurate than C and B is also more accurate than C and if A and B have equal error, we do not know which to pick as the first, A or B. This is where the concept of cost and the methodology of MultiTest comes into play.

2.2. Multiple pairwise comparisons on multiple data sets

To compare multiple algorithms on multiple data sets, one can use Friedman's test, which is the nonparametric equivalent of ANOVA [3,13]. First, all algorithms are ranked on each data set using the average error on the validation folds, giving rank 1 to the one with the smallest error. If the algorithms have no difference between their expected errors, then their average ranks should not be different either, which is what is checked for by Friedman's test. Let r_{ij} be the rank of algorithm $j = 1, \ldots, L$, on data set $i = 1, \ldots, S$, and $R_j = (1/S)\sum_i r_{ij}$ be the average rank of algorithm j. The Friedman test statistic is

$$X^2 = \frac{12S}{L(L+1)}\left[\sum_j R_j^2 - \frac{L(L+1)^2}{4}\right]$$

which is chi-square distributed with $L-1$ degrees of freedom. We reject if $X^2 < \chi^2_{\alpha,L-1}$.

If the test fails to reject, we say that we cannot find any difference between the means of the L algorithms and we do no further processing. If the test rejects, that is, if we know that there is a significant difference between the ranks, we use a post hoc test to check which pairs of algorithms have different ranks.

According to Nemenyi's test, two algorithms have different error rates if their average ranks differ by at least a critical difference $CD = q_\alpha SE$ where $SE = \sqrt{L(L+1)/6S}$, and the values for q_α are based on the Studentized range statistic divided by $\sqrt{2}$.

The subsets of algorithms which have equal error are denoted by underlining them. An example result is

$$\rlap{\underline{A\ B}\ \underline{C\ D}}\lower{3pt}{\underline{}}$$

where algorithms are sorted in ascending average error. We see that there is no difference between A and B, no difference between B and C but there is difference between A and C.

Note that after the post hoc test, we can find subsets of algorithms which have comparable error but we cannot always order them, for example, we cannot always find the best one. Such tests check for equality and a rejection, that is, the absence of an underline, does not imply an ordering. For example, we know that A and C have significantly different errors and that the average errors of A is less than the average errors of C but this does not necessarily mean that A has significantly less error than C; the two-tailed test does not check for this. Nor does it provide us a mechanism to choose between two algorithms which have no significant difference between them, for example A and B. Note also that Nemenyi's test is too conservative, has low power, and may not detect existing differences, even if Friedman's test rejects; this is expected to occur very rarely [3].

The result of Nemenyi's test (or any other test for checking equality) can be used to find the best learner only if one of the following conditions hold; see [12] for details:

- The first one, namely the algorithm with the smallest average, is not underlined. For example, if Nemenyi result is 2 $\underline{\text{4 3}}$ 1, the best can be taken as 2.
- There is a line under the first one and this line does not overlap with any other line(s). If Nemenyi result is $\underline{\text{4 3 2}}$ 1, the best is 2 because it is simpler than 3 and 4.
- There is a line under the first one and this line overlaps with one or more lines but the overlap does not include the first one. If Nemenyi result is $\rlap{\underline{1\ 3\ 4}}\lower{3pt}{\underline{}}$, the best is 1.
- If we have the case above and the overlap does not contain a simpler algorithm, the most simple is selected as the best. If Nemenyi result is $\rlap{\underline{4\ 1\ 3}}\lower{3pt}{\underline{}}$, the best is 1.

If neither of these four cases occur, Nemenyi's test cannot yield the best algorithm. For example, if the result of Nemenyi's test is $\rlap{\underline{4\ 3\ 2}}\lower{3pt}{\underline{}}$, the first underline chooses 2, the second underline chooses 1 which is simpler than 2. But we cannot choose 1 as it has higher expected error than 4. Note that these cases are only for finding the best algorithm; for creating the full ordering, the conditions must be satisfied by any group of algorithms, which is very rarely possible.

2.3. Correction for multiple comparisons

When comparing multiple algorithms, to retain an overall significance level α, one has to adjust the value of α for each post hoc comparison. There are various methods for this. The simple method is to use Bonferroni correction [14] which works as follows: Suppose that we want to compare L algorithms. There are $L(L-1)/2$ comparisons, therefore Bonferroni correction sets the significance level of each comparison to $\alpha/(L(L-1)/2)$. Nemenyi's test is based on this correction, and that is why it has low power. Garcia and Herrera [15] explain and compare the use of various correction algorithms, such as Holm's correction [16], Shaffer's static procedure [17] and Bergmann-Hommel's dynamic procedure [18]. They show that although it requires intensive computation, Bergmann-Hommel, which we adopted in this paper, has the highest power. All of these procedures use $z=(R_i-R_j)/SE$ as the test statistic and compare it with the z value of the suitably corrected α. In a recent paper, García et al. [19] propose new nonparametric tests, two alternatives to Friedman's test and four new correction procedures; their analysis focuses on comparing multiple algorithms against a control algorithm though, and not on ordering.

3. Multi^{2}Test

Our proposed method is based on MultiTest on a single data set [12]. We first review it and then discuss how we generalize it to work on multiple data sets.

3.1. MultiTest

MultiTest [12] is a cost-conscious methodology that orders algorithms according to their expected error and uses their costs for breaking ties. We assume that we have a prior ordering of algorithms in terms of some cost measure. We do not define nor look for statistically significant difference here; the important requirement is that there should be no ties because this ordering is used for breaking ties due to error. Various types of cost can be used [20], for example, the space and/or time complexity during training and/or testing, interpretability, ease of programming, etc. The actual cost measure is dependent on the application and different costs may induce different orderings.

The effect of this cost measure is that, given any two algorithms with the same expected error, we favor the simpler one in terms of the used cost measure. The result of the pairwise test overrides this prior preference; that is, we choose the more costly only if it has significantly less error.

Let us assume that we index the algorithms according to this prior order as $1,2\ldots L$ such that 1 is the simplest (most preferred) and L is the most costly (least preferred). A graph is formed with vertices M_j corresponding to algorithms and we place directed edges as follows: $\forall i,j,\ i<j$, we test if algorithm i has less or comparable expected error to j:

$$H_0 : \mu_i \le \mu_j$$

Actually, we test if the prior preference holds. If this test rejects, we say that M_j, the costlier algorithm, is statistically significantly more accurate than M_i, and a directed edge is placed from i to j, indicating that we override the prior order. After $L(L-1)/2$ pairwise tests (with correction for multiple comparisons), the graph has edges where the test is rejected. The number of incoming edges to a node j is the number of algorithms that are preferred over j but have significantly higher expected error. The number of outgoing edges from a node i is the number of algorithms that are less preferred than i but have significantly less expected error. The resulting graph need not be connected. Once this graph is constructed, a topological sort gives us the order of the algorithms.

Table 1
The result of pairwise tests on *optdigits*. If the entry is 1, the algorithm on the row is statistically significantly more accurate than the algorithm on the column. The algorithms are: *c*45: C4.5 decision tree, *mdt*: multivariate decision tree, *mlp*: multilayer perceptron, *lnp*: linear perceptron, *svl*: support vector machine with linear kernel, *sv*2: support vector machine with quadratic kernel, *svr*: support vector machine with radial (Gaussian) kernel, 5*nn*: 5-nearest neighbor (See Section 4 for experimental details).

	*c*45	*mdt*	*mlp*	*lnp*	*svl*	*sv*2	*svr*	5*nn*
*c*45	0	0	0	0	0	0	0	0
mdt	**1**	0	0	0	0	0	0	0
mlp	**1**	**1**	0	**1**	0	0	0	0
lnp	**1**	0	0	0	0	0	0	0
svl	**1**	**1**	**1**	**1**	0	0	0	**1**
*sv*2	**1**	**1**	**1**	**1**	0	0	0	**1**
svr	**1**	**1**	**1**	**1**	0	0	0	**1**
5*nn*	**1**	**1**	0	**1**	0	0	0	0

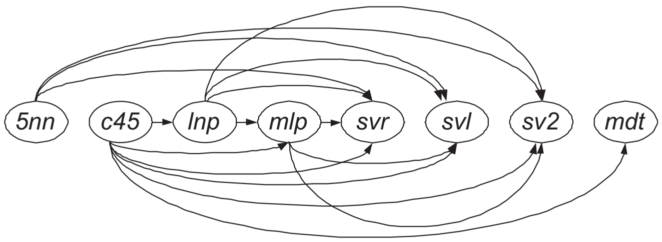


Fig. 1. The directed graph constructed by MultiTest on *optdigits* using the pairwise test results in Table 1 and prior ordering based on training time (5*nn* is the fastest, *mdt* is the slowest to train).

As an example, we show the application of MultiTest to one of our example data sets, *optdigits*. The result of the pairwise tests is shown in Table 1. Fig. 1 shows the directed graph when the prior ordering is based on training time (increasing from left to right). The sample execution of topological sort is shown in Fig. 2. The resulting order after topological sort is 1: *svr*, 2: *svl*, 3: *sv2*, 4: *5nn*, 5: *mlp*, 6: *lnp*, 7: *mdt*, 8: *c45*.

3.2. Multi^{2}Test

We now discuss how MultiTest can be generalized to run over *multiple* data sets. The pseudocode of the method is given in Table 2. First, we apply MultiTest separately on each data set using a pairwise test (with correction for multiple comparisons) and a prior ordering based on cost. We then convert the order found for each data set into ranks such that 1 is the best and L is the worst. These ranks are then given to the post hoc test which does not order the algorithms, but it gives us pairwise statistical differences which we use in MultiTest once more (thus the name Multi^{2}Test), again using the same prior ordering (this time averaged over all data sets after normalization), again with corrections for multiple comparisons. That is, in the outer MultiTest, the directed graph has edges provided by the post hoc test which accumulates the ranks found by MultiTest separately, over all the data sets. The test we use here is two-sided: if the test does not reject, there is no difference; if the test rejects, we prefer the one with lower average rank.[1]

[1] The Matlab code of Multi^{2}Test is available at: http://www.cmpe.boun.edu.tr/~ulas/multi2test/.

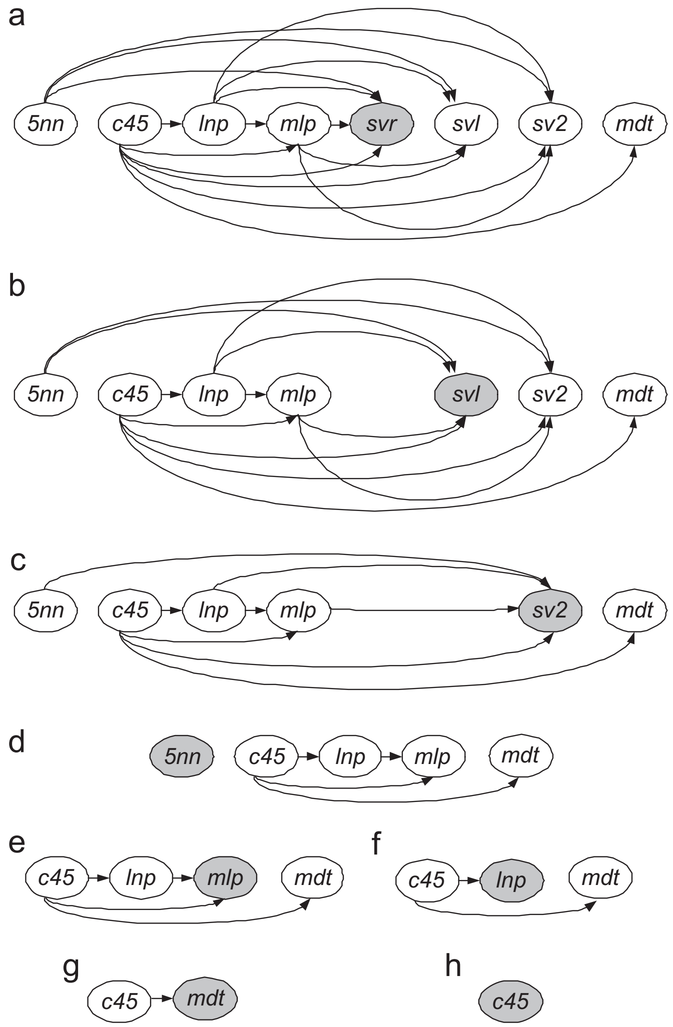


Fig. 2. Sample execution of topological sort on the directed graph generated by MultiTest on *optdigits* using training time as prior cost. The node chosen at each iteration is shaded. (a) There are no algorithms better than *svl*, *sv*2, *svr*, and *mdt* (they do not have outgoing edges), and among them, *svr* is the simplest and is taken first. (b) After *svr* and its incident edges are removed, *svl* is the simplest. (c) Then comes *sv*2. (d) 5*nn*, *mlp* and *mdt* are nodes without any outgoing edges and of the three, 5*nn* is the simplest. (e) Then we choose *mlp*. (f) *lnp* is more accurate than *c*45 and is simpler than *mdt*. (g) *mdt* is more accurate than *c*45 and (h) *c*45 is taken last. The resulting ranking after topological sort is 1: *svr*, 2: *svl*, 3: *sv*2, 4: 5*nn*, 5: *mlp*, 6: *lnp*, 7: *mdt*, 8: *c*45.

As an example for the second pass of Multi^{2}Test, let us assume that we have four algorithms A,B,C,D, according to the prior order of $A<B<C<D$, and the result of post hoc test after the first pass of MultiTest is $\underline{B\,\underline{D}}\,\underline{\underline{A}\,C}$. We then convert the results of post hoc test to pairwise statistically significant differences (Table 3) and together with the prior ordering, the formed directed graph is shown in Fig. 3. Doing a topological sort, we find the final order as 1: B, 2: A, 3: D, 4: C.

4. Results

4.1. Experimental setup

We use a total of 38 data sets where 35 of them (*zoo*, *iris*, *tae*, *hepatitis*, *wine*, *flags*, *glass*, *heart*, *haberman*, *flare*, *ecoli*, *bupa*, *ionosphere*, *dermatology*, *horse*, *monks*, *vote*, *cylinder*, *balance*, *australian*, *credit*, *breast*, *pima*, *tictactoe*, *cmc*, *yeast*, *car*, *segment*, *thyroid*, *optdigits*, *spambase*, *pageblock*, *pendigits*, *mushroom*, *and nursery*) are from UCI [1] and 3 (*titanic*, *ringnorm*, *and twonorm*) are from Delve [21] repositories.

We use eight algorithms:

(1) *c*45: C4.5 decision tree algorithm.
(2) *mdt*: Multivariate decision tree algorithm where the decision at each node is not univariate as in C4.5 but uses a linear combination of all inputs [22].
(3) *mlp*: Multilayer perceptron where with D inputs and K classes, the number of hidden units is taken as $(D+K)/2$.
(4) *lnp*: Linear perceptron with softmax outputs trained by gradient-descent to minimize the cross-entropy.
(5) *svl*: Support vector machine (SVM) with a linear kernel. We use the LIBSVM 2.82 library [23].
(6) *svr*: SVM with a radial (Gaussian) kernel.
(7) *sv*2: SVM with a polynomial kernel of degree 2.
(8) 5*nn*: k-nearest neighbor with $k=5$.

Our methodology is as follows: A data set is first divided into two parts, with $\frac{1}{3}$ as the test set, *test*, and $\frac{2}{3}$ as the training set, *train-all*. The training set, *train-all*, is then resampled using 5×2 cross-validation (cv) [2] where 2-fold cv is done five times (with stratification) and the roles swapped at each fold to generate 10 training and validation folds, tra_i, val_i, $i=1,\dots,10$. tra_i are used to train the base classifiers and the tests are run on the val_i results. We use the test set later to see whether the ranking predicted using the validation set defines a good order on the test set.

4.2. The sign test over averages

Table 4 shows the number of wins and number of losses of each algorithm over each algorithm by simply comparing average validation fold accuracies without any statistical test. The number of wins that are statistically significantly different using the Sign test over 38 runs are shown in bold. We see that for example, *svl* and *svr* are significantly more accurate than the other algorithms, and *mlp* is significantly more accurate than *mdt*.

4.3. Friedman's test and Bergmann-Hommel's dynamic procedure

Table 5 shows the average rankings by Friedman's test and the significant differences using Bergmann-Hommel's procedure for multiple comparisons as the post hoc test using average validation fold accuracies. The table also shows the graphical representation of post hoc test results of compared algorithms with ranks as proposed in [3] (except that we omitted the critical distances since CD changes for each corrected α). The numbers on the line represent the average ranks and bold lines connect the algorithms which have no significant difference.

We see that with respect to the average accuracies, *svl* and *svr* form one group and are statistically significantly different from all other algorithms except *mlp. mlp* is not different from *svl*, is different from *svr*, and is not different from the other group of algorithms, namely, *mdt*, *c*45, 5*nn*, *sv*2, and *lnp*. Bergmann-Hommel results shown as a table of pairwise comparisons are given in Table 5(c).

We should also point out that we cannot use the rankings shown in Table 5(b) to order algorithms. *svr* seems to be the best because it has the lowest average rank but it is not significantly better than *svl* and since *svl* uses the linear kernel and *svr* uses the more expensive Gaussian kernel, it may be better to prefer *svl*. But *mlp* seems as good as *svl* and may be cheaper because it may be using a small number of hidden units whereas *svl* may be storing a large number of support vectors, but we cannot put *mlp* before *svr* because *svr* has significantly lower rank. This implies that this preferrence due to simplicity should be built in the mechanism

Table 2
Multi^{2}Test to rank L supervised algorithms on S data sets.

```
Input: Cost function $C$, Pairwise test $T$, Data sets: $D_i, i=1,\ldots,S$, Algorithms: $M_j, j=1,\ldots,L$, Errors: $Err_{i,j}$
Output: Ranks of algorithms
foreach data set $D_i$, $i=1$ to $S$ do
    order algorithms according to $C$;
    rankalgorithmsusing $C$ andpairwisetest $T$ on $Err$ with MultiTest;
    foreach algorithm$M_j$, $j=1$ to$L$ do
        record rank $r_{ij}$ of algorithm $j$ for data set $i$;
    end
end
calculate $\overline{C}_j$, which is the average normalized cost for each algorithm over all data sets;
apply Friedman'rms test on $R_j = \frac{1}{S}\sum_i r_{ij}$
if Friedman's test rejects the null hypothesis then
    apply posthoc test on $R_j = \frac{1}{S}\sum_i r_{ij}$;
    rank algorithms using $\overline{C}_j$ and pairwise results of post-hoc test with MultiTest;
end
else
    output rank of algorithms according to $\overline{C}_j$;
end
```

Table 3
Tabular representation of post hoc test results for an example run.

	A	*B*	*C*	*D*
A	0	0	0	0
B	**1**	0	**1**	0
C	0	0	0	0
D	0	0	**1**	0

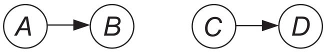


Fig. 3. The directed graph constructed by MultiTest on the example problem using the pairwise test results in Table 3 and the prior ordering: $A<B<C<D$.

for ordering, rather than imposed on the final result and this is exactly what our proposed Multi^{2}Test does.

4.4. *The sign test over pairwise tests*

If instead of using the average accuracies (as we did in Section 4.2), we use the 5 × 2 cv *F* test for pairwise comparison ($\alpha=0.05$), we get Table 6. Here, there are less wins than in Table 4 because to have a win, the difference between the averages should be significant. Again, wins that are significant using the Sign test over 38 data sets are shown in bold.

We again see that *svr* is significantly more accurate than *c*45, *mdt*, *sv*2, and 5*nn*, but it is not more accurate than *mlp* and *lnp* anymore. Note that *svl*, though seems significantly more accurate than other algorithms in Table 4, is no longer so when we use a test instead of just comparing average accuracies. *svl* is not significantly more accurate than *mlp* in Table 6, which explains why it is grouped with *mlp* in Table 5(b).

4.5. *Applying Multi^{2}Test*

Although the above methods give us pairwise comparisons, they cannot be used to order the given algorithms. For this, we use Multi^{2}Test; we show how it is used with two different cost functions, training time and space complexity.

4.5.1. *Training time as cost*

When we use the training time to define prior preferences with MultiTest, we can see three groups of algorithms. The costlier support vector machine variants and *mdt* form one group and are significantly different from the faster group, 5*nn*, *c*45 and *lnp* (see Table 7 (a) and (b)). *mlp*, which is in the middle, is significantly different from the slow *sv2* and *svr*, and the fastest 5*nn*, and has no significant difference from other algorithms (Table 7(c)).

We still do not have an order yet, so we apply the second pass of Multi^{2}Test using the average cost values to define the prior preference. According to the average training time, the prior order is: $5nn<c45<lnp<mlp<mdt<svl<sv2<svr$. Using this prior order and the pairwise test results using Bergmann-Hommel procedure results of Table 7(c), gives us the graph of Table 7(d), where we see that no test result overrides prior order; that is, the second MultiTest pass conforms with the accumulated first MultiTest pass on data sets separately. And therefore, the ranking is: 1: 5*nn*, 2: *c*45, 3: *lnp*, 4: *mlp*, 5: *mdt*, 6: *svl*, 7: *sv2*, 8: *svr*.

4.5.2. *Space complexity as cost*

When we use the space complexity with the same validation errors, we see that, this time, 5*nn* has the highest rank, and forms a group with the complex support vector machine variants *svl*, *svr* and *sv*2 and is significantly different from the simpler group of *lnp*, *c*45, *mdt*, and *mlp* (see Table 8(a) and (b)). We also see that 5*nn* is significantly different from *svr* (Table 8(c)).

When we apply the second pass of Multi^{2}Test according to the average space complexity, the prior order is: $c45<mdt<mlp<lnp<svl<svr<sv2<5nn$. Using this and the pairwise test results using Bergmann-Hommel procedure results of Table 8(c), we get the graph of Table 8(d). So the ranking is: 1: *c*45, 2: *mdt*, 3: *mlp*, 4: *lnp*, 5: *svl*, 6: *svr*, 7: *sv*2, 8: 5*nn*. We see that 5*nn*, which is the best when training time is critical, becomes the worst when space complexity is used.

One may argue that it is useless to apply the second pass of MultiTest, but this is not always the case. We have relatively accurate classifiers and the classifiers do not span a large range of accuracy and the diversity is small. We would expect different orderings going from one data set to another if the classifiers were

Table 4
Number of wins (out of 38) of all algorithms using average accuracies. The bold face entries show statistically significant difference using the Sign test.

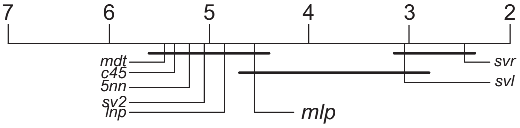
	*c*45	*mdt*	*mlp*	*lnp*	*svl*	*sv*2	*svr*	5*nn*
*c*45	0	19	16	16	11	17	5	15
mdt	19	0	11	16	9	18	6	18
mlp	22	**27**	0	22	9	19	7	24
lnp	22	22	16	0	8	22	8	21
svl	**26**	**29**	**29**	**30**	0	**25**	17	**31**
*sv*2	20	20	18	16	12	0	7	17
svr	**33**	**32**	**31**	**30**	21	**31**	0	**33**
5*nn*	23	20	14	17	7	21	5	0

Table 5
Average ranks and graphical representation of post hoc Bergmann-Hommel procedure results of compared algorithms with ranks using average accuracy.

(a) Average ranks of compared algorithms

*c*45	*mdt*	*mlp*	*lnp*	*svl*	*sv*2	*svr*	5*nn*
5.37	5.45	4.57	4.87	3.05	5.07	2.45	5.18

(b) Graphical representation of Bergmann-Hommel Procedure



(c) Tabular representation of Bergmann-Hommel procedure results

	*c*45	*mdt*	*mlp*	*lnp*	*svl*	*sv*2	*svr*	5*nn*
*c*45	0	0	0	0	**1**	0	**1**	0
mdt	0	0	0	0	**1**	0	**1**	0
mlp	0	0	0	0	0	0	**1**	0
lnp	0	0	0	0	**1**	0	**1**	0
svl	**1**	**1**	0	**1**	0	**1**	0	**1**
*sv*2	0	0	0	0	**1**	0	**1**	0
svr	**1**	**1**	**1**	**1**	0	**1**	0	**1**
5*nn*	0	0	0	0	**1**	0	**1**	0

Table 6
Number of wins of all algorithms using 5 × 2 cv *F* test. The bold face entries show statistically significant difference using the Sign test.

	*c*45	*mdt*	*mlp*	*lnp*	*svl*	*sv*2	*svr*	5*nn*
*c*45	0	5	3	4	2	5	0	4
mdt	5	0	0	2	0	10	0	7
mlp	11	7	0	6	3	10	3	9
lnp	7	3	1	0	0	9	0	5
svl	9	6	4	6	0	13	2	12
*sv*2	7	9	8	6	5	0	1	8
svr	**14**	**14**	10	10	8	**16**	0	**16**
5*nn*	6	4	4	3	1	10	1	0

more diverse and the range spanned by the accuracies of the classifiers were larger. We can construct an example where this is the case: Suppose that we have three algorithms A,B,C according to the prior order of $A<B<C$ and suppose that the result of the range test is $C\underline{A\ B}$. The final order will be 1: C, 2: A, 3: B, which is different from the prior order which is A,B,C. If we choose A as *c*45, B as *mdt* and C as *svr*, and use *breast*, *car*, *nursery*, *optdigits*, *pendigits*, *ringnorm*, *spambase*, *tictactoe*, and *titanic* data sets only, this is what we get using real data using space complexity as prior ordering. We see that the average prior order is *c*45 < *mdt* < *svr*, but the result of Multi^{2}Test is 1: *svr*, 2: *c*45, 3: *mdt* which is different from the prior order. Note that the ordering depends also on the algorithms compared as the critical difference depends on the number of populations compared.

4.6. Testing MultiTest and Multi^{2}Test

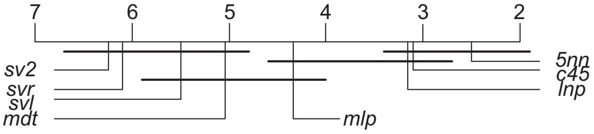
We do experiments on synthetic data to observe the behavior of MultiTest and Multi^{2}Test to see if they work as expected and hence comment on their Type I error and power. In Fig. 4, we have three algorithms (1, 2, 3) numbered in decreasing order of prior preference. Their error rates are taken $p_1=0.5+2\lambda$, p_2=0.5, and $p_3=0.5-2\lambda$, respectively. We simulate a classifier with error probability p as follows: We draw a uniform random number between 0 and 1 and if it is less than p, we take it as an error; we do this N=100 times and the total number of errors divided by N gives us an error rate. When we vary λ from 0 to 0.1, we start from three algorithms of equal error and slightly make them more and more different. For each case, we regenerate data, apply MultiTest, and find an ordering; we do this

Table 7
Average ranks and graphical representation of post hoc Bergmann-Hommel procedure results of compared algorithms used by Multi^{2}Test with training time as the cost measure.

(a) Average ranks of compared algorithms

*c*45	*mdt*	*mlp*	*lnp*	*svl*	*sv*2	*svr*	5*nn*
3.11	5.05	4.37	3.13	5.50	6.24	6.11	2.50

(b) Graphical representation of Bergmann-Hommel Procedure



(c) Tabular representation of Bergmann-Hommel procedure results

	*c*45	*mdt*	*mlp*	*lnp*	*svl*	*sv*2	*svr*	5*nn*
*c*45	0	**1**	0	0	**1**	**1**	**1**	0
mdt	**1**	0	0	**1**	0	0	0	**1**
mlp	0	0	0	0	0	**1**	**1**	**1**
lnp	0	**1**	0	0	**1**	**1**	**1**	0
svl	**1**	0	0	**1**	0	0	0	**1**
*sv*2	**1**	0	**1**	**1**	0	0	0	**1**
svr	**1**	0	**1**	**1**	0	0	0	**1**
5*nn*	0	**1**	**1**	0	**1**	**1**	**1**	0

(d) MultiTest graph for the second pass of Multi^{2}Test

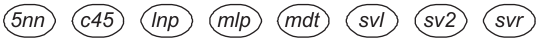
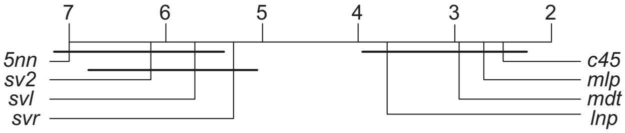


Table 8
Average ranks and graphical representation of post hoc Bergmann-Hommel procedure results of compared algorithms used by Multi^{2}Test with space complexity as the cost measure.

(a) Average ranks of compared algorithms

*c*45	*mdt*	*mlp*	*lnp*	*svl*	*sv*2	*svr*	5*nn*
2.50	2.95	2.71	3.68	5.71	6.13	5.32	7.00

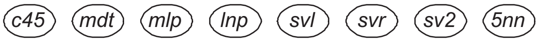
(b) Graphical representation of Bergmann-Hommel Procedure



(c) Tabular representation of Bergmann-Hommel procedure results

	*c*45	*mdt*	*mlp*	*lnp*	*svl*	*sv*2	*svr*	5*nn*
*c*45	0	0	0	0	**1**	**1**	**1**	**1**
mdt	0	0	0	0	**1**	**1**	**1**	**1**
mlp	0	0	0	0	**1**	**1**	**1**	**1**
lnp	0	0	0	0	**1**	**1**	**1**	**1**
svl	**1**	**1**	**1**	**1**	0	0	0	0
*sv*2	**1**	**1**	**1**	**1**	0	0	0	0
svr	**1**	**1**	**1**	**1**	0	0	0	**1**
5*nn*	**1**	**1**	**1**	**1**	0	0	**1**	0

(d) MultiTest graph for the second pass of Multi^{2}Test



1000 times and we count the total number of times MultiTest finds all possible orderings as a function of λ. The plots we see in Fig. 4 are those of three example orderings—we did not include all to avoid cluttering of the figures. When $\lambda=0$, all algorithms have the same error rate and MultiTest returns the prior preference of 1–2–3. As we increase λ, the error of the most costly algorithm decreases and the error of the least costly algorithm increases. When 3 becomes significantly better than 1, but not 2, we get the ordering of 2–3–1. In the end when $\lambda=0.1$, all algorithms are statistically significantly different from each other and MultiTest returns the algorithms in increasing order of error rate 3–2–1 completely reversing the prior order. This shows that MultiTest indeed works as expected. The fact that an ordering has high probability when it should be chosen and has low probability when it should not be chosen indicates that the methodology has low Type I error and high power.

In Fig. 5, this time we compare four algorithms on multiple data sets using Multi^{2}Test. Their error rates are $\alpha+3\lambda$, $\alpha+\lambda$, $\alpha-\lambda$, and $\alpha-3\lambda$, respectively. They are again numbered in decreasing order of prior preference. The number of data sets is 30 and base error rates of the classifiers (α) on each data set takes a random offset between 0.45 and 0.55. In the beginning ($\lambda=0$), although the base error rates are different, all algorithms have nearly the same error rate on each data set and as expected, Multi^{2}Test returns the prior preference 1–2–3–4 as the ordering. As we increase λ, the difference between the error rates starts to increase and this starts moving costlier algorithms ahead of the

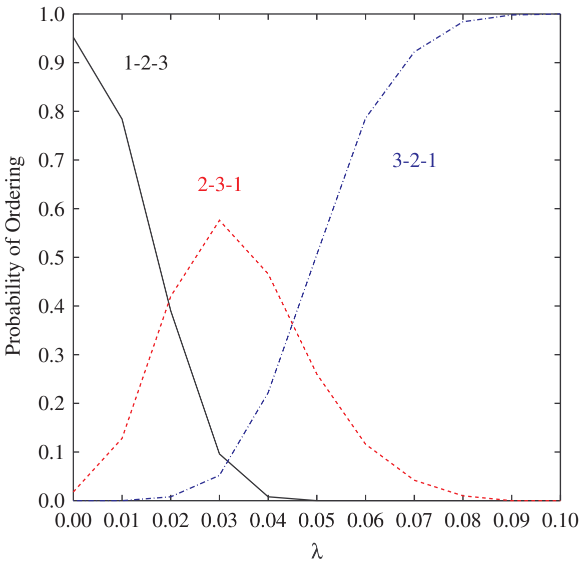


Fig. 4. Probabilities of different orderings found by MultiTest on synthetic data as a function of a multiplier parameter λ that varies the difference between errors of algorithms. Only three example orderings are shown for clarity.

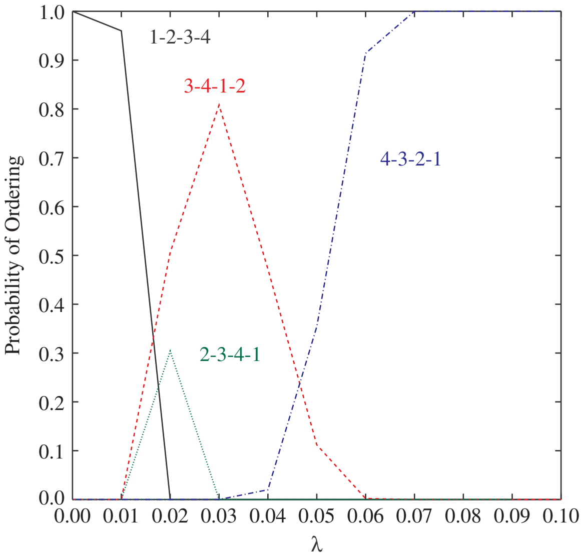


Fig. 5. Probabilities of different orderings found by Multi^{2}Test on synthetic data as a function of a multiplier parameter λ that varies the difference between errors of algorithms.

simpler algorithms because they start having less error. As these difference between the error rates of the algorithms get significant, Multi^{2}Test starts reversing the orderings, starting with 1 and 4 first (leading to the order of 2–3–4–1), and then between 1 and 3, and 2 and 4 (leading to the order of 3–4–1–2) and so on. In the end when $\lambda = 0.1$, since all algorithms are statistically significantly different from each other, Multi^{2}Test returns the algorithms in increasing order of error rate 4–3–2–1. Again, we see that Multi^{2}Test works as expected. How fast the probability of a certain ordering rises and falls (and hence how much orderings overlap) depend on the pairwise test used inside MultiTest, its significance α and the validation set size N.

4.7. *Verification of results on test*

We also did experiments to verify the results of Multi^{2}Test on the *test* set. What we do is we run the same Multi^{2}Test on the test data and check if the orderings we find using the validation set is a good predictor for what we find on the test set. We find that though there are small differences, for the most part, the predicted orderings match what we find on the test set.

When we use the training time as the cost measure, out of the 38 data sets, on 19 data sets, we find the same ordering on validation and test sets. There are 10 data sets where there are big rank changes. We observe this change five times with 5*nn*, three times with *c*45 and twice with *mdt*. We believe that this is because decision tree and nearest neighbor algorithms have high variance and hence their accuracies may differ slightly on different data which may lead to different orderings. A high variance in the algorithm's accuracy over different folds would make the statistical test less likely to reject and in such a case, the prior cost term would determine the final ordering. In the final ranking, this information as to whether the ordering is due to the result of the test or the cost prior can be differentiated.

When we consider space complexity as the cost measure, we see that on 24 out of the 38 data sets, the same ordering retained. This time there are only five big changes and again mostly due to decision trees; two with *c*45, two with *mdt* and one with *mlp*.

From these experiments, we conclude that the rankings produced by MultiTest are stable except for some high-variance algorithms, and the overall results produced by Multi^{2}Test on validation data is a good predictor of the real ranking on the test data.

5. Discussions and conclusions

We propose a statistical methodology, Multi^{2}Test, a generalization of our previous work, which compares and orders multiple supervised learning algorithms over multiple data sets. Existing methods in the literature can find statistical differences between two algorithms, or find subsets of algorithms with comparable error, but our proposed method compares and orders the given algorithms, and allows, for example, to choose the best of an arbitrary number of algorithms over an arbitrary number of data sets.

The ranks of the algorithms for Friedman's test or the number of wins in the Sign test may seem appropriate to order algorithms but the difference between wins and losses and ranks can be small and not significant. If *A* has 1 more win than *B* out of 38 data sets, but is 10 times more costly to implement, we would prefer *B*. We need to make sure that the gain in accuracy is worth the increase in cost. If the difference is due to chance, cost should override. That is why Multi^{2}Test is needed.

One may argue that instead of using the cost measure as prior preference, one could also combine the two measures of cost and accuracy into a single number for example by taking a weighted sum and order accordingly. We would still face the ordering problem in this case. There will be ties (differences too small to be considered significant) and the problem of how to break ties; we use the second criterion (cost in this case) for tie-breaking. Combining multiple criteria using, for example, a weighted summation also has the problem of setting the weights.

There is a significant body of literature on multiple criteria decision analysis and optimization. An example is the ELECTRE algorithm [24,25] where the idea is to choose an action, or rank actions (in our case algorithms) according to several criteria, which in our case may be the performance on different data sets. ELECTRE finds a minimal subset of actions that is preferred to other actions so that with further computation, the best one can be chosen. In our case, just the performance on data sets would not be enough to find a single best and one would need to use an extra measure, such as cost as we do in MultiTest. ELECTRE also allows setting different weights to different criteria but in our case, all data sets would have equal weight. In an application where we have multiple cost criteria, for example space *and* time complexity, an approach as used by ELECTRE may be useful to define an overall measure combining multiple measures of cost.

The ranking produced by Multi^{2}Test uses the generalization error of algorithms and a prior cost measure. This prior preference allows ordering algorithms whose expected errors do not have a statistically significant difference between them. This is a realistic setting, considering that in real life, in many applications, error is not the sole criterion but some measure of cost is also critical. We see that the ranking produced by Multi^{2}Test uses this cost measure effectively. If an algorithm is in rank i after Multi^{2}Test, this implies that all the algorithms that follow it with rank $j > i$ are either significantly worse or are equally accurate and costlier. The cost measure is used to break ties when there is no significant difference in terms of accuracy. If certain algorithms incur a cost that is not affordable for a particular situation, they may just be removed from the final ranked list and the current best can be found; or equivalently, Multi^{2}Test can be run without them.

Note that MultiTest or Multi^{2}Test do not significantly increase the overall computational and/or space complexity because the real cost is the training and validation of the algorithms. Once the algorithms are trained and validated over data sets and these validation errors are recorded, applying the calculations necessary for MultiTest or Multi^{2}Test is simple in comparison.

Our implementation uses the 5×2 cv F test for pairwise comparison of algorithms on a single data set. One could use other pairwise tests or other resampling schemes. For example, 10×10 folding [10] will have the advantage of decreasing Type I and II errors but will increase the computational complexity. MultiTest and Multi^{2}Test are statistical frameworks, and any resampling method and a suitable pairwise statistical test on some loss measure with appropriate α correction could be used. For example, the same methodology can be used to compare regression algorithms over a single or multiple data sets. These are interesting areas for further research.

If one has groups of data sets for similar applications, it would be better to order algorithms separately on these. For example if one has six different data sets for different image recognition tasks, and four different data sets for speech, it is preferable that Multi^{2}Test be run twice separately instead of once on the combined 10. Multi^{2}Test results are informative: Let us say we have the ranking of L algorithms on S data sets. Instead of merging them to find one overall ordering as Multi^{2}Test does, these rankings may allow us to define a measure of similarity which we can then use to find groups of similar algorithms; such similar ranks of algorithms also imply a similarity between data sets. These would be other interesting uses of Multi^{2}Test results.[2]

Acknowledgments

We would like to thank Mehmet Gönen for discussions. This work has been supported by the Turkish Academy of Sciences in the framework of the Young Scientist Award Program (EA-TÜBA-GEBİP/2001-1-1), Boğaziçi University Scientific Research Project 07HA101, Turkish Scientific Technical Research Council (TÜBİTAK) EEEAG 107E127, 107E222 and 109E186.

References

[1] A. Asuncion, D.J. Newman, UCI machine learning repository, 2007.
[2] T.G. Dietterich, Approximate statistical tests for comparing supervised classification learning algorithms, Neural Computation 10 (1998) 1895–1923.
[3] J. Demsar, Statistical comparisons of classifiers over multiple data sets, Journal of Machine Learning Research 7 (2006) 1–30.
[4] E. Alpaydın, Combined 5×2 cv F test for comparing supervised classification learning algorithms, Neural Computation 11 (1999) 1885–1892.
[5] X. Zhu, Y. Yang, A lazy bagging approach to classification, Pattern Recognition 41 (2008) 2980–2992.
[6] F. Wilcoxon, Individual comparisons by ranking methods, Biometrics Bulletin 1 (1945) 80–83.
[7] M. Gönen, E. Alpaydın, Regularizing multiple kernel learning using response surface methodology, Pattern Recognition 44 (2011) 159–171.
[8] A.P. Bradley, The use of the area under the ROC curve in the evaluation of machine learning algorithms, Pattern Recognition 30 (1997) 1145–1159.
[9] S.G. Alsing, K.W. Bauer Jr., J.O. Miller, A multinomial selection procedure for evaluating pattern recognition algorithms, Pattern Recognition 35 (2002) 2397–2412.
[10] R.R. Bouckaert, Estimating replicability of classifier learning experiments, in: Proceedings of the International Conference on Machine Learning ICML'04, 2004, pp. 15–22.
[11] C. Nadeau, Y. Bengio, Inference for the generalization error, Machine Learning 52 (2003) 239–281.
[12] O.T. Yıldız, E. Alpaydın, Ordering and finding the best of $K > 2$ supervised learning algorithms, IEEE Transactions on Pattern Analysis and Machine Intelligence 28 (2006) 392–402.
[13] M. Friedman, The use of ranks to avoid the assumption of normality implicit in the analysis of variance, Journal of the American Statistical Association 32 (1937) 675–701.
[14] A. Dean, D. Voss, Design and Analysis of Experiments, Springer, New York, 1999.
[15] S. García, F. Herrera, An extension on "statistical comparisons of classifiers over multiple data sets" for all pairwise comparisons, Journal of Machine Learning Research 9 (2008) 2677–2694.
[16] S. Holm, A simple sequentially rejective multiple test procedure, Scandinavian Journal of Statistics 6 (1979) 65–70.
[17] J.P. Shaffer, Modified sequentially rejective multiple test procedures, Journal of the American Statistical Association 81 (1986) 826–831.
[18] G. Bergmann, G. Hommel, Improvements of general multiple test procedures for redundant systems of hypotheses, in: P. Bauer, G. Hommel, E. Sonnemann (Eds.), Multiple Hypotheses Testing, 1988, pp. 100–115.
[19] S. García, A. Fernández, J. Luengo, F. Herrera, Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: experimental analysis of power, Information Sciences 180 (2010) 2044–2064.
[20] P.D. Turney, Types of cost in inductive concept learning, in: Proceedings of the Workshop on Cost-Sensitive Learning, ICML'00, 2000, pp. 15–21.
[21] C.E. Rasmussen, R.M. Neal, G. Hinton, D. van Camp, M. Revow, Z. Ghahramani, R. Kustra, R. Tibshirani, Delve data for evaluating learning in valid experiments, 1995–1996.
[22] O.T. Yıldız, E. Alpaydın, Linear discriminant trees, in: Proceedings of the International Conference on Machine Learning, ICML'00, 2000, pp. 1175–1182.
[23] C.C. Chang, C.J. Lin, LIBSVM: a library for support vector machines, 2001.
[24] B. Roy, P. Vincke, Multicriteria analysis: survey and new directions, European Journal of Operational Research 8 (1981) 207–218.
[25] J. Figueira, V. Mousseau, B. Roy, ELECTRE methods, in: J. Figueira, S. Greco, M. Ehrgott (Eds.), Multiple Criteria Decision Analysis: State of the Art Surveys, Springer Verlag, Boston, Dordrecht, London, 2005, pp. 133–162.

Aydin Ulaş received his BS, MS and PhD degrees in Computer Science from Boğaziçi University, İstanbul, in 1999, 2001 and 2008, respectively. He is currently doing his post doc in the University of Verona, Italy and working on the FP7 Project SIMBAD (Similarity-based Pattern Analysis and Recognition). His research interests include model selection, classifier combination, statistical comparison of classification algorithms, medical imaging, bioinformatics and machine learning. He has served as member of the scientific committee of different international conferences, and he is a reviewer for several international conferences and journals. He is a member of IEEE Computational Intelligence Society and IAPR Turkish Chapter (TÖTIAD).

Olcay Taner Yıldız received the BS, MS, and PhD degrees in Computer Science from Boğaziçi University, İstanbul, in 1997, 2000, and 2005, respectively. He did his postdoc at the University of Minesota in 2005. He is currently an Assistant Professor in Işik University, İstanbul, Turkey. His research interests include model selection, decision trees, natural language processing, and robotics.

[2] We are grateful to an anonymous reviewer for this suggestion.

Ethem Alpaydın received his BSc from the Department of Computer Engineering of Boğaziçi University in 1987 and the degree of Docteur es Sciences from Ecole Polytechnique Fédérale de Lausanne in 1990. He did his postdoctoral work at the International Computer Science Institute, Berkeley in 1991 and afterwards was appointed as Assistant Professor at the Department of Computer Engineering of Boğaziçi University. He was promoted to Associate Professor in 1996 and Professor in 2002 in the same department. As visiting researcher, he worked at the Department of Brain and Cognitive Sciences of MIT in 1994, the International Computer Science Institute, Berkeley in 1997 and IDIAP, Switzerland in 1998. He was a Fulbright Senior Scholar in 1997/1998 and received the Research Excellence Award from the Boğaziçi University Foundation in 1998 and 2008, the Young Scientist Award from the Turkish Academy of Sciences in 2001 and the Scientific Encouragement Award from the Turkish Scientific and Technical Research Council in 2002. His book Introduction to Machine Learning was published by The MIT Press in October 2004. Its German edition was published by Oldenbourg Verlag in May 2008, and its Chinese edition was published by Huazhang Press in June 2009. The Turkish edition is in preparation. He is a senior member of the IEEE, an editorial board member of The Computer Journal (Oxford University Press) and an Associate Editor of Pattern Recognition (Elsevier). The new edition of Introduction to Machine Learning, second edition was published by The MIT Press in February 2010.