Omnivariate Rule Induction Using a Novel Pairwise Statistical Test

Olcay Taner Yildiz

Abstract—Rule learning algorithms, for example, RIPPER, induces univariate rules, that is, a propositional condition in a rule uses only one feature. In this paper, we propose an omnivariate induction of rules where under each condition, both a univariate and a multivariate condition are trained, and the best is chosen according to a novel statistical test. This paper has three main contributions: First, we propose a novel statistical test, the combined 5×2 cv t test, to compare two classifiers, which is a variant of the 5×2 cv t test and give the connections to other tests as 5×2 cv F test and k-fold paired t test. Second, we propose a multivariate version of RIPPER, where support vector machine with linear kernel is used to find multivariate linear conditions. Third, we propose an omnivariate version of RIPPER, where the model selection is done via the combined 5×2 cv t test. Our results indicate that 1) the combined 5×2 cv t test has higher power (lower type II error), lower type I error, and higher replicability compared to the 5×2 cv t test, 2) omnivariate rules are better in that they choose whichever condition is more accurate, selecting the right model automatically and separately for each condition in a rule.

Index Terms—Rule induction, model selection, statistical tests, support vector machines

1 Introduction

A rule contains a conjunction of propositions and a class code that is the label assigned to an instance that is covered by the rule. An example rule containing two propositions for class C_2 is

$$\text{IF } (x_2 = \text{'red'}) \text{ AND } (x_1 < 1.5) \text{ THEN } C_2. \quad (1)$$

The propositions are of the form $x_i = v$, $x_i < \theta$ or $x_i \geq \theta$, depending on, respectively, whether the input feature x_i is discrete or numeric (Rules can also be extended from propositions to the first-order case, but this is beyond the scope of this paper). A rule set is an ordered list of such rules. A test instance is classified by sequentially checking rules in a rule set. The first rule that covers the test instance assigns its class to the instance.

Rule induction algorithms have a number of desirable properties:

1. For each class C_i, there is a rule set in conjunctive normal form. Each rule set merges rules with ORs, and each rule consists of conditions merged via ANDs. Such rule sets are easy to understand, allowing knowledge extraction and validation by application experts.
2. These methods are nonparametric in that they assume no a priori form on the model or class densities and fit their complexity to that of data.
3. They learn fast and can be used on very large data sets with a large number of instances.
4. They do their own feature extraction/dimensionality reduction and can be used on data sets with a large number of features. For an excellent review of rule induction algorithms, see [1].

Because of these reasons, rule induction algorithms are more and more frequently used in many data mining and pattern recognition applications and are preferred over other methods like artificial neural networks. Popular rule induction algorithms are C4.5RULES [2], PART [3], CN2 [4], and RIPPER [5]. C4.5RULES generates a decision tree and then converts it to a set of rules by writing each path from the root to a leaf as a rule; PART grows a partial decision tree and generates a single rule from the best performing leaf; RIPPER and CN2 directly generate a set of rules. We did a survey on the separate-and-conquer rule learning algorithms in the literature, which will be given in Section 2.1.

A rule as in (1) is univariate in that conditions in the rule use only one feature. As such, each condition defines an axis-aligned split (e.g., $x_1 < 1.5$ versus $x_1 \geq 1.5$) and a rule that is a conjunction of such conditions define hyperrectangles in the input space. It is obvious that if the underlying class discriminant is not axis aligned, then univariate conditions will not be appropriate. We can define multivariate conditions where several features are used to define a condition. For example, a multivariate linear condition is of the form:

$$w_1 x_1 + w_2 x_2 + \cdots + w_d x_d + w_0 < 0, \quad (2)$$

for suitable values of w_j, $j = 0, \ldots, d$. The univariate condition is a special case, where one of w_j, $j = 1, \ldots, d$ is 1, and all others are 0. This possibility of generalizing from univariate to multivariate nodes has been previously noticed for decision trees, and tree induction methods that use multivariate decision nodes have been proposed [6], [7], [8], [9]. Details of these approaches will be given in Section 2.2.

• *The author is with the Department of Computer Engineering, Işik University, P.K. TR-34980 Sile, Istanbul, Turkey. E-mail: olcaytaner@isikun.edu.tr.*

Manuscript received 9 June 2011; revised 7 Jan. 2012; accepted 21 July 2012; published online 13 Aug. 2012.
Recommended for acceptance by Y. Chen.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-2011-06-0337.
Digital Object Identifier no. 10.1109/TKDE.2012.155.

Although multivariate nodes have been used in decision trees nearly for three decades, learning multivariate rules directly from data is rare. Our first contribution is the proposal of the multivariate version of RIPPER namely R.MULTI. There are three main differences between R.MULTI and RIPPER:

- R.MULTI learns rules containing only multivariate linear conditions, which are in the form of (2), whereas RIPPER just contains univariate conditions.
- R.MULTI extracts multivariate linear conditions via linear support vector machine (SVM), whereas RIPPER explores the univariate feature space exhaustively.
- R.MULTI uses cross validation (cv) for model selection, whereas RIPPER uses minimum description length (MDL).

Our second contribution is the proposal of a new test, the combined 5×2 cv t test, which is a one-sided variant of the 5×2 cv t test proposed by Dietterich [10]. We also give the theoretic connections to other tests such as 5×2 cv F test and k-fold paired t test. The 5×2 cv t test uses only the difference of the first fold error values of the algorithms, and this is not robust due to the variance in folds. We replace this with a statistic that uses all 10 differences of error values and is, therefore, more robust. The combined 5×2 cv F test uses square of the differences of error values and cannot be used as a one-sided test.

Previously, omnivariate decision trees (ODTs) have been proposed, where at each decision node, univariate and multivariate candidate nodes are trained, and the more accurate, as chosen by a statistical test, is used [11]. Our third and main contribution is the combination of the previous two contributions into a single algorithm, where we propose an omnivariate version of rule learning, R.OMNI, in which conditions may be univariate or multivariate, and the more accurate is chosen by the combined 5×2 cv t test. Although one can advocate that using the combined 5×2 cv F test may be enough for model selection, experimental results on 54 data sets show that R.OMNI using the proposed combined 5×2 cv t test generates simpler and as accurate rule sets compared to R.OMNI that uses the combined 5×2 cv F test.

Fig. 1 shows example rule sets generated by R.UNI, R.MULTI, and R.OMNI algorithms for Iris data set. In this data set, R.UNI and R.MULTI generate rules containing only univariate and multivariate conditions, respectively. On the other hand, R.OMNI matches the complexity of a condition with the complexity of the subproblem defined by the data training the decision and generates two rules: one with a univariate and the other with a multivariate condition.

The paper is organized as follows: In Section 2, we review rule induction approaches. We derive our proposed combined 5×2 cv t test and discuss its connections to other tests in Section 3. In Sections 4 and 5, we propose our novel multivariate R.MULTI and omnivariate R.OMNI rule induction methods. We give the experimental results in Section 6 and conclude in Section 7.

R.UNI:
IF $F_3 < 1.9$ THEN Class: iris-setosa
ELSE IF $F_3 < 4.7$ THEN Class: iris-versicolor
ELSE Class: iris-virginica

R.MULTI:
IF $4.81F_1 + 5.59F_2 - 15.77F_3 - 6.24F_4 > 8.03$
THEN Class: iris-setosa
ELSE IF $-0.1F_1 + 4.21F_2 - 8.96F_3 - 11.2F_4 > -15.03$
THEN Class: iris-versicolor
ELSE Class: iris-virginica

R.OMNI:
IF $F_3 < 1.9$ THEN Class: iris-setosa
ELSE IF $2.05F_1 + 4.52F_2 - 28.78F_3 - 0.33F_4 > 10.51$
THEN Class: iris-versicolor
ELSE Class: iris-virginica

Fig. 1. Rule sets generated by R.UNI, R.MULTI, and R.OMNI algorithms for *Iris*. Rules are ordered, and a test instance is classified by the first rule it matches. There is a default class if no rule covers an instance. In this simple data set, each rule contains only one condition.

2 RULE INDUCTION

There are two main groups of rule learning algorithms. Separate-and-conquer algorithms and divide-and-conquer algorithms. Separate-and-conquer algorithms first find the best rule that explains part of the training data. After *separating* the examples those are covered by this rule, the algorithms *conquer* remaining data by finding next best rules recursively. Consequently, previously learned rules directly influence the data of the other rules.

Divide-and-conquer algorithms greedily find the split that best separates data in terms of some predefined impurity measure such as information gain, entropy, Gini index, and so on. After *dividing* examples according to the best split, the algorithms *conquer* each part of the data by finding next best splits recursively. In this case, previously learned splits in the parent nodes directly influence the data of the descendant nodes.

2.1 Separate-and-Conquer Approach

2.1.1 Hill-Climbing Approach

PRISM [12] is one of the oldest rule induction techniques, which uses hill climbing in learning rules. It starts with an empty rule and adds conditions as long as the rule covers negative examples. The algorithm compares possible refinements with the purity measure $-log\frac{p}{p+n}$. PREPEND [13] puts the new learned rule before the previously learned rules. The logic behind is that general rules will be learned first and if we put these rules at the end of the list, they will also cover exceptions.

The two previous algorithms do not have any pruning step. Since hill climbing is a greedy search method, pruning the rule set may improve the performance. For this reason, several hill-climbing algorithms do use pruning. REP [14] prunes the rule set by deleting the last condition in a rule, I-REP [15] (predecessor of RIPPER) prunes the rule set starting with the first condition in a rule, GROW [16] deletes a final sequence of conditions, SWAP-1 [17] finds

the best replacement for a condition. Like the postpruning in the decision tree induction, these algorithms first find the rule set that covers all positive examples in the training set, which overfit the training data. Then, they prune the rule set using another set, prune set.

RIPPER (R.UNI) learns rules from scratch starting from an empty rule set. It has two phases: In the first phase, it builds an initial set of rules, one at a time, and in the second phase, it optimizes the rule set m times, typically twice [5].

In the first phase, RIPPER learns rules one by one, where rules are added to a rule set to minimize the total description length. First, propositional conditions are added one at a time to a rule, at each step choosing the condition that maximizes the information gain. Choosing the best is done exhaustively by searching all possible split points. The rule is then pruned to alleviate overfitting. Conditions are removed one by one, where each time the condition that mostly increases a rule value metric (based on the accuracy on the prune set) is selected for removal. In RIPPER, one stops removing conditions when the rule value metric cannot be increased further.

In the second phase, rules in the rule set are optimized. Two alternatives are grown for each rule. The first candidate, the replacement rule, is grown starting with an empty rule, whereas the second candidate, the revision rule, is grown starting with the current rule. These two rules and the original rule are compared, and the one with the smallest description length is selected and put in place of the original rule.

JOJO [18] and SWAP-1 [17] are bidirectional hill-climbing rule induction algorithms, where they not only allow adding conditions but also allow removing conditions from a rule. SWAP-1 checks if a new learned condition can be dropped or can be replaced by a new condition to improve the rule.

PN-RULE [19] claim that existing sequential covering algorithms try to achieve maximum precision for each condition, which causes the rule set to be unsuccessful for the rare classes. This is due to the fact that usually, rare classes are considered as false positives to the general rules. They use a two-phase design to overcome this problem, where in the first phase (P-phase) they predict the presence of the target class, and in the second phase (N-phase), they predict the absence of the target class.

DATASQUEEZER [20] is one of the newest rule learning algorithms that performs greedy hill-climbing search. The main characteristics of DATASQUEEZER are 1) data reduction is performed by using the prototypical concept learning before generating rules (based on FINDS algorithm in [21]); 2) it exhibits log-linear asymptotic complexity with the size of training data and, therefore, is as fast as other competitive rule learners.

2.1.2 Beam Search

AQ [22] is the predecessor of all rule induction systems. AQ and some other algorithms as CN2 [23], POSEIDON [24], BEXA [25] use beam search as top down to induce rules from most general to most specific manner. On the other hand, DLG [26] and LERILS [27] use beam search as bottom up to induce rules from most specific to most general manner.

2.1.3 Best First Approach

Best first search can be seen as a beam-search technique with an infinite beam size. ML-SMART [28] used that fact to generate a set of candidate rules. Since the number of alternative rules can grow exponentially, ML-SMART uses pruning heuristics to remove unpromising rules.

PROGOL [29] is also a best first rule induction algorithm working top-down manner. PROGOL makes its best first search according to the A^* technique [30]. According to the A^* technique, each rule have its score as the sum of $\mathrm{f(n)} + \mathrm{g(n)}$. f(n) is the performance of the rule, which is calculated as the number of positive examples covered minus the number of negative examples covered minus the length of the rule. g(n) is the heuristic that how the rule will behave if it was completed. PROGOL implements g(n) by estimating the number of conditions required to complete the rule.

2.1.4 Stochastic Approach

SFOIL [31] and MILP [32] used stochastic search to remove local minima problem of hill climbing. When the hill-climbing search reaches an optima (local optima), they restart the search for best rule from other unexamined rule.

SIA [33] effectively uses genetic algorithm to generate best rule set. SIA maintains a list of candidate rules as in beam search called a generation of the genetic algorithm. This generation produces children (new rules), by crossover operations such as exchanging the conditions of the two rules, by mutation as generating new rules for a parent rule.

More recently, ant colony optimization (ACO) has been carried out for the extraction of rule-based classifiers. The first application of ACO to the rule induction task is the ant-miner algorithm [34]. Since then, several extensions and modifications of this sequential covering algorithm have been presented [35], [36].

2.1.5 Rough-Set Approach

The fuzzy-rough set (FRS) methodology is one of the widely studied tool for reducing dimension and producing classification systems. For example, Shen and Chouchoulas [37] use the FRS methodology to reduce dimension and build a rule-based classifier by extracting some fuzzy rules from a perceptron network classifier in the reduced space. On the other hand, there are many works that directly generate rule base classifiers using the FRS approach. The fuzzy-rough classification tree classifier [38] use fuzzy-rough hybrids to measure the dependency of decision attributes in the decision tree generation mechanism. Hu et al. [39] discover fuzzy classification rules based on the well-known Apriori algorithm.

Some of the classifiers have limitations such as working only in discrete domain and, therefore, requiring a discretization step for numeric features [40], [41], or not considering the theoretical properties of the lower and upper approximations, such as topologic and algebraic properties [42]. On the other hand, Zhao et al. [43] build a rule-based classifier by using one generalized FRS model after proposing a new concept named as "consistence degree."

2.2 Divide-and-Conquer Approach

Friedman [44] is the first who used linear discriminant analysis (LDA) for constructing decision trees. The algorithm constructs binary univariate decision trees, where the variable can be an original, transgenerated, or adaptive variable. LDA is applied to construct the adaptive variable.

LTREE [45] constructs multivariate decision trees with binary splits. Like Friedman's approach, LTREE generates new features using LDA and uses these extracted features in the binary split. These extracted features can also be used in the descendant nodes of the original node.

CART [6] finds the coefficients of the features using a step-wise procedure, where in each step, one cycles through the features $x_1, x_2, \ldots, x_d$ doing a search for an improved linear combination split. Each instance is normalized by centering each value of each feature at its median and then dividing by its interquartile range.

OC1 [7] is an extension to the basic CART. A perturbation vector is added to the discriminant once there is convergence that is no improvement in the impurity. Such perturbation helps CART to make conjugate jumps in the discriminant hyper-space. Another extension is running CART 20-50 times and selecting the best discriminant according to the impurity measure.

In neural trees [46], at each node of the decision tree, there is a neural network trained with its corresponding data. Once the weights of the neural networks are found, they can be used to channel the instance to the correct branch. The nodes are binary; therefore, they propose to use an heuristic to group $K > 2$ classes into two metagroups before training the neural network.

FACT [47] creates a K-ary multivariate tree, where each decision node can have can have K branches. Here, K represents the number of classes in the data set. Contrary to the binary trees, in FACT, each branch has its own discriminant calculated using LDA, and an instance is forwarded to the ith branch that minimizes expected risk.

QUEST [48] is an improved version of FACT and uses binary splits instead of K-ary splits at each decision node. Like neural trees, they need to group $K > 2$ classes into two supergroups and propose to use two-means clustering for this aim. QUEST also differs from FACT in the way that it does not assume equal variances and uses the roots of the quadratic equation (due to the equation in quadratic discriminant analysis) as candidates for the split point.

In LDT [9], there are two optimization steps to find the best multivariate split. In the inner optimization step, LDA is used to separate two distinct groups of classes C_L and C_R as much as possible. In the outer optimization problem, LDT searches the best split of classes into two groups, C_L and C_R.

In LMDT [8], with K classes, as in FACT, a node is allowed to have K branches, and each branch has its own discriminant. Linear discriminants are trained to minimize the classification error, rather than an impurity measure.

Tibshirani and Hastie [49] propose a tree-based maximum margin classifier, where they search the line that partitions the classes into two groups, that has the maximum margin.

Bennett and Blue [50] investigate decision trees with support vector classifiers at each node, but they do not discuss adaptive construction of the tree topology, i.e., the tradeoff between the overall tree complexity and the complexity of support vector classifiers at each node remains to be investigated.

3 Pairwise Statistical Tests

Let X_{ij}^t denote the Bernoulli random variable representing the outcome of classifier $i = 1, \ldots, K$ on instance $t = 1, \ldots, N$ of validation fold $j = 1, \ldots, L$. $X_{ij}^t = 1$ if the outcome is error, and $X_{ij}^t = 0$ if the outcome is correct. The average error of learner i on fold j is then

$$Y_{ij} = \sum_{t=1}^{N} X_{ij}^t / N. \tag{3}$$

This corresponds to the *misclassification error rate* in classification. Y_{ij} are the sum of independent and identically distributed random variables (X_{ij}^t) and by the central limit theorem are approximately normal distributed with mean μ_i. For each learning algorithm i, the expected error is the sample average m_i (estimator of μ_i), which is calculated as

$$m_i = \sum_{j=1}^{L} Y_{ij} / L. \tag{4}$$

In the following sections, we will give the derivations of three well-known statistical tests, and our proposed test, where all of them have the null hypothesis $H_0 : \mu_1 = \mu_2$.

3.1 5×2 cv t Test

Let A_j show the difference between the error values of the two learners on fold j; so the difference between the error values of the two learners on fold 1 of replication i is $A_{(2i-1)} = Y_{1(2i-1)} - Y_{2(2i-1)}$ and on fold 2 of replication i is $A_{(2i)} = Y_{1(2i)} - Y_{2(2i)}$. s_i^2 is the estimated variance of the replication i:

$$s_i^2 = (A_{(2i-1)} - \hat{A}_i)^2 + (A_{(2i)} - \hat{A}_i)^2, \tag{5}$$

where $\hat{A}_i$ is the average of the differences on replication i: $\hat{A}_i = \frac{(A_{(2i-1)} + A_{(2i)})}{2}$. Given that, s_i^2 will be

$$\begin{aligned} s_i^2 &= \left(\frac{\left(A_{(2i-1)} - A_{(2i)}\right)}{2}\right)^2 + \left(\frac{\left(A_{(2i)} - A_{(2i-1)}\right)}{2}\right)^2 \\ &= \frac{\left(A_{(2i-1)}^2 + A_{(2i)}^2\right)}{2} - A_{(2i-1)} A_{(2i)}. \end{aligned} \tag{6}$$

Under the null hypothesis that the two means are equal, A_j/σ is unit normal ($\mathcal{Z}$), and assuming that $A_{(2i-1)}$ and $A_{(2i)}$ are independent normals (two folds of replication i), s_i^2/σ^2 is chi-square distributed with one degree of freedom ($\mathcal{X}_1^2$). Assuming also that the s_i^2 are independent

$$M = \frac{\sum_{i=1}^{5} s_i^2}{\sigma^2} = \frac{\sum_{i=1}^{10} A_i^2 - 2 \sum_{i=1}^{5} A_{(2i-1)} A_{(2i)}}{2\sigma^2} \tag{7}$$

is chi-square distributed with five degrees of freedom. We know that if $Z \sim \mathcal{Z}$ and $X \sim \mathcal{X}^2_n$ and if Z and X are independent, then $Z/\sqrt{X/n}$ is t distributed with n degrees of freedom. Then, the test statistic

$$t'_d = \frac{A_1/\sigma}{\sqrt{M/5}} = \frac{\sqrt{10}A_1}{\sqrt{\sum_{i=1}^{10} A_i^2 - 2\sum_{i=1}^{5} A_{(2i-1)}A_{(2i)}}} \tag{8}$$

is t-distributed with five degrees of freedom. Since $t^2_n \sim F_{1,n}$, the test statistic

$$F'_d = \frac{10A_1^2}{\sum_{i=1}^{10} A_i^2 - 2\sum_{i=1}^{5} A_{(2i-1)}A_{(2i)}} \tag{9}$$

is F-distributed with one and five degrees of freedom. Then, the null hypothesis is accepted with α level of confidence if $t'_d \in (-t_{\alpha/2,5}, t_{\alpha/2,5})$ or $F'_d < F_{\alpha,1,5}$ [10].

3.2 5×2 cv F Test

As explained above, under the null hypothesis that the two means are equal, A_j/σ is unit normal ($\mathcal{Z}$), so their square A_j^2/σ^2 is chi-square distributed with one degree of freedom. Since the sum of n chi-square distributed variables is also chi-square distributed, the sum of squares of differences

$$T = \frac{\sum_{i=1}^{10} A_i^2}{\sigma^2} \tag{10}$$

is chi-square distributed with 10 degrees of freedom. According to (7), sum of the variances of the differences, M, is chi-square distributed with five degrees of freedom. We know that if $X \sim \mathcal{X}^2_m$, $Y \sim \mathcal{X}^2_n$, and if X and Y are independent, then the ratio $\frac{X/m}{Y/n}$ is F-distributed with m and n degrees of freedom. So, if we also assume that the sum of squares of the differences and the sum of the variances of the differences are independent of each other, the test statistic

$$F'_a = \frac{T/10}{M/5} = \frac{\sum_{i=1}^{10} A_i^2}{\sum_{i=1}^{10} A_i^2 - 2\sum_{i=1}^{5} A_{(2i-1)}A_{(2i)}} \tag{11}$$

is F-distributed with 10 and 5 degrees of freedom. Then, the null hypothesis is accepted with α level of confidence if $F'_a < F_{\alpha,10,5}$ [51].

3.3 k-Fold Paired t Test

We know that if a sample with k examples is normally distributed, then

$$\frac{\sqrt{k}(m-\mu)}{S} \tag{12}$$

is t distributed with $k-1$ degrees of freedom. Since there are L normally distributed A_is (assumption) and we test $\mu = 0$,

$$t'_k = \sqrt{L}\frac{\bar{A}}{S} \tag{13}$$

is t distributed with $L-1$ degrees of freedom, where $\bar{A}$ and S^2 are defined as

$$\bar{A} = \frac{\sum_{i=1}^{L} A_i}{L} \tag{14}$$

$$\begin{aligned} S^2 &= \frac{\sum_{i=1}^{L} \left(A_i - \bar{A}\right)^2}{L-1} \\ &= \frac{\sum_{i=1}^{L} A_i^2 - 2\left(\sum_{i,j=1,i\neq j}^{L} A_i A_j\right)/(L-1)}{L}. \end{aligned} \tag{15}$$

If we put the values of $\bar{A}$ and S in (13), we get the test statistic ($L = 10$)

$$t'_k = \frac{\sum_{i=1}^{10} A_i}{\sqrt{\sum_{i=1}^{10} A_i^2 - 2/9\sum_{i,j=1,i\neq j}^{10} A_i A_j}}. \tag{16}$$

Since $t^2_n \sim F_{1,n}$, the test statistic

$$F'_k = \frac{\left(\sum_{i=1}^{10} A_i\right)^2}{\sum_{i=1}^{10} A_i^2 - 2/9\sum_{i,j=1,i\neq j}^{10} A_i A_j} \tag{17}$$

is F-distributed with one and nine degrees of freedom. Then, the null hypothesis is accepted with α level of confidence if $t'_k \in (-t_{\alpha/2,9}, t_{\alpha/2,9})$ or $F'_k < F_{\alpha,1,9}$.

3.4 Combined 5×2 cv t Test

In this section, we give the details of our proposed combined 5×2 cv t test. The motivation of our test is like the motivation in [51], i.e., the 5×2 cv t test uses A_1 in the numerator. This is arbitrary; actually, there are 10 possible A_i that can be placed there and one may notice that depending on which one we use, the test sometimes accepts and sometimes rejects. This is disturbing because this depends on the order of folds and replications that should not effect the test. As explained above, Alpaydin [51] has previously used this to improve the 5×2 cv t test and proposes the 5×2 cv F test that uses all 10 A_i. The drawback of 5×2 cv F test is that it uses square of the differences of error values and cannot be used for a one-sided test.

So, if $A_i/\sigma \sim \mathcal{Z}$, then $\bar{A}$ (defined in (14)) is $\mathcal{N}(0, 1/10)$ and, therefore, $\sqrt{10}\bar{A}/\sigma \sim \mathcal{Z}$. If we place $\bar{A}$ instead of A_1 in (8), we get

$$t'_y = \frac{\sqrt{10}\bar{A}}{\sqrt{M/5}} = \frac{\sum_{i=1}^{10} A_i}{\sqrt{\sum_{i=1}^{10} A_i^2 - 2\sum_{i=1}^{5} A_{(2i-1)}A_{(2i)}}}, \tag{18}$$

which is t distributed with five degrees of freedom. Since $t^2_n \sim F_{1,n}$, the test statistic

$$F'_y = \frac{\left(\sum_{i=1}^{10} A_i\right)^2}{\sum_{i=1}^{10} A_i^2 - 2\sum_{i=1}^{5} A_{(2i-1)}A_{(2i)}} \tag{19}$$

is F-distributed with one and five degrees of freedom. Then, the null hypothesis is accepted with α level of confidence if $t'_y \in (-t_{\alpha/2,5}, t_{\alpha/2,5})$ or $F'_y < F_{\alpha,1,5}$. We expect this test to be more robust because it uses $\bar{A}$ (the average of 10 A_i values) instead of one A_1.

In general, when comparing two classifiers A and B, we perform two one-sided tests, namely ($\mu_a \leq \mu_b$ and $\mu_a \geq \mu_b$). If one of them rejects, we can conclude that one has significantly larger error rate than the other. For example, if the test $\mu_a \leq \mu_b$ is rejected, we can conclude that A has

larger error than B. If both tests accept, we can conclude they have the same error rate. On the other hand, in two-sided tests, one only tests for equality ($\mu_a = \mu_b$), if the test accepts, no problem, but if the test rejects, one cannot statistically decide if A has lower error rate than B or B has lower error rate than A; therefore, one cannot use two-sided tests for decisions like that.

One tailed tests make it easier to reject the null hypothesis when the alternative is true. A large sample, two-sided, 0.05 level t test puts a probability of 0.025 in each tail. It needs a t statistic of less than -1.96 to reject the null hypothesis of no difference in means. A one-sided test puts all of the probability into a single tail. It rejects the hypothesis for values of t less than -1.645. Therefore, a one-sided test is more likely to reject the null hypothesis when the difference is in the expected direction, which makes one-sided tests very attractive to machine learning practitioners whose definition of success is having a statistically significant result.

3.5 Discussion

If we compare the test statistic of our proposed test, F'_y, with the test statistics of the other tests, F'_d (5×2 cv t test), F'_k (k-fold paired t test), and F'_a (5×2 cv F test), we see

- $F'_d > F'_y$, when $10A_1^2 > (\sum_{i=1}^{10} A_i)^2$, or $A_1^2 > 10\bar{A}^2$, that is, 5×2 cv t test rejects more than the combined 5×2 cv t test, when A_1 is significantly larger ($\sqrt{10}$ times) than the mean of A_is. As explained above, this makes 5×2 cv t less robust than our proposed the combined 5×2 cv t test since the result of the test depends on the order of folds and replications.
- $F'_a > F'_y$, when $\sum_{i=1}^{10} A_i^2 > (\sum_{i=1}^{10} A_i)^2$, that is, 5×2 cv F test rejects more than the combined 5×2 cv t test, when the sum of squares is larger than the square of sums. If the first classifier is better than the second in half of the folds, and if the reverse is true for the other half of the folds, F'_a will be far larger than F'_y, and 5×2 cv F test will reject the equality of performances of those two classifiers, whereas the combined 5×2 cv t test accept it.
- $F'_k > F'_y$, when $\sum_{i \neq j}^{10} A_i A_j / 9 > \sum_{i=1}^{5} A_{(2i-1)} A_{(2i)}$, that is, k-fold paired t test rejects more than the combined 5×2 cv t test, when the average of the products of differences in k folds (of one replication) is larger than the average of the products in two folds of the five replications. For small samples, k-fold cv will produce very small test sets, which in turn increase variance and F'_k. Therefore, k-fold paired t test due to its nature, may be more sensitive to outliers than the combined 5×2 cv t test.

4 Multivariate Rule Induction (R.Multi)

R.MULTI is the multivariate version of RIPPER where each condition is a linear combination of the features, as in (2). To get multivariate rules, one can first train a multivariate tree and then write paths from the root to leaves as a set of multivariate rules, as C4.5RULES does for univariate trees. However, Cohen [5] has shown that learning rules directly is faster and more accurate than learning a tree and than converting it to rules—RIPPER is faster and more accurate than C4.5RULES. We, therefore, opted for learning multivariate rules directly and toward this aim, we hereby propose a multivariate version of RIPPER namely R.MULTI.

4.1 R.Multi versus R.Uni

R.MULTI differs from original R.UNI in several respects: First, because we take a weighted sum, discrete attributes should be converted to a numeric form; this is done by 1-of-L encoding by defining L dummy 0/1 variables for a discrete attribute with L possible values.

Second, in training a multivariate linear decision, an SVM with linear kernel is used to find the weight vector $\boldsymbol{w} = [w_1, w_2, \ldots, w_d]^T$ and the threshold w_0 that best separates the two classes. The details of finding $\boldsymbol{w}$ and w_0 will be explained in Section 4.2. We choose a linear SVM for the following main reasons: 1) As far as our knowledge, a linear SVM is one of the best learning algorithms to linearly discriminate two classes, 2) SVM-based algorithms are discriminant based, and therefore, there is no need to estimate the class densities or the exact posterior probability values as in Bayesian classifiers, and 3) SVM-based algorithms are formulated as convex optimization problems, and there is a single optimum that we can solve analytically. So, we are not bothered with heuristics for learning rates as in other linear discriminants such as linear perceptron or logistic regression.

Third, RIPPER uses MDL to check model complexity and avoid overfitting, whereas R.MULTI uses cv as the model selection method. This is done in two places: to stop adding conditions to a rule, and to stop adding rules to the rule set. The MDL calculation used in RIPPER cannot be generalized to the multivariate case because of the combinatorial number of possible hyperplanes that can be drawn [7]. We have, therefore, decided to use cv as the model selection method in R.MULTI, instead of MDL. For this, at each stage, we leave out one-third of the data set as validation set and use the remaining two-thirds for actual training. We stop adding a condition to a rule, or adding a rule to the rule set, if the error on the validation set stops decreasing. Similarly, in the optimization phase of R.MULTI, the two rules, revision and replacement, and the original rule are compared according to the error rate on the validation set, and the one with the smallest validation error rate is selected and put in place of the original rule.

4.2 Finding Best Multivariate Condition: Linear SVM

In SVM, the classification task is defined as a maximization problem [52]. The distance from the separating hyperplane to the instances closest to it on either side is called margin, and the optimal separating hyperplane is the one that maximizes the margin. For the nonseparable case, we require

$$r^t \left(\boldsymbol{w}^T \boldsymbol{x}^t + w_0 \right) \geq 1 - \epsilon^t \tag{20}$$

for each data point $\boldsymbol{x}^t$ with output r^t. If $\epsilon^t = 0$, the instance is on the separating hyperplane. If $0 < \epsilon^t < 1$, the instance is correctly classified but it is in the margin; otherwise, it is

misclassified. Maximizing margin is equivalent to minimizing $\|\boldsymbol{w}\|$, and after adding penalty term ($\sum_t \epsilon^t$) to the objective function, we get the formulation

$$L_p = \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_t \epsilon^t - \sum_t \alpha^t \left[r^t(\boldsymbol{w}^T\boldsymbol{x}^t + w_0) - 1 + \epsilon^t\right] - \sum_t \mu^t \epsilon^t, \tag{21}$$

where α^t and μ^t are Langrange multipliers, and C is the penalty factor. When the quadratic problem is solved, instances whose $\alpha^t > 0$ are called support vectors. $\boldsymbol{w}$ is calculated as the weighted sum of these support vectors

$$\boldsymbol{w} = \sum_t \alpha^t \boldsymbol{x}^t, \tag{22}$$

and the threshold w_0 is calculated as

$$w_0 = \frac{1}{N}\sum_t r^t - \boldsymbol{w}^T x^t, \tag{23}$$

where N is the sample size.

5 Model Selection in Rule Induction: Omnivariate Rule Induction (R.Omni)

The model selection problem in rule induction can be defined as choosing the best model at each condition in a rule. In approximating the real (unknown) discriminant with univariate conditions, we are limited to a piecewise approximation using axis-aligned hyperplanes. With multivariate linear conditions, we can use arbitrary hyperplanes and, thus, approximate the discriminant better. The omnivariate variant, R.OMNI, at each condition, trains and compares the two possible conditions, univariate and linear multivariate, and uses a statistical test to check for statistically significant difference. Only if the test indicates that the multivariate condition has significantly higher accuracy do we choose the multivariate condition; otherwise, we choose the univariate condition due to its simplicity.

Each condition type has a certain bias; using multivariate linear condition, for example, we are assuming that the input space can be divided using hyperplanes into localized regions (volumes) where classes or groups of classes are linearly separable. Using a rule with the same type of conditions everywhere, we assume that the same bias is appropriate in all cases. The omnivariate approach [11] advocates the view that this assumption is not always correct and that at each condition of the rule, which corresponds to a different subproblem defined by the subset of the training data reaching that condition, a different model may be appropriate, and that the right model should be found and used. For example, we expect that while training the initial conditions in a rule (when we have more data), a multivariate model may be used; but once we have added a number of conditions, we have easier problems in effectively smaller dimensional subspaces, and at the same time, we have smaller training data and simple, for example, univariate, splits may suffice and generalize better.

The major differences between the ODT algorithm [11] and the omnivariate rule induction algorithm R.OMNI are as follows:

```
1  Ruleset OmnivariateRuleInduction(X, V)
2      RS = {}
3      Classes C_i ordered in increasing prior probability
4      for p = 1 to K − 1
5          Pos = C_p, Neg = C_{p+1}, ..., C_K
6          RS_p = {}
7          while X contains positive samples
8              Divide X into Grow set G and Prune set P
9              r = GrowRuleOmnivariate(G)
10             PruneRuleOmnivariate(r, P)
11             if CalculateError(r)> 0.5
12                 break
13             else
14                 RS_p = RS_p + r
15                 Remove examples covered by r from X
16         for i = 1 to 2
17             OptimizeRulesetOmnivariate(RS_p, X, V)
18             SimplifyRulesetOmnivariate(RS_p, V)
19         RS = RS + RS_p
20     return RS
```

Fig. 2. Pseudocode for learning an omnivariate rule set using cv.

- ODT generates decision trees using a divide-and-conquer approach (Section 2.2), whereas R.OMNI generates rule sets using a separate-and-conquer approach (Section 2.1). Although decision trees can be converted into rule sets, as Cohen has pointed in [5], learning rules directly is both fast and accurate than learning a tree and than converting it to rules.
- ODT uses LDA to find the splits in the multivariate nodes, whereas R.OMNI uses a linear SVM to find the splits in the multivariate conditions.
- ODT compares candidate models (univariate or multivariate) using the combined 5×2 cv F test, whereas R.OMNI does it with the proposed combined 5×2 cv t test.
- ODT chooses between three candidate models (univariate, multivariate linear, and multivariate nonlinear), whereas R.OMNI chooses between two candidate models (univariate and multivariate linear).

The pseudocode for learning omnivariate rule set from examples using cv is given in Fig. 2. We start learning from an empty rule set (Line 2), and learn rule set for each class C_i one at a time. To do this, we sort the classes increasingly according to prior probabilities (Line 3), and we try to separate each class C_p (positives) from the remaining classes $C_{p+1}, \ldots, C_K$ (negatives) (Line 5). Rules are grown (Line 8), pruned (Line 9) and added (Line 13) one by one to the rule set. We stop adding rules when 1) the training error of the rule is larger than 0.5 (Line 10) or 2) if there are no remaining positive examples (Line 7). After learning a rule set, it is optimized twice (Line 16) and pruned (Line 17). The difference of R.OMNI from R.UNI occurs in two places: First, the rule is grown and pruned with separate data sets called grow set G and pruning set P. Second, the optimization and simplification are done using a validation data set V.

The pseudocode for optimizing a rule set using cv is given in Fig. 3. In the optimization phase, two alternatives are grown for each rule (Line 2). The first candidate, the replacement rule, is grown (Line 4) and pruned (Line 5)

```
1  Ruleset OptimizeRulesetOmnivariate(RS, X, V)
2      for each rule r in RS
3          Divide X into Growset G and Pruneset P
4          r_replace = GrowRuleOmnivariate(G)
5          PruneRuleOmnivariate(r_replace, P)
6          r_revise = GrowRuleOmnivariate(G)
7          PruneRuleOmnivariate(r_revise, P)
8          RS_replace = RS − r ∪ r_replace
9          RS_revise = RS − r ∪ r_revise
10         E = CalculateError(RS, V)
11         E_replace = CalculateError(RS_replace, V)
12         E_revise = CalculateError(RS_revise, V)
13         r_min = The rule with min(E, E_replace, E_revise)
14         RS = RS − r ∪ r_min
15     return RS
```

Fig. 3. Pseudocode for optimizing rule set RS on training set X and validation set V.

starting with an empty rule, whereas the second candidate, the revision rule, is grown (Line 6) and pruned (Line 7) starting with the current rule. These two rules and the original rule are compared, and the one with the smallest error on the validation set V (Lines 10-13) is selected and put in place of the original rule (Line 14).

The pseudocode for simplifying a rule set using cv is given in Fig. 4. In simplifying the rule set, rules are pruned in reverse order (Line 3). We prune a rule from the rule set if its removal decrease the error on the validation set X (Lines 4-8).

The pseudocode for growing an omnivariate rule from examples using a cv model selection technique is given in Fig. 5. We start learning from an empty rule (Line 2) and add conditions one by one. To add a condition, we train two different candidate models, univariate model (Line 6) and multivariate linear model (Line 7). We first generate 10 training and validation sets using 5×2 fold cv (Line 4), and find the best model using the validation errors of the models and the statistical test t (Lines 9-13). The idea is to keep the architecture simple, unless the additional complexity is justified by the significant decrease in error, and in applying the test, we take complexity into account as follows [53]: When comparing a univariate model with the multivariate linear model, we test if the univariate model has an expected error rate less than or equal to the expected error of the multivariate linear model:

$$H_0 : \mu_{uni} \leq \mu_{multi} \text{ versus } H_1 : \mu_{uni} > \mu_{multi}.$$

By assuming a prior preference of a univariate model to a multivariate linear model, we would like to test whether it is supported by the data, the hypothesis follows the prior and is one sided. If the test does not reject, we favor the univariate model: Either $\mu_{uni} < \mu_{multi}$, that is, the simpler model indeed has less error, and we choose it because it is more accurate; or, $\mu_{uni} = \mu_{multi}$, and we prefer the simpler model. The multivariate linear model is favored only if the test rejects, i.e., when the additional complexity is justified by the significant decrease in error and the test (data) overrides our prior preference. That is, accuracy is checked first and given equal accuracies, the simpler model is favored. When we find the best condition (model), we add the condition to the rule and remove examples covered by that condition from the data set (Line 16). We stop adding

```
1  Ruleset SimplifyRulesetOmnivariate(RS, X)
2      E = CalculateError(RS, X)
3      for each rule r in RS in reverse order
4          RS_removed = RS − r
5          E_removed = CalculateError(RS_removed, X)
6          if E_removed < E
7              RS = RS − r
8              E = E_removed
9      return RS
```

Fig. 4. Pseudocode for simplifying rule set RS on data set X.

conditions to a rule when there are no negative examples in the data set (Line 3).

6 Experiments

6.1 Comparison of Pairwise Statistical Tests

6.1.1 Experimental Setup

The seven classification algorithms we use because of their low time/space complexity are as follows:

1. *max* decides based on the prior class probability without looking at the input. All test instances are assigned to the class with the maximum prior. It is not a learning algorithm in the usual sense, but *any* plausible learning algorithm must have smaller error rate than *max*; it is indeed surprising that *max* is sometimes quite accurate.
2. *mean* is the nearest mean classifier that keeps the mean vector for each class and assigns instance to the class whose mean has the smallest euclidean distance to the instance [54]. This corresponds to assuming that classes are Gaussian distributed with a shared covariance matrix whose diagonals are equal and off diagonals are 0.
3. *lda* is the well-known linear classification algorithm.

```
1  Rule GrowRuleOmnivariate(X)
2  r = {}
3  while X covers negative examples
4      Generate k training and validation sets
       using 5×2 fold cross-validation
5      for j = 1 to k
6              Use exhaustive search to find
               best univariate condition c_uni
7              Use SVM with linear kernel to find
               best multivariate condition c_linear
8          E_c^j = CalculateError(c, V_j)
9      Test H_0 : μ_uni ≤ μ_multi using statistical test t
10     if H_0 is accepted
11         r = r ∪ c_uni
12     else
13         r = r ∪ c_multi
14     Remove examples from X that are covered by c
15 return r
```

Fig. 5. Pseudocode for growing an omnivariate rule using data set X.

4. *lnp* linear perceptron with softmax outputs trained by gradient-descent to minimize cross entropy.
5. RIPPER.
6. *c45* is the archetypal decision tree method [2].
7. *nn* [55] is the 1-nearest neighbor classification algorithm and uses the euclidean distance.

These classification algorithms are tested on 40 data sets from the UCI machine learning repository [56]. Each data set is resampled using 5×2 cv [10] where twofold cv is done five times (with stratification) and the roles swapped at each fold to generate 10 training and testing folds.

6.1.2 One-Sided Pairwise Test Results

In the first part of our experiments, we compare our proposed combined 5×2 cv t test with the 5×2 cv t test. Toward this aim, we look at the performance of the tests in comparing the expected error of two different algorithms. We have seven different algorithms; therefore, we can make 21 different pairwise comparisons (*max* versus *mean*, *max* versus $lda, \ldots, c45$ versus *nn*). For each pair, we test the one-sided hypothesis $H_0 : \mu_1 \leq \mu_2$, where μ_1 and μ_2 are the expected error of the first and second algorithms, respectively. This test is done on 40 data sets where for each data set the rejection probability is calculated on 1,000 runs (of 5×2 folds). As in [51], to compare the type I and II errors of the statistical tests, for each data set and for each algorithm pair, we calculate the normalized distance between the expected error of the two algorithms:

$$z = \frac{m_1 - m_2}{(s_1 + s_2)/2}, \tag{24}$$

where m_1 and m_2 are the average expected error of the first and second algorithms and s_1 and s_2 denote the standard deviation of the error values of the first and second algorithms on 1,000 runs. A small difference in expected error result in a smaller z measure, which implies we have similar algorithms and a rejection would be a type I error. On the other hand, a large z measure denotes large difference between expected error, and we expect larger rejection rates for a test to have lower type II error (and higher power). Since we have 40 data sets and 21 classifier pairs, we have 840 different z value and rejection probability pair for the two tests, and we plot these in Fig. 6. For visualization purposes, we have also fitted a function to the data of both tests. We see from the figure that for $z > 1.6$, the 5×2 cv t test rejects only 20 percent of the cases whereas the combined 5×2 cv t test rejects almost 95 percent of the cases, indicating that the combined test has higher power (lower type II error). Similarly, when $z < 0$, the combined test has a lower probability of reject indicating that it has lower type I error.

Replicability of an experiment is a measure of how well the outcome of the experiment can be reproduced [57]. When an experiment is repeated n times with different randomizations of a given data set, the experiment is consistent if its outcome is the same for all n experiments and is almost consistent if its outcome is the same for all $n-1$ experiments. The replicability is defined as the probability that two runs of the statistical test on the same data set will produce the same outcome. Bouckaert and Frank [57] estimated this probability as

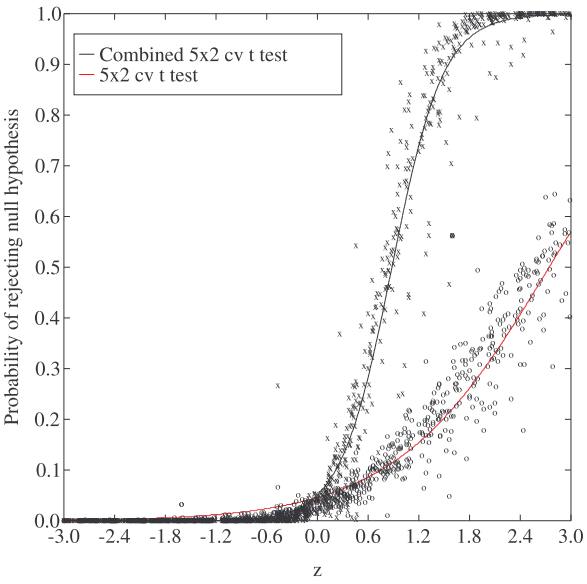

Fig. 6. Comparison of type I and II errors of the combined 5×2 cv t with the 5×2 cv t test on 40 classification problems. x-axis is $z = \frac{m_1 - m_2}{(s_1+s_2)/2}$, y-axis is the rejection probability of $H_0 : \mu_1 \leq \mu_2$.

$$R(n,k) = \frac{k(k-1) + (n-k)(n-k-1)}{n(n-1)}, \tag{25}$$

where n denotes the number of experiments and k denotes the number of times the test rejects the null hypothesis.

According to the one-sided test results, our proposed combined 5×2 cv t test (0.8928) has higher replicability than the 5×2 cv t test (0.8039).

6.1.3 Two-Sided Pairwise Test Results

In the second part of our experiments, we compare our proposed combined 5×2 cv t test with the three other tests explained in this paper. Similar to Section 6.1.2, we look at the performance of the tests in comparing the expected error of two different algorithms. In this case, for each pair, we test the two-sided hypothesis $H_0 : \mu_1 = \mu_2$, where μ_1 and μ_2 are the expected error of the first and second algorithms, respectively. We again have seven algorithms, 40 data sets, 1,000 runs making a total of 840,000 comparisons. To compare the type I and II errors of the statistical tests, for each data set and for each algorithm pair, we now calculate the absolute normalized distance between the expected error of the two algorithms:

$$z = \frac{|m_1 - m_2|}{(s_1 + s_2)/2}. \tag{26}$$

A large z measure denotes large difference between expected error, and we expect larger rejection rates for a test to have lower type II error (and higher power). We see from Fig. 7 that for $z > 2.1$, the 5×2 cv t test rejects only 20 percent and 5×2 cv F test rejects only 30 percent of the cases whereas the combined 5×2 cv t test rejects almost 95 percent of the cases, indicating that the combined test has higher power (lower type II error).

A small difference in expected error result in a smaller z measure, which implies we have similar algorithms and a rejection would be a type I error. We see from Fig. 7 that for small z values ($z \leq 0.3$), 5×2 cv F test has the lowest type I

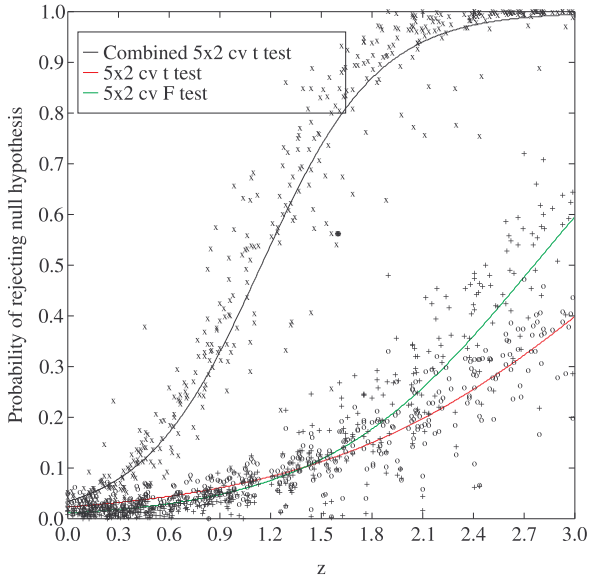

Fig. 7. Comparison of type I and II errors of the combined 5×2 cv t with the 5×2 cv t test and 5×2 cv F test on 40 classification problems. x-axis is $z = \frac{|m_1 - m_2|}{(s_1 + s_2)/2}$, y-axis is the rejection probability of $H_0 : \mu_1 = \mu_2$.

error, then comes the 5×2 cv t test, and the combined test has the highest type I error. Note that despite higher type I error, only our proposed combined 5×2 cv t test has the tendency to reject the null hypothesis while the error difference gets larger.

According to the two-sided test results, again our proposed combined 5×2 cv t test (0.8891) has the highest replicability, followed by the 5×2 cv F test (0.8674) and 5×2 cv t test (0.8411) has the lowest replicability.

6.2 Comparison of Omnivariate Rule Induction with Multivariate and Univariate Rule Induction

6.2.1 Experimental Setup

We use a total of 54 data sets from UCI [56] and Delve [58] repositories. Our methodology in the division of training, validation, and test sets is as follows: A data set is first divided into two parts, with $1/3$ as the test set, *test*, and $2/3$ as the training set, *train-all*. The training set, *train-all*, is then resampled using 2×5 cv where fivefold cv is done two times (with stratification) and the roles swapped at each fold to generate 10 training and validation folds, tra_i, val_i, $i = 1, \ldots, 10$. tra_i are used to learn the rule sets. The optimization and simplification of the rule sets are done using val_i. The test set, *test*, is used to estimate the generalization error of the rule sets.

We compare nine algorithms in the second part of our experiments:

1. C45: C4.5 is the archetypal decision tree method [2].
2. CN2: CN2 rule learning algorithm.
3. PART: PART rule learning algorithm.
4. CART: Multivariate decision tree learning algorithm using backfitting to find the discriminant at each decision node.
5. LDT: Multivariate decision tree learning algorithm using LDA to find the discriminant at each decision node.
6. R.UNI: Ripper the proper, original univariate rule induction algorithm.
7. R.MULTI: Our proposed multivariate rule induction algorithm, where an SVM with linear kernel is used to find multivariate linear conditions. We use the LIBSVM 2.82 library that implements the linear SVM [59]. We tune the regularization parameter C using val_i on each data set.
8. R.OMNI(F): Our proposed omnivariate rule induction algorithm, where combined 5×2 cv F test is used to compare univariate and multivariate candidate models for each decision condition.
9. R.OMNI(t): Our proposed omnivariate rule induction algorithm, where combined 5×2 cv t test is used to compare univariate and multivariate candidate models for each decision condition.

In our simulations, we saw that the optimization phase should be repeated twice, also for the multivariate case, as in RIPPER. These two optimization iterations do decrease error rate and rule set size, but repeating it further does not have such a drastic effect and does not justify the extra training time.

Our comparison criteria are the generalization error of the algorithms, the number of rules and the number of conditions in those rules they generate, and the time required to train the algorithm.

Since we have more than two algorithms, on each data set, we cannot compare the average performances on 54 data sets and one needs to resort to nonparametric tests. Friedman's test and its posthoc Nemenyi's test use ranks instead of the absolute performances [60]. On each data set, the performance values of the algorithms are sorted from the best to the worst so that the best one gets the rank of 1, the second 2, and so on, until we get to 9. We then use nonparametric tests to check for significant difference in average ranks over the 54 data sets.

According to *Nemenyi's test*, two neighboring algorithms lead to classifiers with significantly different performance ranks at significance level α if the difference of their average ranks is greater than or equal to the critical difference

$$CD = q_\alpha \sqrt{\frac{L(L+1)}{6M}}, \tag{27}$$

where L is the number of algorithms, M is the number data sets, and q_α is the Studentized range statistic divided by $\sqrt{2}$. This allows us to find cliques of equally good subsets that we can represent by underlining them.

6.2.2 Results

Table 1 shows the average and standard deviations of expected error of rule sets produced by C45, CN2, PART, CART, LDT, R.UNI, R.MULTI, R.OMNI(F), and R.OMNI(t) algorithms. With respect to the expected error, R.MULTI is better than both omnivariate algorithms that seem to be similar and are better than R.UNI proper. The results of the Nemenyi's test on the expected error is given in Fig. 8a. We see that there are three cliques: (R.OMNI(t), R.OMNI(F), R.MULTI, LDT), (R.OMNI(t), R.OMNI(F), LDT, PART, R.UNI, CART, C45), and (PART, CART, R.UNI, CN2, C45).

TABLE 1
The Average and Standard Deviations of Expected Error of Rule Sets Produced by C45, CN2, PART, CART, LDT, R.UNI, R.MULTI, R.OMNI(F), and R.OMNI(t) Algorithms

Dataset	C45	CN2	PART	CART	LDT	R.UNI	R.MULTI	R.OMNI(F)	R.OMNI(t)
artificial	0.7± 1.6	0.0± 0.0	0.0± 0.0	0.9± 2.9	0.0± 0.0	0.4± 1.2	0.0± 0.0	0.0± 0.0	0.0± 0.0
australian	13.9± 0.0	22.4± 2.1	16.5± 1.6	14.2± 1.2	12.9± 0.9	14.2± 0.7	13.5± 0.5	14.4± 0.8	13.9± 0.3
balance	40.5± 2.7	36.1± 1.5	29.2± 4.2	15.8± 3.6	11.2± 1.3	34.6± 1.1	9.9± 1.8	9.8± 1.3	8.9± 0.8
breast	6.9± 1.2	7.5± 0.9	7.7± 1.7	3.5± 0.8	4.9± 0.5	5.9± 1.6	3.5± 0.5	4.0± 0.6	3.6± 0.6
bupa	38.5± 4.4	38.7± 4.8	39.1± 3.7	39.7± 5.6	40.7± 2.4	37.8± 5.0	37.2± 3.4	37.8± 2.7	38.4± 3.6
car	12.5± 1.9	28.7± 0.3	5.2± 1.1	5.9± 1.1	7.1± 0.6	20.4± 2.1	7.6± 1.1	7.8± 0.8	7.2± 1.1
cmc	50.9± 2.5	56.2± 1.1	50.1± 1.9	51.9± 3.4	51.2± 2.2	51.0± 4.1	56.5± 1.2	55.1± 2.5	56.9± 0.9
credit	16.1± 1.2	18.1± 1.6	20.7± 2.9	17.1± 1.7	15.5± 0.8	15.5± 0.6	14.7± 1.2	15.2± 1.1	15.1± 0.9
cylinder	32.2± 1.4	26.2± 2.2	29.4± 3.2	35.3± 4.9	28.5± 2.5	33.3± 2.9	28.3± 3.1	28.5± 2.4	29.1± 2.4
dermatology	12.5± 1.5	15.3± 2.4	12.8± 1.7	14.3± 2.1	3.9± 1.2	12.5± 0.7	5.0± 0.8	7.0± 1.5	7.1± 1.3
donors	7.8± 0.7	10.8± 0.7	7.4± 0.5	7.3± 0.5	5.2± 0.2	6.1± 0.3	5.3± 0.2	5.2± 0.2	5.5± 0.1
ecoli	21.3± 2.7	19.4± 1.5	19.1± 3.0	18.3± 4.9	15.0± 3.5	17.6± 1.2	17.0± 3.9	19.0± 3.3	17.5± 4.2
flags	38.7± 3.5	46.4± 3.0	42.4± 3.5	52.5± 3.6	50.9± 3.2	44.8± 2.4	49.3± 4.6	44.9± 4.1	49.9± 6.9
flare	12.3± 0.8	14.1± 1.7	16.0± 2.3	12.8± 2.1	11.9± 0.0	13.2± 1.4	11.9± 0.0	12.1± 0.6	11.9± 0.0
german	29.4± 1.2	29.4± 2.0	27.4± 2.4	29.3± 1.5	23.5± 1.0	28.0± 3.1	23.8± 2.2	24.2± 1.5	23.0± 1.9
glass	37.4± 3.4	40.4± 2.2	38.0± 5.3	42.0± 5.1	42.8± 3.0	45.8± 4.2	40.4± 3.4	40.0± 4.8	40.8± 3.6
haberman	26.4± 0.3	27.3± 3.0	31.8± 4.0	26.8± 0.7	26.7± 0.6	26.2± 0.5	26.2± 0.9	26.2± 0.5	26.8± 0.7
hayesroth	26.7± 1.5	21.1± 1.8	28.5± 5.1	20.7± 4.0	16.4± 4.0	23.5± 4.2	16.7± 2.1	17.5± 3.0	18.4± 1.6
heart	30.9± 4.0	29.7± 4.0	25.4± 3.7	21.3± 4.7	17.4± 2.7	32.0± 4.1	17.4± 2.9	17.3± 3.8	17.8± 2.9
hepatitis	22.3± 4.0	20.0± 4.0	22.5± 3.3	22.3± 1.6	20.4± 1.6	24.0± 4.4	18.5± 2.3	22.5± 3.1	19.6± 2.0
horse	18.4± 2.8	14.4± 2.2	13.9± 2.8	17.2± 2.2	14.7± 1.9	15.2± 2.2	12.0± 2.4	12.6± 2.3	11.9± 2.0
iris	7.6± 1.1	6.1± 3.3	8.2± 1.5	7.1± 3.5	2.7± 1.0	11.2± 0.9	4.5± 1.6	9.8± 2.3	7.8± 2.4
ironosphere	14.2± 4.4	9.6± 2.3	13.2± 3.8	12.2± 2.2	12.0± 1.7	16.6± 2.9	10.7± 1.8	12.2± 1.8	12.5± 2.4
krvskp	1.2± 0.4	4.3± 0.6	1.5± 0.3	3.2± 0.9	3.7± 0.4	1.4± 0.6	2.0± 0.6	2.1± 0.8	2.5± 0.9
letter	18.4± 0.5	58.8± 1.1	16.7± 0.5	19.5± 0.7	17.4± 0.7	21.0± 0.5	56.6± 2.7	25.7± 1.3	56.9± 2.2
magic	17.1± 0.4	21.0± 0.5	16.6± 0.3	16.5± 1.0	16.7± 0.2	16.0± 0.4	24.2± 3.6	15.5± 0.5	17.0± 0.9
mammographic	20.2± 0.5	24.7± 1.1	22.7± 1.9	19.0± 1.2	18.9± 0.7	19.7± 0.7	18.7± 1.7	18.9± 1.3	19.2± 0.7
monks	14.7± 6.1	0.0± 0.0	2.8± 2.1	3.7± 3.8	17.0± 5.0	0.0± 0.0	12.6±10.6	4.3± 8.8	14.0±10.4
nursery	5.5± 0.5	22.4± 0.6	2.4± 0.3	3.1± 1.0	6.3± 0.3	5.9± 0.6	7.7± 0.8	15.3± 5.2	8.5± 1.5
ocr	23.6± 4.2	29.4± 2.7	25.1± 1.7	32.5± 4.4	12.6± 1.7	33.0± 2.8	5.8± 1.6	9.1± 3.2	7.0± 1.4
optdigits	15.0± 0.7	32.7± 2.7	11.3± 0.9	11.2± 0.8	6.3± 0.6	14.8± 1.1	7.6± 0.4	8.3± 1.9	7.5± 0.8
pageblock	3.7± 0.5	4.0± 0.3	3.3± 0.3	3.8± 0.4	4.4± 0.4	3.5± 0.3	4.2± 0.3	3.7± 0.7	3.3± 0.4
parkinsons	15.4± 4.5	15.8± 1.6	14.3± 7.3	18.8± 5.8	11.5± 1.5	15.4± 1.6	11.2± 1.3	12.6± 2.4	13.1± 2.2
pendigits	5.9± 0.5	11.2± 0.9	5.3± 0.7	4.6± 0.4	3.6± 0.5	7.0± 0.4	6.5± 0.8	9.0± 3.7	6.8± 1.9
pima	28.9± 2.7	28.4± 2.3	27.6± 2.9	30.5± 4.1	22.8± 1.0	25.8± 1.8	23.2± 0.8	24.4± 1.6	23.3± 1.6
postoperative	29.0± 0.0	33.5± 6.7	35.2± 7.0	29.0± 0.0	30.0± 3.1	29.0± 0.0	29.0± 0.0	29.0± 0.0	29.0± 0.0
promoters	24.4±10.3	16.4± 3.3	22.2± 5.1	25.6± 7.0	28.6± 4.9	20.6± 3.5	11.9± 3.2	13.9± 3.9	13.3± 5.4
ringnorm	12.0± 0.7	6.3± 0.3	6.5± 0.7	17.5± 1.0	22.7± 0.3	8.3± 1.0	11.2± 1.1	9.6± 0.8	13.7± 1.8
satellite47	14.6± 1.3	20.1± 1.0	13.7± 1.2	14.7± 1.4	16.9± 0.9	15.8± 0.9	17.2± 0.7	15.3± 1.4	15.9± 1.9
spambase	9.3± 1.2	11.4± 0.9	6.3± 0.6	7.6± 0.5	9.9± 0.4	9.2± 0.7	8.9± 0.9	8.2± 0.6	8.8± 0.9
spect	20.3± 2.5	22.1± 3.3	21.9± 2.9	21.4± 2.5	20.1± 2.1	21.3± 4.0	18.8± 2.9	19.0± 2.4	19.1± 2.9
splice	9.8± 0.9	17.7± 1.4	11.1± 1.0	10.8± 1.1	9.0± 1.0	6.7± 0.8	6.3± 0.2	6.4± 0.3	6.2± 0.4
tae	52.1± 5.6	61.2± 5.3	51.3± 3.9	55.8± 7.7	54.2± 8.7	65.4± 0.0	47.7± 6.5	50.8± 5.2	54.0± 9.4
tictactoe	22.8± 1.6	9.4± 2.1	11.6± 3.4	15.5± 6.9	28.9± 1.9	1.6± 0.2	1.6± 0.0	1.6± 0.0	1.6± 0.0
titanic	21.8± 0.5	21.1± 0.2	21.2± 0.3	21.1± 0.2	22.5± 0.4	22.4± 0.6	22.0± 0.7	22.0± 0.6	21.9± 0.4
transfusion	24.0± 0.0	22.7± 0.9	24.9± 1.8	24.0± 0.0	22.9± 1.2	22.8± 1.1	24.0± 0.0	23.8± 0.6	24.0± 0.0
twonorm	17.5± 0.6	14.2± 0.7	8.3± 0.6	2.1± 0.2	2.0± 0.1	12.0± 0.6	2.0± 0.1	2.9± 0.2	2.5± 0.3
vehicle	29.8± 2.4	45.7± 5.4	26.8± 2.6	30.3± 3.9	28.8± 2.3	43.3± 6.7	29.1± 2.1	30.8± 2.2	30.0± 1.9
vote	5.1± 0.4	7.7± 1.6	6.3± 1.3	5.4± 0.8	5.2± 1.2	6.0± 2.0	4.3± 0.3	6.3± 2.4	5.9± 2.0
wave	24.7± 1.2	38.1± 3.6	21.6± 1.1	16.8± 0.9	14.5± 0.4	23.9± 1.0	17.6± 0.3	17.5± 0.3	17.4± 0.3
wine	10.3± 3.6	8.5± 3.8	10.3± 3.8	6.7± 2.9	2.2± 1.4	11.0± 6.5	1.5± 1.2	5.3± 4.2	2.3± 1.4
winequality	45.6± 1.3	52.8± 2.5	45.5± 1.6	45.6± 1.4	45.3± 0.7	45.9± 0.8	47.4± 0.5	46.5± 0.7	46.3± 1.0
yeast	47.9± 3.2	52.3± 1.0	46.6± 2.2	49.0± 4.5	44.9± 1.1	45.6± 1.9	48.7± 1.8	45.8± 2.2	46.8± 1.7
zoo	15.9± 4.5	18.9± 4.0	16.8± 3.8	29.5± 9.0	25.7± 5.3	14.6± 2.6	14.9± 3.2	15.4± 3.8	14.1± 4.0

As we have explained in Section 4, a linear SVM is a powerful classifier, so R.MULTI is better than R.UNI in many data sets, with significant differences occurring especially on discrete data sets such as *balance*, *car*, *hayesroth*, and *promoters*. Some data sets are also inherently complex, and in those data sets, again R.MULTI is significantly better than R.UNI such as *dermatology*, *ocr*, *optdigits*, and *tae*. Contrarily, on the handcrafted data set *monks*, R.UNI can learn the optimal rules and is significantly better than R.MULTI, which overfits for this case.

Although, in general, R.MULTI is more accurate than R.UNI, there are also some other data sets, where R.UNI is more accurate. This implies that it is worth using multivariate and/or rules in certain cases. We also see that R.OMNI having both univariate and multivariate conditions outperforms R.UNI on the same 12 data sets where R.MULTI also outperforms R.UNI (*balance*, *breast*, *car*, *german*, *hayesroth*, *heart*, *ironosphere*, *optdigits*, *promoters*, *tae*, *vehicle*, *wine*), and it outperforms R.MULTI on four data sets where R.UNI also outperforms R.MULTI (*magic*, *satellite47*, *pageblock*, *ringnorm*). This shows that R.OMNI uses whichever type of condition is more appropriate, selecting the right model appropriately.

The results of the Nemenyi's test on the number of rules and number of conditions are given in Figs. 8b and 8c,

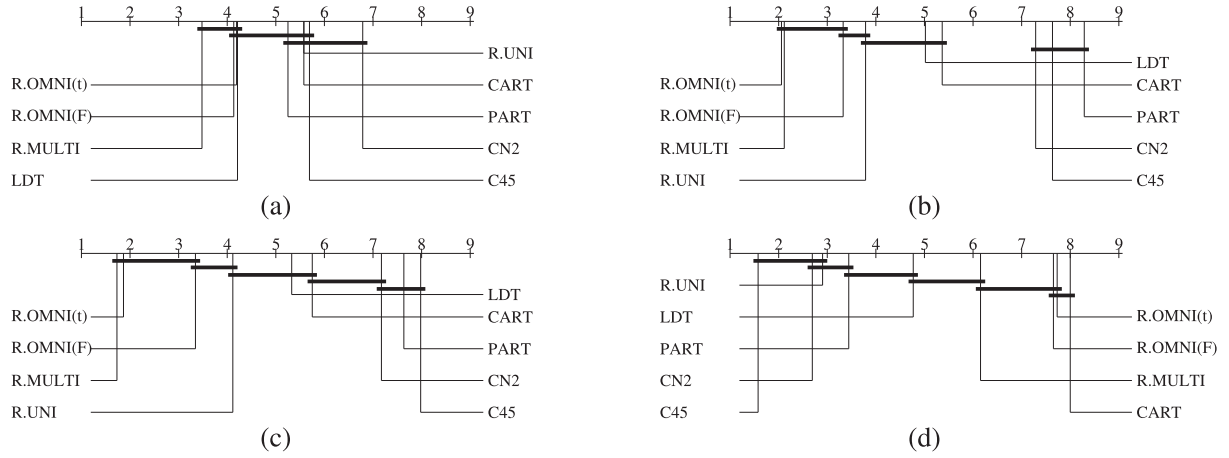

Fig. 8. The result of Nemenyi's test on the (a) expected error, (b) number of rules, (c) number of conditions, (d) training time of rule sets produced by C45, CN2, PART, CART, LDT, R.UNI, R.MULTI, R.OMNI(F), and R.OMNI(t) algorithms.

respectively. With respect to the rule set complexity, in terms of number of rules, there are four cliques: (R.OMNI(t), R.OMNI(F), R.MULTI), (R.OMNI(F), R.UNI), (R.UNI, LDT, CART), and (CN2, C45, PART). In terms of number of conditions, there are five cliques: (R.OMNI(t), R.OMNI(F), R.MULTI), (R.OMNI(F), R.UNI), (R.UNI, LDT, CART), (CART, CN2), and (CN2, C45, PART). This is mainly due to the fact that multivariate conditions are more complex than univariate conditions, and one multivariate condition is equivalent to multiple univariate conditions. Usually, R.MULTI generates significantly smaller rules on data sets with discrete attributes. For example, on *balance*, R.UNI generates on the average 4.0 rules whereas R.MULTI generates 1.5; on *mushroom*, R.UNI generates 8.1 rules compared to 1.1 rules of R.MULTI; and on *tictactoe*, R.UNI generates 8.2 rules compared to R.MULTI's 1.0.

Since R.OMNI uses a combined 5×2 cv F or t test, it requires 10 runs of univariate and multivariate condition optimizations. Therefore, it is the slowest of the three algorithms. R.UNI algorithm requires $\mathcal{O}(dNlogN)$ (sorting N data for d features using Quicksort) to find the best condition, whereas R.MULTI requires $\mathcal{O}(N^3)$ (solving convex optimization problem) to find the best condition. Therefore, if d is significantly larger than N, R.MULTI is faster than R.UNI; otherwise, R.UNI is faster than R.MULTI. The result of Nemenyi's test on the training time of the rule sets produced by C45, CN2, PART, CART, LDT, R.UNI, R.MULTI, R.OMNI(F), and R.OMNI(t) algorithms is shown in Fig. 8d. Now, there are six cliques: starting with the fastest univariate learners (C45, CN2, R.UNI), (CN2, R.UNI, PART), (PART, LDT), continuing with the multivariate learners (LDT, R.MULTI), (R.MULTI, R.OMNI(F), R.OMNI(t)), and the slowest omnivariate learners with CART (R.OMNI(F), R.OMNI(t), CART).

The number of times univariate and multivariate linear models selected in rule sets produced by omnivariate algorithms R.OMNI(F) and R.OMNI(t) are given in Table 2. Also, the selection counts of those models as a function of the position of a condition in a rule for R.OMNI(F) and R.OMNI(t) algorithms are given in Fig. 9. In R.OMNI(F), linear models are selected less than univariate models, whereas in R.OMNI(t) univariate models are selected less than the multivariate linear models. In 41 data sets out of 54 data sets, a linear model is chosen more than the univariate model by R.OMNI(t). On the other hand, in 28 data sets out of 54 data sets, a univariate model is chosen more than the linear model by R.OMNI(F).

The difference is mainly due to the nature of two statistical tests. In Section 6.1.3, we showed that combined 5×2 cv F test is more conservative than the combined 5×2 cv t test. That is, combined 5×2 cv t test rejects the null hypothesis that univariate model and multivariate model have the same expected error more than the combined 5×2 cv F test. If the null hypothesis is rejected more, since usually the linear model is better than the univariate model, the linear model is favored more. On the other hand, if the null hypothesis is not rejected (like usually in 5×2 cv F test), the simple model, namely univariate model, is favored more.

If we look at Fig. 9, we see that multivariate conditions are usually selected early on, closer to the beginning of the rule, where the complexity of the problem is high. We see that when there is few data, it is more likely that the appropriate condition is univariate, whereas with more data, a multivariate condition becomes more likely. When learning a rule, we add one condition at a time, and each condition covers a number of instances that are removed from the data. While learning the initial conditions (when there are more data), it is more likely to have a multivariate condition; as we proceed down the rule, we have less remaining data and the condition is more likely to be univariate—the multivariate condition overfit and is rejected by the statistical test.

7 Conclusions

In this paper, first, we propose a novel test, the combined $5 \times 2\, t$ test, to use as a one-sided test in comparing two classifiers. Although there are statistical tests in the literature such as the $5 \times 2\, t$ test [10] or the 5×2 F test [51], they have their own drawbacks. The $5 \times 2\, t$ test uses only the difference of the first fold error values of the algorithms, which is not robust due to the variance in folds. We replace this with a statistic that uses all differences of error values and is, therefore, more robust. The 5×2 cv F test uses square of the differences of error values and cannot be used for a one-sided test.

TABLE 2
The Number of Times Univariate and Multivariate Models Selected in Rule Sets Produced by R.OMNI(F) and R.OMNI(t) on 10 Runs

Dataset	R.OMNI(F)		R.OMNI(t)		Dataset	R.OMNI(F)		R.OMNI(t)		Dataset	R.OMNI(F)		R.OMNI(t)	
	UNI	LIN	UNI	LIN		UNI	LIN	UNI	LIN		UNI	LIN	UNI	LIN
artificial	10	12	1	17	heart	4	11	1	11	promoters	1	10	2	8
australian	30	1	9	1	hepatitis	8	7	1	10	ringnorm	256	0	61	69
balance	0	13	0	13	horse	13	10	3	10	satellite47	48	28	22	39
breast	2	10	0	10	iris	16	5	11	9	spambase	111	36	19	55
bupa	18	23	4	23	ironosphere	18	11	12	15	spect	8	10	0	11
car	9	91	5	91	krvskp	34	64	4	58	splice	0	29	0	25
cmc	23	9	27	21	letter	5143	320	142	546	tae	12	17	6	20
credit	10	9	7	5	magic	548	65	105	77	tictactoe	0	10	0	10
cylinder	17	30	4	23	mammographic	27	9	14	12	titanic	14	6	16	3
dermatology	25	26	14	37	monks	31	20	5	19	transfusion	30	0	0	0
donors	8	14	0	13	nursery	356	32	25	155	twonorm	9	43	0	48
ecoli	50	28	19	35	ocr	36	83	6	87	vehicle	46	83	18	99
flags	27	22	22	27	optdigits	48	134	14	159	vote	16	2	17	2
flare	20	0	0	0	pageblock	94	32	54	55	wave	3	33	0	30
german	5	22	0	18	parkinsons	21	6	5	8	wine	11	13	0	20
glass	38	29	14	43	pendigits	467	149	73	276	winequality	22	32	15	33
haberman	4	8	28	1	pima	11	13	0	18	yeast	141	36	35	65
hayesroth	9	18	2	20	postoperative	0	0	0	0	zoo	37	13	30	17

To validate our novel test, we compare it with three well-known tests namely 5×2 cv t test, 5×2 F test, and k-fold paired t test. We see that the one-sided combined 5×2 t test has higher power (lower type II error); that is, it rejects when there is a large difference between the error values of two algorithms. Similarly, when there is not any significant difference between the error values of algorithm, the combined test accepts; that is, it has a lower probability of reject indicating that it has lower type I error. The combined test has also higher replicability; that is, tests performed by different practitioners with the same pair of algorithms, the same data sets, and the same hypothesis test present similar results.

In the second part of our work, we propose two novel rule induction algorithms R.MULTI and R.OMNI, where R.MULTI uses multivariate conditions and R.OMNI chooses between univariate and multivariate conditions using cv.

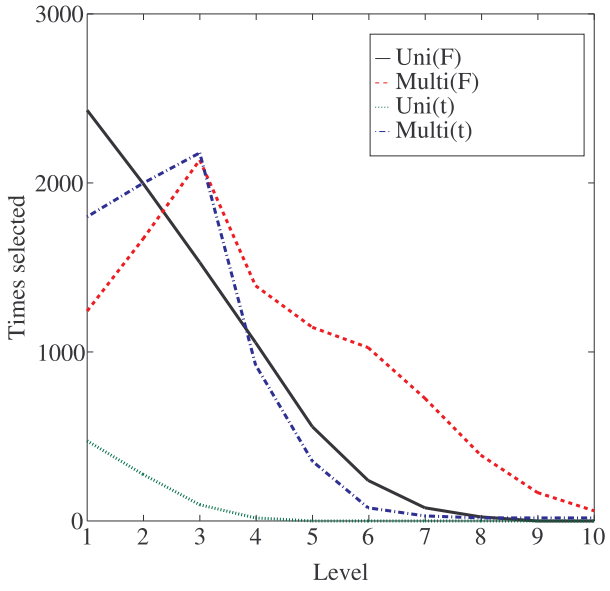

Fig. 9. The number of times univariate and multivariate linear models selected in rule sets produced by R.OMNI(F) and R.OMNI(t) on 10 validation runs with respect to the level of the condition in the rule.

Our simulation results show that R.UNI and R.MULTI algorithms perform well on different data sets and R.OMNI performs as well as the better of them. Multivariate conditions are more complex but rules can be generated by using a smaller number of them.

As a future work, we want to combine an ensemble of multivariate rules. Boosted version of R.UNI, SLIPPER, is given in [61]. Since multivariate conditions have more variance than univariate conditions, we expect bagged or boosted R.MULTI to have smaller error rate.

REFERENCES

[1] J. Fürnkranz, "Separate-and-Conquer Learning," *Artificial Intelligence Rev.*, vol. 13, pp. 3-54, 1999.
[2] J.R. Quinlan, *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.
[3] E. Frank and I.H. Witten, "Generating Accurate Rule Sets without Global Optimization," *Proc. 15th Int'l Conf. Machine Learning*, pp. 144-151, 1998.
[4] P. Clark and R. Boswell, "Rule Induction with CN2: Some Recent Improvements," *Proc. European Working Session Machine Learning*, pp. 151-163, 1991.
[5] W.W. Cohen, "Fast Effective Rule Induction," *Proc. 12th Int'l Conf. Machine Learning*, pp. 115-123, 1995.
[6] L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone, *Classification and Regression Trees*. John Wiley and Sons, 1984.
[7] S.K. Murthy, S. Kasif, and S. Salzberg, "A System for Induction of Oblique Decision Trees," *J. Artificial Intelligence Research*, vol. 2, pp. 1-32, 1994.
[8] C.E. Brodley and P.E. Utgoff, "Multivariate Decision Trees," *Machine Learning*, vol. 19, pp. 45-77, 1995.
[9] O.T. Yildiz and E. Alpaydin, "Linear Discriminant Trees," *Proc. 17th Int'l Conf. Machine Learning*, pp. 1175-1182, 2000.
[10] T.G. Dietterich, "Approximate Statistical Tests for Comparing Supervised Classification Learning Classifiers," *Neural Computation*, vol. 10, pp. 1895-1923, 1998.
[11] O.T. Yildiz and E. Alpaydin, "Omnivariate Decision Trees," *IEEE Trans. Neural Networks*, vol. 12, no. 6, pp. 1539-1546, Nov. 2001.
[12] J. Cendrowska, "PRISM: An Algorithm for Inducing Modular Rules," *Int'l J. Man-Machine Studies*, vol. 27, pp. 349-370, 1987.
[13] G.I. Webb and N. Brkič, "Learning Decision Lists by Prepending Inferred Rules," *Proc. Workshop Machine Learning and Hybrid Systems*, 1993.
[14] C.A. Brunk and M.J. Pazzani, "An Investigation of Noise-Tolerant Relational Concept Learning Algorithms," *Proc. Eighth Int'l Workshop Machine Learning*, pp. 389-393, 1991.

[15] J. Fürnkranz and G. Widmer, "Incremental Reduced Error Pruning," *Proc. 11th Int'l Conf. Machine Learning,* pp. 378-383, 1994.
[16] W.W. Cohen, "Efficient Pruning Methods for Separate-and-Conquer Rule Learning Systems," *Proc. 13th Int'l Joint Conf. Artificial Intelligence,* pp. 988-994, 1993.
[17] S.M. Weiss and N. Indurkhaya, "Reduced Complexity Rule Induction," *Proc. 12th Int'l Joint Conf. Artificial Intelligence,* pp. 678-684, 1991.
[18] D. Fensel and M. Wiese, "Refinement of Rule Sets with JOJO," *Proc. Sixth European Conf. Machine Learning,* pp. 378-383, 1993.
[19] M.V. Joshi, R.C. Agarwal, and V. Kumar, "Mining Needles in a Haystack: Classifying Rare Classes via Two-Phase Rule Induction," *Proc. ACM SIGMOD Conf.,* pp. 91-102, 2001.
[20] L.A. Kurgan, K.J. Cios, and S. Dick, "Highly Scalable and Robust Rule Learner: Performance Evaluation and Comparison," *IEEE Trans. Systems, Man, and Cybernetics-Part B: Cybernetics,* vol. 36, no. 1, pp. 32-53, Feb. 2006.
[21] T. Mitchell, *Machine Learning.* McGraw-Hill, 1997.
[22] R.S. Michalski, "On the Quasi-Minimal Solution of the Covering Problem," *Proc. Fifth Int'l Symp. Information Processing,* pp. 125-128, 1969.
[23] P. Clark and T. Niblett, "The CN2 Induction Algorithm," *Machine Learning,* vol. 3, pp. 261-283, 1989.
[24] F. Bergadano, S. Matwin, R.S. Michalski, and J. Zhang, "Learning Two-Tiered Descriptions of Flexible Concepts: The POSEIDON System," *Machine Learning,* vol. 8, pp. 5-43, 1992.
[25] H. Theron and I. Cloete, "BEXA: A Covering Algorithm for Learning Propositional Concept Descriptions," *Machine Learning,* vol. 24, pp. 5-40, 1996.
[26] G.I. Webb, "Learning Disjunctive Class Descriptions by Least Generalization," Technical Report TR C92/9, Deakin Univ., School of Computing and Math., Geelong, Australia, 1992.
[27] M. Chisholm and P. Tadepalli, "Learning Decision Rules by Randomized Iterative Local Search," *Proc. 19th Int'l Conf. Machine Learning,* pp. 75-82, 2002.
[28] F. Bergadano, A. Giordana, and L. Saitta, "Automated Concept Acquisition in Noisy Environments," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 10, no. 4, pp. 555-578, July 1988.
[29] S.H. Muggleton, "Inverse Entailment and PROGOL," *New Generation Computing,* vol. 13, pp. 245-286, 1995.
[30] P.E. Hart, N.J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Trans. Systems Science and Cybernetics,* vol. SSC-4, no. 2, pp. 100-107, July 1968.
[31] U. Pompe, M. Kocačič, and I. Kohonenko, "SFOIL: Stochastic Approach to Inductive Logic Programming," *Proc. Second Slovenian Conf. Electrical Eng. and Computer Science,* pp. 189-192, 1993.
[32] M. Kovačič, "Stochastic Inductive Logic Programming," PhD dissertation, Dept. of Computer and Information Science, Univ. of Ljubljana, 1994.
[33] G. Venturini, "SIA: A Supervised Inductive Algorithm with Genetic Search for Learning Attributes Based Concepts," *Proc. Sixth European Conf. Machine Learning,* pp. 280-296, 1993.
[34] R. Parpinelli, A.A. Freitas, and H.S. Lopes, "Data Mining with an Ant Colony Optimization Algorithm," *IEEE Trans. Evolutionary Computation,* vol. 6, no. 4, pp. 321-332, Aug. 2002.
[35] D. Martens, M.D. Backer, J. Vanthienen, M. Snoeck, and B. Baesens, "Classification with Ant Colony Optimization," *IEEE Trans. Evolutionary Computation,* vol. 11, no. 5, pp. 651-665, Oct. 2007.
[36] J.L. Olmo, J.R. Romero, and S. Ventura, "Using Ant Programming Guided by Grammar for Building Rule-Based Classifiers," *IEEE Trans. Systems, Man, and CyberneticsPart B: Cybernetics,* vol. 41, no. 6, pp. 1585-1599, Dec. 2011.
[37] Q. Shen and A. Chouchoulas, "A Rough-Fuzzy Approach for Generating Classification Rules Pattern Recognition," *Pattern Recognition,* vol. 35, pp. 2425-2438, 2002.
[38] R. Bhatt and M. Gopal, "FRCT: Fuzzy-Rough Classification Trees," *Pattern Analysis and Applications,* vol. 11, pp. 73-88, 2008.
[39] Y.C. Hu, R.S. Chen, and G.H. Tzeng, "Finding Fuzzy Classification Rules Using Data Mining Techniques," *Pattern Recognition Letters,* vol. 24, pp. 509-519, 2003.
[40] R. Yasdi, "Learning Classification Rules from Database in the Context of Knowledge Acquisition and Representation," *IEEE Trans. Knowledge and Data Eng.,* vol. 3, no. 3, pp. 293-306, Sept. 1991.
[41] Y. Tsai, C. Cheng, and J. Chang, "Entropy-Based Fuzzy Rough Classification Approach for Extracting Classification Rules," *Expert Systems with Applications,* vol. 31, pp. 436-443, 2006.
[42] X. Wang, E. Tsang, S. Zhao, D. Chen, and D. Yeung, "Learning Fuzzy Rules from Fuzzy Samples Based on Rough Set Technique," *Information Sciences,* vol. 177, pp. 4493-4514, 2007.
[43] S. Zhao, E.C. Tsang, D. Chen, and X. Wang, "Building a Rule-Based Classifiera Fuzzy-Rough Set Approach," *IEEE Trans. Knowledge and Data Eng.,* vol. 22, no. 5, pp. 624-638, May 2010.
[44] J.H. Friedman, "A Recursive Partitioning Decision Rule for Non-Parametric Classification," *IEEE Trans. Computers,* vol. C-26, no. 4, pp. 404-408, Apr. 1977.
[45] J. Gama, "Discriminant Trees," *Proc. 16th Int'l Conf. Machine Learning,* pp. 134-142, 1999.
[46] H. Guo and S.B. Gelfand, "Classification Trees with Neural Network Feature Extraction," *IEEE Trans. Neural Networks,* vol. 3, no. 6, pp. 923-933, Nov. 1992.
[47] W.Y. Loh and N. Vanichsetakul, "Tree-Structured Classification via Generalized Discriminant Analysis," *J. Am. Statistical Assoc.,* vol. 83, pp. 715-725, 1988.
[48] W.Y. Loh and Y.S. Shih, "Split Selection Methods for Classification Trees," *Statistica Sinica,* vol. 7, pp. 815-840, 1997.
[49] R. Tibshirani and T. Hastie, "Margin Trees for High-Dimensional Classification," *J. Machine Learning Research,* vol. 8, pp. 637-652, 2007.
[50] K. Bennett and J. Blue, "A Support Vector Machine Approach to Decision Trees," *Proc. Int'l Joint Conf. Neural Networks,* pp. 2396-2401, May 1998.
[51] E. Alpaydin, "Combined 5×2 cv F Test for Comparing Supervised Classification Learning Classifiers," *Neural Computation,* vol. 11, pp. 1975-1982, 1999.
[52] V. Vapnik, *The Nature of Statistical Learning Theory.* Springer Verlag, 1995.
[53] O.T. Yildiz and E. Alpaydin, "Ordering and Finding the Best of $K > 2$ Supervised Learning Algorithms," *IEEE Trans. Pattern Analysis Machine Intelligence,* vol. 28, no. 3, pp. 392-402, Mar. 2006.
[54] E. Alpaydin, *Introduction to Machine Learning,* second ed. The MIT Press, 2010.
[55] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning.* Springer Verlag, 2001.
[56] A. Asuncion and D.J. Newman, "UCI Machine Learning Repository," http://www.ics.uci.edu/~mlearn/MLRepository.html, 2007.
[57] R. Bouckaert and E. Frank, "Evaluating the Replicability of Significance Tests for Comparing Learning Algorithms," *Proc. Pacific-Asia Conf. Advances in Knowledge Discovery and Data Mining,* pp. 3-12, 2004.
[58] C.E. Rasmussen, R.M. Neal, G. Hinton, D. van Camp, M. Revow, Z. Ghahramani, R. Kustra, and R. Tibshirani, "Delve Data for Evaluating Learning in Valid Experiments," http://www.cs.toronto.edu/~delve/, 1995/1996.
[59] C.-C. Chang and C.-J. Lin, *LIBSVM: A Library for Support Vector Machines,* http://www.csie.ntu.edu.tw/~cjlin/libsvm, 2001.
[60] J. Demsar, "Statistical Comparisons of Classifiers over Multiple Data Sets," *J. Machine Learning Research,* vol. 7, pp. 1-30, 2006.
[61] W.W. Cohen and Y. Singer, "A Simple, Fast, and Effective Rule Learner," *Proc. 16th Nat'l Conf. Artificial Intelligence,* pp. 335-342, 1999.

Olcay Taner Yildiz received the BSc, MSc, and PhD degrees in computer science from Boğaziçi University, Istanbul, in 1997, 2000, and 2005, respectively. He did postdoctoral work at the University of Minnesota in 2005. He is a full-time associate professor of computer engineering in the Department of Computer Science and Engineering at Işik University, Turkey. He worked on machine learning, specifically model selection and decision trees. His current research include software engineering, natural language processing, and bioinformatics.