Calculating the VC-Dimension of Decision Trees

Özlem Aslan
Department of Computer Engineering
Boğaziçi University
TR-34342, Istanbul, Turkey
Email: ozlem.aslan1@boun.edu.tr

Olcay Taner Yıldız
Department of Computer Engineering
Işık University
TR-34980, Istanbul Turkey
Email: olcaytaner@isikun.edu.tr

Ethem Alpaydın
Department of Computer Engineering
Boğaziçi University
TR-34342, Istanbul, Turkey
Email: alpaydin@boun.edu.tr

***Abstract*—We propose an exhaustive search algorithm that calculates the VC-dimension of univariate decision trees with binary features. The VC-dimension of the univariate decision tree with binary features depends on (i) the VC-dimension values of the left and right subtrees, (ii) the number of inputs, and (iii) the number of nodes in the tree. From a training set of example trees whose VC-dimensions are calculated by exhaustive search, we fit a general regressor to estimate the VC-dimension of any binary tree. These VC-dimension estimates are then used to get VC-generalization bounds for complexity control using SRM in decision trees, i.e., pruning. Our simulation results shows that SRM-pruning using the estimated VC-dimensions finds trees that are as accurate as those pruned using cross-validation.**

I. Introduction

One of the main goals of machine learning is to extract the "best" model from a labelled training sample where the "best" model is the one which makes the most accurate predictions on an unlabeled test sample. There is a dependency between the complexity and the accuracy of a model: If the model is too complex, we can easily learn the training data but we risk overfitting, that is, the model does not learn a general rule but the particular training data and gives large error on unseen test data. If the model is too simple, even the training data may not be learned well and the error will be large on both training and test data.

The Vapnik-Chervonenkis (VC) theory [1] defines this model selection problem as *predictive learning* and provides a mathematical framework to solve it. With respect to the VC-theory and Structural Risk Minimization (SRM), the optimal model can be found by minimizing the VC generalization bounds estimated from the data sample. In order to calculate the VC generalization bounds of a model, we require (i) the VC-dimension of the model representing its complexity, and (ii) the error of the model on the training data. For example, in a two-class classification problem, the generalization bound of a model is given as [2]

$$E_g = E_t + \frac{\epsilon}{2}\left(1 + \sqrt{1 + \frac{4E_t}{\epsilon}}\right) \quad (1)$$

where E_t is the training error and

$$\epsilon = a_1 \frac{V[\log(a_2 N/h) + 1] - \log(\nu)}{N} \quad (2)$$

Here, V represents the VC dimension of the model, ν represents the confidence level (it is recommended to use $\nu = \frac{1}{\sqrt{N}}$ for large sample sizes), and a_1 and a_2 are empirically fitted constants.

Given a dataset, the N data points can be labeled in 2^N different ways as positive and negative. For any of these labelings, if we can separate the positive data points and negative data points by some hypothesis h from the hypothesis class H, then we say H shatters N points. The maximum number of points that can be shattered by H is called VC-dimension of H and measures the capacity of the hypothesis class of H.

In this work, we use decision trees as our hypothesis class. Decision trees are tree-based structures where (i) each internal node implements a decision function, $f_m(\boldsymbol{x})$, (ii) each branch of an internal node corresponds to one outcome of the decision, and (iii) each leaf corresponds to a class. If the decision function $f_m(\boldsymbol{x})$ uses only one dimension of the input vector $\boldsymbol{x}$, the decision tree is called a *univariate* decision tree [3]. ID3 is one of the most known univariate decision tree algorithm with discrete features [4].

Determining the optimal complexity of a decision tree is important. With complex decision trees, we do not learn a general rule but memorize the particular training data which gives large error on unseen test data. With too simple trees, even the training data may not be learned well and the error will be large on both training and test data.

Although the VC dimension of some classifiers are known, it is not possible to obtain the VC-dimension of all [1], and the only possibility is to estimate it experimentally [5], [6]. The proposed approach depends on the fact that for two-class problems, the deviation of the expected training error of the classifier from 0.5 (which is the worst a two-class classifier can do) is correlated with the VC-dimension of that classifier and the data size. Vapnik et al. (1994) derive a theoretical formula for this correlation. A set of experiments on artificial sets are done and based on the frequency of the errors on these sets, a best fit for the theoretical formula is calculated. Shao et al. (2000) use optimized experimental design to improve the VC-estimates. The algorithm starts with the uniform experiment design used in [5] and by making pairwise exchanges between design points, the optimized design is obtained. The mean square error (MSE) is used as the criterion to identify good and bad design points.

After estimating the VC-dimension, one can plug it into the VC generalization bound formula and do model selection

978-1-4244-5023-7/09/$25.00 ©2009 IEEE

ISCIS September 14-16, 2009
METU
Northern Cyprus Campus

using SRM. Shao et al. (2000) first estimate the VC-dimension for penalized linear estimators experimentally which they use for model selection in regression tasks. According to the experiment results, the VC-dimension obtained by the optimized design achieves better model selection performance than two classical model selection criteria namely Akaike's final prediction error and generalized cross-validation [7].

As similar to above discussion, there is no explicit formula to calculate the VC-dimension of a decision tree. In this work, we focus on the easiest case of univariate trees with binary features and we believe that the VC-dimension of a univariate decision tree with binary features depends on the number of binary features, the number of decision nodes as well as the tree structure. Our approach is the following: For a number of simple trees, we use an exhaustive search algorithm to calculate the exact VC-dimensions. Then, using these as a training set, we fit a regressor so that we can estimate the VC-dimension of any tree. We use these VC-dimension estimates in pruning to validate that they are indeed good estimates. Note that, we are discussing the VC dimension of hypotheses classes defined as families of decision trees that share the tree structure and differ only in the variables being tested along the internal nodes.

This paper is organized as follows: In Section II, the exhaustive search algorithm is described. We give experimental results in Section III, show how we can use the VC-dimension estimates for complexity control in the decision trees in Section IV and conclude in Section V.

II. Exhaustive Search Algorithm

Let say we want to find the VC-dimension of a decision tree hypothesis class H. Given a dataset with d dimensions where each dimension is binary, the number of possible distinct data points is 2^d. The idea is as follows: Considering a subset of size N of these 2^d possible instances, if for each possible class labellings of these N points, there is an instantiation h of our assumed decision tree hypothesis class H that classifies it correctly, we say that the decision tree has VC dimension of at least N. The VC-dimension of H is the largest N.

The exhaustive search algorithm is illustrated in Figure 1 for a dataset where $d = 2$ input features and $N = 3$ data points. We want to determine the VC-dimension of a decision tree structure with 2 nodes representing the hypothesis class H. Note that each feature can only be used once on a path from the root node to a leaf node. Because of this, we have only two (nonisomorphic) hypotheses h_1 and h_2 in hypothesis class H. For the first hypothesis h_1, the first feature is used for the split in the first node and the second feature is used for the split in the second node, for the second hypothesis h_2; the second feature is used for the split in the first node and the first feature is used for the split in the second node.

The number of input features is 2, therefore the number of distinct data points is $2^2 = 4$. Since we explore $N = 3$ data points in example, all the combinations of 3 points from 4 points are shown. For each data combination with $N = 3$,

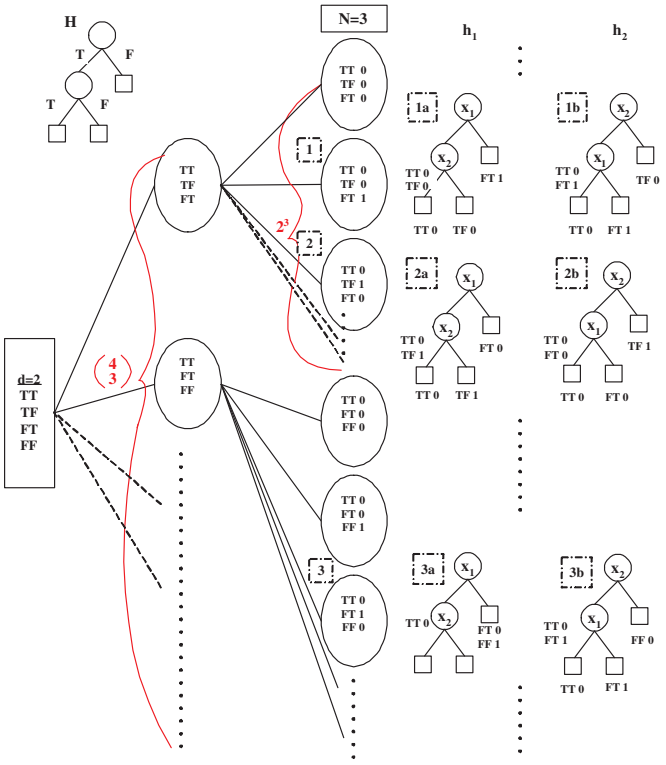

Fig. 1. Example showing finding VC-Dimension of univariate decision tree for a dataset with $d = 2$ and $N = 3$.

```
VCDimension ExhaustiveSearch(H, d, data)
1   N = 1
2   while TRUE
3       dataComb = getDataCombination(data, N)
4       successful = FALSE
5       while dataComb != NULL
6           classifiedAllCombinations = TRUE
7           classComb = getClassCombination(N)
8           while classComb != NULL
9               if not treeClassify(H, dataComb, classComb)
10                  classifiedAllCombinations = FALSE
11                  break
12              classComb = getNextClassCombination(N)
13          if classifiedAllCombinations
14              successful = TRUE
15              break
16          else
17              dataComb = getNextDataCombination(data, N)
18      if successful
19          N = N + 1
20      else
21          break
22  return N − 1
```

Fig. 2. The pseudocode of the exhaustive search algorithm for finding VC-dimension of univariate decision tree: H: Decision tree hypothesis class, d: Number of inputs in the dataset, $data$: Universal set for d dimensional input

we check for each possible class labeling, if one of h_1 or h_2 classifies it correctly.

- For the first class labeling corresponding to the first data combination (shown as case 1), both hypotheses h_1 and h_2 classify the data correctly (Trees 1a and 1b). Note that only one of the hypotheses is enough.
- For the second class labeling corresponding to the first data combination (shown as case 2), again both hypotheses h_1 and h_2 classify the data correctly (Trees 2a and 2b). Note that second node is not required in Tree 2b in order to classify the data.
- For the third class labeling corresponding to the second data combination (shown as case 3), the first hypothesis h_1 can not classify the data correctly (Tree 3a), whereas the second hypothesis h_2 classifies the data correctly (Tree 3b).

The pseudocode for finding VC-Dimension of univariate decision tree with binary features is given in Figure 2. Given a dataset with d dimensions, instead of generating all possible combinations beforehand, we generate all possible data combinations having N data points iteratively (Line 3 and Line 17). For each data combination, we generate all possible class labellings iteratively (Line 7 and Line 12). For each possible class labeling of a data combination, we check if there is an hypothesis h from the decision tree hypothesis class H that classifies the data correctly (Line 9). Recall that hypothesis check is done by changing the features used in the nodes of the decision tree. If there is not such an hypothesis (Line 10, 11), we break the class labeling search and continue the search with the next data combination (Line 17). If there is such an hypothesis, we iterate to next class labeling(Line 12). If for all class labellings of a data combination we can find a decision tree hypothesis h (Lines 13, 14), we increment N (Lines 18, 19) and continue the search. If all subsets N of 2^d are iterated and no subset is classified for all class labellings, then the search is over and VC dimension is taken as $N-1$.

The computation is incremental such that if a data combination is classified for a data size, then data size is incremented by one and the algorithm continues to search. For a data size N and dimension d, there are $\binom{2^d}{N}$ possible data combinations. For each data combination with size N, there are 2^N possible class labellings. In the worst case, we need to check each element of the decision tree hypothesis class H, therefore, the computational complexity of the algorithm is

$$\sum_{N=1}^{V} \binom{2^d}{N} 2^N |H| \tag{3}$$

This is the reason why we can run the exhaustive search algorithm only on few H and on cases with small d and $|H|$. For example, the full tree with depth 4 and for 4 input features requires 2 days to complete on a quad-core computer, whereas the full tree with depth 5 and for 5 input features will require approximately 10^{13} days. We then fit a regressor to this data and check if we can extrapolate to other trees.

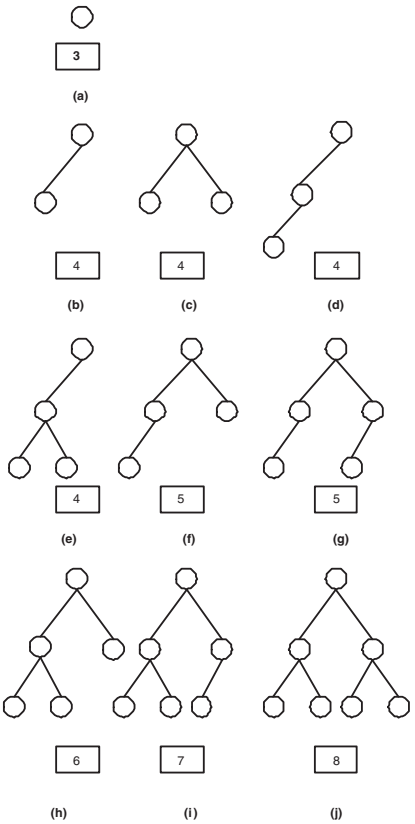

Fig. 3. VC-dimension of univariate decision trees for datasets with 3 input features. Only the internal nodes are shown.

III. Experiments

A. Exhaustive search results

Just as the computational complexity grows exponentially, as the depth of the tree increases, the number of possible trees grows exponentially, too. Because of this reason, we thoroughly searched decision trees with depth up to four. We then try to capture a recursive formula for the VC-dimension in terms of the number of nodes in the decision tree, the depth of the decision tree and the VC-dimensions of left and right subtrees.

In order to decrease the number of trees to a specific depth, we use the fact that two isomorphic trees have the same VC-dimension. Therefore, we start by creating trees with depth 1 (which is one tree), then create other decision trees by adding nodes to the last level (leaves) of the previous trees (with depth minus one), while making sure that we are not creating isomorphic trees.

Figures 3 and 4 show the VC-dimensions of decision trees for datasets with 3 and 4 input features. It can be seen that the VC-dimension increases as the number of nodes in the decision tree increases, but there are exceptions where the VC-dimension remains constant though the number of nodes increases. For example, in Figure 3, the VC-dimension increases going from Figure 3a to Figure 3b and from Figure 3h to Figure 3i as the number of nodes increases. On the other

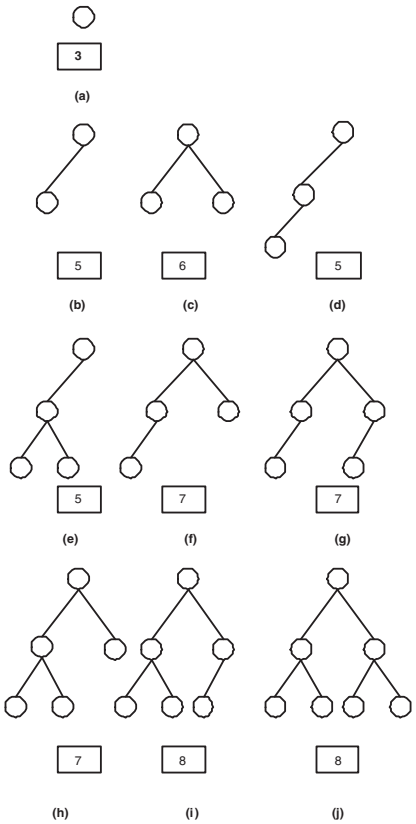

Fig. 4. VC-dimension of univariate decision trees for datasets with 4 input features. Only the internal nodes are shown.

hand, the VC-dimension remains constant going from Figure 3b to Figure 3c and from Figure 3d to Figure 3e although the number of nodes increases.

There is an increase in the VC-dimension as the depth of the decision tree increases. However this dependency is not trivial. For example in Figure 4, although going from depth 1 to depth 2 results in an increase in the VC-dimension, some of the decision trees with depth 2 (for example, Figure 4b) have the same VC-dimension as the decision trees with depth 3 (Figures 4d and 4e). We also see that, a decision tree with depth 2 (Figure 4c) has larger VC-dimension than a tree with depth 3 (Figures 4d or 4e).

When we look at the subtrees to capture a relation between the VC-dimension of the decision tree and the VC-dimension of its subtrees, we see that it is not straightforward. However, it is obvious that the VC-dimension of a decision tree is greater than or equal to the VC-dimension of its subtrees. Note that in comparing the VC-dimension of the parent tree with its subtrees, the subtree can use one less input feature than its parent tree. For example the VC-dimension of decision tree in Figure 4d is greater than or equal to its left subtree, which is the decision tree in Figure 3b.

B. Estimating the VC-Dimension by regression

As the number of input features and/or the number of nodes increase, the search space grows exponentially. This dramatic

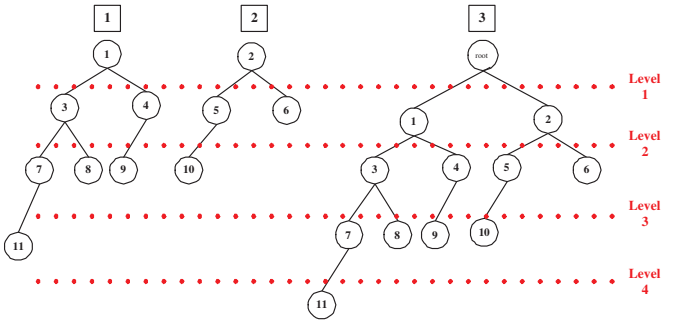

Fig. 5. Merging two decision trees to construct new decision tree. The numbers in the nodes represent the order those nodes are added into the new decision tree.

increase especially starts to occur when $d > 4$ and/or the tree depth is larger than 4. In order to calculate the VC-dimensions of larger decision trees, we need to decrease the search space if possible without sacrificing from accuracy. There are three dimensions of the search space:

- All possible data combinations with N data points selected from 2^d distinct instances.
- All class labelings of N data points.
- All hypotheses in the decision tree hypothesis class H.

Since one single data combination is enough for proving that the VC-dimension of a decision tree hypothesis class H is at least N, we decided to reduce the search space by considering only a subset of all possible data combinations. In our experiments, we take 1000 data combinations for each decision tree structure instead of all $\binom{2^d}{N}$.

As we have explained previously, the main purpose of this paper is to relate the VC-dimension of a decision tree to the VC-dimension values of its left and right subtrees. Therefore, we not only need to find the VC-dimension of a decision tree, but we also need to have the VC-dimensions of its left and right subtrees. In order to achieve this, we go in the reverse direction, and merge two trees whose VC-dimensions are known and then we find the VC-dimension of the merged tree. Figure 5 shows this merging process, which is done level by level. Thus, starting at the first level, nodes 1 and 2 are selected and added to the new tree. Continuing with the next level which is level two, nodes 3, 4, 5 and 6 are added to the new tree. In the third level, nodes 7, 8, 9 and 10 are added to the new decision tree. Finally, in the fourth level, node 11 is added to the new decision tree.

After finding the VC-dimension of a set of decision trees, we try to estimate a general formula to relate the VC-dimension of a decision tree to (i) the VC-dimension values of its left and right subtrees (V_l, V_r) (ii) the number of decision nodes in the tree (M), and (iii) the number of input features in the corresponding dataset (d). We used linear regression that takes these as inputs and trained it on a set of 154 instances. The relationship of V_l and V_r to the VC-dimension of the decision tree seems to be linear. For the relationship between d and M to the VC-dimension of the decision tree we tried

two (d and $\log d$) and three (M, $\sqrt{M}$ and $\log M$) functions respectively. Using a least squares fit, the best regressor turns out to be

$$V = 0.7152 + 0.6775 V_l + 0.6775 V_r - 0.6600 \log d + 1.2135 \log M \quad (4)$$

with an R^2 value of 0.9487 which indicates that this is a good fit.

IV. Complexity Control using VC-Dimension Estimates

In this section to show that our estimated VC-dimension values are useful, we use them for complexity control in decision trees. Controlling complexity in decision trees could be done in two ways. We can control the complexities of the decision nodes by selecting the appropriate model for a node (an omnivariate decision tree can have univariate, linear multivariate or nonlinear multivariate nodes [8]), or we can control the overall complexity of the decision tree via pruning. Since this paper covers only univariate trees with binary features, we take the second approach and use the VC-dimension estimates found in the previous section for pruning.

In postpruning (CVprune), we compare the performance of the subtree with a leaf replacing the subtree on a separate validation set. If there is overfitting, we expect the more complex subtree to learn the noise and perform worse than the simple leaf. Here the validation set determines the number of nodes after pruning and it is not possible to determine the number of nodes beforehand. In our experiments, for CVprune, 20 percent of the data is put aside as a pruning set.

When we prune a node using SRM, we first find the VC generalization error using Equation 1 where V is the VC-dimension and E_t is the training error of the subtree. Then, we find the training error of the node as if it is a leaf node. Since the VC-dimension of a leaf node is 1, we can find the generalization error of the tree as if it is pruned. If the generalization error of the leaf node is smaller than the generalization error of the subtree, we prune the subtree, otherwise we keep it. We call this type of pruning SRMprune and compare it with CVprune. For the sake of generality, we also include the results of trees before any pruning is applied (NOprune).

We used three different functions:

$$\begin{aligned} F_1 &= x_0 x_1 + x_0 x_2 + x_1 x_2 \\ F_2 &= x_0 x_1 + x_0 x_2 + x_0 x_3 + x_1 x_2 + x_1 x_3 + x_2 x_3 \\ F_3 &= x_0 x_1' + x_0' x_1 \end{aligned}$$

Our comparison criteria are generalization error (on the validation folds of 5×2 cross-validation), complexity (as measured by the total number of decision nodes in the tree). Three points must be considered when the synthetic datasets are generated:

- The number of input features. We used d = 8 and d = 12 in our experiments, where the extra inputs act as redundant input variables that should not appear in the tree.
- Noise level. We used five different noise levels such as ρ = 0.0, 0.01, 0.05, 0.1, and 0.2 to flip the output.
- Training sample size drawn from the possible set of half of 2^d instances. We used four different sample size percentage such as S = 10, 25, 50, 100.

Table I shows the average and standard deviations of error rates and the number of nodes of decision trees generated using NOprune, CVprune and SRMprune from three functions for d = 12, ρ = 0.0, and S = 100. We see that, the first two functions are discovered without any error by all three techniques. On the other hand, to learn F_3 (xor), which requires nonorthogonal splits, one must generate larger trees. In this case, CV prunes trees more aggresively (tree size of 83 compared to 175) by sacrificing from accuracy (8.5 compared to 3.9 percent).

Table II shows the average and standard deviations of error rates and the number of nodes of decision trees generated using NOprune, CVprune and SRMprune for different number of inputs and for ρ = 0.2, S = 100, and $F = F_2$. We see that for small number of inputs, pruning can not reduce the error rate much, but for large number of inputs the tree without pruning overfits and, (with redundant inputs) pruning especially SRMprune can reduce the error rate nearly to its minimum (Since the noise level is 20 percent the minimum error rate will be near this number). On the other hand, both CV and SRM pruning works well, they reduce the number of nodes significantly. For d = 8, the tree size is reduced from 60 to 10, for d = 12, it is reduced from 870 to 9.

Table III shows the average and standard deviations of error rates and the number of nodes of decision trees generated using NOprune, CVprune and SRMprune for different noise levels and for d = 12, S = 50, and $F = F_1$. As expected, when the noise level increases, pruning helps. For example, although the noise level is as low as 0.01, NOprune generates nearly 10 times larger trees than CVprune and SRMprune. Another important point is that as the noise level is increasing, the size of the pruned trees remains nearly the same. If we compare SRMprune with CVprune, we see that SRMprune generates more accurate trees without sacrificing from tree size.

Table IV shows the average and standard deviations of error rates and the number of nodes of decision trees generated using NOprune, CVprune and SRMprune for different sample sizes and for d = 8, ρ = 0.05, and $F = F_3$. For this case, as the sample size decreases, overfit occurs, and both CV and SRM pruning do not help much, their error rates also increases as the sample size decreases.

V. Conclusions and Future Work

We estimate the VC-dimension of univariate decision trees with binary features via an exhaustive search algorithm. Due to the exponential time complexity of the algorithm, it is applied up to five dimensional inputs and on decision trees with depth up to five. These results are then used to find a linear recursive fit that relates the VC-dimension of a tree to the VC-dimension of its left and right subtrees, the number of nodes in the tree and the number of input features in the dataset. These estimated values are used in pruning using SRM and when compared with cross-validation pruning, we see that SRM pruning using our estimated VC-dimension values work

TABLE I

THE AVERAGE AND STANDARD DEVIATIONS OF ERROR RATES AND THE NUMBER OF NODES OF DECISION TREES GENERATED USING NOPRUNE, CVPRUNE AND SRMPRUNE ON THE THREE FUNCTIONS FOR $d = 12$, $\rho = 0.0$, AND $S = 100$.

Function	Error Rate			Number of Nodes		
	NOprune	CVprune	SRMprune	NOprune	CVprune	SRMprune
F_1	0.0± 0.0	0.0± 0.0	0.0± 0.0	5.0± 0.0	5.0± 0.0	5.0± 0.0
F_2	0.0± 0.0	0.0± 0.0	0.0± 0.0	9.0± 0.0	9.0± 0.0	9.0± 0.0
F_3	3.9± 2.8	8.5± 7.0	3.9± 2.8	177.6±115.8	83.3±59.5	174.9±115.6

TABLE II

THE AVERAGE AND STANDARD DEVIATIONS OF ERROR RATES AND THE NUMBER OF NODES OF DECISION TREES GENERATED USING NOPRUNE, CVPRUNE AND SRMPRUNE FOR DIFFERENT NUMBER OF INPUTS AND FOR $\rho = 0.2$, $S = 100$, AND $F = F_2$.

d	Error Rate			Number of Nodes		
	NOprune	CVprune	SRMprune	NOprune	CVprune	SRMprune
8	38.1± 4.1	37.8± 5.3	35.3± 2.7	57.5± 6.3	3.8± 3.3	12.8± 7.9
12	35.5± 1.2	28.2± 3.0	21.0± 0.6	869.2±15.1	4.2± 1.5	9.0± 0.0

TABLE III

THE AVERAGE AND STANDARD DEVIATIONS OF ERROR RATES AND THE NUMBER OF NODES OF DECISION TREES GENERATED USING NOPRUNE, CVPRUNE AND SRMPRUNE FOR DIFFERENT NOISE LEVELS AND FOR $d = 12$, $S = 50$, AND $F = F_1$.

ρ	Error Rate			Number of Nodes		
	NOprune	CVprune	SRMprune	NOprune	CVprune	SRMprune
0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0	5.0± 0.0	5.0± 0.0	5.0± 0.0
0.01	3.6± 0.5	1.5± 0.3	1.5± 0.3	62.5±11.0	5.0± 0.0	5.0± 0.0
0.05	12.2± 0.8	5.0± 0.5	5.0± 0.5	167.0±10.6	5.0± 0.0	5.0± 0.0
0.1	21.7± 0.9	12.8± 4.7	10.6± 0.2	283.2±13.0	5.2± 2.2	5.0± 0.0
0.2	35.7± 1.4	29.3± 5.4	20.6± 0.9	419.5±13.7	2.6± 1.6	5.0± 0.0

TABLE IV

THE AVERAGE AND STANDARD DEVIATIONS OF ERROR RATES AND THE NUMBER OF NODES OF DECISION TREES GENERATED USING NOPRUNE, CVPRUNE AND SRMPRUNE FOR DIFFERENT SAMPLE SIZES AND FOR $d = 8$, $\rho = 0.05$, AND $F = F_3$.

S	Error Rate			Number of Nodes		
	NOprune	CVprune	SRMprune	NOprune	CVprune	SRMprune
100	19.0± 5.9	25.3±14.9	15.8± 8.6	36.3±10.6	8.4± 5.1	23.8±18.9
50	23.7±14.7	28.9±17.2	23.4±14.6	19.4± 9.1	4.4± 3.3	18.1± 9.7
25	27.0±11.7	37.4±15.7	27.0±11.7	9.4± 4.1	1.3± 1.7	9.4± 4.1
10	41.7±17.1	45.0±17.2	41.7±17.1	5.3± 0.9	0.9± 1.4	5.3± 0.9

well and find trees that are as accurate as CV pruning without the overhead of cross-validation or needing to leave out some data for traininig set.

In this paper, we only consider the VC-dimension of univariate decision trees with binary features. The approach can easily by extended to univariate decision trees with discrete and/or continuous features as

- Discrete features having $L > 2$ different values can be converted to L binary features using 1-of-L encoding.
- Continuous features can be converted to discrete features whereafter the same procedure for discrete features can be applied.

In this work, the discussion in only about the two-class classification tasks. Extension to K-class case does need the estimation of VC-dimension for multiclass problems [9]. Ridella and Zunino (2001) extended the theoretical framework of Vapnik et al. (1994) to yield bounds to the generalization error for multiclass problems.

Contrary to the approach taken here, one can estimate the VC-dimension of the univariate decision tree empirically using uniform and/or optimized experiment design. The VC-dimension estimates then can be used for complexity control in the decision tree [10].

REFERENCES

[1] V. Vapnik, *The Nature of Statistical Learning Theory*. New York: Springer Verlag, 1995.

[2] V. Cherkassky and F. Mulier, *Learning From Data*. John Wiley and Sons, 1998.

[3] J. R. Quinlan, *C4.5: Programs for Machine Learning*. San Meteo, CA: Morgan Kaufmann, 1993.

[4] ——, "Induction of decision trees," *Machine Learning*, vol. 1, pp. 81–106, 1986.

[5] V. Vapnik, E. Levin, and Y. L. Cun, "Measuring the vc-dimension of a learning machine," *Neural Computation*, vol. 6, pp. 851–876, 1994.

[6] X. Shao, V. Cherkassky, and W. Li, "Measuring the vc-dimension using optimized experimental design," *Neural Computation*, vol. 12, pp. 1969–1986, 2000.

[7] V. Cherkassky and Y. Ma, "Comparison of model selection for regression," *Neural Computation*, vol. 15, pp. 1691–1714, 2003.

[8] O. T. Yıldız and E. Alpaydın, "Omnivariate decision trees," *IEEE Transactions on Neural Networks*, vol. 12, no. 6, pp. 1539–1546, 2001.

[9] S. Ridella and R. Zunino, "Empirical measure of multiclass generalization performance: The k-winner machine case," *IEEE Transactions on Neural Networks*, vol. 12, pp. 1525–1529, 2001.

[10] Z. Yang, W. Zhu, and L. Ji, "Slit: Designing complexity penalty for classification and regression trees using the srm principle," in *Proceedings of the Lecture Notes in Computer Science*, vol. 3971, 2006, pp. 895–902.