Oblivious Decision Trees, Graphs, and Top-Down Pruning

Ron Kohavi and Chia-Hsin Li
Computer Science Department
Stanford University
Stanford, C\ 94305
{ronnyk,jamie}iDCS Stanford EDU

Abstract

We describe a supervised learning algorithm, EODG that uses mutual information to build an oblivious decision tree The tree is then converted to an Oblivious read-Onre Decision Graph (OODG) b\ merging nodes at the same level of the tree For domains that art appropriate for both decision trees and OODGs, per formance is approximately the same aS THAT of C45), but the number of nodes in the OODG is much smalle r The merging phase that converts the oblivious decision tree to an OODG provides a new way of dealing with the replication problem and a new pruning mechanism that works lop down starting from tin root The pruning mechanism is well suited for finding symmetries and aids in recovering from splits on irrelevant features that mav happen during the tree consLrm tion

1 Introduction

Dmsion trees provide a hypothesis space for supervised mat hine learning algorithms that is well suited for many data^ets encountered in the real world (Breiman, Friedman Olshen & Stone 1984 Quinlan 1993) The tree structure, used to represent concepts suffers from some well-known problems, most notably the replication problem disjunctive concepts such as $(A \wedge B) \vee (C \wedge D)$ are inefficiently represented (Pagallo & Haussler 1990) A. related problem, the fragmentation problem sometimes called over-branching or over-partitioning (Fayyad 1991), is specific to the top-down recursive partitioning that is used to build the trees In the common top-down decision tree construction algorithms including CART (Breiman et al 1984), ID3, and C4 5 (Quinlan 1993), the training set is partitioned according to a test at each node (usually a test on a single feature) After a few splits, the number of instances at the node diminishes to a point where distinguishing between the actual pat-'?rn (signal) and random events (noise) becomes difficult The fragmentation problem is especially apparent when there are mulli-valued features that are used as tests or specific intervals that need to be identified in continuous features, the multi-way splits quickly partition the dataset into small sets of instances While there have been attempts by Quinlan (1993) and Fayyad (1994) to atlack the problem directly there is no agreed-upon method and (4 5 s default parameters do not even default to such splits

The smallest decision trees for most symmetric functions, such as majority, *'in* of *n* " and parity, require exponentially-sized trees While parity is unlikely, to occur in practice m of *n* ' is more common (Spademan 1988) Induction algorithms that search for small trees tend to generalize poorly on these functions

Kohavi (1994a) introduced Oblivious read-Once Decision Graphs (OODGs) as an alternative representation structure for supervised classification learning OODGs retain most of the advantages of decision trees, while overcoming the above problems OODGs are similar to Ordered Binary Decision Diagrams (Bryant 1986) which have been used in the eugineenng community to represent state-graph models of systems allowing verification of finite-state systems with up Lo 10120 states Much of the work on OBDDs carries over to OODGa We refer the reader to Kohavi (1994a) for a discussion of related work

The HOODG algorithm for bottom-up induction of OODGs was shown to be effective for nominal features (Kohavi 19946) but the algorithm could not cope with continuous features and lacked a global measure of improvement that could help cope with irrele\ant features

In this paper, we introduce EODG (Entropv-based Oblivious Decision Graphs) a top-down induction algorithm for inducing oblivious read-once decision graphs. The EODG algorithm uses the mutual information of a single split across the whole level to determine the appropriate tests for the interior nodes of the tree AH instances are involved at every choice point in the tree construction and, therefore the algorithm does not suffer from fragmentation of the data as much as recursive partitioning algorithms do After an oblivious decision tree is built, it is pruned and converted into an oblivious

decision graph by merging nodes in a top-down sweep

The paper is organized as follows In Section 2 we briefly review the OODG structure and its properties Section 3 describes the EODG algorithm Section *4* describes the experimental results Section 5 describes future work, and Section 6 concludes with a summary

2 Oblivious Read-Once Decision Graphs

In this section, we formally define decision graphs and then specialize them to oblivious read-once decision graphs

Given n discrete fealures (attributes) X_1 X_2 , X_n, with domains D_1, D_n respectively, the instance space \ is the cross-product of the domains i e , $D_1 \times \times D_n$ A k-classification function is a function / mapping each instance in the instance space Lo one of *k* classes, i e , $f \quad X \mapsto \{0, \quad , k-1\}$

A decision graph for a *k*-classification function over features X_1 X_2 X_n with domains D_1, D_2 D_n is a directed acyclic graph (D4G) with the following properties

1 There are exactly *k* nodes called category nodes, that are labelled 0 1, , *k*— 1, and have outdegree zero

2 Non-category nodes are called branching nodes Each branching node is labelled by some feature X_i and has \D,\ outgoing edges, each labelled by a distinct value from D_i

3 There it. one distinguished node—the root—that *is* the only node with indegret zero

The class assigned by a decision graph to a given feature assignment (an instance), is determined by tracing the unique path from the root to a category node, branching according to the labels on the edges

In a read-once decision graph, each test occurs at most once along any path from the root to a leaf In a levelled decision graph the nodes are partitioned into a sequence of pairwise disjoint sets, the levels such that outgoing edges from each level terminate at the next level An oblivious decision graph is a levelled graph such that all nodes at a given level are labelled by the same feature

Previous work on OODGs (Kohavi 1994a, Kohavi 19946) defined read-once as allowing a *feature* to be tested only once For continuous features, this restriction is is inappropriate, thus we allow continuous features to be tested at different levels as long as the thresholds are different

An OODG is an oblivious read-once decision graph A constant node is a node such that all edges emanating from it terminate at the same node of the subsequent level Figure 2 shows an OODG derived by the EODG algorithm for the mushroom problem (a post-processing

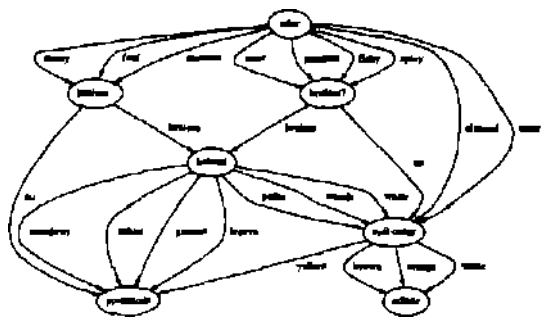

Figure 1 The OODG (constant nodes removed) induced by EODG for the mushroom concept

algorithm removes constant nodes to make the graphs more readable)

OODGs have many interesting properties These include an OODG for nominal features is canonical for any total function if an ordering on the features is given any symmetric Boolean function *(c g* parity, majority and m of n) can be represented by an OODG of size $O(n^2)$ and the width of levels in OODGs is bounded by mm $\left\{ 2^{\prime} \; k^{2^{\lfloor n/2 \rfloor}} \right\}$ for Boolean inputs We refer the reader to Kohavi (1994a) for a more detailed description of these properties

3 The EODG algorithm

The EODG algorithm has three phases growing an oblivious decision tree (ODT) pruning complete levels from the bottom and merging nodes from the top down We now describe the phases in detail

3 1 Top-down Construction Using Mutual Information

The top-down construction of the ODT is performed using mutual information in a manner similar to *c4* 5 (Quinlan 1991) The main difference is that instead of doing recursive partitioning, i e finding a split and constructing the subtree recursively, an iterative process is used

The general mechanism for growing ODTs is simple decide on a test with *k* outcomes that will be done at all nodes in the current level of the tree Partition the instances at each node *p* according to the test and create a new level containing *k* nodes for each node p in the current level The *k* nodes are connected from the parent node p to the *k* children by edges labelled with the *k* possible outcomes Figure 2 shows this algorithm

The find_split routine determines which split to conduct at all the nodes at the current level and decides when to terminate While the test can be of any type, our implementation uses multi-waysplits on nominal features (the test outcome is the feature value) and threshold splits on continuous feature To determine which

Input a set *T* of libelled instances
Output an oblivious decision tree for classifying instances
Each set created by this algorithm has a corresponding node in the graph

```
1  S = {T} // S is a set of sets of instances
   // t is a test with k outcomes to conduct at all nodes
   // in this level find_split returns empty split
   //to terminate
2  while (t = find_split(S)) {
3      S' = {}
4      foreach s ∈ S {
5          Split s into k sets according to the test t and
           add the sets to S'
6          For each of the k sets, create a node and
           connect to the node representing s Label
           edge by the test outcome
7      }  // end foreach
8      S = S'
9  }  // end while
```

1 lgure *2* The algorithm for growing the oblivious decision trees

feature to split on, and what threshold to pick for con Urinous features, we use mutual information Our notation follows that of Cover & Thomas (1991) The conditional entropy $H(Y \mid X_1 \quad X_\ell)$ of the label V given features X_1, X_2 , X/ represents the amount of information about *Y* after we have seen the values of $X_1 \; X_2 \quad X_\ell$ and is defined as

$$H(Y \mid X_1 \quad , X_\ell) = - \sum_{x_1 \in X_1} \quad \sum_{x_\ell \in X_\ell, y \in Y} \Delta$$
$$\Delta = p(y \; x_1, \quad x_\ell) \log(p(y|x_1, \quad x_\ell))$$

diven a new feature A , wt define the mutual information as the difference in the conditional entropies

$$I(Y, X \mid X_1, \quad X_\ell) = H(Y \mid X_1, \quad X_\ell) - H(Y \mid X_1, \quad X_\ell \; X)$$

If *t* is zero, the mutual information is defined as

$$I(Y, X) = - \sum_{y \in Y} p(y) \log(p(y)) - H(Y \mid X)$$

Because the mutual information is biased in favor of testa with many outcomes, we define the adjusted mutual information as the mutual information divided by log *k,* where *k* is the number of outcomes for *X* Given *S,* a set of sets of instances (each set of instances is a node in the graph), find split checks all the possible features and determines the one that has the highest adjusted mutual information For continuous features, we try all possible thresholds between two adjacent values of a feature Computing the best threshold split after the values for a given feature have been sorted can be done in linear time in the number of instances by shifting instances from the left child node to the right child node as the threshold is increased The technique is exactly the same as that used in Breiman et al (1984) and Quinlan (1993) exrept that we determine one split for all the nodes at a level and shift instances from the left child to the right child for all nodes at the current level

When building a decision tree, it is clear that one should not split on a nominal feature twice, since no information can be gained However, a surprising phenomenon that occurs in OODGs, which has no equiva lent in decision tree indurtion, is that it may be useful to split pure nodes i t , nodes that contain instances of only one class, because the merge process may benefit for example, if we are left with one instance per node then all nodes are trivially pure If we split on a relevant feature some instances will go left and some instances will go right these nodes will then be merged and the correct target concept might be identified

3 2 Merging Nodes

The main advantage of building ODTs over regular decision trees is that nodes at the same level can be merged because they contain the same test We begin by describing how to convert an ODT into an OODG We then describe how this merging can be used to conduct top-down pruning, a technique that we believe is more robust to noise than bottom-up pruning and also aids in overcoming splits on irrelevant features by merging the children to a single node as in Figure 2

Two subtrees in a decision tree are isomorphic if they art both leaves labelled with the same class, or if the root nodes contain the same test and the corresponding children, reached by following the edges with the same labels on both subtrees, are the roots of isomorphic subtrees Two isomorphic subtrees may be merged without changing the classification behavior of the decision structure While subtree isomorphism may be performed for decision trees too it is unlikely that subtrees will be isomorphic without the leveling restriction

The merging process is a post-processing step that is executed after the ODT has been grown For each level we check all pairs of nodes for isomorphism and merge those that are isomorphic The complexity of the merge process is dominant in the induction of OODGs Atea(h level with fe nodes, $0\{k^2\}$ isomorphic tests have to be conducted However, because of the kite theorem (Kohavi 1994a), there is a limit on the number of nodes that may be at lower levels, and many merges must succeed at higher levels where the number of nodes is much less than the number of leaves of the ODT

One advantage of merging nodes comes from the abil ity to assign classes to nodes with no instances If a split is done and no instances fall to one of the children the child is labelled with the majority class of the parent

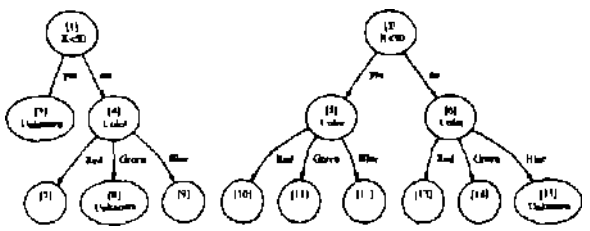

Figure 3 Two compatible subtrees The numbers in brackets are the node numbers referred to in the text

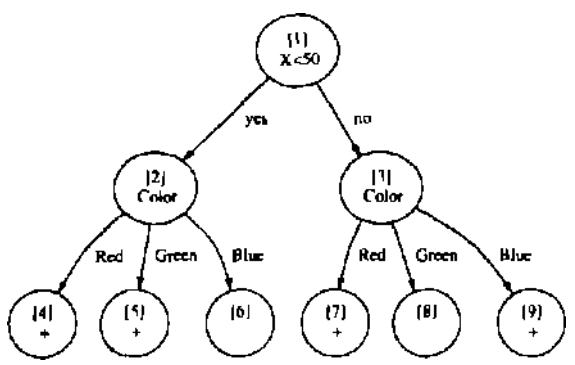

Figure 4 The merged subtree of the compatible subtrees above

When budding the ODJ *we* leave the node labelled as unknown" until the merge phase is done because (he merge phase may assign it a class Intuitively we would like an unknown node to match anything when we check whether two subtrees can be merged Two subtrees; are defined as compatible if either at least one root is labelled unknown' (in which case it is a leaf) or if the root nodes make the same test and the corresponding children are the roots of compatible subtrees Figures 3 shows two compatible subtrees and Figure *4* shows the merged subtree

The merging of compatible subtrees changes the bias of the algorithm bv assuming that a child node that is marked unknown' is likely to behave the same as another child if they belong to compatible subtrees In order to strengthen the usefulness of the merging process we split all the nodes at a level that are not leaves labelled "unknown " even those that are pure Node 4 in Figure 3 is pure but it has no instances that have the color green If the node is merged with node 6, it will correctly classfy all instances in the training set, but will classify instances that reach the node and have the color green the same way that node 6 classifies them

In noisy situations it may be that two subtrees are almost compatible and should be merged, but the above definition of compatibility will not do so Two subtrees are K-compatible if when we merge them, the number of newly misclassified instances (i e instances correctly classified by the subtrees but not by the merged tree) is

---

Input A training set and a test for each level of the oblivious decision tree
Output An oblivious read-once decision graph

```
1  // The bottom-up pruning of levels
2  Foreach i from 1 to no levels in full-grown tree do
3       acc_i = cross-validation acc of tree grown to level i
4  Let ℓ be the level for which acc was the highest
5  Let ODT be the oblivious tree grown to level ℓ
6  // Top-down merging of ODT
7  Foreach i from 1 to ℓ {
8       Repeat
9            For level i, find a pair of nodes that are approx
             compatible and merge them
10 } // End foreach
```

Figure 5 The bottom-up pruning and top-down merging/pruning

---

less than *h* Compatible subtree are 0-compatible When two nodes are considered for merging we allow them to misclassify *k* instances where *k* is defined similarly to the pruning method used in C4 5 (Quinlan 199 3) the number of misclassifications of each subtree is increased to the high-value of a confidence interval and the merge takes place only if a similar increase in the merged subgraph results in fewer misclassifications than the sum of the adjusted misclassifications of the children We have used a 909c < onfidence interval

3 3 Bottom-up Pruning

Intuit ively, the test for merging nodes checks whether the reduced error-rate due to the nodes being separate is significant statistically The merging of K-compatible nodes works well in noisy situations only if the misclassification rates at the leaves are non-zero If the tree overfits the data and has a resubstitution error rate near zero, any merging of subtrees at the top will seem to introduce many misclassifications that are unlikely to happen in a random sample with a large number of instances

In order to get a more 'honest" estimate of the misclassification error at the leaves, we use cross-validation to decide how many levels of the ODT to prune We fix the order of tests (and thresholds) at the levels, and cross validate every level to get an accuracy estimate Building an ODT given an ordering is an extremely fast process, so cross-validating the tree at different levels is not an expensive process Note however that the estimate is likely to be slightly optimistic because the split tests were determined using all the data Figure 5 shows the bottom-up pruning algorithm and the top-down merge/prune phase

Dataset	train size	test size	features continuous	features nominal	no classes
australian all	690	10-CV	6	8	2
balance-scale all	625	10-CV	4	0	3
breast all	683	10-CV	10	0	2
chess all	3196	10-CV	0	36	2
cleve all	296	10-CV	6	7	2
crx all	653	10-CV	6	9	2
diabetes all	768	10-CV	8	0	2
flare all	1066	10-CV	2	8	2
german all	1000	10-CV	24	0	2
mushroom all	5644	10-CV	0	22	2
segment all	2310	10-CV	19	0	7
soybean-large all	562	10-CV	0	35	19
tic-tac-toe all	958	10-CV	0	9	2
vehicle all	846	10-CV	18	0	4
vote	301	10-CV	0	16	2
monk1	124	432	0	6	2
monk2	169	132	0	6	2
monk2-local	169	432	0	17	2
monk3	122	432	0	6	2
parity5+5	100	1024	0	10	2
shuttle-small	3866	1934	9	0	7
waveform (21)	300	4700	21	0	3

Table 1 The domains

Dataset	C4.5	C4.5Rules	EODG	EODT	p-value
australian all	85.36±1.19	85.22±1.03	82.61±1.85	83.19±1.59	0.06018
balance-scale all	76.94±1.81	77.10±2.31	86.70±1.22	82.20±1.95	0.99997
breast all	95.32±0.87	95.76±0.63	95.32±0.72	94.59±0.69	0.49119
chess all	99.50±0.13	99.47±0.12	98.50±0.25	98.75±0.22	0.00251
cleve all	74.98±2.53	76.33±3.54	78.76±2.54	76.71±2.88	0.86370
crx all	84.08±1.08	83.93±1.27	84.24±1.29	85.31±1.16	0.54261
diabetes all	71.75±1.02	72.40±0.95	74.74±1.23	74.61±1.55	0.95714
flare all	82.36±1.35	81.80±1.22	82.64±1.41	82.83±1.35	0.71643
german all	72.50±1.41	73.40±1.31	70.70±1.78	71.40±1.87	0.16188
mushroom all	100.0±0.00	100.0±0.00	100.0±0.00	100.0±0.00	0.50000
segment all	96.36±0.33	96.10±0.45	94.76±0.53	96.45±0.42	0.00321
soybean-large all	92.54±1.43	91.28±1.00	81.13±1.87	84.29±1.79	0.00002
tic-tac-toe all	85.59±1.08	98.85±0.57	89.45±0.96	83.20±1.43	0.99718
vehicle all	69.84±1.77	71.86±1.56	54.61±2.72	66.78±1.02	0.00027
vote	95.64±0.52	95.87±0.45	94.91±0.31	94.49±0.61	0.08810
monk1	75.70±2.06	100.0±0.00	100.0±0.00	97.22±0.79	1.00000
monk2	65.00±2.29	65.28±2.29	68.52±2.23	68.75±2.23	0.82534
monk2-local	70.40±2.20	65.74±2.29	98.84±0.50	95.60±0.98	1.00000
monk3	97.20±0.79	96.30±0.91	97.22±0.79	97.22±0.79	0.50712
parity5+5	50.00±1.56	50.00±1.56	50.00±1.56	50.00±1.56	0.50000
shuttle-small	99.50±0.16	99.64±0.14	99.74±0.11	99.74±0.11	0.89174
waveform (21)	70.40±0.66	72.60±0.65	65.77±0.69	67.13±0.68	0.00000

Table 2 The accuracies for C4.5, C4.5-rules, EODG, EODT. The last column indicates the p-value for a t-test, indicating whether EODG is better than C4.5 (values > 5) or vice versa

Dataset	C4 5	EODG	EODT	EODG time in seconds
australian	58 6	25 7	804 8	148 8
balance-scale	80 0	35 5	1533 8	67 5
breast	20 4	14 7	279 2	29 9
chess	56 2	69 4	14335 0	2019 7
cleve	45 0	12 9	152 9	32 1
crx	57 7	25 3	764 7	133 7
diabetes	122 8	18 0	319	142 6
flare	2 8	4 6	26 6	151 2
german	158 4	17 2	292 6	192 5
mushroom	30 5	31 6	329	226 4
segment	81 8	179 1	6694 4	482 9
soybean-large	66 5	80 9	761 1	278 7
tic-tac-toe	129 1	64 6	2018 2	99 0
vehicle	178 6	252 3	10209 6	574 1
vote	15 1	7 0	223 9	38 4
monk1	18 0	7 0	53 0	3 6
monk2	31 0	21 0	214 0	14 3
monk2-local	47 0	16 0	125 0	14 2
monk3	12 0	4 0	16 0	4 0
parity5+5	23 0	11 0	63 0	17 8
shuttle-small	17 0	14 0	45 0	103 0
waveform (21)	51 0	30 0	317 0	53 8

Table 3 Comparison of the sizes of the induced structures The last column indicates the time in CPU seconds on a Sparc-10 for EODG to train and (est once (equivalent to one fold in ten-fold CV)

4 Experiments

To estimate the performance accuracy of EODG, we chose a few artificial datasels and real-world datasets from the UC Irvine repository (Murphy & Aha 1994) The artificial datasets are standard benchmark datasets such as the monk problems (Thrun etal 1991), from the real-world domains, we chose ones that had at least 200 instances Because we have not implemented classification of instances with unknown values we removed all such instances from the datasets tested Table 1 shows the characteristics of the different domains used For the artificial domains and for shuttle, we ran a train/test-set sequence For the other domains, we did 10-fold cross-validation

Table 2 shows the accuracies of C4 5, C4 5-rules, EODG and EODT (the first two stages of EODG) The last column is the p-va]ue for a paired t-test on the cross-validation folds of C4 5 and EODG (For the rune on a single test-set, we did an unpaired t-test using the sum of the estimated standard deviations from the test set) For the single runs, we did an unpaired t-test Values greater than 0 5 indicate that EODG is better and values less than 0 5 indicate that C4 5 is better On many datasets, the behavior of C4 5 and EODG is similar, there are datasets for which EODG's accuracy is worse then C4 5 (soybean-large, vehicle, waveform), and there are datasets for which the accuracy is better (balance-scale cleve tic-tac-toe, monkl, monk2-!ocal) It is interesting to note that on some problems EODT performs better, indicating that the symmetry bias assumed in the merge phase of EODG is not appropriate but the oblivious restriction doe* not hurt much

One of the main advantages of EODG is the small graphs it produces compared to C4 5 Table 3 shows the average sizes Tor thf graphs created for each dataeet EODG clearly creates smaller graphs and when it creates significantly larger graphs (soybean, vehicle) it is also a worse performer The graphs created by EODT are huge

5 Future Work

The bottom-up pruning phase of EODG prunes the ODT, which may be inappropriate For example, parity is badly represented as a tree, and thus the pruning stage reduces the number of features too much Initial experiments that involve pruning after merging showed that it may be better (e *g* , parity5+5 gives 100% accuracy), but the algorithm becomes too expensive

The EODG algorithm can be improved along a few directions We have not implemented handling of unknown values, but we believe that an approach similar to that used in decision trees (Quinlan 1993) should work well We are currently conducting tests on a single fea-

ture at a node, other tests, such as oblique splits are possible Because EODG suffers less from lack of instances at lower levels, using such tests might not suffer from the same problems that have occurred in decision trees The disadvantage of such an extension is the loss of comprehensibility that is one of the most important characteristics of decision trees and graphs over other hvpothesis spaces

We noticed that for continuous features, many splits art made on different thresholds Since we only merge nodes at the same level, we might consider multi-way splits on real features, as suggested in Fayyad & Irani (1993)

A different approach to feature selection and ordering, which is more feasible in OODGs then in decision trees is to try different ordenngs on the features An ODT on a dataset is defined by a set of tests one per level and hence the space of possibilities is much smaller than that of derision trees (although still ver> large)

6 Summary

We reviewed the characteristics of oblivions read-once decision graphs as the underlvmg hypothesis spa* t for in dution algorithms We introduced EODG, a top-down algorthm that constructs oblivious decision trees using information gain as a splitting criteria and then merges nodes to form oblivious read-once decision graphs 'The merging of nodes from the top provides a new wa\ of pruning nodes in some observed <ases, this corrects a split on an irrelevant feature by merging the children

OODGs have a differenl bias from that of decision Tress, and thus some concepts that are hard to represent is trees are easy to represent as OODGs and vice-versa (independent of the specific induction algorithm used) Wf have shown that for some real-world dalasets at UC Irvine the bias of EODG is appropriate and for the problems where performance is approximately the same as that of C4 5, EODG produces much smaller graphs

Acknowledgments

Acknowledgments The work in this paper was done using the MLC++ library parti} funded by ONR grant N00014-94 1-0448 and NSF grants IRI-9116399 and IR1-9411106 We thank Scott Benson and the anonymous reviewers for comments on the paper We thank Yael Kleefeld and Ya-Huei Wang for their support

References

Breiman, L , Friedman, J H , Olshen, R A& Stonp, C J (1984), *Classification and Regression Trees,* Wadsworth International Group

Bryant, R E (1986), 'Graph-based algorithms for boolean function manipulation , *IEEE Transac tions on Computers* C-35(8), G77-691

Cover, T M & Thomas, J A (1991) *Elements of Information Theory,* John Wiley & Sons Inc

Fayyad, U M (1991), On the induction of decision trees for multiple concept learning PhD thesis EECS Dept, Michigan University

Fayyad U M (1994), Branching on attribute values in decision tree generation, in Proceedings of the twelfth national conference on artificial intelligence AAA! Press and MIT Press pp 601-606

Fayyad U M & Irani, K B (1993), Multi-interval dis--cretization of continuous attributes for classification learning, in R Bajcsy, ed "Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence' , Morgan Kaufmann

Kohavi, R (1994a), Bottom-up induction of oblivious, read-once decision graphs in "Proceedings of the European Conference on Machine Learning" Available by anonymous ftp from starry Stanford EDU pub/ronnyk/suroHL94 ps

Kohavi, R (19946), Bottom up induction of oblivious read-once decision graphs strengths and limitations, in "Twelfth National Conference on Artificial Intelligence pp 613-618 Available by anonymous ftp from Starry Stanford EDU pub/ronnyk/aaai94 pa

Murph\ P M & Aha D W (1994), UCI repository of machine learning databases For information contact ml-repositorv'QJics uci cdu

Pagallo, G & Ilaussler D (1990), Boolean feature discovery in empirical learning ' *Machine Learning* 5,71-99

Quinlan, J R (1993), *C4 5 Proqrams for Machine Learning* Morgan Kaufmann, Los Alios, California

Spackman, A K (1988) Learning c ategoncal criteria in biomedical domains, in Proceedings of the Fifth Internati)nal Machine Learning (onference , Morgan Kaufmann, pp 36-46

Thrun etal (1991) The monk s problems A performance comparison of different learning algorithms Technical Report (. MU-CS-01-197 Carnegie Mellon University