Feature Transformation and Multivariate Decision Tree Induction

Huan Liu & Rudy Setiono

School of Computing, National University of Singapore, Singapore 119260.
{liuh,rudys}@comp.nus.edu.sg

Abstract. Univariate decision trees (UDT's) have inherent problems of *replication*, *repetition*, and *fragmentation*. Multivariate decision trees (MDT's) have been proposed to overcome some of the problems. Close examination of the conventional ways of building MDT's, however, reveals that the fragmentation problem still persists. A novel approach is suggested to minimize the fragmentation problem by separating hyperplane search from decision tree building. This is achieved by *feature transformation*. Let the initial feature vector be $\mathbf{x}$, the new feature vector after feature transformation T is $\mathbf{y}$, i.e., $\mathbf{y} = T(\mathbf{x})$. We can obtain an MDT by (1) building a UDT on $\mathbf{y}$; and (2) replacing new features $\mathbf{y}$ at each node with the combinations of initial features $\mathbf{x}$. We elaborate on the advantages of this approach, the details of T, and why it is expected to perform well. Experiments are conducted in order to confirm the analysis, and results are compared to those of C4.5, OC1, and CART.

1 Introduction

Among many algorithms for univariate decision tree (UDT) induction, C4.5 [18] is one that is widely used and well received. The basic algorithm is divide-and-conquer. Let the classes be $\{C_1, C_2, ..., C_k\}$. To construct a decision tree from dataset D, we go through one of these steps: (1) if D contains one or more examples, all belonging to a single class C_j. Stop; (2) if D contains no example, the most frequent class at the parent of this node is chosen as the class. Stop; or (3) if D contains examples belonging to a mixture of classes, information gain is then used as a heuristic to split D into partitions (branches) based on the values of a *single feature*. Start recursion on each branch to build subtrees. C4.5 also devised sophisticated methods to deal with continuous features, overfitting, and so on [18]. Note that at each decision node, a univariate (single feature) test is performed to partition the data. Researchers noticed that this univariate test leads to (1) the replication problem - subtrees that are replicated in a UDT; (2) the repetition (or repeated testing) problem - features that are repeatedly tested (more than once) along a path in a UDT; and (3) the fragmentation problem - data is gradually partitioned into small fragments [16, 3]. Replication and repetition always imply fragmentation, but fragmentation may occur without any replication or repetition if many features need to be tested [6]. Thus reduced is the UDT's capability of expressing concepts succinctly, which renders many concepts either difficult or impossible to express by UDT's. In addition, the univariate tests in tree building are also found sensitive to noise and minor variations in the data [20]. Multivariate decision trees (MDT's) can overcome these problems. In building an MDT, each test can be based on one or more features, i.e., a multivariate test. For example, the replication problem can be solved by testing Boolean combinations of the features; the repeated testing problem can be handled by forming linear combinations of the features. In [16, 3], it is shown that these combinations improve accuracy and reduce tree size.

2 Classical Way of Building MDT's and Its Limitations

Multivariate decision tress have been proposed to overcome the replication and repetition problems. In essence, an MDT is built in a way similar to building a UDT with one difference - performing univariate or multivariate tests. In building a UDT, at each test, we need to select the most promising feature. Many heuristic measures were proposed, such as impurity [2], information gain [17], and more in [14]; in building an MDT, however, we need to find the most promising hyperplane (a set of features is needed). It is known that selecting the best set of features for test at a node is an NP-complete problem [9]. In order to find an "optimal" hyperplane for a multivariate test, researchers offered various suggestions such as Regression [2], Linear Programming [1, 3], Randomization [15], and Simulated Annealing [9]. In the following, we examine the conventional procedure of MDT building with reference to UDT building.

2.1 Common aspects of the MDT building

Many issues in building MDT's are the same as those for UDT's. Both types of trees are built from labeled examples. In finding a linear combination of initial features for an MDT test, however, we encounter some issues that are not found in a UDT. Testing on a single feature is equivalent to finding an axis-parallel hyperplane to partition the data, while testing on multiple features we try to find an oblique hyperplane. The latter is significantly more complicated than the former. All existing methods of building an MDT can be described in the following procedure (the conventional way):

1. Select features that should be included in a linear combination;
2. Find the coefficients of a multivariate test that result in the best partition of the examples;
3. Build the MDT recursively; and
4. Prune the MDT to avoid overfitting.

In Step 1, a UDT only examines at maximum N features where N is the number of features describing a dataset, but an MDT needs to look at $\sum_{i=1}^{M} \binom{N}{i}$, where M is the best number of features for the current oblique hyperplane. A UDT does not require Step 2 which is also a very time-consuming step [9].

Basically, we need to find an optimal orientation for the hyperplane in an M-dimensional space. In order to avoid exhaustive search for M features and corresponding coefficients, researchers offered some solutions. Breiman et al. [2] suggested a perturbation algorithm that modifies one coefficient of the hyperplane at a time. Murthy et al. [15] implemented OC1 that uses a randomized algorithm to avoid local minima in modifying coefficients (and several coefficients can be modified at a time). Brodley and Utgoff [3] reviewed more algorithms for finding the coefficients of a hyperplane (e.g., linear programming, a recursive least squares procedure, the Pocket algorithm, and the thermal training procedure). Heath et al [9] employed a simulated annealing algorithm. All in all, various solutions have been proposed for finding optimal coefficients, and no absolute conclusion on which solution is the best [3]. Step 3 is the same for both MDT's and UDT's with an MDT having fewer recursions since it induces a simpler tree in general (though every node is more complicated). Step 4 is also common to both types of trees. The difference is that an MDT node contains more information (an oblique hyperplane) compared to a UDT node (an axis-parallel hyperplane). Extra care should be taken in pruning an MDT node.

2.2 Limitations of the conventional procedure

From the head-to-head comparison above, we realize (1) it is indeed *a complicated process* to induce an MDT following *exactly* the way in which a UDT is built; (2) in this process, simulated annealing, randomization algorithms, or heuristics have to be used for selecting a split point at node in an MDT to avoid exhaustive search; (3) searching for optimal hyperplanes has to be repeated as many times as the number of multivariate test nodes. For example, if a final MDT has 10 internal nodes, that means ten oblique hyperplanes have been formed; (4) the fragmentation problem persists. The last point is obvious since hyperplane search is intertwined with tree expansion as in univariate decision tree induction. For example, if the data is partitioned equally at each test, the size of the data for a test at a new level k (which is 0 at the tree root) decreases by a factor of 2^k. Sample shortage can cause inappropriate estimation for methods such as regression, neural networks, and simulated annealing. Because a hyperplane (multivariate test) is generally more powerful in partitioning data into uniform classes than a univariate test is, powerful multivariate tests have to be based on small fragments of data due to the fragmentation problem.

3 A Novel Perspective

The conventional way of MDT building has limitations. We propose here a different perspective. As we know that hyperplane search at each multivariate test could be complicated and time-consuming, can we separate hyperplane search from tree induction? The separation will also avoid finding hyperplanes with data fragments. Our proposal is to build an MDT via *feature transformation*.

Let the initial feature vector be $\mathbf{x} = (x_1, x_2, ..., x_N)$, the new feature vector after applying feature transformation T is $\mathbf{y} = (y_1, y_2, ..., y_M)$, where $\mathbf{y} = T(\mathbf{x})$. We obtain a linear MDT by (1) building a UDT with $\mathbf{y}$; and (2) replacing new feature vector $\mathbf{y}$ at each node with the combinations of initial feature vector $\mathbf{x}$.

Algorithm BMDT

Input: dataset D with feature vector $\mathbf{x}$
Output: an MDT
BMDT (D)
1. `Trans` = `buildTrans`(D) with $\mathbf{y} = T(\mathbf{x})$;
2. D_Y = `Trans`(D) /*D_Y with feature vector $\mathbf{y}$*/;
3. $Tree_Y$ = `createUDT`(D_Y);
4. MDT = `Repl`($Tree_Y$) with $\mathbf{y} = T(\mathbf{x})$

`buildTrans`() creates a data transformer (`Trans`) that can change the original data D into D_Y which is described by feature vector $\mathbf{y}$. In the process, we also obtain feature transformation T. `createUDT`() induces a univariate decision tree, $Tree_Y$. `Repl`() replaces all y_i in $Tree_Y$ with combinations of x_j's. By transforming features from $\mathbf{x}$ to $\mathbf{y}$, we can indeed create an MDT about $\mathbf{x}$ by inducing a UDT of $\mathbf{y}$. Steps 2 and 4 in BMDT are straightforward. The important components of BMDT are (1) `buildTrans`() and (2) `createUDT`(). Our choices are a feedforward neural network (NN) for `buildTrans`() and C4.5 for `createUDT`().

Transformation via a neural network

The tool for transformation is a standard 2-layer feedforward neural network [19] which has the input, hidden, and output units. The first layer of weights is formed by the input and hidden units, and the second layer by the hidden and output units. At each layer, the units are fully connected. The weighted inputs are fed into hidden units and summed. A squashing function (e.g., logistic) is normally applied at hidden units, thus an *activation value* is obtained. The activation values are then fed to the output units where the values are summed at each output unit, and the class labels are determined. The typical issues about such NN's are (1) how many hidden units should be used; and (2) input/output coding of the data. In BMDT, we simply use an excessive number of hidden units (e.g., if there are N inputs (features), use $N/2$ hidden units), and rely on a pruning algorithm [8, 21], to find the sufficient and necessary number of hidden units for good performance from the neural net. After pruning, the number of hidden units is usually much smaller than that of the input units. The input and output coding schemes are standard, for example, using normalized values for continuous data, binary coding for nominal data, and thermometer coding for ordinal discrete data.

In transforming features using a neural network, the remaining hidden units are our new features $\{y_1, y_2, ..., y_M\}$. The activation values of an original example form a new example in the feature space $\mathbf{y}$. As such, we obtain transformed data (D_Y in BMDT) as well as transformed features. Two more issues remain: (1) when should the network training and pruning stop (i.e., achieving sufficiently high accuracy without overfitting the data); and (2) how to find the combination of initial features $\{x_1, x_2, ..., x_N\}$ for each new feature in $\mathbf{y}$. The solution for (1) exists, i.e., by employing cross validation during network training and pruning. The solution for (2) is found by understanding how an activation value is obtained in the net. We will discuss it now.

Rewriting univariate tests in x

Let's look at how an activation value y_j is obtained. If we use the hyperbolic tangent function (δ) at the hidden units, we have

$$y_j = \delta(\sum_{i=1}^{N} w_{ij} x_i + \tau_j) \qquad (1)$$

where w_{ij} are weights on the connections from input i to hidden unit j, and τ_j is the bias for hidden unit j. Note that in (1) δ is a squashing function which normalizes new features in $\mathbf{y}$; and it is a one-to-one mapping function. The complexity thus introduced is that the transformation is not linear any more. However, this does not prevent us from building a linear MDT. After a UDT is induced from D_Y, a typical test is "$y_j < c$" and c is some threshold value determined by tree induction. Applying equation (1), we obtain an equivalent form of the test, "$\delta(\sum_{i=1}^{N} w_{ij} x_i) < c$". Taking the inverse δ, the test is "$(\sum_{i=1}^{N} w_{ij} x_i) < \delta^{-1}(c)$", a linear inequality involving the initial features $\mathbf{x}$. A linear MDT is thus built. It is obvious that in this context, a nonlinear MDT and a linear MDT are equivalent. We choose the linear form since it is simpler in representation, faster to test, and easier for us to compare with the results of other linear MDT builders such as OC1 and CART.

Why use a neural network?

A feature transformer should satisfy the following conditions: (1) it generates new features; (2) new features preserve information; (3) it compresses information so there are fewer new features than original ones; (4) reverse transformation is possible; and (5) it is simple and standard. Among many choices (see related work), we select a standard two-layer feedforward network for the task. This is because it satisfies all the conditions above. After the network training and pruning, hidden units are new features ($\mathbf{y}$). Neural network research [23, 22] demonstrates that hidden units are a powerful compressor of input information. Excessive hidden units can be pruned away without impairing the network's capability. More often than not, pruning can improve its performance by avoiding overfitting caused by excessive hidden units - hence, after condensing and pruning, hidden units (the new features) preserve the information represented by input units (the original features); the number of hidden units is usually much smaller than that of input units. As is shown earlier, it is straightforward to rewrite a multivariate test of y_i by a combination of original $\mathbf{x}$.

Analysis

Before we discuss empirical study, some analysis can help us to know why this novel approach deserves such a study and is worthy of our effort. With BMDT, training just one neural network has made it possible to find all hyperplanes required for a multivariate decision tree, instead of searching for as many hyperplanes as the number of multivariate test nodes in a decision tree. The power of an MDT lies in optimal or near optimal hyperplanes, the search of such a hyperplane can, however, be intractable [9]. If possible, we should minimize the number of such search.

All training data is used in neural network training and pruning to transform original features $\mathbf{x}$ to new features $\mathbf{y}$. New features $\mathbf{y}$ are then used in tree induction. This two-phase procedure effectively decouples hyperplane search from tree induction, unlike the classical way of MDT building in which a new hyperplane is searched as a new test node is required, and every new hyperplane has to be formed with less data (i.e., the fragmentation problem). In other words, the feature transformation approach (BMDT) extracts hyperplane search out of tree induction so that the fragmentation problem is minimized.

Pruning multivariate decision tree is not as straightforward as in univariate decision tree pruning. Overfitting does occur in MDT's as pointed out in [3], which is partially due to sample shortage that causes inappropriate estimation for methods such as regression, neural networks, and simulated annealing, but pruning the entire node with a multivariate test may result in even more classification errors. Brodley and Utgoff suggested that in such a case, one should try to reduce the error by eliminating some features from the multivariate test, in other words, no node is pruned. Overfitting is handled in the proposed approach through pruning in a univariate decision tree. No special treatment is needed. In addition, inappropriate estimation caused by sample shortage is limited because the new approach minimizes the fragmentation problem.

4 Empirical Study and Discussion

Our analysis above shows that the proposed approach should perform well. The analysis will be further confirmed via experiments. In particular, we wish to verify whether (1) BMDT can create MDT's; and (2) MDT's constructed by BMDT are comparable to, if not better than, those by other methods, such as CART, OC1, and C4.5, in terms of tree size and predictive accuracy. We select datasets commonly used and publicly available from the UC Irvine data repository [13] with different data types (nominal, continuous, and mixed), additional two data sets (Par12a and Mux12) are taken from [27]. They are summarized in Table 1. For nominal attributes, their values are binarized. Doing so is not biased against C4.5 since researchers [24] observed that this would improve univariate decision trees' accuracy. Five classification algorithms are used in the experiments, they are BMDT, CART, OC1, C4.5, and NN (a standard 2-layer feedforward neural network with pruning). CART is chosen because it is the very first one that proposed multivariate tests; OC1 is the latest one and publicly available, Murthy et al. [15] also compared OC1 with other multivariate decision tree induction algorithms with favorable results. C4.5 is chosen due to its reputable performance and used here for reference as base average tree size and average accuracy.

Results are presented in Tables 2 and 3. Also included are the results of a two-tailed t-test. The P-values are calculated for C4.5, OC1, and CART against BMDT based on 10-fold cross validation. Included also are the results of the

Name	#Data	#A	Type
1 Monk1	432	17	binary
2 Monk2	432	17	binary
3 Monk3	432	17	binary
4 Par12a	1024	12	binary
5 Mux12	1024	12	binary
6 TicTacToe	958	27	binary
7 Vote	300	48	binary
8 BreastCancer	699	9	continuous
9 Bupa	345	6	continuous
10 Ionosphere	351	34	continuous
11 Iris	150	4	continuous
12 Pima-diabetes	768	8	continuous
13 Sonar	208	60	continuous
14 Australian	690	14	mixed
15 HeartDisease	297	13	mixed
16 Housing	506	13	mixed

Table 1. Dataset Summary. #Data - data size, Type - attribute type, and #A - number of attributes.

neural networks (NN) based on which BMDT builds MDT's. We want to understand whether the difference between every pair is statistically significant. If it is significant, we further check which result is better by looking at the two average values. We observe the following from experiments:

Predictive accuracy (Table 2): First of all, BMDT's accuracy is not significantly different from NN's. However, BMDT performs significantly differently from C4.5 in 7 out of 16 cases. For all the 7 cases, BMDT's accuracy is better except for one (Mux12). When the results of BMDT are statistically different from those of OC1 and CART, BMDT always achieves higher average accuracy. The numbers of cases for which BMDT achieves significantly higher average accuracy over OC1 and CART are 6 and 4 out of 16 cases, respectively.

Tree size (Table 3): BMDT builds trees that are significantly smaller than those of C4.5's in all cases. In 7 cases, trees created by BMDT are significantly different from those of OC1's, in which two of the OC1's trees are smaller. In 9 cases, trees by BMDT are significantly different from those of CART's, in which only one of CART's trees is smaller.

An example: The dataset is Pima-diabetes. In Table 3, it is seen that C4.5 creates a UDT with average tree size of 122.4 nodes, BMDT builds an MDT with average tree size of 3 nodes. That means the MDT has one root and two leaves (class labels). Hence, there is only one oblique hyperplane at the root. Choosing one out of ten MDT's to show how it is obtained, we have the following: (1) after pruning, the network contains one hidden unit and 4 out of the 8 original inputs, i.e., after transformation, the new feature vector $\mathbf{y}$ is one dimensional in the new space; (2) building a UDT over the feature y, we have "if $y \leq 0$ then class 0, else class 1"; (3) inverse transformation and the network weights of the first layer

Name	BMDT	NN	C4.5	OC1	CART
Monk1	100	100	100	100	100
(P-value)		-	-	-	-
Monk2	100	100	93.7*	99.77	99.31
(P-value)		-	(**0.0025**)	(0.3434)	(0.3434)
Monk3	100	100	100	100	97.22*
(P-value)		-	-	-	(**0.0133**)
Par12a	100	100	100	99.49*	99.68
(P-value)		-	-	(**0.0109**)	0.1355
Mux12	99.6	99.39	100.0	99.44	99.32
(P-value)		(0.6324)	(**0.0982**)	(0.6822)	(0.3826)
TicTacToe	98.0	97.71	93.8*	91.75*	86.33*
(P-value)		(0.1801)	(**0.0001**)	(**0.0034**)	(**0.0001**)
Vote	96.3	96.33	97.3	93.10*	95.17
(P-value)		(0.7583)	(0.1944)	(**0.0422**)	(0.4073)
B-Cancer	96.1	96.0	95.3	94.99	94.71
(P-value)		(0.7583)	(0.2423)	(0.4033)	(0.2777)
Bupa	68.8	68.18	61.1*	66.09	65.22
(P-value)		(0.5565)	(**0.0475**)	(0.5289)	(0.3443)
Ionosphere	88.1	88.34	90.0	88.32	86.61
(P-value)		(0.5986)	(0.4249)	(0.9376)	(0.6827)
Iris	95.3	91.33	94.0	96.00	94.00
(P-value)		(0.3005)	(0.6975)	(0.7914)	(0.6510)
Pima	76.4	76.32	70.9*	72.40*	72.40*
(P-value)		(0.8996)	(**0.0094**)	(**0.0362**)	(**0.0362**)
Sonar	86.1	86.55	85.5	85.58	87.02
(P-value)		(0.3521)	(0.8519)	(0.9524)	(0.8482)
Australian	85.7	86.52	84.2	83.04*	86.09
(P-value)		0.3955	(0.2014)	(**0.0922**)	(0.7988)
Heart	82.8	83.15	72.0*	73.74*	77.78
(P-value)		0.7944	(**0.0046**)	(**0.0068**)	0.1335
Housing	85.8	85.96	82.0*	82.41	80.83*
(P-value)		0.5912	(**0.0256**)	(0.2625)	(**0.0470**)

Table 2. 10-fold cross validation results - accuracy (%). P-values in bold mean that results are significantly different from BMDT's. * indicates that BMDT is significantly better.

(input to hidden units) give the oblique hyperplane (in original attributes $\mathbf{x}$): if $5.67x_1 + 1.28x_2 + 2.81x_6 + 44.27x_7 \leq 316.20$ then class 0, else 1. In contrast to the UDT (109 nodes and each node with one attribute) built on the initial features $\mathbf{x}$ created by C4.5, the MDT has only three nodes and an oblique hyperplane with four attributes. In this case, the MDT has the simplest possible tree but a more complex node. Another point to note is that for this particular fold cross validation, UDT's accuracy is 64.94%, while MDT's is 76.62%.

We summarized the results with the following discussion.

– We can expect that BMDT performs well in general. This is because it overcomes the replication and repetition problems by still constructing multivariate decision trees, as well as minimizes the fragmentation problem through feature transformation. The latter cannot be achieved by the traditional way of constructing MDT's. Via feature transformation, BMDT circumvents the difficult problem of choosing M out of N features every time when a multi-

Name	BMDT	C4.5	OC1	CART
Monk1	9.8	14.8*	7.0	9.4
(P-value)		(**0.0001**)	(**0.0013**)	(0.5594)
Monk2	8.0	87.6*	5.4	7.4
(P-value)		(**0.0001**)	(0.1174)	(0.7418)
Monk3	10.0	15.0*	3.0	3.0
(P-value)		(**0.0001**)	(**0.0001**)	(**0.0001**)
Par12a	11.0	31.0*	11.4	27.8*
(P-value)		(**0.0001**)	(0.7472)	(**0.0899**)
Mux12	33.6	212.0*	29.6	63.0*
(P-value)		(**0.0001**)	(0.4846)	(**0.0028**)
TicTacToe	5.6	71.8*	17.6*	20.8*
(P-value)		(**0.0001**)	(**0.0787**)	(**0.0185**)
Vote	3.0	6.6*	3.2	3.2
(P-value)		(**0.0001**)	(0.3434)	(0.3434)
B-Cancer	3.2	22.2*	5.0	5.8*
(P-value)		(**0.0001**)	(0.1105)	(**0.0533**)
Bupa	5.8	79.2*	9.2	18.6
(P-value)		(**0.0001**)	(0.3961)	(0.1495)
Ionosphere	3.0	24.8*	8.6*	21.4*
(P-value)		(**0.0001**)	(**0.0552**)	(**0.0005**)
Iris	5.0	8.2*	5.0	5.2
(P-value)		(**0.0011**)	-	(0.3434)
Pima	3.0	122.4*	11.4*	26.2*
(P-value)		(**0.0001**)	(**0.0196**)	(**0.0722**)
Sonar	3.0	31.0*	8.2*	16.4*
(P-value)		(**0.0001**)	(**0.0056**)	(**0.0002**)
Australian	5.4	68.7*	4.8	6.2
(P-value)		(**0.0001**)	(0.8226)	(0.7571)
Heart	3.0	51.9*	9.2*	7.4*
(P-value)		(**0.0001**)	(**0.0566**)	(**0.0008**)
Housing	9.6	53.8*	4.2	8.8
(P-value)		(**0.0001**)	(0.1510)	(0.8640)

Table 3. 10-fold cross validation results - tree size. P-values in bold and * have the same meanings as in Table 2.

variate test node is created; tree pruning also becomes simpler. A feedforward neural network preserves and condenses information. Network pruning plays a key role in removing irrelevant information and in avoiding overfitting data. Referring to Tables 2 and 3, we notice that better average accuracy obtained by BMDT is usually due to a tree with fewer nodes.

– It is clear in Table 2 that average accuracy rates of BMDT and NN are the same except for one case that BMDT's is better but the difference is not statistically significant. Is there any advantage to have BMDT over NN? This question brings us back to the old question whether we need a decision tree when a neural network is present, or vice versa. It boils down to an issue of explicitness of learned representations (rules versus weights). The learned representation of a decision tree framework is explicit and allows one to examine the rules and explain how a prediction is made, while trained neural networks are largely black boxes[1]. As we see, the explicitness requires

[1] There are efforts at opening these black boxes [25, 7, 22].

extra work in BMDT. If no explanation for prediction is needed, rather than BMDT, NN should be the first choice due to its high accuracy.

- Adding multivariate tests, in a conventional way of constructing MDT's [15, 3], decreases the number of nodes in the tree, but increases the number of features tested per node. It is not the case for BMDT: owing to feature transformation, testing at a node is still univariate, but the number of nodes in the tree decreases. Recall that the dimensionality of $\mathbf{y}$ is usually much smaller than that of $\mathbf{x}$, this is because $\mathbf{y}$ corresponds to the hidden units after network pruning and $\mathbf{x}$ to the input units.
- Although oblique trees are usually small in terms of nodes in a tree, each node is more complicated than a node in an axis-parallel tree. Hence, there is a tradeoff to consider between the two types of trees: a large tree with simple nodes vs. a small tree with complicated nodes. This also indicates that MDT's are a complimentary alternative to UDT's. Adopting which type of trees depends on the problems (i.e., data) at hand.
- How can one include unordered features in a linear combination test? Different solutions have been suggested. In CART, Breiman et al [2] suggested to form a linear combination using only the ordered features. Another solution is to map each multi-valued unordered feature to m numeric features, one for each observed value of the feature [26]. This is equivalent to the binary coding mentioned earlier when the neural net input coding was discussed.

5 Related Work

In order to overcome the replication problem and alleviate the fragmentation problem, researchers have suggested various solutions. [16] proposed compound boolean features; [12, 28] employed constructive induction to construct new features defined in terms of existing features; [27] tried global data analysis - assessing the value of each partial hypothesis by recurring to all available training data. However, all these solutions are mainly designed for boolean data. The approach proposed in this work can handle all types of data (namely nominal, continuous, or mixed).

Feature transformation can be achieved by other forms. One popular approach is principle component analysis (PCA). The basic idea is to calculate eigenvalues of the covariance matrix of the data with $\mathbf{x}$ and select eigenvectors whose values are greater than a threshold (e.g., 0.9) to form a transformation matrix with M (the number of new features) rows and N (the number of original features) columns. The big disadvantage of PCA is that it cannot take advantage of the class information, though it is available. By discarding those eigenvectors with eigenvalues less than the threshold, PCA cannot fully preserve the input information.

Feature transformation may also be linked to techniques such as discretization [5, 11] and subset selection [10, 4]. Discretization transforms continuous data into discrete one. Subset selection, as its name implies, chooses a subset of original features. Although both techniques can reduce the dimensionality of the data, it is clear that neither new nor compound features are generated.

6 Summary

Univariate decision trees inherit the problems of replication, repetition, and fragmentation. Multivariate decision trees can overcome some of these problems. Close examination of the conventional way of building an MDT reveals that the fragmentation problem persists. Further investigation shows that the intertwining of hyperplane search and tree induction is the major cause since the conventional way of the MDT building follows exactly the way in which a univariate decision tree is built. We propose to separate hyperplane search from tree induction via feature transformation. A neural network is employed to transform original features to new features that preserve and compress information in data. It is chosen because of its many advantages over other methods. Instead of searching hyperplanes as the tree grows, applying feature transformation to the MDT building is, in effect, finding all the hyperplanes before tree induction. In this way, all training data is used for hyperplane search so that the fragmentation problem is minimized. We experimented with the new approach (BMDT) to prove the benefits of using this approach. The reduction of tree nodes from UDT's to MDT's is significant with equal or better accuracy - this indicates the effectiveness of BMDT in attacking the problems of replication, repetition, and fragmentation. The important conclusion is that it is possible and worthwhile to separate hyperplane search from tree induction; and doing so also makes building a multivariate decision tree simpler.

Acknowledgments

Thanks to S. Murthy and S. Salzberg for making OC1 publicly available. Thanks also to J. Yao, M. Dash, X. Ma for helping obtain the results of OC1 and CART reported in this paper.

References

1. K.P. Bennett and O.L. Mangasarian. Neural network training via linear programming. In P.M. Pardalos, editor, *Advances in Optimization and Parallel Computing*, pages 56–67. Elsevier Science Publishers B.V., Amsterdam, 1992.
2. L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone. *Classification and Regression Trees*. Wadsworth & Brooks/Cole Advanced Books & Software, 1984.
3. C.E. Brodley and P.E. Utgoff. Multivariate decision trees. *Machine Learning*, 19:45–77, 1995.
4. M. Dash and H. Liu. Feature selection methods for classifications. *Intelligent Data Analysis: An International Journal*, 1(3), 1997. `http://www-east.elsevier.com/ida/free.htm`.
5. U.M. Fayyad and K.B. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pages 1022–1027. Morgan Kaufmann Publishers, Inc., 1993.

6. J.H. Friedman, R. Kohavi, and Y. Yun. Lazy decision trees. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 717–724, 1996.
7. L. Fu. *Neural Networks in Computer Intelligence.* McGraw-Hill, 1994.
8. B. Hassibi and D.G. Stork. Second order derivatives for network pruning: Optimal brain surgeon. *Neural Information Processing Systems*, 5:164–171, 1993.
9. D. Heath, S. Kasif, and S. Salzberg. Learning oblique decision trees. In *Proceedings of the Thirteenth International Joint Conference on AI*, pages 1002–1007, France, 1993.
10. K. Kira and L.A. Rendell. The feature selection problem: Traditional methods and a new algorithm. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 129–134. Menlo Park: AAAI Press/The MIT Press, 1992.
11. H. Liu and R. Setiono. Chi2: Feature selection and discretization of numeric attributes. In J.F. Vassilopoulos, editor, *Proceedings of the Seventh IEEE International Conference on Tools with Artificial Intelligence, November 5-8, 1995*, pages 388–391, Herndon, Virginia, 1995. IEEE Computer Society.
12. C Matheus and L. Rendell. Constructive induction on decision trees. In *Proceedings of International Joint Conference on AI*, pages 645–650, August 1989.
13. C.J. Merz and P.M. Murphy. UCI repository of machine learning databases. `http://www.ics.uci.edu/~mlearn/MLRepository.html`. Irvine, CA: University of California, Department of Information and Computer Science, 1996.
14. John Mingers. An empirical comparison of selection measures for decision-tree induction. *Machine Learning*, 3:319–342, 1989.
15. S Murthy, S. Kasif, S. Salzberg, and R. Beigel. Oc1: Randomized induction of oblique decision trees. In *Proceedings of AAAI Conference (AAAI'93)*, pages 322–327. AAAI Press / The MIT Press, 1993.
16. G. Pagallo and D. Haussler. Boolean feature discovery in empirical learning. *Machine Learning*, 5:71–99, 1990.
17. J.R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.
18. J.R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, 1993.
19. D.E. Rumelhart, J.L. McClelland, and the PDP Research Group. *Parallel Distributed Processing*, volume 1. Cambridge, Mass. The MIT Press, 1986.
20. I.K. Sethi. Neural implementation of tree classifiers. *IEEE Trans. on Systems, Man, and Cybernetics*, 25(8), August 1995.
21. R. Setiono. A penalty-function approach for pruning feedforward neural networks. *Neural Computation*, 9(1):185–204, 1997.
22. R. Setiono and H. Liu. Understanding neural networks via rule extraction. In *Proceedings of International Joint Conference on AI*, 1995.
23. R. Setiono and H. Liu. Analysis of hidden representations by greedy clustering. *Connection Science*, 10(1):21–42, 1998.
24. J.W. Shavlik, R.J. Mooney, and G.G. Towell. Symbolic and neural learning algorithms: An experimental comparison. *Machine Learning*, 6(2):111–143, 1991.
25. G.G. Towell and J.W. Shavlik. Extracting refined rules from knowledge-based neural networks. *Machine Learning*, 13(1):71–101, 1993.
26. P.E. Utgoff and C.E. Brodley. An incremental method for finding multivariate splits for decision trees. In *Machine Learning: Proceedings of the Seventh International Conference*, pages 58–65. University of Texas, Austin, Texas, 1990.
27. R. Vilalta, G. Blix, and L. Rendell. Global data analysis and the fragmentation problem in decision tree induction. In M. van Someren and G. Widmer, editors, *Machine Learning: ECML-97*, pages 312–326. Springer-Verlag, 1997.
28. J. Wnek and R.S. Michalski. Hypothesis-driven constructive induction in AQ17-HCI: A method and experiments. *Machine Learning*, 14, 1994.