Converting A Trained Neural Network To A Decision Tree
DecText - Decision Tree Extractor

Olcay Boz
AT&T Labs
200 Laurel Ave.
Middletown, NJ, USA.
E-mail: olcay@att.com

Abstract—
Neural Networks are very successful in acquiring hidden knowledge in datasets. Their most important weakness is that the knowledge they acquire is represented in a form not understandable to humans. Understandability problem of Neural Networks can be solved by extracting Decision Rules or Decision Trees from the trained network. There are several Decision Rule extraction methods and Mark Craven's TREPAN which extracts MofN type Decision Trees from trained networks. We introduced new splitting techniques for extracting classical Decision Trees from trained Neural Networks. We showed that the new method (DecText) is effective in extracting high fidelity trees from trained networks. We also introduced a new discretization technique to make DecText be able to handle continuous features and a new pruning technique for finding simplest tree with the highest fidelity.

I. Introduction

Neural Networks have very good generalization capability. The most important weakness of Neural Networks is that they are like black boxes. Understanding the reasoning behind a Neural Network's output is not easy. Knowledge acquired by a Neural Network is represented by its topology, by the weights on the connections and by the activation functions of the hidden and output nodes. These representations are not easily understandable.

Symbolic learning techniques produce more understandable outputs but they are not as good as connectionist learning techniques in generalization [1], [9], [6].

In this paper, we present new splitting techniques for extracting classical Decision Trees from trained Neural Networks. Decision Trees are more understandable and they can easily be converted to If-Then rules. We show that the new method developed is able to create decision trees which are close in accuracy to the neural networks they are extracted from and their fidelity to the networks are high. **Fidelity** is the ability of the extracted decision tree to imitate the neural network it is extracted from. Fidelity is important if the rules extracted are going to be used for understanding the neural network. We concentrate on increasing fidelity more than increasing accuracy of the extracted decision tree. We called the Decision Tree extraction algorithm **DecText** (**Dec**ision **T**ree **ext**ractor). DecText is applicable to classification problems only. We also developed a new discretization and a new pruning algorithms. The discretization algorithm is used for handling continuous variables while extracting Decision Tree from a trained Neural Network. Pruning algorithm tries to minimize tree size while trying to maximize fidelity of the Decision Tree to the Neural Network.

II. Rule Extraction From Neural Networks

Lack of explanation capability is one of the most important reasons why Neural Networks do not get the necessary interest in some parts of the industry. In most of the real world applications (especially in safety critical applications) users want to know the reasoning behind the conclusion of a learning system or an expert system.

Extracting If-Then rules is usually accepted as the best way of extracting the knowledge represented in the Neural Network. Here are some advantages of extracting rules from neural networks:

- The rules extracted form the trained net can be used in other systems, like expert systems.
- Rules may be used to discover previously unknown features in the data (data mining).
- Rules may be used for explaining the reasoning behind the output of the system.
- Generalization of the system can be improved by having a better feature representation.
- Rules can also be used in refining incomplete domain knowledge.

Decision trees can also be easily converted to If-Then rules. Therefore, converting a neural network to a decision tree is as good as extracting If-Then rules.

Quality of the rules can be measured by looking at Accuracy, Fidelity, Comprehensibility and Complexity of the rules. Fidelity is the fraction of instances on which Neural Network and the extracted rules give the same output. Comprehensibility is measured by the size of the rule set and by the number of antecedents in each rule. Large rule sets are difficult to understand.

A. Trepan

TREPAN was developed by Craven [4], [3]. It converts a Neural Network to a MofN Decision Tree. TREPAN uses MofN type splits like ID2-of-3 [7]. An MofN expression is satisfied when at least M of its N conditions are satisfied.

TREPAN has an Oracle which determines the class predicted by the network for each instance presented to it. Instances may be created randomly. Oracle is also used to determine class labels for the tree leafs and to select splits for creating tree's internal nodes. While creating new random instances, TREPAN uses empirical distributions to model discrete valued features and kernel density estimates [10] to model continuous valued features.

TREPAN uses local models which makes use of constraints. This improves accuracy for some domains but creates problems for some others because of scarce data at a node. TREPAN tries to overcome this deficiency by comparing the model at a node with the model of its closest parent which uses its own local model. If they are close, parent's model is used at the node, else local model is created. Creating local models by using MofN type constraints is much more difficult and expensive than by using regular constraints.

TREPAN uses a heuristic search which aims to increase the fidelity for constructing MofN splits. It first finds a binary split by using information gain. TREPAN splits two valued features into two. If a feature has more than two values, binary splits based on each value is considered (e.g. computer = desktop? computer = laptop?). The binary split selected is used as a seed for the search process. Information gain is used as the evaluation function in the search. Candidate splits are created by adding a new member to the set and keeping m constant or by adding a new member and incrementing the value of m by one.

TREPAN uses two stopping criteria. If a node covers the instances of a single class with high probability, it is made a leaf. It finds the proportion of examples (p_c) (the proportion of the number of instances classified as the most common class to the number of total instances at the node) and the confidence interval for the class. It queries the oracle until $prob(p_c < (1-\epsilon)) < \delta$). ϵ and δ are parameters to the algorithm. TREPAN also stops tree generation by limiting the depth of the tree.

For labeling the leafs, p_c is found first. If p_c does not satisfy the first stopping criterion, TREPAN tries to find out if there is enough confidence in p_c to label that node with the dominating class. If there are not enough instances new random instances are created until they satisfy $\forall_{\mathbf{i} \neq \mathbf{c}}[\mathbf{prob}(\mathbf{p_c} < \mathbf{p_i}) < \delta]$.

III. DecText - Decision Tree Extractor

DecText is a Decision Tree extraction algorithm. The Decision Tree extracted from the network mimics the network's behavior. It is the network's representation in a more understandable form. DecText tries to make Decision Tree learn the concept represented by the Neural Network.

We have developed new splitting techniques, a new discretization method which uses Neural Network for making DecText be able to handle continuous variables and a new tree pruning technique which tries to optimize fidelity between the tree and the network.

One of the problems with classical Decision Tree creation algorithms is that splitting tests toward the leafs are selected using fewer instances than splitting tests up in the tree. Therefore splitting tests toward the leafs are less reliable. DecText can use unlabeled instances. Therefore if we have unlabeled instances available, we can use fixed number of instances to choose splitting features at any level in the tree. In some problem domains unlabeled instances are readily available (e.g. data from sensors etc.). If we have a mathematical model of the problem, we can create new instances by using that model. If we have neither, we can create a model of the dataset and we can create random instances by using the model.

We handle continuous features by first discretizing them. Discretization can be done globally (before creating the Decision Tree) or locally at each node of the tree. Discretization algorithm creates N + 1 discrete values for the feature. If a continuous feature is used for splitting, N + 1 subnodes are created and that feature is not used again.

Random instances are used only for choosing the best splitting feature and for labeling nodes which have no training data instances available to label them.

To maximize fidelity all leafs are labeled by using the outputs of Neural Network on training data instances. If there are no training instances available at a node, n random instances are created (by using its parent's data model if local data modeling is being used or by using global data model if global data modeling is being used) and their outputs from the Neural Network are used for labeling the leaf. n is by default equal to the number of instances in the training dataset.

By default, we do not use a stopping criteria. We stop creating subnodes only if all the instances at a node are classified the same by the Neural Network or if the node is empty. User may limit the depth of the tree. If the tree reaches that depth all the nodes at that depth are made leafs and they are labeled either by the available dataset instances or by using random instances (if the node is empty). User may also enter a fidelity stopping value. If fidelity at a node is equal to or greater than the fidelity stopping value entered by the user, that node is made a leaf and is labeled by using the Neural Network's outputs of training set instances.

Fidelity at a nonterminal node is found by finding the dominating class (C_D) from the training data instances at that node. Class values are found by using the outputs of the Neural Network for each instance. Then, the number of instances ($|S_{C_D}|$) classified as the dominating class (C_D) is found. $|S_{C_D}|/|S|$ ($|S|$: number of training instances at the node) gives the fidelity at that node.

A. Splitting Methods

We have developed 4 new splitting methods. SetZero splitting criteria tests the Neural Network's outputs for each feature to find the most relevant feature. SSE, ClassDiff and Fidelity splitting criteria check the partitions created by the features and choose the feature which creates the best partitions according the criteria they use.

A.1 SetZero Split

In SetZero splitting algorithm, the feature which effects the outputs of the Neural Network the most is chosen for splitting the dataset. Changes in the value of more relevant features effect classification more than the changes in the value of less relevant features. Therefore, the most relevant feature has the most predictive power on classification. For each node by using the dataset and the feature set available at that node we can find the feature with the most predictive power for the dataset at that node.

We call the new method SetZero because for finding the relevance of a feature, we set its value to zero in each instance (to simulate removing the corresponding input node from the

Neural Network) and find the difference in Neural Network's response to the new instance and the original instance.

For each instance, the output values for the original instance are found. Then, a feature from the instance is removed and the output values for this new instance are found. The difference between the original output values and the new output values tell us how much the feature removed effects the outputs. The effect of each feature on all the outputs for all the instances are found. Then, the average effect of each feature on each output and on the network is found.

If an instance is $\{f_1, f_2, \ldots, f_{nF}\}$ and the outputs from the Neural Network for this instance are $\{o_1, o_2, \ldots, o_{nC}\}$, for finding the effect of f_1 on the outputs, a new instance $\{0, f_2, \ldots, f_{nF}\}$ is created. This instance is presented to the Neural Network and the outputs $\{o_1^*, o_2^*, \ldots, o_{nC}^*\}$ are found. The absolute difference $|o_1 - o_1^*|$ tells us how much f_1 is effecting o_1.

$\sum_i^m (freq(o_i, S)/|S|)|o_i - o_i^*|$ tells us how much f_1 is effecting the network. $freq(o_i, S)$ is the frequency of the instances classified as o_i in the dataset. To be able to delete a feature by setting its value to zero more reliably, we train the network by using logistic activation function and scale the inputs to the range [0.1, 0.9]. In this way, Neural Network will not be trained by using value 0. Therefore, weights will not be able to compensate for the value 0.

A.2 SSE Split

SSE (Sum Squared Error) is calculated using the formula:

$$SSE = \sum_i^n \sum_j^m (t_{i,j} - o_{i,j})^2 \tag{1}$$

t: Teaching output **o**: NN's output **n**: # of instances **m**: # of classes

Splitting feature should be chosen in a way to maximize the possibility of a single class dominating each partition created. In SSE split, we measure the quality of each partition by finding SSE value for each partition. To be able to find an SSE value for a partition we have to have a teaching output. For unlabeled (i.e. random) instances we find the output by testing them on the network. We find the classification for all the instances. Then we find the dominating class in the dataset. We assume that all the instances in that partition should have been classified as the dominating class. Then, we set the teaching output to dominating class.

If the instances are all classified the same with high confidence by the Neural Network then SSE will be low. If they are classified as different classes with low confidence then SSE will be high. Classification confidence in feedforward classification Neural Networks can be measured by comparing output values for each class if one-of-m output representation is used. In one-of-m representation, each class is represented by an output node in the network. The class corresponding to the node with the highest value is considered as the classification for that instance. If the value for that output is close to 0.9 and the value for all the other outputs are close to 0.1, we can say that Neural Network is very confident about the output. We used 0.9 for the expected class value and 0.1 for the others. If there are 5 classes and if the expected class is C_2 then expected output will be $\{0.1, 0.9, 0.1, 0.1, 0.1\}$.

In SSE Split, for feature f_i, data subsets (S_{ij}) for each partition are found. Then, for each S_{ij}, the dominating class is found by using Neural Network's outputs. Expected classification is set to dominating class for S_{ij} and SSE_{ij} is calculated for S_{ij}. To find SSE_i value for feature f_i, all SSE_{ij} values are added for all partitions. Feature with the smallest SSE_i value will be used for splitting.

A.3 ClassDiff Split

Like SSE split ClassDiff split also measures the quality of the partitions created by each feature. The reason we call this new algorithm ClassDiff splitting is that we use the difference between the maximum average NN output value and the second maximum average NN output value in the subsets to measure how good a partition is.

In Neural Networks using one-of-m representation for classification, the difference between the highest output unit value and the second highest output unit value gives a good measure of the confidence of classification.

ClassDiff value for a subset is calculated like this: First a $TotalOut$ vector with nC (nC: number of classes) members is created and each member is initialized to 0. Then, each instance is tested on the Neural Network to find nC output values(Out_i). Out_i is added to $TotalOut_i$. $TotalOut_i$ is divided by $|S|$ ($|S|$: Number of instances in the dataset) to find $AverageOut_i$. Then, the maximum valued $AverageOut$ ($AverageOutMax$) and second maximum valued $AverageOut$ ($AverageOutMax2$) are found. $ClassDiff$ value is the difference between these two ($ClassDiff = AverageOutMax - AverageOutMax2$).

For finding the splitting feature f_i at a node, for each partition S_{ij} of f_i $classDiff_{ij}$ is calculated. Then, $classDiffPart_i$ is found by adding all $[(|S_{ij}|/|S|) * classDiff_{ij}]$ values for all the partitions of feature f_i. $|S_{ij}|$ is the number of instances in the jth partition of the ith feature. The feature with the highest classDiffPart value is the most relevant feature.

A.4 Fidelity Split

Fidelity split chooses the feature which creates partitions with the highest total fidelity between the Neural Network and Decision Tree. Fidelity at a node is found like this: First the dominating class for the subset at a node (by using Neural Network's outputs) is found. Then, it is assumed that all the instances in that subset are supposed to be classified as the dominating class. The frequency of the instances classified as the dominating class in this subset gives us the fidelity at that node.

For finding the most relevant feature at a node, for each feature f_i, the dataset is partitioned into as many datasets as the number of values for feature f_i. For each partition S_{ij} of f_i $Fidelity_{ij}$ is found. Then, $FidelityPart_i$ is calculated by adding all $Fidelity_{ij}$ values for all the partitions of that feature. The feature with the highest $FidelityPart$ value is chosen as the most relevant feature.

B. Training Data modeling For Creating Random Instances

At each node, for finding the most relevant feature for splitting the dataset at that node, we use a certain number of instances. The default value used in DecText is the number of instances in the training set. But the value is user definable. If we have n training data instances at a node and if we need m instances for finding the splitting feature, we create (m - n) random instances. Random instances are created by using a model of the dataset. To model the training dataset we used Kernel Density Estimate (KDE)[10] for continuous features and empirical distributions for discrete features. We tested both global model and the local model. In both we used local constraints for the features.

Empirical distribution is the frequency of values in the dataset. KDE (Kernel Density Estimation) is calculated by using the formula:

$$f(x) = \frac{1}{m}\sum_{j}^{m}\left[\frac{1}{\sqrt{2\pi\sigma}}e^{-(\frac{x-\mu_j}{2\sigma})^2}\right] \tag{2}$$

m : number of training examples
μ_j : value of the feature for the jth example
σ : width of Gaussian kernel

The method we used is a very simple way of creating a model of the dataset and its effectiveness is limited. We can use a more complicated technique to create a better model from the data. If we use a complicated model creation method locally, it will be computationally very expensive because it has to be created at each node. Another problem is that, toward the end of the tree number of training instances get scarce. This reduces the reliability of the model. Constraints at the nodes help to reduce the size of the data domain but it may not be enough in all applications for creating a reliable model.

We believe that creating random instances by using a global model has advantages. When a global model is used, user will have more flexibility in choosing the modeling technique he will use. User may also use domain knowledge or dependencies between the features while creating the model. More complicated models can be used since it will be created only once. Therefore, it will not be computationally very expensive.

We used global modeling with local random instance creation. By doing this we take advantage of using the whole dataset for creation of the model and the constraints at each node. Global model is more reliable than the local model because it uses more instances.

C. ClassDiff Discretization

We developed a new discretization algorithm which uses Neural Network's outputs for finding cutpoints and for stopping creation of cutpoints. We call this new algorithm ClassDiff Discretization. The reason is that, we use the difference between the maximum average NN output value and the second maximum average NN output value in the subsets. For more information on how to find ClassDiff value for a subset please refer to section III-A.3.

Discretization is similar to splitting. The cutpoints found by the discretization algorithm should maximize the probability of having one class dominate the subsets created by the cutpoints.

For finding the cutpoints, instances are first sorted by increasing value of the feature F. The mid points of instances where classification of the Neural Network change are found. For each candidate cut point, the dataset is divided into two subsets and ClassDiff value is calculated for each subset.

To find the ClassDiff value for a cut point, the ClassDiff values for the left and right subsets created by the cutpoint are found. The final value is calculated by using the formula:

$$\begin{aligned} ClassDiff_p(S) \;=\; & \frac{|S_{left}|}{|S|}ClassDiff(S_{left}) + \\ & \frac{|S_{right}|}{|S|}ClassDiff(S_{right}) \end{aligned} \tag{3}$$

The cut point with the maximum $ClassDiff_p$ value is chosen as the final cut point. The subsets which has a ClassDiff value which is smaller than DiscStop are further partitioned recursively. DiscStop is user definable and the default is 0.4. This is a heuristic value. It generally gives good results with not too many cutpoints.

D. Fidelity Pruning

After the tree is extracted, it is first checked if any of the non-terminal nodes has children all classified the same. Such non-terminal nodes are made leafs and children's label is assigned to them as their label. Checking non-terminal nodes starts from the bottom of the tree and it goes toward the root. This kind of pruning does not change fidelity of the tree.

Fidelity Pruning tries to optimize fidelity of the tree while trying to make it as simple as possible. m random instances by using the data model for the whole training dataset are created first. m is user defined. The default value for m is the number of instances in the training dataset. For all the non-terminal nodes of the tree the node is made a leaf and fidelity of the new tree is tested by using the random dataset. If fidelity is better than maxFidelity, new fidelity is assigned to maxFidelity and tree is saved in best tree. At the end best tree is returned as the pruned tree.

While extracting the Decision Tree the dominating class at each non-terminal node is saved. While pruning when a non-terminal node is turned into a leaf it is labeled by using the dominating class saved while extracting the tree.

IV. Evaluation Of DecText

In this section we will show that DecText extracts high fidelity Decision Trees from trained Neural Networks. We also show that our discretization algorithm is much more flexible than entropy discretization and it performs better than entropy discretization. We also show that Fidelity Pruning reduces number of nodes and tree size without affecting fidelity or accuracy much.

First we report accuracy results for Neural Network, C4.5 and DecText for each dataset. We also compare number of nodes in the C4.5 trees and DecText trees. To be able to test the capabilities of the new algorithm better, for any of the tests we did not use pruning (except for the tests of the pruning algorithm). For comparing tree sizes and accuracy we created unpruned trees by using C4.5 (we used the command c4.5 -m1 -c100 -f file base -u).

A. Datasets Used For Evaluating DecText

We used the datasets Vote, Vote-3, Heart, and Housing. Vote dataset contains votes from U.S. House of Representatives during the 98th congress [8]. Heart dataset [5] is concerned about heart disease diagnosis. The data was collected from the Cleveland Clinic Foundation. In this dataset we changed the feature "number of major vessels" from continuous to discrete with values "0,1,2,3". Housing dataset concerns housing values in suburbs of Boston. The original class attribute was continuous representing median house value in $1000's. We converted this to a discrete valued attribute with 2 (0 and 1) values each representing the ranges $0 - $20000, $20001 - $50000.

Following Buntine, Niblet and Craven [2], [3] we removed the third feature "physician fee freeze" from the voting dataset and created vote-3 dataset. "physician fee freeze" feature makes the learning task much easier by partitioning the dataset very well between the two classes.

The datasets we used and their characteristics are listed in Table I.

TABLE I

DATASETS AND THEIR CHARACTERISTICS

Dataset	#Insts.	#Feats.	#Cont. Feats.	#Disc. Feats.	#Clas.	%Insts. Maj. Class
Vote	435	16	-	16	2	61
Vote-3	435	15	-	15	2	61
Heart	297	13	6	7	2	53
Housing	506	13	12	1	2	41

For training the Neural Network we used backpropagation momentum algorithm. For learning factor we used 0.2 and for momentum factor we used 0.1 as starting points and changed those values to try to get the best classification from the Neural Network. We also tested different number of epochs (cycles) and different number of hidden units for each dataset. For the hidden and output units we used logistic function as the activation function. In our tests for all the data sets we used 10(8,1,1) cross validation.

The number of hidden units we used for each dataset are:

Vote: 40, Vote-3: 40, Heart: 20, Housing: 20

The average number of epochs (cycles) we used for each dataset are :

Vote: 150, Vote-3: 170, Heart: 200, Housing: 1800

B. Initial Test Results For DecText

Test set accuracy results for Neural Networks, C4.5 and DecText are listed in Table II. The results are the average of 10-fold cross validation for all datasets. The datasets are chosen so that Neural Network has better classification accuracy.

Fidelity results from DecText are listed in table III. Accuracy and fidelity results are the results without using random instances and without any pruning. We used SetZero split for these initial tests. For discretization, entropy discretization was used. Later we will show that we can improve these results by using random instances and by using ClassDiff discretization technique.

TABLE II

TEST SET ACCURACY (IN PERCENTAGE) RESULTS FOR NEURAL NETWORK, DECTEXT, AND C4.5 FOR EACH DATASET

Method	Vote	Vote-3	Heart	Housing
C4.5	94.6	85.3	70.0	81.0
NN	**95.4**	**93.0**	**83.5**	**90.8**
DecText	94.2	88.2	77.6	86.0

TABLE III

TEST SET FIDELITY (IN PERCENTAGE) RESULTS FOR DECTEXT (WITHOUT RANDOM INSTANCES).

Method	Vote	Vote-3	Heart	Housing
DecText	96.0	94.2	84.5	88.0

We also compare the number of nodes in the trees and depth of the trees created by DecText and C4.5. The results are listed in tables IV. Later we will show that we can reduce the number of nodes in a DecText tree by pruning without reducing (or sometimes by improving) fidelity and accuracy much. The results show that if all the features have discrete values number of nodes in the trees are almost equal with C4.5. For continuous valued features DecText creates more nodes. This is mainly because C4.5 uses binary splits for continuous valued features. Although the number of nodes decreases if binary splits are used, depth of the tree increases.

TABLE IV

NUMBER OF NODES AND SIZE OF THE TREES CREATED BY C4.5 AND DECTEXT.

Method	Vote	Vote-3	Heart	Housing
# Nodes				
C4.5	**72.1**	**146.5**	**127.7**	**59.0**
DecText	85.0	149.8	164.0	134.0
Tree Size				
C4.5	**7.1**	**9.9**	9.1	10.5
DecText	7.7	**9.9**	**7.8**	**7.7**

We first tested new splitting techniques we developed. We also tested gain splitting technique to compare the results we get by using the new techniques. We report the fidelity results in percentages and we also report the tree depth and the number of nodes in the trees. All the results are the averages of 10-fold cross validation tests.

As it can be seen from the tables below, SetZero split gives much better results for the Vote and Vote-3 datasets. These datasets have more relevant features. SetZero sorts the features by their relevancy to the classes in the problem. The other splitting techniques sort the features by the quality of the partitions created. So, if there are highly relevant features in the dataset SetZero will create much better results. If there are not any relevant features in the dataset other splitting techniques will create better results. Gain split creates the smallest trees.

TABLE V

FIDELITY, ACCURACY, TREE SIZE AND # NODES VALUES FOR NEW SPLITTING TECHNIQUES.

Method	Vote	Vote-3	Heart	Housing
Fidelity %				
SetZero	**96.0**	**94.2**	84.5	86.2
SSE	94.4	92.8	**88.6**	87.4
ClassDiff	94.6	92.8	88.0	**87.8**
Fidelity	95.6	92.1	86.2	87.2
Gain	95.3	92.3	84.8	86.6
Accuracy %				
SetZero	94.2	88.1	76.2	82.2
SSE	94.4	88.1	**79.6**	83.8
ClassDiff	94.6	88.6	78.9	83.4
Fidelity	**95.1**	88.4	78.6	83.2
Gain	94.4	**89.5**	77.3	**84.2**
Tree Size				
SetZero	7.7	9.9	7.5	9.1
SSE	8.5	10.0	6.4	7.9
ClassDiff	8.5	9.9	6.6	7.6
Fidelity	8.8	9.9	**6.1**	7.6
Gain	**6.2**	**7.5**	6.7	**7.5**
# Nodes				
SetZero	85.0	150.0	163.8	165.8
SSE	73.0	126.0	74.9	105.5
ClassDiff	72.4	127.0	76.3	103.7
Fidelity	73.0	115.0	**74.0**	100.8
Gain	**45.4**	**85.0**	75.0	**87.4**

TABLE VI

FIDELITY VALUES IN PERCENTAGE OF NEW SPLITTING TECHNIQUES (LOCAL VS GLOBAL RANDOM INSTANCES - ENTROPY DISCRETIZATION)

Method	Vote	Vote-3	Heart	Housing
SetZero				
Local	96.5	94.5	85.5	**87.3**
Global	**97.2**	**95.2**	**86.5**	86.2
SSE				
Local	95.8	93.7	**89.7**	**88.1**
Global	**96.5**	**94.0**	**89.7**	87.0
ClassDiff				
Local	95.8	93.3	**91.0**	88.0
Global	**96.5**	**93.6**	87.9	**88.6**
Fidelity				
Local	95.1	**94.0**	**91.0**	**87.9**
Global	**96.0**	93.5	88.2	**87.9**
Gain				
Local	**96.3**	93.0	**87.6**	86.0
Global	96.2	**93.2**	87.2	**87.0**

TABLE VII

FIDELITY, ACCURACY, TREE SIZE AND # NODES VALUES OF NEW SPLITTING TECHNIQUES WITH FIDELITY PRUNING (NO RANDOM INSTANCES - ENTROPY DISCRETIZATION)

Method	Vote	Vote-3	Heart	Housing
Fidelity %				
SetZero	**96.3**	**93.7**	86.2	81.8
SSE	95.8	92.8	**90.0**	81.0
ClassDiff	**96.3**	93.3	89.0	**83.2**
Fidelity	96.0	91.9	89.3	80.4
Gain	95.8	91.9	88.0	75.0
Accuracy %				
SetZero	94.4	89.1	77.2	79.4
SSE	94.4	88.1	81.0	77.4
ClassDiff	**95.8**	**89.5**	79.3	**80.4**
Fidelity	94.7	88.1	**81.1**	78.4
Gain	94.4	89.4	79.7	73.4
Tree Size				
SetZero	6.1	8.3	6.4	6.7
SSE	**4.4**	6.9	4.6	5.3
ClassDiff	5.1	7.5	4.2	5.6
Fidelity	5.0	**6.7**	**3.9**	**4.8**
Gain	4.6	6.9	4.7	4.9
# Nodes				
SetZero	45.1	64.6	79.4	65.0
SSE	**18.1**	42.1	34.6	44.0
ClassDiff	21.1	47.2	35.8	40.8
Fidelity	18.4	**37.3**	**30.7**	34.0
Gain	22.9	38.2	33.2	**31.8**

C. Random Instances

In this section, we report the tests on DecText by using random instances. Random instances are created while choosing the best splitting feature or while labeling a node if there are no training data instances available at the node. The number of instances to be used at each node is set to the number of instances in the training dataset. We tested creating random instances by using local model, and global model.

Creating random instances for finding the splitting features and for labeling empty nodes improves fidelity results in almost all of the cases. For most of the tests using global model gives slightly better results than using local model. Running the algorithm couple of times and choosing the tree with the best fidelity improves the results. In some domains because of the constraints of the features we may not be able to create as many random instances we need to. In these cases we use as many random instances we can create.

D. Fidelity Pruning Algorithm

The results of the pruning algorithm are shown in Table VII. They show that tree size and number of nodes decreased considerably. Accuracy and fidelity increased for most of the cases. The only exception is the housing data. This may be because housing data has mostly continuous features. This needs to be investigated further.

E. Entropy vs ClassDiff Discretization

By fidelity comparison, Entropy and ClassDiff Discretization algorithms give very close results. ClassDiff Discretization results are slightly better than Entropy Discretization results. Housing dataset is dominated by continuous features. Heart dataset has more discrete features and discrete features are usually chosen at the top of the tree for splitting. Therefore goodness of ClassDiff algorithm is more obvious from the Housing dataset results.

Another advantage of using ClassDiff Discretization is that the level of discretization is user controllable by using ClassDiff parameter. This parameter is used when we decide if we should further find cutpoints in a partition or not. If ClassDiff parameter is small (i.e. 0.1) we will get fewer cutpoints. If it is high (i.e. 0.8) we will get more cutpoints. Number of cutpoints will effect the fidelity and the complexity of the extracted tree. The fewer the cutpoints the more understandable the tree is.

TABLE VIII

FIDELITY RESULTS FOR ENTROPY DISCRETIZATION AND CLASSDIFF DISCRETIZATION.

Method	Heart Entropy	Heart ClassDiff	Housing Entropy	Housing ClassDiff
SetZero	84.5	**84.8**	86.2	**87.4**
SSE	**88.6**	88.3	87.4	**88.0**
ClassDiff	88.0	**88.3**	87.8	**88.2**
Fidelity	86.2	**86.6**	87.2	**88.2**
Gain	**84.8**	83.5	86.6	**87.6**

F. Number Of Random Instances

Number of random instances used at each node for finding the splitting feature effects the fidelity considerably. Increasing the number of random instances used for finding splitting features increased fidelity.

For vote dataset we got the best results by using SetZero split by using 435 random instances created by using global data model. We got 97.2% fidelity for that dataset.

For vote-3 dataset we got the best results from SetZero split by using 500 random instances created by using global model. We got 96.4% fidelity for that dataset.

For Heart dataset we got the best results by using SSE split by using 5000 random instances created by using a local data model. We used ClassDiff discretization with a ClassDiff stopping value of 0.3. We got 94.4% fidelity for that dataset.

For housing dataset we got the best results by using ClassDiff split by using 506 random instances created by using local data model. We used ClassDiff discretization with a ClassDiff stopping value of 0.3. We got 91.0% fidelity for that dataset.

Here is the comparison of the results we reported in the beginning of the chapter and the best results we got.

TABLE IX

COMPARISON OF INITIAL FIDELITY RESULTS WITH THE BEST FIDELITY RESULTS.

Method	Vote	Vote-3	Heart	Housing
Basic	96.0	94.2	84.5	88.0
Best	**97.2**	**96.4**	**94.4**	**91.0**

V. Conclusion

Two of the most important citeria used for evaluating inductive machine learning algoithms are generalization and understandability. Generalization determines how well the model created by the algorithm fit to unseen data. Neural Networks are very good at generalizing on unseen data. But they are like black boxes. The way they represent the knowledge they acquire is not very comprehensible for humans. For safety critical real world applications understanding the reasoning behind the Neural Network's output is very important.

Several researchers addressed the problem by extracting rules from Neural Networks. Most of these methods were not applicable to general Neural Networks. Some required binary inputs. Some required special training algorithms. The ones applicable to general Neural Networks were computationally very expensive.

Craven addressed the problem by extracting MofN Decision Trees from trained Neural Network. MofN Decision Trees, we believe, are harder to understand and deal with than regular Decision Trees. MofN type Decision Trees are more suitable for MofN type problems.

We developed a new Decision Tree extraction technique (DecText). DecText extracts classical Decision Trees from trained Neural Networks and prunes the tree in a way to maximize the fidelity between the tree and the Neural Network. We also introduced a new discretization method for making DecText be able to handle continuous features and a pruning method to minimize tree size while trying to maximize fidelity..

We introduced 4 new splitting techniques for extracting Decision Trees from Neural Networks. SetZero split works better than all when there are highly relevant features in the problem. SSE and ClassDiff work better in other cases. Fidelity split does not perform as good as SSE or ClassDiff most of the time. We also compared the results of splitting techniques with entropy gain split. We showed that most of the time the new splitting techniques perform better than gain split. The reason entropy gain split and also fidelity split do not perform as good as the others is that entropy gain split and fidelity split use only classification for each instance. They do not take into account the values of the outputs of NN. Output values of NN give us very valuable information about the network. For entropy gain split outputs 0.1, 0.9 and 0.49, 0.51 are the same because in both classification is the second class. But the first one is a much better classification then the second one.

References

[1] L. Atlas, R. Cole, J. Connor, M. El-Sharkawi, R. J. Marks, V. Muthusamy, and E. Barnard. Performance comparisons between backpropagation networks and classification trees on three real-world applications. In *Advances in Neural Information Processing Systems*, volume 2, pages 622–629, 1990.

[2] W. Buntine and T. Niblett. A further comparison of splitting rules for decision-tree induction. *Machine Learning*, 8:75, 1992.

[3] M. W. Craven. *Extracting Comprehensible Models from Trained Neural Networks*. PhD thesis, Department of Computer Sciences, University of Wisconsin-Madison, 1996. (Also appears as UW Technical Report CS-TR-96-1326).

[4] M. W. Craven and J. W. Shavlik. Extracting tree-structured representations of trained networks. In *Advances in Neural Information Processing Systems*, volume 8, pages 24–30, Denver, CO, 1996. MIT Press.

[5] R. Detrano, A. Janosi, W. Steinbrunn, M. Pfisetrer, J. Schmid, S. Sandhu, K. Guppy, S. Lee, and V. Froelicher. International application of a new probability algorithm for the diagnosis of coronary artery disease. *American Journal of Cardiology*, 64:304–410, 1987.

[6] D. Fisher and K. McKusick. An emprical comparison of id3 and back propagation. In *Proceedings of the eleventh international Joint Conference on Artificial Intelligence*, pages 788 – 793, Detroit, 1989. Morgan Kaufmann.

[7] P. Murphy and M. Pazzani. Id2-of-3: Constructive induction of n-of-m concepts for discriminators in decision trees. In *Proceedings of the Eighth International Machine learning Workshop*, pages 183–187, Evanston, IL, 1991. Morgan Kaufmann.

[8] J. C. Schlimmer and D. H. Fisher. A case study of incremental concept induction. In T. Kehler and S. Rosenschein, editors, *Proceedings of the Fifth National Conference on Artificial Intelligence, Philadelphia*, volume 1, pages 496–501, San Mateo, CA, 1986.

[9] J. W. Shavlik, R. J. Mooney, and G. G. Towell. Symbolic and neural learning algorithms: An experimental comparison. *Machine Learning*, 6:111–143, 1991.

[10] B. W. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman & Hall, London, 1986.