Extracting decision trees from trained neural networks

R. Krishnan*,[1], G. Sivakumar, P. Bhattacharya

Department of Computer Science and Engineering, Indian Institute of Technology, Powai, Mumbai 400 076, India

Abstract

In this paper we present a methodology for extracting decision trees from input data generated from trained neural networks instead of doing it directly from the data. A genetic algorithm is used to query the trained network and extract prototypes. A prototype selection mechanism is then used to select a subset of the prototypes. Finally, a standard induction method like ID3 or C5.0 is used to extract the decision tree. The extracted decision trees can be used to understand the working of the neural network besides performing classification. This method is able to extract different decision trees of high accuracy and comprehensibility from the trained neural network.

Keywords: Rule extraction; Decision trees; Data mining; Knowledge discovery; Classification

1. Introduction

Machine learning techniques have been applied to many real-life industrial problems with success [1]. Two of the commonly used techniques are artificial neural networks [2] and decision trees [3–5]. Artificial neural networks (ANNs) have empirically been shown to *generalize* well on several machine learning problems. Reasonably satisfactory answers to the questions like how many examples are needed for the neural network to learn a concept and what is the best neural network architecture for a particular problem domain (given a fixed number of training examples) are available in the form of a *learning theory* [6], so it is now possible to train neural networks without guesswork. This makes them an excellent tool for *data mining* [7], where the focus is to learn data relationships from huge databases. However, there are applications like credit approval and medical diagnosis where explaining the reasoning of the neural network is important. The major criticism against neural networks in such domains is that the decision making process of neural networks is difficult to understand [8]. This is because the knowledge in the neural network is stored as real-valued parameters (weights and biases) of the network, the knowledge is encoded in distributed fashion and the mapping learnt by the network could be non-linear as well as non-monotonic. One may wonder why neural networks should be used when comprehensibility is an important issue. The reason is that predictive accuracy is also very important and neural networks have an appropriate *inductive bias* for many machine learning domains. The predictive accuracies obtained with neural networks are often significantly higher than those obtained with other learning paradigms, particularly decision trees.

Decision trees have been preferred when a good understanding of the decision process is essential such as in medical diagnosis. Decision tree algorithms execute fast, are able to handle a high number of records with a high number of fields with predictable response times, handle both symbolic and numerical data well and are better understood and can easily be translated into *if–then–else*

rules. However there are a few shortcomings of decision tree algorithms [9].

1. Tree induction algorithms are unstable, i.e. addition or deletion of a few samples can make the tree induction algorithm yield a radically different tree. This is because the partitioning features (splits) are chosen based on the sample statistics and even a few samples are capable of changing the sample statistics. The split selection (i.e. selection of the next attribute to be tested) is greedy. Once selected, there is no backtracking in the search. So the tree induction algorithms are subject to all the risks of hillclimbing algorithms, mainly that of converging to locally optimal solutions.
2. The size of a decision tree (the total number of internal nodes and leaves) depends on the complexity of the concept being learnt, the sample statistics, noise in the sample and the number of training examples. It is difficult to control the size of the decision trees extracted and sometimes very large trees are generated making comprehensibility difficult. Most decision tree algorithms employ a *pruning* mechanism [10] to reduce the decision tree size. However, pruning may sometimes reduce the generalization accuracy of the tree.
3. Only one hypothesis is finally presented to the user. There may be advantages in maintaining all consistent hypotheses which such algorithms may miss out. Presenting different decision trees to the user is desirable as he will be in a position to choose the one where he is most familiar with the domain attributes or attributes which make the data gathering easy. Although most decision tree algorithms seem to have a bias towards generating smaller trees, they are not guaranteed to yield the smallest possible tree consistent with the training data. In fact, this is an NP-hard problem [11]. Also, it has been found [12] that smaller consistent decision trees are on an average less accurate than slightly larger trees. So a method that can give decision trees of different sizes is preferable. Besides the user may not mind a slightly bigger decision tree if it contains decision variables he understands.

In the machine learning literature, neural networks and decision trees have often been viewed as competing and incompatible paradigms. There have been numerous papers *comparing* them on different machine learning domains [13,14]. The conclusions have more often favored neural networks as the better technique. This is because neural networks can learn any input–output mapping (the *universal approximation property*), while decision trees can learn concepts describable by *hyperrectangles*. However, Quinlan [15] has pointed out in a study that neural networks seem more suited to problems that require knowledge of many input variables simultaneously whereas decision trees seem to do better when the decision making involves looking at each input variable in sequence (one followed by the other).

The learning mechanisms of neural networks and decision trees are quite different and incompatible, hence it has proved difficult to combine the merits of both. Recent work in Rule Extraction from neural networks [16–19] suggests that it is possible to get comprehensible representations of the knowledge stored in neural networks. In this paper, we present a methodology to capture the best of both worlds, i.e. the accuracy of neural networks and the comprehensibility of decision trees. Our proposed method is able to (i) to provide an alternate mechanism for generating decision trees that utilizes trained neural networks, (ii) to enhance comprehensibility of trained neural networks by extracting a decision tree which is close to the neural network in functionality, and (iii) to provide a mechanism for generating more than one decision tree from the data. Our method can be used to obtain a range of decision trees with varying levels of comprehensibility and accuracy. The extracted decision tree can be used to either understand the working of the neural network or as a classifier in its own right.

The organization of the rest of the paper is as follows. In Section 2, we explain the proposed method. In Section 3, we evaluate our method empirically on three well-known machine learning domains. In Section 4, we discuss related work and in Section 5, we present the conclusions.

2. The proposed method

Our method is motivated by the Rule Extraction as Learning paradigm of Craven and Shavlik [18]. In this paradigm, the task of extracting knowledge from a trained neural network is viewed as an inductive learning problem. The target concept we are trying to learn is the function represented by the trained neural network. The trained neural network is viewed as a black-box, i.e. one is not allowed to know the internals of the neural network architecture. The only interface permitted with the neural network is presenting a input vector and obtaining the network's classification. By observing the network's classification on a large number of inputs we hope to understand the behavior of the network.

The basic idea behind our method is to extract decision trees from inputs generated from the trained neural network instead of doing it directly from the data. Our method has three main steps. In the first step, we aim to understand what are the kind of inputs for which the trained neural network gives a desired output classification. By obtaining a large number of such inputs, we would be able to utilize the benefits of neural network training like robustness to noise, stable learning in the presence of outliers in the data and better generalization. We call such inputs *prototypes* for the class. In the second step we aim to select the best out of these prototypes for

inducing the decision tree. There are two motivations for this. The first is that decision trees induced from a smaller dataset tend to be smaller. Secondly, the training set might contain useful information that could be used to constrain the prototypes. In the final step, we induce a decision tree from the selected prototypes. We now explain each of these steps in detail.

2.1. Generation of prototypes

Given a trained network and a desired output vector, a *prototype* is an input vector that is unambigously classified by the neural network. In this case we can use *queries* for generating prototypes because the trained neural network can act as an *oracle* for obtaining the classification corresponding to any input vector presented to it. The advantage of learning with queries is that they can be used to search specific regions of high information content. In our method, we ask *membership queries* [20] to the neural network. In a membership query, we present an input vector to the neural network obtain the network's classification for it. The process of classification in a trained network involves only matrix multiplications, hence it is fast and efficient. Based on the network's classification, we decide whether the sample vector could have come from one of the original classes, i.e. if it is a prototype input. The hope is that if we can collect a large number of prototypes for each output class of the network, they would contain sufficient information about what the network has learnt and hence could be used to infer what the network has learnt.

One candidate for obtaining prototypes is by using network inversion algorithms [21]. However, such algorithms have long convergence times, besides at the end of the inversion only one prototype vector is obtained. We require a querying mechanism that should be able to search large regions of the input space for the prototypes, should be able to integrate and make use of information gained from earlier queries while asking its next query and should be *anytime*, i.e. if the querying process is stopped arbitrarily, it should have a set of answers at that point. Hence, we have adapted a genetic algorithm (GA) based prototype extraction method proposed by Eberhart [22] for our use.

In this method the GA is used as a search mechanism. The input to the GA are vectors of the same size as the neural network input vector. The inputs are random but conform to the range of the original inputs. The size of the GA population can be chosen to depend upon the dimensionality of the neural network training data as well as the training set size. To obtain the prototype vectors, first we fix a desired output vector. The GA is run so that the fitness value returned by the evaluation function (which in this case is just the feedforward network calculation) is minimum. In the implementation of the GA we have used, the most desirable candidates from the population are those having low fitness values. The fitness function we have employed for the genetic algorithm is

$$fitness = \sum_{i=1}^{n} abs(ANN(\mathbf{x}_i) - \mathbf{t}_i),$$

where the suffix i denotes the ith bit of the output vector, $\mathbf{t}_i$ is the desired output vector and $ANN(\mathbf{x}_i)$ is the output vector obtained by presenting the input vector $\mathbf{x}_i$ to the trained neural network. The above fitness function assigns a lower fitness to that input whose output vector is close to the desired output vector. The crossover and mutation rates for the GA were set at 0.6 and 0.001, respectively.

With the above scheme for obtaining prototypes, two strategies can be envisaged to obtain prototypes. The first is to use the GA-based prototype algorithm to generate a large number of prototypes and then use a prototype selection mechanism to reduce the number of prototypes. The second is to incorporate a term in the GA fitness function to obtain dissimilar prototypes. We have chosen to implement the first strategy because we experimentally found that the second strategy requires longer convergence times.

2.2. Prototype selection

After the prototypes are extracted, the next decision is whether to induce the decision tree directly from the prototypes or perform a *prototype selection*. One objective of the prototype selection is to filter out those prototypes that are unlikely given the training set distribution and retain only those that satisfy the training set distribution. Although the GA generates prototypes conforming to the input data ranges, it could still generate *outliers*. This is because we are not considering interactions between the various components of the input vector.

If an a priori knowledge of the distribution of the data is available, it can be used in performing a distribution modeling. For instance, the data could have been generated by a well-understood process having a fixed (but unknown) mean and standard deviation. Maximum likelihood techniques [23] can be used for estimating the parameters of the distribution on such occasions. However, in most real-life applications, distributions are multimodal and clustered, also one does not usually have an a priori knowledge of the distribution. Hence, we decided to use a *non-parametric* distribution modeling technique in our method.

The other aim of the prototype selection is motivated by a recent study that empirically shows that the removing randomly selected training instances often results in trees that are substantially smaller and as accurate as those built on all available training instances [24]. This suggests that a method of selecting a subset of prototypes may be able to give decision trees of user-defined size.

However, since we are interested in extracting many decision trees, we wanted a classification model whose performance could be used to filter prototypes selectively. One option is the probabilistic neural network [25], which is a Parzen window classifier whose window width can be learnt. The other option is Kohonen's learning vector quantization (LVQ) [26].

2.2.1. Selecting prototypes using the probabilistic neural network

Parzen [27] introduced an estimator for estimating a univariate probability density function(PDF) from a random sample. This was extended by Cacoullus [28] to the multivariate case. The estimated density computed by both these methods converges asymptotically to the true density as the sample size increases. For a single population whose sample size is n, the estimated density function is

$$g(x) = \frac{1}{n\sigma} \sum_{i=0}^{n-1} W\left[\frac{x - x_i}{\sigma}\right]. \quad (1)$$

Here W is a weighting function that has large values for small distances between the unknown and the training sample and decreases rapidly to zero as the distance increases, and σ controls the width of the area of influence and is set to smaller values for larger sample sizes. A common choice for the weighting function is the *Gaussian* function. This is simply because the Gaussian function is a well-behaved function that satisfies the conditions required by Parzen's method and has also been found to do well in practice. For the multivariate case the estimator is

$$g(x) = \frac{1}{(2\pi)^{p/2}\sigma^p n} \sum_{i=0}^{n-1} \mathrm{e}^{-\|x - x_i\|^2/2\sigma^2}. \quad (2)$$

In the above equation, p is the dimensionality of the input vector and n is the number of data samples.

Probabilistic neural networks are the neural network implementation of the above method and was first discussed in [25]. They have fast training times and produce outputs with Bayes posterior probabilities. The learning in PNNs consists of finding an appropriate value of σ. This is very crucial because too small a value of σ causes the effect of neighbors in a cluster to get lost whereas a large σ blurs the population density. The method we use to optimize σ is by using *jackknifing*. This involves using all but one sample for training and one sample for testing. This provides an *unbiased* indication of the performance of the classifier. This process is used for all samples, each time using a different sample.

For prototype selection using the PNN, the following method is used.

1. Classify the prototypes using the PNN. This generates an output vector $\mathbf{o}_i$.
2. For each output vector $\mathbf{o}_i$ find o_1 and o_2 the maximum and second maximum values in the output vector.
3. if $o_1 - o_2 > threshold$ then select this prototype for the tree induction phase, else reject it.

By choosing various values of the thresholds, it is possible to control the number of prototypes filtered. Setting the threshold to a large value will cause a large number of prototypes to be rejected.

2.2.2. Selecting prototypes using learning vector quantization

Kohonen's learning vector quantization (LVQ) algorithm offers another way of filtering prototypes by providing data compression via selection of codebook vectors to represent a group of vectors. It differs from the PNN in that prototype selection is done by the choice of the codebook vectors, i.e. by the LVQ algorithm itself and not by a classification model. We have used the LVQ-PAK [29] for the experiments reported here.

2.3. The induction algorithm

We decided to use ID3 as the induction algorithm because of its simplicity and also because we wanted to be sure that the improvements we obtained were due to our method. In its simplest form, ID3 does no pruning. C4.5 (or its successor C5.0) or any other inducer can be used its place however. Building the decision tree involves recursive partitioning of the training examples. Each time a node is added to the tree, some subset of the input features are used to pick the logical test at that node. The feature that results in the maximum *information gain* is selected for the test at that node. The feature that results in the maximum gain is the one that minimizes the entropy function H:

$$H = \sum_i - p_i \log\, p_i,$$

where p_i is the probability of the ith class for that feature. After the test is picked, it is used to partition the examples and the process is continued recursively until each subset contains instances of one class or satisfies some statistical criterion. For the purpose of the experiments reported here, we have used the ID3 implementation with the MLC++ package [30] and an evaluation version of C5.0.[2] The algorithm for our proposed method is shown in Table 1. A flowchart of the algorithm is given in Fig. 1.

3. Empirical evaluation

We applied our method to generate decision trees from three machine learning datasets, the Iris dataset, the Wine

[2]Thanks to Dr. J.R. Quinlan for providing the evaluation version of C5.0

Table 1
The proposed method

1. Given a dataset, train an appropriate feedforward neural network using backpropagation. Test the network for generalization until satisfactory performance is achieved.
2. Query the trained network by using the GA-based querying technique. The output of the querying phase is a set of prototypical inputs near the decision boundaries of the network. The number of prototypes collected can be decided based on the size of the network and the training set size.
3. Use the ID3 induction algorithm to induce the decision tree using the prototypes obtained in step 2. Then test the decision tree on an independent test set. If performance is acceptable, then STOP, else GOTO step 4 or 5 depending upon whether the PNN or the LVQ is used. The measure of this performance could be either the size of the decision tree or its accuracy.
4. Use the probabilistic neural network to filter prototypes. Start with a low value of δ where δ is the difference between the maximum and the second maximum output activations of the output neurons. At each iteration increment δ by a fixed amount. Check if sufficient number of prototypes are obtained. If YES then GOTO step 3 else HALT.
5. Use the LVQ to generate desired number of codebook vectors from the prototypes. GOTO step 3.

dataset and the Wisconsin University Breast-Cancer dataset. These datasets were obtained from [31].

3.1. The Iris dataset

This data set contains 150 samples, three classes of 50 instances each of the Iris plant. Out of these 75 samples were used for training the network and 75 for testing. The classes are Setosa, Versicolor and Virginica. There are four features for each class, viz., sepal length, sepal width, petal length and petal width. A feedforward network with four input units, two hidden units and three output units was used. After training the network, the proposed method was run on the above network with 100 prototypes extracted for each class. The prototype selection phase was not used in this case as sufficiently accurate trees were obtained without it. The generalization accuracies (on the test set) were as follows:

- Neural network accuracy: 98 %.
- ID3 accuracy: 96%, tree size nine nodes, five leaves.
- Proposed method tree accuracy: 98%, tree size seven nodes, four leaves.

The decision trees generated by ID3 and our proposed method are shown in Figs. 2 and 3, respectively. Thus, for the Iris dataset, the proposed method is able to extract a smaller, more accurate decision tree as compared to directly inducing the decision tree from the data.

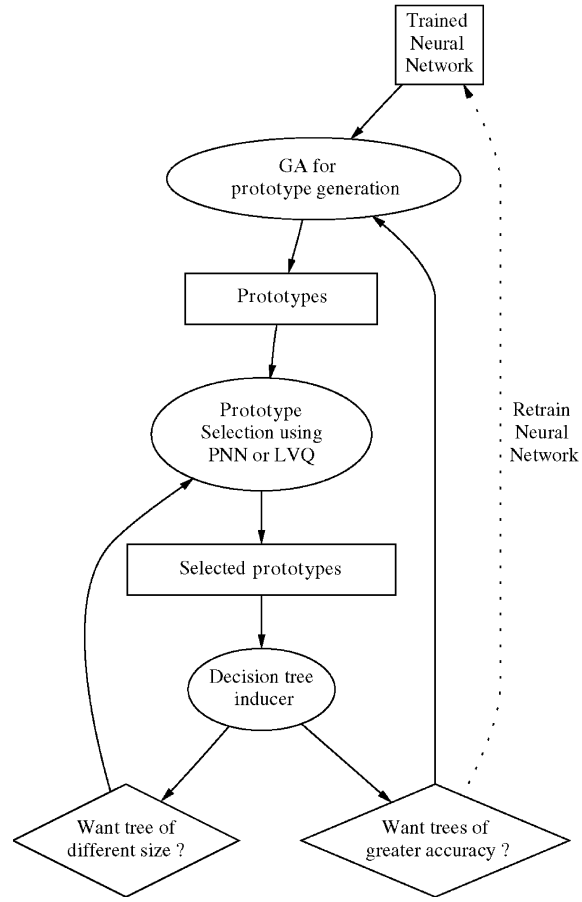

Fig. 1. The proposed algorithm.

3.2. The Wisconsin breast cancer dataset

This database has nine inputs and two outputs. The input features are clump thickness, uniformity of cell size, uniformity of cell shape, marginal adhesion, single epithelial cell size, bare nuclei, bland chromatin, normal nucleoli and mitoses. There are 699 training instances in the original dataset, however 16, of these had missing values and hence were deleted. Out of the remaining 683 training instances, 444 were labelled *Benign* and 239 as *Malignant*. These instances are divided into a training set of size 341 and a test set of size 342.

We trained a neural network with nine input units, two hidden units and two output units. We first used our method without the prototype selection phase on the above dataset. The number of prototypes extracted for each class was 1000. However, the accuracy of the decision tree obtained was comparably less than that of the neural network trained on the same data. Hence it

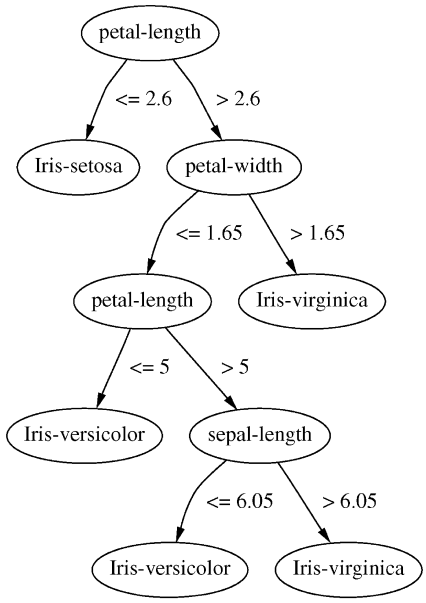

Fig. 2. Decision tree for Iris data using ID3 (nine) nodes, five leaves. 96% accuracy).

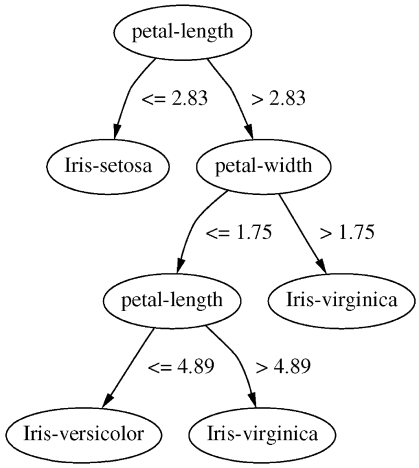

Fig. 3. Decision tree for Iris data using the proposed method (seven nodes, four leaves, 98% accuracy).

was decided to use the prototype selection phase for this dataset. We extracted a number of decision trees by varying the PNN δ from 0.01 to 0.30. The tree size (number of nodes + leaves) almost scaled linearly with δ. This gives a method for controlling the size of the tree. We also studied the effect of using the LVQ for selecting the prototypes. For comparison purposes, the number of LVQ codebook vectors was kept the same as the number of prototypes selected by the PNN. The accuracy of the trees obtained using the PNN and LVQ prototypes is shown in Fig. 7. It can be seen that the LVQ modeling yields more accurate trees than the PNN modeling. The main difference between the two modeling schemes is that the PNN modeling makes use of the training set (and is hence susceptible to influence of outliers in the training set) while the LVQ modeling does not make use of the training set. The results were as follows:

- Neural network accuracy: 97%.
- ID3 Decision tree accuracy: 96.8%, 39 nodes, 20 leaves. This tree though accurate is too big to comprehend.
- Proposed method (using PNN modeling): Best tree had an accuracy of 94.74%, 3 nodes, 2 leaves.
- Proposed method (using LVQ modeling): Best tree had an accuracy of 95.91%, 13 nodes, 7 leaves.

All the decision trees for this dataset also a high fidelity (above 90%) where fidelity is defined as the percentage of samples on which the neural network and decision tree show the same classification. Thus this experiment confirms that the proposed method is able to extract accurate trees in domains with high input dimensionality. Some of the decision trees extracted using the proposed method for this dataset are shown in Figs. 4–6.

3.3. *The wine dataset*

The purpose of experiments on this dataset is to study the effect of the various GA parameters on the quality of the prototypes generated. This dataset is the result of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars. The analysis determined the quantities of 13 constituents found in each of the three types of wines. The dataset has 13 dimensional continous inputs and has 178 instances, out of which 119 were used as training instances and the remaining as test instances. The number of instances in each class is: class 1 – 59(40, 19), class 2 – 71(47, 24), class 3 – 48 (32,16). The figures in brackets indicate the partitioning of the data into training and test sets. We have chosen this dataset to perform the following

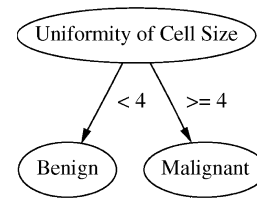

Fig. 4. Decision tree for cancer data using the proposed method (three nodes. two leaves, 94.74% accuracy).

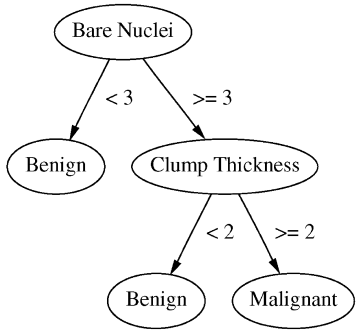

Fig. 5. Decision tree for cancer data using the proposed method (five nodes, three leaves, 93.27% accuracy).

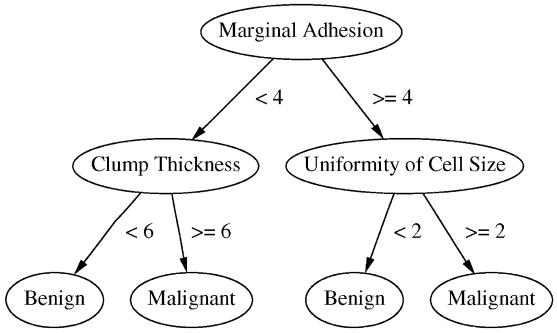

Fig. 6. Decision tree for cancer data using the proposed method (seven nodes, four leaves, 93.57% accuracy).

experiments. In all the experiments, the crossover rate of the GA was 0.6 and the mutation rate was 0.001. In the first experiment, we study the effect of varying the number of prototypes on the accuracy of the ID3 and C5.0 induction algorithms. Prototypes were generated across five runs using a different random seed for the GA each time. The best performance of ID3/C5.0 with reference to the number of prototypes is shown in Fig. 8. The corresponding decision tree sizes (i.e. number of leaves in the tree) is shown in Fig. 9. From these plots, it can be seen that the decision tree accuracy increases as the number of prototypes is increased. However the size of the tree also increases, hence the number of prototypes cannot be increased indefinitely if we want comprehensible trees.

In the next experiment, we studied the effect of changing the GA population on the accuracy and size of the ID3/C5.0 decision trees. The number of trials was kept constant. The results are shown in Figs. 10 and 11. Although there is no discernable relationship between the population size and the decision tree accuracy in this case, the size of the tree shows an increasing trend with increase in population size. The probable reason for this is the increased diversity that gets introduced into the prototypes with an increase in population size.

4. Related work

In this section, we review related methods. Tickle et al. [32] have proposed a method called DEDEC (Decision

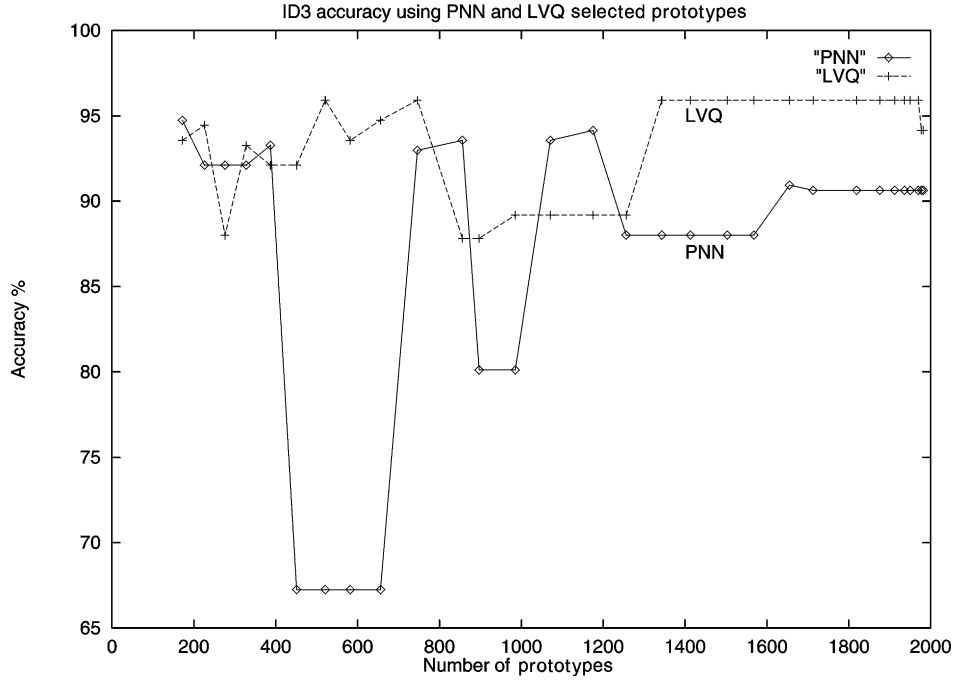

Fig. 7. Comparative accuracies of ID3 generated decision trees for PNN and LVQ selected prototypes on the cancer data.

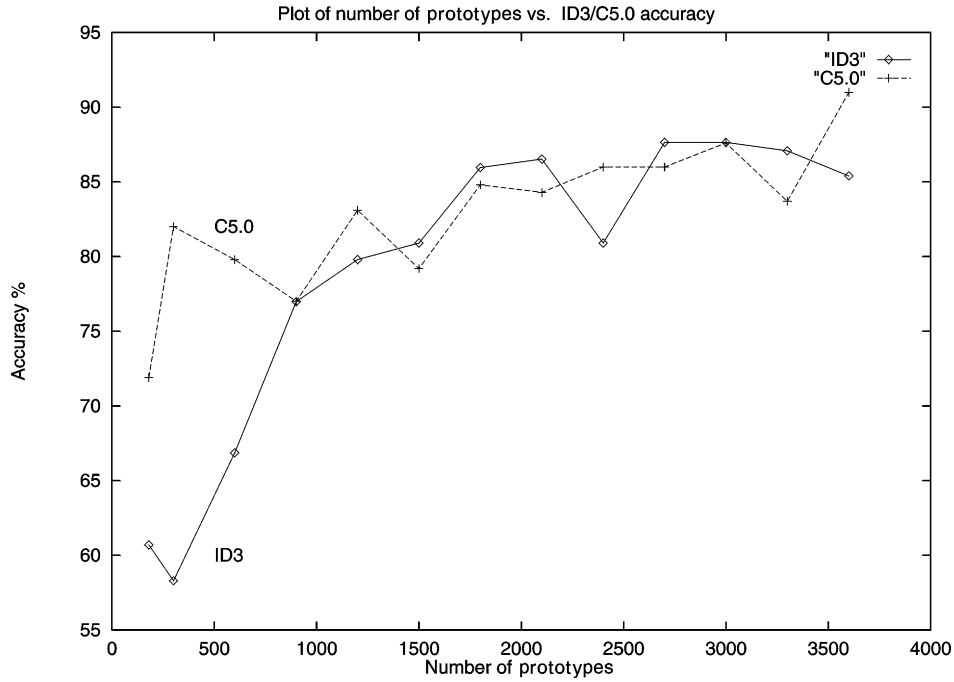

Fig. 8. Comparative accuracies of ID3 and C5.0 for varying number of GA prototypes on the wine data.

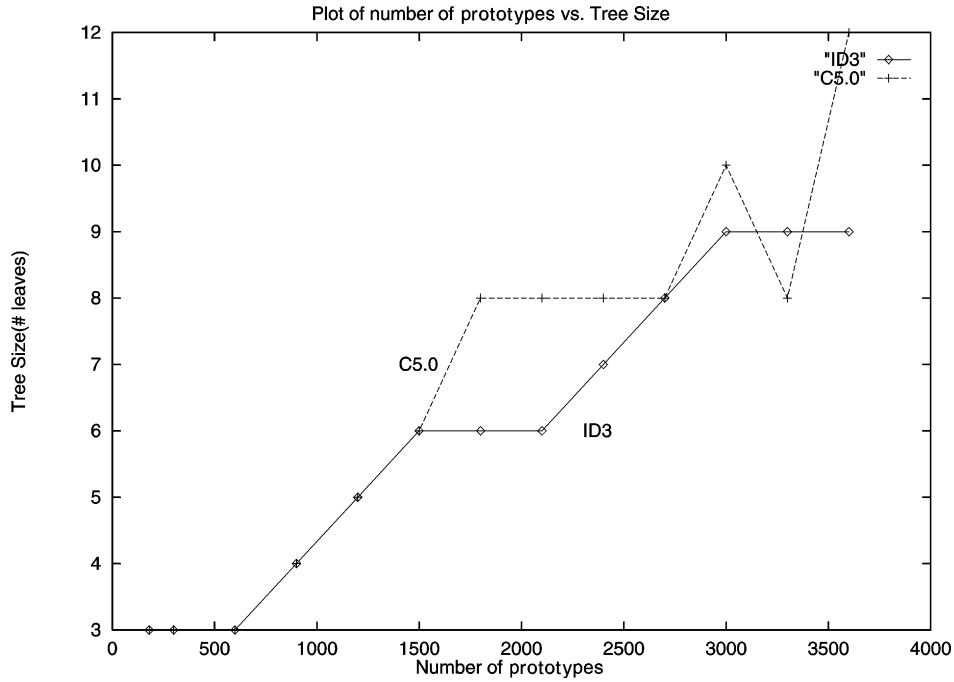

Fig. 9. Comparative tree sizes of ID3 and C5.0 for varying number of GA prototypes on the wine data.

Detection by Rule Extraction from Neural Networks). The *cascade correlation* algorithm is used to train the network. DEDEC uses information extracted by analyzing the weight vectors of the trained ANN to rank the input variables in terms of their relative importance. Examples are then generated for this input and a symbolic induction algorithm is used to generate the rules. This method is able to deal with only boolean or discrete inputs.

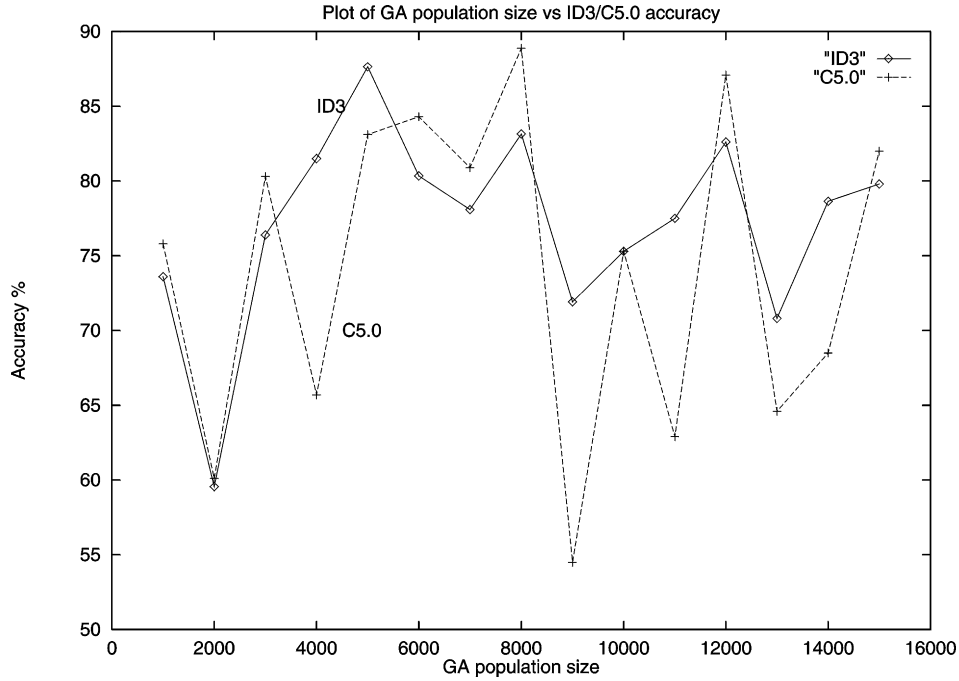

Fig. 10. Comparative tree sizes of ID3 and C5.0 for varying number of GA population sizes on the wine data.

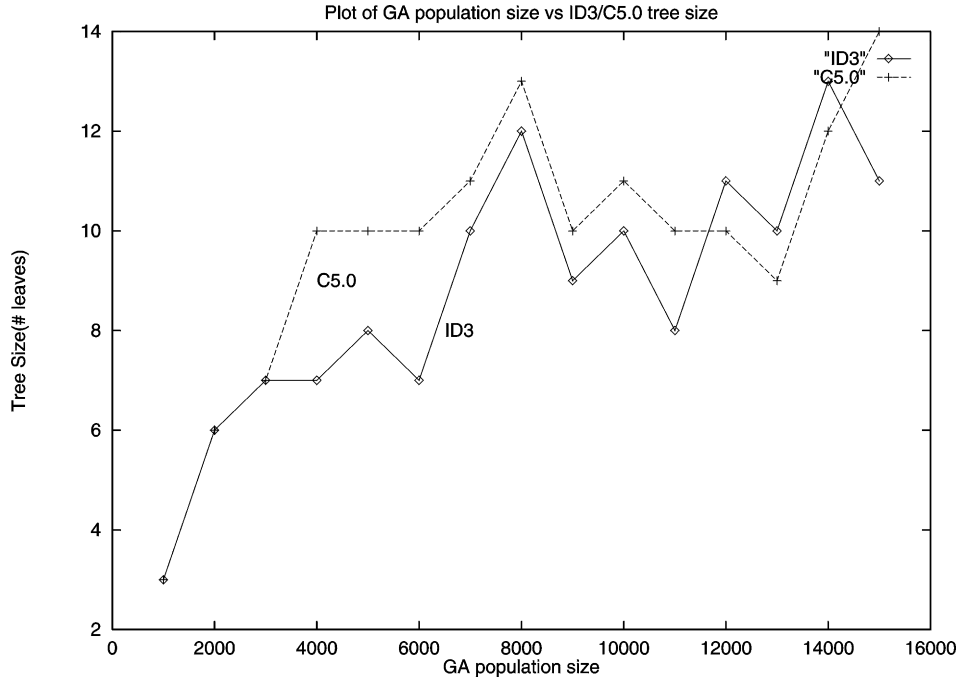

Fig. 11. Comparative tree sizes of ID3 and C5.0 for varying number of GA prototypes on the wine data.

One of the most recent and more sophisticated algorithms is the TREPAN algorithm [19]. TREPAN builds decision trees by recursively partitioning the instance space. However, unlike other decision tree algorithms where the amount of training data used to select splitting tests and label leaves decreases with the depth of the tree, TREPAN uses membership queries to generate additional training data. For drawing the query instances,

TREPAN uses *empirical distributions* to model discrete valued features and *kernel density estimates* to model continous features. The following are some of the main differences between TREPAN and our proposed method. Firstly, TREPAN assumes that the training data for the network is available. Although this is a reasonable assumption, there may be cases where a neural network which forms a subsystem of a larger software system may have been trained on proprietary data. For instance, the data may have been obtained from an extensive market survey or financial analysis. The proposed method can work independent of the database. This could also prove useful in working with large databases. Secondly, TREPAN compares the marginal distributions for each feature separately. Such a modeling of the data distributions does not take dependencies among features into account. Since our method constructs the entire query vector in one go, the dependencies among the features are taken into consideration. Thirdly, TREPAN does a narrow form of active learning in its sampling strategy. Since the instances are sampled based on the currently considered feature for the node splitting, it is difficult to ensure that most parts of the input space have been sampled. Since our method incorporates a more *global* search mechanism in the form of a GA, the above problem is less likely to occur. Finally, TREPAN extracts only one decision tree from the network. In many applications, it could be more desirable to extract a range of decision trees with varying sizes (and hence comprehensibility) and accuracy and leave it to the user to make the decision which tree he wants to use. In contrast, our method is able to extract many decision trees of varying size.

5. Conclusions

In this paper, we have proposed a method for extracting decision trees from trained neural networks instead of extracting them directly from data. The proposed method is independent of the architecture of the trained network. In fact, it can be used to understand any black box that can be queried.

We have used a genetic algorithm to present membership queries to the trained neural network and obtain prototypes. We have then suggested a methodology to contol the size of the decision tree based on selecting a subset of the generated prototypes. We have applied two non-parametric distribution modeling techniques, the probabilistic neural network and learning vector quantization for the prototype subset selection. Decision trees of varying sizes have been obtained using the proposed method on popular machine learning datasets and accurate and comprehensible trees have been obtained.

Although our method has yielded good results on the domains tested, the following improvements should be possible. Firstly, we have chosen the genetic algorithm as the query mechanism for the neural network. Other methods of query construction could be used to improve upon the quality of prototypes obtained, for example, the *selective sampling* method of Cohn et al. [33]. Secondly, for domains with a large number of inputs the genetic algorithm population would have to be very large to sample the domain effectively. Our method will benefit by using a suitable *feature subset selection* mechanism to reduce the number of input features.

References

[1] P. Langley, H.A. Simon, Applications of machine learning and rule induction, Commun. ACM 38 (11) (1995) 55–64.

[2] S. Haykin, Neural Networks : A Comprehensive Foundation, McMillan, New York, 1994.

[3] J. Quinlan, Induction of decision trees, Mach. Learning (1986) 81–106.

[4] J. Quinlan, C4.5: Programs for Machine Learning, Morgan Kaufmann, San Mateo, CA, 1993.

[5] L. Breiman, J. Friedman, R. Olshen, C. Stone, Classification and Regression Trees, Wadsworth and Brooks, Monterey, CA, 1984.

[6] M. Vidyasagar, A Theory of Learning and Generalization with Applications to Neural Networks and Control Systems, Springer, London, 1996.

[7] U. Fayyad, R. Uthurusamy, Data Mining and Knowledge Discovery in Databases, Commun. ACM 39 (11) (1996) 24–26.

[8] R. Andrews, J. Diederich, A.B. Tickle, A survey and critique of techniques for extracting rules from trained artificial neural networks, Technical Report, Neurocomputing Research Centre, Queensland University, Australia, 1995.

[9] T.G. Dietterich, Editorial in Mach. Learning 24 (1996) 1–3.

[10] F. Esposito, D.J. Marchette, G.W. Rogers, A comparative analysis of methods for pruning decision trees, IEEE Trans. Pattern Anal. Mach. Intell. 19 (5) (1997) 476–491.

[11] L. Hyafil, R.L. Rivest, Constructing optimal binary trees is NP-complete, Inform. Process. Lett. 5 (1) (1976) 15–17.

[12] P.M. Murphy, M.J. Pazzani, Exploring the decision forest: an empirical investigation of Occam's Razor in decision tree induction, J. Artif. Intell. Res. 1 (1) (1994) 257–275.

[13] T.G. Dietterich, H. Hild, G. Bakiri, A comparison of ID3 and backpropagation for English text-to-speech mapping, Mach. Learning 18 (1995) 51–80.

[14] J.W. Shavlik, R. Mooney, G.G. Towell, Symbolic and neural learning algorithms: an experimental comparison, Mach. Learning 6 (1991) 111–143.

[15] J. Quinlan, Comparing connectionist and symbolic learning methods, in: S.J. Hanson, G.A. Drastal, R.L. Rivest (Eds.), Computational Learning Theory and Natural Learning Systems, vol. 1, MIT Press, Cambridge, MA, 1994, pp. 445–456.

[16] L.M. Fu, Rule generation from neural networks, IEEE Trans. Systems Man Cybernet. 24 (7) (1994) 1114–1124.

[17] G. Towell, J. Shavlik, Extracting refined rules from knowledge-based neural networks, Mach. Learning 13 (1993) 71–101.

[18] M.W. Craven, J.W. Shavlik, Using sampling and queries to extract rules from trained neural networks, in: Proc. 11th Int. Conf. on Machine Learning, San Fransisco, 1994.
[19] M.W. Craven, Extracting comprehensible models from trained neural networks, Ph.D. Thesis, University of Wiconsin, Madison, 1996.
[20] D. Angluin, Queries and concept learning, Mach. Learning 2 (1988) 319–342.
[21] M.L. Rossen, Closed form inversion of backpropagation networks: theory and optimization issues, in: Advances in Neural Information Processing Systems, vol. 3, Morgan Kaufmann, San Mateo, CA, 1991, pp. 868–872.
[22] R.C. Eberhart, The role of genetic algorithms in neural network query-based learning and explanation facilities, in: Proc. Int. Workshop on Combinations of Genetic Algorithms and Neural Networks, Baltimore, 1992.
[23] R.O. Duda, P.E. Hart, Pattern Classification and Scene Analysis, Wiley, New York, 1973.
[24] T. Oates, D. Jensen, The effects of training set size on decision tree complexity, in: Proc. 14th Int. Conf. on Machine Learning, 1997.
[25] D. Specht, Probabilistic neural networks, Neural Networks 3 (1990) 109–118.
[26] Teuvo Kohonen, Self-Organizing Maps, Springer, Berlin 1995.
[27] E. Parzen, On estimation of a probability density function and mode, Ann. Math. Statist. 33 (1962) 1065–1076.
[28] T. Cacoullos, Estimation of multivariate density, Ann. Inst. Statist. Math. Tokyo 18 (2) (1966) 179–189.
[29] Teuvo Kohonen, Jari Kangas, Jorma Laaksonen, Kari Torkkola, LVQ-PAK: a program package for the correct application of learning vector quantization algorithms, in: Proc. Int. Joint Conf. on Neural Networks, IEEE press, Baltimore, 1992, pp. 725–730.
[30] Ronny Kohavi, G. John, R. Long, D. Manley, K. Pfleger, MLC ++ : A Machine Learning Library in C ++ , in Tools with Artificial Intelligence, IEEE Computer Society Press, 1994, pp. 740–743. The software MLC ++ is available by anonymous FTP to starry.stanford.edu in /pub/ronnyk/mlc directory.
[31] University of California, Irvine Repository of Machine Learning databases, obtainable by anonymous FTP to ftp.ics.uci.edu in the /pub/machine-learning-databases directory.
[32] A. Tickle, M. Orlowski, J. Diederich, DEDEC: a methodology for extracting rules from trained artificial neural networks, in: R. Andrews, J. Diederich (Eds.), Rules and Networks, Queensland University of Technology, Australia, 1996.
[33] D. Cohn, L. Atlas, R. Ladner, Improving generalization with active learning, Mach. Learning 15 (2) (1994) 201–221.