Decision Tree Pruning Using Backpropagation Neural Networks

BOONSERM KIJSIRIKUL[1] and KONGSAK CHONGKASEMWONGSE[2]
Department of Computer Engineering, Chulalongkorn University,
Phathumwan, Bangkok, 10330, Thailand
email: boonserm[1], g41kck[2]@mind.cp.eng.chula.ac.th

Abstract

Neural networks have been widely applied to various tasks, such as handwritten character recognition, autonomous robot driving, determining the consensus base in DNA sequences. In this paper, we describe the use of backpropagation neural networks for pruning decision trees. Decision tree pruning is indispensable for making the overfitting trees more accurate in classifying unseen data. In decision trees, the overfitting can occur when the size of the tree is too large compared to the number of training data. Many methods for decision tree pruning have been proposed, and all of them remove some nodes from the tree to reduce its size. However, some removed nodes may have a significance level or some contribution in classifying new data. Therefore, instead of absolutely removing nodes, our proposed method employs a backpropagation neural network to give weights to nodes according to their significance. Experimental results on twenty domains demonstrate that our method outperforms error-based pruning.

1 Introduction

Neural networks have been widely applied to various tasks, such as handwritten character recognition [4], autonomous robot driving [8], determining the consensus base in DNA sequences [1]. In this paper, we describe the use of backpropagation neural networks for preventing overfitting in decision tree learning. Overfitting is a widely observed phenomenon that often occurs when hypotheses are as complex as the training data and when the data contains noise. In decision trees, the overfitting phenomenon can occur when the size of the tree is too large compared to the number of training data. It has been shown that the overfitting trees do not perform well on new data. There are two broad classes of techniques for preventing the overfitting, i.e. (1) approaches that stop growing the tree earlier, before it reaches the point where it perfectly classifies the training data, and (2) approaches that allow the tree to overfit the data, and then post-prune the tree. Between these approaches, the second approaches of pruning trees have been found to be more successful in practice [7].

Many methods for decision tree pruning have been proposed, including cost-complexity pruning [3], reduced error pruning and pessimistic pruning [9], error-based pruning [10], MDL-based pruning [5], pruning with misclassification costs [2], etc. While these methods can differ in several ways, they share the same basic technique of *completely* removing some nodes from the tree to reduce its size. However, some removed nodes may have a significance level or some contribution in classifying new data. Thus these methods are unable to make use of these kind of nodes.

This paper presents a novel method for decision tree pruning, called *soft-pruning*. Instead of absolutely removing nodes, our proposed method gives weights to nodes according to their significance. The significance level or the weight of a node is determined by a backpropagation neural network. We run experiments on twenty domains, in the UCI repository of machine learning databases [6], to compare our method with error-based pruning that is one of the most effective method for tree pruning and is employed by a well-known decision tree learner C4.5 [10]. The results demonstrate that our method performs significantly better on five domains, worse on one domain and equally well on the rest.

The paper is organized as follows. Section 2 briefly describes the method of C4.5 for inducing and pruning decision trees. Section 3 presents our method for soft-pruning decision trees. Section 4 describes the experiments and reports the results of soft-pruning compared to those of C4.5's pruned and unpruned trees. Finally, the conclusion is given in Section 5.

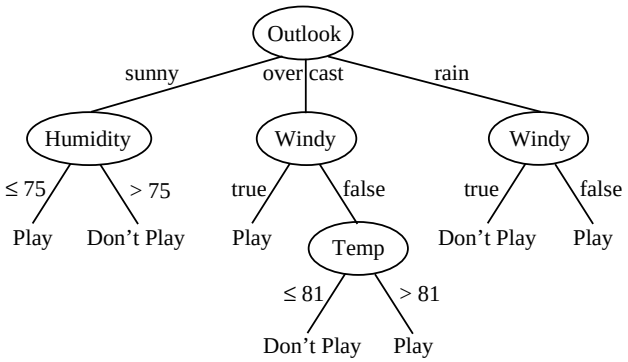

Figure 1: An example of a decision tree.

Table 1: Training examples for the concept *PlayTennis*.

Outlook	Temp(°*F*)	Humidity(%)	Windy?	Class
sunny	75	70	true	Play
sunny	80	90	true	Don't Play
sunny	85	85	false	Don't Play
sunny	72	95	false	Don't Play
sunny	69	70	false	Play
overcast	72	90	true	Play
overcast	83	78	false	Play
overcast	64	65	true	Play
overcast	81	75	false	Don't Play
rain	71	80	true	Don't Play
rain	65	70	true	Don't Play
rain	75	80	false	Play
rain	68	80	false	Play
rain	70	96	false	Play

2 Decision Tree Induction & Error-Based Pruning

In this paper, we employ the method of C4.5 for inducing decision trees. The obtained trees are then pruned by our method that will be described later in Section 3. In this section, we first briefly explain the method of C4.5 for inducing and pruning decision trees (see [10] for details).

A decision tree is a structure that contains *leaf* nodes and *decision* nodes. A leaf node indicates a class of examples, and a decision node specifies some test to be carried out on a single attribute value, with one branch and subtree for each possible outcome of the test. Figure 1 shows an example of a decision tree for the concept *PlayTennis*. In this example, there are seven leaf nodes, each of which belongs to one of two classes of examples, i.e. `Play` and `Don't Play`. There are five test nodes in this tree which are `Outlook`, `Humidity`, `Windy`, `Windy` and `Temp`. This tree is generated from training examples shown in Table 1 (table adapted from [10]).

To construct a tree, C4.5 employs the recursive partitioning method that continues to add a test node into the tree. If all examples in a branch belong to the same class, the subtree becomes a leaf node with the class as label. Otherwise, a test is chosen which separates the examples into at least two partitions according to the outcome of the test. This process continues until each subset in the partition contains examples of a single class. C4.5 uses the gain ratio criterion [10] to select the best test node at each non-leaf node.

To prune the tree, C4.5 employs the error-based pruning algorithm [10]. The algorithm is a kind of post-pruning algorithm; it allows the tree to overfit the examples and then post-prune the tree. The algorithm starts from the bottom of the tree and examines each non-leaf subtree. If replacement of this subtree with a leaf would lead to a lower estimated error, then prune the tree. The algorithm estimates the error of a leaf by computing a statistical confidence interval of the resubstitution error (error on the training set for the leaf) assuming an independent binomial model and selecting the upper bound of the confidence interval. The estimated error for a subtree is the sum of the errors for the leaf nodes underneath it. Because leaf nodes have fewer examples than their parents, their confidence interval is wider, possibly leading to larger estimated errors, hence they may be pruned [2]. For the tree in Figure 1, the process of C4.5's pruning gives a simplified tree shown in Figure 2.

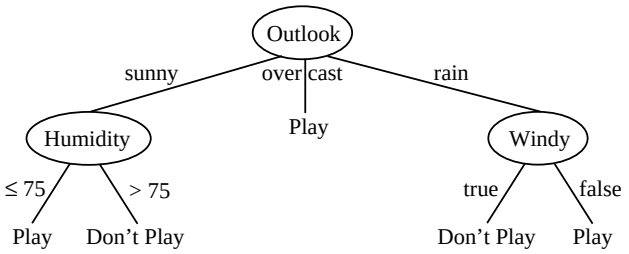

Figure 2: An example of a pruned tree.

3 Soft-Pruning

This section presents our method of soft-pruning decision trees. The method is composed of three steps, i.e. (1) converting a tree to rules, (2) constructing a neural network from the rules, and (3) training the network. To illustrate our method, consider a decision tree in Figure 1 which classifies data into two classes, i.e. `Play`

and `Don't Play`.

Given a decision tree, we first construct a set of rules by generating a rule for each leaf node in the tree. Each attribute test (e.g. `Outlook=sunny`) along the path from the root to the leaf becomes a rule antecedent and the classification at the leaf becomes the rule consequent. From the above tree, we obtain the rules shown in Table 2.

Table 2: A set of rules obtained from the decision tree in Figure 1.

(1) IF (Outlook=sunny) $\wedge$ (Humidity$\leq$75) THEN Play
(2) IF (Outlook=sunny) $\wedge$ (Humidity$>$75) THEN Don't Play
(3) IF (Outlook=overcast) $\wedge$ (Windy=true) THEN Play
(4) IF (Outlook=overcast) $\wedge$ (Windy=false)$\wedge$ (Temp$\leq$81) THEN Don't Play
(5) IF (Outlook=overcast) $\wedge$ (Windy=false)$\wedge$ (Temp$>$81) THEN Play
(6) IF (Outlook=rain) $\wedge$ (Windy=true) THEN Don't Play
(7) IF (Outlook=rain) $\wedge$ (Windy=false) THEN Play

The reason for converting the decision tree to a rule set before pruning is that converting to rules allows distinguishing among the different contexts in which a decision node is used. Because each distinct path through the tree node produces a distinct rule, the pruning decision regarding that attribute test can be made differently for each path. In contract, if the tree itself were pruned, the only two choices would be to remove the decision node completely, or to retain it in its original form. For example, if a test node x in one rule examining a particular property of examples has less significance when it is combined with another node y, it should be removed. However, if node x, without node y, in another rule plays an important role in classifying data, it should be retained.

Therefore, converting the tree to rules provides us with more flexibility in pruning. Moreover, we incorporate an additional technique for allowing more flexibility in pruning decision trees. Here, instead of *completely* pruning nodes, we give higher weights to important nodes and lower weights to unimportant ones. The significance level of each node is determined in terms of a weight trained by a backpropagation neural network (BNN) which will be described later.

Having a set of rules, we construct a neural network whose structure is determined as follows. The antecedents of a rule are used as input units that are linked to one hidden unit which represents the rule. Therefore, the number of hidden units in the network is the same as the number of rules. Each class is represented by one output unit of the network, and thus the number of output units is equal to the number of classes. The links from hidden units to output units are fully connected. Note that all hidden and output units include bias weights. All weights of all links and bias weights are trained by the standard backpropagation algorithm [11]. Figure 3 shows an example of a neural network constructed from the rules in Table 2.

To train the network, we first randomly initialize the weights of the network, and adjust the weights by the backpropagation algorithm. In our experiment, all units in the network use the sigmoid function. The training examples of the network are the same as those used to create the decision tree except that only examples that are correctly classified by the tree are used. In the training process, each training example is evaluated with all input units and the truth values of the input units are determined. The input units whose truth values are true are set to 1, whereas the units whose truth values are false are set to 0. The network is repetitively trained by using training examples until it converges or the number of training iterations exceeds the predefined threshold. After having been trained, the network can be used to classify unseen data. The unseen data is evaluated with input units as in the training process. The truth values of the input units are then fed into the network, and the output with highest value will be taken as the prediction.

In case of data with no missing value, the value of an input unit will be 1 or 0 according to its truth value. However, for data with missing value, we assign to the input unit the following value:

$$\frac{\textit{no. of training examples satisfying that input unit}}{\textit{no. of all training examples}}$$

The obtained value is between 0 and 1. This is similar to a technique for dealing with missing value data described in [10].

4 The Experimental Results

This section presents the results of soft-pruning decision trees. We compare our method with C4.5 [10]. For each dataset, we employ C4.5 to generate an unpruned decision tree and convert the tree to a rule set. Having a rule set, we then build our BNN as described above.

To evaluate our soft-pruning method, we ran experiments comparing the method with C4.5's unpruned

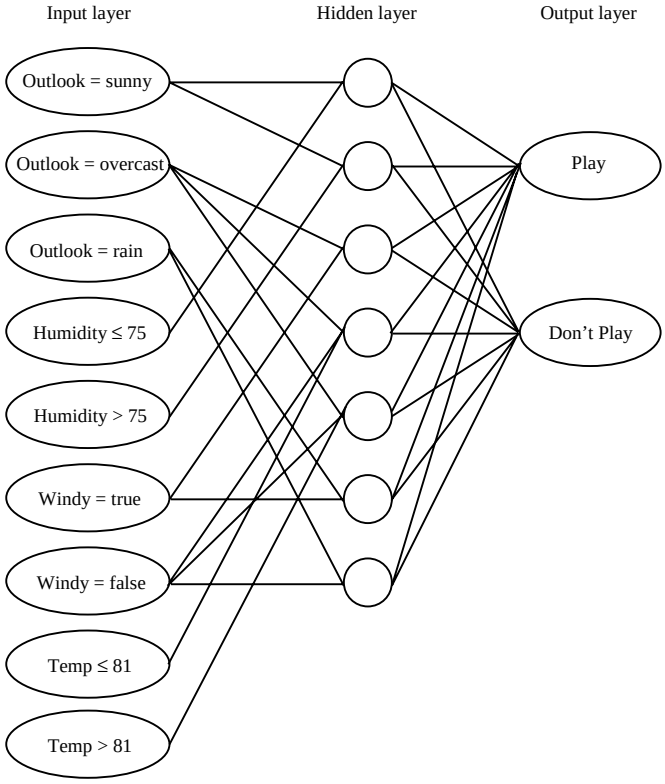

Figure 3: The structure of the neural network for the rule set in Table 2.

and pruned trees. Twenty datasets of multi-class learning domains from the UCI repository [Merz et al., 1997] were used. In case of datasets where training and test sets were already provided, the results were evaluated on the given test sets. In the other datasets providing no test set, the results were averaged using 6-fold cross-validation. In 6-fold cross validation, the data was partitioned into six disjoint subsets. Each subset was used as a test set once, and the remaining subsets were used as the training set.

Table 3 report the summary of the datasets and the classification accuracies of soft-pruning and C4.5. In the table, 6CV denotes the experiment that uses six-fold cross-validation. The fifth and sixth columns in the table show the accuracies of C4.5's unpruned and C4.5's pruned trees, respectively. The accuracies of soft-pruning are given in the last column. The row "Win-lose-tie (Soft-pruning - XXX)" in the table shows the performance comparison between soft-pruning and the corresponding systems; where XXX is the C4.5's unpruned tree or the C4.5's pruned tree. Superscripts denote confidence levels for the difference in accuracy between soft-pruning and the corresponding system, using a one-tailed paired t test: $+$ or $-$ is 90%, $++$ or $--$ is 95%, $+++$ or $---$ is 99%; no superscripts denote confidence levels that are below 90%($+$ indicates higher accuracy of soft-pruning, whereas $-$ indicates lower accuracy of soft-pruning). Note that the highest accuracy for each data set is shown in bold face.

The results show that soft-pruning was more accurate than C4.5's unpruned trees in 17 datasets, and less in only 1 dataset. Compared with C4.5's pruned trees, soft-pruning achieved a higher accuracy in 14 datasets and lower in 5 datasets. If we include confidence levels in the comparison, we can see that soft-pruning performs significantly better than both C4.5's unpruned and pruned trees on 5 datasets. Soft-pruning performed significantly worse than C4.5's pruned trees only on one dataset. These results demonstrate the usefulness of soft-pruning decision trees. The better results are due to more flexibility in pruning of our method: (1) converting a tree to rules before pruning will produce a non-leaf node of the tree in more than two rules, and thus this node can be separately soft-pruned according to its participation in the rules, and (2) a node that has some degree of significance for a rule will not be completely pruned, and will be assigned with an appropriate weight by the neural network.

Table 3: The percent accuracies of soft-pruning and C4.5.

Data Set	#Train	#Test	#Classes	C4.5 (Unpruned)	C4.5 (Pruned)	Soft-Pruning
Allbp	2800	972	3	96.81	**97.84**	97.43
Allhyper	2800	972	5	**98.87**	98.56	98.77
Allhypo	2800	972	5	99.49	99.49	**99.79**
Allrep	2800	972	4	98.77	**99.07**	98.97
Anneal	798	100	6	97.00	95.00	**98.00**
Balance-scale	625	6CV	3	69.76^{+++}	65.77^{+++}	**89.92**
Glass	214	6CV	6	66.34	66.34	**67.79**
Image	210	2100	7	89.43	**91.00**	89.90
Iris	150	6CV	3	**95.33**	94.00	**95.33**
LED	2000	500	10	75.40	75.20	**76.00**
LED 17	2000	500	10	66.40	$\mathbf{74.60}^{--}$	69.40
Lymphography	148	6CV	4	73.70	78.38	**78.39**
Primary-tumor	339	6CV	22	42.19	41.32^{++}	**45.15**
Satimage	4435	2000	6	84.90^{++}	85.45	**86.80**
Segment	2310	6CV	7	96.97	96.97	**97.19**
Shuttle	43500	14500	7	99.97^{+}	99.95^{++}	**99.99**
Soybean	307	376	19	85.64	**86.70**	86.44
Waveform	5000	6CV	3	75.76^{+++}	75.82^{+++}	**80.62**
Waveform+noise	5000	6CV	3	75.22^{+++}	75.30^{+++}	**80.56**
Wine	178	6CV	3	93.30	93.30	93.30
Average				84.06	84.54	86.49
Win-lose-tie(Soft-pruning - XXX)				17-1-2	14-5-1	

5 Conclusion

We have proposed a novel method for decision tree pruning, called soft-pruning. The main contribution of our work is the method of employing a backpropagation neural network to prune the decision tree. The results comparing soft-pruning and C4.5 show that soft-pruning performs better than standard C4.5's pruned and unpruned trees.

One disadvantage of soft-pruning is that it loses the ease of interpretability of the original trees. One of our future research is to transform the obtained network into a more understandable representation such as probabilistic rules.

References

[1] C. F. Allex, J. W. Shavlik, and F. R. Blattner, "Neural Network Input Representations that Produce Accurate Consensus Sequences from DNA Fragment Assemblies", *Bioinformatics*, 15, 1999, pp. 723-728.

[2] J. P. Bradford, C. Kunz, R. Kohavi, C. Brunk, and C. E. Brodley, "Pruning Decision Trees with Misclassification Costs", *Proceedings of the 10th European Conference on Machine Learning (ECML-98)*. Springer-Verlag, 1998.

[3] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*, Belmont, CA: Wadsworth, 1984.

[4] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation Applied to Handwritten Zip Code Recognition", *Neural Computation*, 1(4), 1989.

[5] M. Mehta, J. Rissanen, and R. Agrawal, "MDL-based Decision Tree Pruning", *Proceedings of the First International Conference on Knowledge Discovery and Data Mining*, AAAI Press, 1995, pp. 216-221.

[6] C. J. Merz, P. M. Murphy, and D. W. Aha, "UCI repository of machine learning databases", Department of Information and Computer Science, University of California, Irvine, CA. `http://www.ics.uci.edu/~mlearn/MLRep-ository.html`, 1997.

[7] T. Mitchell, *Machine Learning*, New York: McGraw-Hill, 1997.

[8] D. A. Pomerleau, "Knowledge-Based Training of Artificial Neural Networks for Autonomous Robot Driving", In J. H. Connel and S. Mahadevan (Eds.), *Robot Learning*, Boston: Kluwer Academic Publishers, 1993, pp. 19-43.

[9] J. R. Quinlan, "Simplifying Decision Trees", *International Journal of Man-Machine Studies*, 27, 1987, pp. 221-234.

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

[11] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Internal Representations by Error Propagation", In D. E. Rumelhart and J. L. McClelland (Eds.) *Parallel distributed processing*, 1, Cambridge, MA:MIT Press, 1986.