Improving Decision Tree Pruning through Automatic Programming

Stig-Erland Hansen
stig.e.hansen@hiof.no

Roland Olsson
roland.olsson@hiof.no

Abstract

Automatic Design of Algorithms through Evolution (ADATE) is a machine learning system for program synthesis with automatic invention of recursive help functions. It is well suited for automatic improvement of other machine learning algorithms since it is difficult to design such algorithms based on theory alone which means that experimental tuning, optimization and even design of such algorithms is essential for the machine learning practitioner.

To demonstrate the feasibility and usefulness of "learning how to learn" through program evolution, we used the ADATE system to automatically rewrite the code for the so-called error based pruning that is an important part of Quinlan's C4.5 decision tree learning algorithm.

We evaluated the resulting novel pruning algorithm on a variety of machine learning data sets from the UCI machine learning repository and found that it generates trees with seemingly better generalizing ability. The same meta-learning may be applied to most machine learning methods.

keywords: Automatic programming, decision tree learning, meta learning

1 Introduction

Classification is one of the most popular and well studied fields of machine learning. The problem is how to predict which class an instance belongs to based on its attributes, for example predicting whether a mushroom is poisonous by considering its color and height. Systems making such predictions are known as classifiers, and are learned, that is induced, using already seen data where the class of each instance is known. In other words, classifiers make predictions about the future based on the past.

Classifier inducers search for the best possible classifier for a particular data set. Their search space is virtually infinite in terms of the number of classifiers, which means that it is practically impossible to conduct an exhaustive search that is guaranteed to find an optimal classifier.

Therefore, the classifier inducers reduce the search space by favoring certain classifiers over others. This is known as the inductive bias of a classifier inducer.

It is the inductive bias and how well it fits a specific data set that determines the accuracy of the induced classifier. Thus, for a specific data set it is important to find the classification algorithm with the appropriate inductive bias. However, what if the

This paper was presented at the NIK-2007 conference; see http://www.nik.no/.

inductive bias could be adapted to the data set through a general method? Such a method could be applied to any classification algorithm and make specific improvements beneficial for a specific domain or a specific type of domain. In this paper, we show that the ADATE automatic programming system[1] is such a method for meta-learning.

There are many types of existing classifier inducers that would be possible to improve through automatic programming, but decision tree inducers have at least three beneficial properties, making them more suitable than other algorithms. First, decision tree inducers are heavily based on heuristics, suggesting that there is room for improvements. Since they have been very thoroughly researched, any generally valid improvements are a testament to the benefits of an automated experimental approach and the limitations of the human brain.

Second, the algorithmic complexity of decision tree inducers is relatively low and they are reasonably fast in both training and classification which reduces the overall computational requirement for meta-learning through automatic programming.

Third, decision trees have a central place in numerous other classification algorithms. For instance, there are rule inducers utilizing decision trees, model tree inducers which create decision trees with models at the leaves, and decision trees are often preferred in ensemble algorithms like bagging[2] and boosting[3]. As a result, potential improvements to a decision tree algorithm could be incorporated into other algorithms as well.

Decision tree learning is comprised of building and pruning. A decision tree is typically built by recursively selecting the most promising attribute and splitting the training set accordingly until all instances belong to the same class or all attributes have already been used. The role of pruning is to simplify the decision tree either during or after building.

It seems that pruning may have a greater potential for improvement than building. Breiman et al. observed relatively little difference between the type of decision tree produced when different methods for selecting the most promising attribute were used, and found it more important to choose the right pruning algorithm [4]. Therefore, we chose to improve decision tree pruning.

This article focuses on how ADATE can be used to improve decision tree pruning. Section 2 gives a short introduction to ADATE. Section 3 explains how the ADATE specification for pruning was written. Section 4 describes how the ADATE system was executed and how the final synthesized pruning algorithm was selected. Section 5 presents experimental results for the synthesized algorithm on synthetic and real world data sets and Section 6 outlines how ADATE can be employed to improve other classification algorithms.

2 Automatic Programming

Automatic programming, also known as program induction, is an emerging technology where either a reasonably small program or one or more parts of a potentially huge software system are automatically synthesized. For more information on automatic programming in general, please see the JMLR special issue [5] recently co-edited by one of us or the online notes for Ricardo Aler's tutorial at ICML-2006[6].

ADATE [1] is one of the leading systems for induction of programs and is able to induce recursive functional programs with automatic invention of recursive help functions as needed.

It could solve standard algorithmic problems such as sorting, deletion from binary search trees, generating all sublists of a given length for a given list and so on, more than a decade ago and has been steadily improved since then.

In addition to algorithmically sophisticated program transformations that generate new expressions, ADATE contains abstraction (ABSTR) that introduces new functions, and case/lambda-distribution/lifting (CASE-DIST) that manipulates the scope of variables and functions. Both of the latter two transformations preserve semantics and are not as combinatorially difficult as the expression synthesis transformations.

The main challenge when designing a program evolver such as ADATE is to reduce overall run time, which depends on the size s of the final program that is to be produced and the average execution time δ required to run a program on all training inputs.

In a previous paper [7], we showed that the total time required for program evolution in ADATE under certain assumptions is $O(\delta s^3)$. Therefore, the size s of the part of an overall system that is selected for automatic improvement should be kept small. However, by repeatedly selecting different parts of a possibly huge code, it is possible to improve all of it if the execution time δ is reasonably small.

The decision tree pruning experiments reported in this paper have $\delta \approx 0.5$s when running 960 training input trees on an Intel Pentium D 940 CPU core. The best synthesized program has a code size of 331 bits using ADATE's syntactic complexity measure which in this case corresponds to 73 lines of Standard ML code.

3 Pruning

The biggest problem in classifier induction is to overly adapt the classifier to specific details of the training set. In order to avoid such overfitting, decision tree learning employs pruning, which, as the name suggests, involves simplification of a decision tree by eliminating seemingly overfitted branches. There are two approaches to pruning, called pre-pruning and post-pruning.

Pre-pruning takes place during tree induction and tries to stop the induction process as soon as overfitting occurs. However, it suffers from the horizon effect[8] in that it cannot explore the result of allowing the induction process to continue, making premature and overdue stopping hard to avoid [4].

Post-pruning, on the other hand, takes place after the induction process has ended and tries to simplify the tree by removing superfluous nodes. In this way, it wastes more resources than pre-pruning, yet, at the same time, it is more robust in simplification of decision trees because it has access to the full tree, allowing thorough exploration of the impact of removing nodes. Since the increased robustness normally out-weighs the increased computation, post-pruning is preferred over pre-pruning. Therefore, we will focus on improving post-pruning in this article.

Existing Pruning Algorithms

Most post-pruning algorithms are implemented quite similarly. They navigate either top-down by starting at the top of the tree and moving downwards to the leaves, or bottom-up by starting at the bottom of the tree and moving upwards to the root. During navigation, the true error rate is estimated for the different parts of the tree to decide which should be retained and which should be pruned.

Although estimation of the true error rate varies considerably between pruning methods, they normally base their estimation either on the training set or on a distinct validation set. Since the validation set is separate, the measurements from it are not biased as opposed to the measurements from the training data. This may explain the observations made by Mingers in his comparison of pruning algorithms which lead him to conclude that the pruning algorithms utilizing a separate validation set produce more accurate trees[9].

However, in his study, Mingers did not take into account the lower amount of data available for training when a validation set is used, possibly resulting in a worse starting point. Another study performed by Esposito, Malerba and Semeraro took this into consideration and came to the opposite conclusion of Mingers[10].

We have chosen to study meta-learning for so-called error based pruning (EBP) which is the pruning algorithm used in Ross Quinlan's popular C4.5 decision tree system [11].

It traverses the tree bottom-up while choosing at each internal node whether to retain the node as it is, replace it with a leaf containing the majority class, or replace the node with the most frequently used child according to the training set, a process known as tree grafting. The operation with the lowest estimated error rate is performed.

Estimation of the true error rate of a leaf is based on some questionable statistical reasoning around the error rate observed in the training set. Assuming that the observed error rate is governed by a binomial distribution, the estimated error rate is the upper limit of a confidence interval with a specific confidence level, where 25% is the default [11]. The estimated error rate of a tree is the sum of estimated error rates of the children weighted by the fraction of the instances reaching each particular child.

Specification

The ADATE system requires a specification which presents the problem it should solve. This specification mainly consists of the following three parts.

The f Function.

This function contains the code that is to be rewritten through automatic programming. It is implemented in a small subset of the mostly functional programming language Standard ML and may be just a small part of a big code that is to be automatically improved.

In this paper, the `f` function provided in the specification is a partial implementation of EBP, where we omitted tree grafting and estimate the error rate naively, yielding inappropriate estimates when the number of errors is either close to zero or close to the number of instances. We started ADATE with this "naive EBP" to see if it would be automatically improved to an algorithm competitive with EBP or even superior to it.

The `f` function takes an unpruned decision tree as input, and returns a transformed decision tree along with an estimated error rate for the tree. Optimally, the `f` function should transform an unpruned decision tree to the simplest, most accurate tree possible.

Available Functions.

The functions available to ADATE when synthesizing `f` fall into two groups. The first group comprises constructors and functions for creation and manipulation of the built-in types in ADATE. This includes boolean functions for comparing floating point values and constants, and arithmetic functions for manipulation of floating point values.

The second group contains constructors for the various list and tree types employed to represent decision trees.

ADATE Training Inputs and Evaluation Function.

In this paper, an ADATE training input consists of a decision tree to be pruned. Associated with each tree is a unique synthetic data set which is split into a training partition and a test partition. Each decision tree was generated by the ID3 [12] decision tree inducer using the training partition before starting ADATE. When ADATE is running, each pruned tree returned by the f function is tested on its associated decision tree test partition. The sum of the classification errors that a pruned tree makes on all test partitions determines how good it is and is used as a heuristic evaluation function that guides the ADATE search for better pruning algorithms.

There does not seem to be a correspondence between the type of decision tree learner used and the pruning algorithm [9], so any synthesized pruning algorithm working well with ID3 should work just as well with any other decision tree learner.

A synthetic data set is generated as follows. First, we create a two-layer feed forward neural network. This network always has exactly two output nodes but we chose to vary the number of input nodes, which equals the number of input attributes, between 10 and 12. We only use 0 or 1 as network inputs and also a binary target attribute, the value of which is given by which output node that has the greatest value for given neural net inputs. The neural net weights are chosen randomly between -0.5 and 0.5. Thus, each output node is simply a random linear combination of the input nodes. A complete synthetic data set is obtained by feeding the network with all 2^n inputs for a given number n of input attributes.

As mentioned above, each data set is divided into a training partition and a test partition, where we vary the portion utilized for training from 0.2 to 0.35 in steps of 0.05. In order to emulate how certain information often is missing from real world data sets, some attributes are removed completely from the data set. The number of attributes removed is varied between 0 and 4. This removal of attributes is an attempt to model the indeterminism, caused by not having sufficiently many sufficiently well chosen attributes, that is encountered in most real world data sets.

By using all combinations of the number of input attributes, the fraction used for training and the number of removed attributes, 60 different data sets are created. This procedure is repeated many times to create larger numbers of ADATE training, validation and test inputs which all are multiples of 60 in this paper.

4 Experimental Procedure

The ADATE system was executed on a cluster with 16 computers over a period of three weeks. The 16 computers have a total of 22 CPU cores that are a mix of Intel Pentium D and Pentium 4.

Before performing this execution, we conducted a set of small experiments in order to try to determine the optimal size of the training set with respect to both overfitting and synthesis speed. The size should be small to obtain a short execution time δ, but at the same time it should be big enough to avoid excessive overfitting. Three specifications with 60, 120 and 240 instances were executed, and the 120 and 240 specifications yielded the most accurate programs on a separate synthetic validation set with 1200 instances. Although the 120 specification did not show any excessive signs overfitting, the danger of eventual overfitting when more execution time is allowed lead us to choose 240. During execution of ADATE, the training size was increased to 480 and eventually to 960 as we observed some signs of overfitting by studying how classification accuracy on the validation set increased with program size.

```
fun estimateError( c, n ) =
let
    val v1 = tanh( tanh( tanh( (n - c) /  n ) ) )
    val v2 = sqrt( tanh( tanh( sqrt( n ) ) ) )
    val v3 = tanh( sqrt( sqrt( c ) ) )
in
    (v1 + v2) / v3
end
```

Figure 1: The estimateError function of the best pruning algorithm produced by ADATE.

Selecting the Pruning Algorithm

The ADATE population contains many programs at any given time. Therefore, some method must be applied to finally select the program that is likely to produce the most accurate decision trees in general.

Since more computation time is available per program during this final selection, we chose to use a larger validation set with 15000 instances and selected the program with the total least number of classification errors on this set.

The selected pruning algorithm has a structure similar to that of the initial pruning algorithm and the tree traversal was not changed much by ADATE. However, the selected program contains an auxiliary error estimation function generated by ADATE that is different from anything we have seen and still apparently quite practically useful for pruning. The arguments of this function are the number of instances `n` that reach a given tree node and the number `c` of these instances that are correctly classified by the subtree corresponding to the node.

We rewrote the error estimation function to a more readable implementation in Standard ML where we have changed the ADATE generated variable names. The function is given in Figure 1, and consists of an arithmetic expression that is somewhat hard to interpret.

Note that the `tanh` function, being a scaled version of a sigmoid function, often is used in the nodes of feed forward neural nets and that weight optimization for such nets typically results in a "black box effect" that makes the nets hard to analyze in detail. Unfortunately, the automatically synthesized function in Figure 1 also suffers from this black box effect.

Thus, we may notice experimentally that this ADATE generated function results in better pruning than the confidence interval upper limit estimation used by the EBP of C4.5, but it is hard to analyze it theoretically.

5 Results and Analysis

The synthesized pruning algorithm (SYNTH), which was partially shown in Figure 1, is evaluated empirically across both synthetic and real world data sets by comparing it with four other pruning algorithms, namely no pruning (NOP), the naive implementation of EBP (NEBP) that ADATE used as its initial program, EBP without tree grafting (WEBP) and EBP without tree grafting(TWEBP), where the confidence level is tuned using 10 fold cross validation.

The pruning algorithms are compared over multiple data sets using the Friedman test [13, 14, 15] which will try to reject the null hypothesis of all algorithms producing equally

Table 1: Average ranks for each pruning algorithm on synthetic data.

#Removed Attribs	#Data sets	NOP	NEBP	WEBP	TWEBP	SYNTH
0	12000	3.426-	2.936-	3.078-	3.010-	2.549
1	12000	3.885-	3.318-	2.618-	2.632-	2.548
2	12000	4.245-	3.532-	2.360+	2.395+	2.469
3	12000	4.445-	3.534-	2.278+	2.360	2.384
4	12000	4.483-	3.330-	2.349+	2.440-	2.398
Total	60000	4.097-	3.330-	2.536-	2.567-	2.470

accurate decision trees. It is rank based and the algorithms are ranked for each data set according to the accuracy of the decision tree they produce, where the most accurate pruning algorithm receives the highest rank, rank 1. If two or more pruning algorithms are equally accurate, the average rank is assigned to them.

If the null hypothesis is rejected at significance level 0.05, the post hoc test proposed by Holm[16] is employed to investigate which of the pruning algorithms that are truly different from SYNTH.

Synthetic Data

The synthetic data sets for testing are generated using the same procedure that was employed when creating the training and validation sets for ADATE, but the procedure is repeated many more times. We used 1000 repetitions, resulting in 60000 new data sets with 12000 data sets for each specific number of attributes to remove. In this way, it is possible to determine with statistical significance whether the synthesized pruning algorithm produces more accurate trees in this domain compared to the other pruning algorithms.

Table 1 contains the average rank of each algorithm when a specific number of attributes are removed from the data sets. The last row shows the average rank across all 60000 data sets. A + means the algorithm is significantly better than SYNTH, and a - means it is significantly worse.

The ADATE generated SYNTH pruning performs overall better than all the other pruning algorithms. It has a rank considerably higher than NOP and NEBP, both in total and for each type of data sets where a specific number of attributes are removed.

This shows that the ADATE system has been able to significantly improve the initial pruning algorithm, NEBP, for this domain. On the other hand, the difference in rank is not that great compared to WEBP and TWEBP, and it has a somewhat lower average rank than WEBP for the data sets where two, three, and four attributes are removed. However, SYNTH performs so much better on the data sets where no attributes are removed that it overall produces the most accurate decision trees.

A surprising observation is that WEBP has a slightly higher rank than TWEBP on average. This is probably caused by the limited number of training instances for some of the data sets. For instance, the smallest training set contains only 102 training instances.

Real World Data

All of the 14 data sets are taken from the UCI machine learning repository [17]. They have only discrete attributes since the decision tree inducer, ID3, learning the initial unpruned decision trees does not support continuous attributes in its original form. Additionally, it

Table 2: Ranks for each pruning algorithm on real world data sets. Obviously, smaller numbers are better.

Data sets	NOP	NEBP	WEBP	TWEBP	SYNTH
agaricus-lepiota	3(0)	3(0)	3(0)	3(0)	3(0)
audiology	2(24.5)	2(24.5)	4(28.4)	5(28.9)	2(24.5)
car	2.5(5.6)	2.5(5.6)	5(6.3)	4(5.7)	1(5.4)
dna	5(8.4)	4(7.3)	1.5(6.1)	1.5(6.1)	3(6.9)
hayes-roth	4.5(28.8)	4.5(28.8)	1.5(27.5)	1.5(27.5)	3(28.1)
kr-vs-kp	1.5(0.3)	1.5(0.3)	5(0.5)	3.5(0.4)	3.5(0.4)
monks-1	3(2.2)	3(2.2)	5(2.9)	1(2.0)	3(2.2)
monks-2	1.5(29.3)	1.5(29.3)	4.5(34.3)	4.5(34.3)	3(30.1)
monks-3	5(2.7)	2.5(1.1)	2.5(1.1)	2.5(1.1)	2.5(1.1)
nursery	1(1.2)	2.5(1.5)	5(3.2)	4(2.4)	2.5(1.5)
promoters	2(25.2)	3(25.3)	5(30.3)	4(26.4)	1(23.5)
soybean	4(7.9)	2.5(7.8)	2.5(7.8)	5(8.1)	1(7.5)
tic-tac-toe	4.5(14.9)	4.5(14.9)	2.5(14.8)	1(13.9)	2.5(14.8)
house votes	5(6.7)	4(6.4)	2(4.8)	1(4.4)	3(5.1)
Avg	3.180(11.3)	2.929(11.1)	3.500(12.0)	2.964(11.5)	2.430(10.8)

does not support missing values, so any missing values are replaced with the most frequent attribute value.

Table 2 shows the ranks given to each algorithm for the different data sets according to the average error percent found using 10 fold cross validation. This error percent is shown in parenthesis. The last row includes the average rank and error percent of all data sets.

SYNTH performs better than the other pruning algorithms with a total average rank of 2.4 compared to 2.9 for the best of the others.

Still, there is not enough statistical evidence to conclude that some of the algorithms are truly different from one another for these real world data sets and the null hypothesis could not be rejected.

The table contains some unexpected results. First, performing no pruning for some of these data sets seems to be a good strategy and it has the highest rank for several of the data sets. In addition, it has an average error percent lower than TWEBP and WEBP. Second, the naive version of EBP, NEBP, performs better than TWEBP and WEBP both in terms of rank and error percent.

Detailed information about the amount of pruning employed by the different algorithms is reported in table 3. Each row contains the average size of the pruned trees produced for the different data sets, except the last row containing the average tree sizes across all data sets.

There is a distinct pattern in the amount of pruning performed by the different algorithms compared to each other for these data sets. Naturally, NOP creates the largest trees since no pruning is employed. NEBP is similar to NOP and performs almost no pruning, while TWEBP and WEBP perform the most pruning of all the algorithms. This is rather surprising considering the close algorithmic relationship between the three algorithms, yet it is consistent with the rank and error percent of these algorithms where the amount of pruning performed by NEBP appears to be better suited for these data sets.

SYNTH prunes more than NOP and NEBP and less than WEBP and TWEBP, a strategy that is the most appropriate on average for these data sets considering the ranks

Table 3: Average number of nodes in the pruned trees for each pruning algorithm on real world data sets.

Data sets	NOP	NEBP	WEBP	TWEBP	SYNTH
aga	38.0	38.0	38.0	38.0	38.0
aud	136.5	136.5	79.4	83.2	131.4
car	387.6	342.8	180.4	221.2	329.8
dna	720.2	449.8	181.4	172.2	393.4
hay	59.6	52.1	29.7	30.9	47.3
krkp	91.4	90.6	60.4	66.4	67.0
mon1	96.7	96.7	66.9	77.8	94.7
mon2	446.4	434.1	1.0	1.0	403.4
mon3	60.0	17.6	17.6	17.6	17.6
nur	1133	1010.9	501	727.2	879.3
pro	37.8	33.0	19.4	17.0	32.2
soy	248.1	186.1	138.7	143.4	169.9
tic	319.6	319.6	163.9	216.1	282.7
vot	66.4	54.7	13.9	4.0	29.8
Avg	274.4	233.0	106.6	129.7	208.3

and error percents observed.

6 Further Work

The next natural step is to try to improve other classification algorithms in a similar manner to how decision tree pruning was improved. For instance, it would be interesting to explore whether a boosting algorithm like AdaBoost[18] could be improved, either by letting ADATE synthesize the best base classifier to use with AdaBoost for a particular domain or improving the algorithm as a whole. One problem with boosting is that it can be computationally expensive since the base classification algorithm is executed numerous times. Thus, this base algorithm should be fast to execute.

7 Conclusion

We have shown that the ADATE system can be used to improve decision tree pruning with statistical significance for a fairly general domain. The automatically generated pruning algorithm outperformed the other pruning algorithms when tested on synthetic data sets selected according to a probability distribution that attempts to model the effects of not having enough information available in real world data sets.

In addition, it performed better than the other pruning algorithms across real world data sets, although the differences between the algorithms were not statistically significant according to the Friedman test.

Our results show that ADATE can be used to improve decision tree learning, both in general and for a particular domain of application, and that it might be possible to automatically improve other classification algorithms as well.

References

[1] Olsson, R.: Inductive functional programming using incremental program transformation. Artificial Intelligence **1** (1995) 55–83

[2] Breiman, L.: Bagging predictors. Machine Learning **24**(2) (1996) 123–140

[3] Schapire, R.E.: The strength of weak learnability. Machine Learning **5**(2) (1990) 197–227

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

[5] Olsson, R., Schmid, U., editors: Special issue on inductive programming. Journal of Machine Learning Research **7** (2006)

[6] Aler, R.: Automatic inductive programming. In: Tutorial at the International Conference on Machine Learning, Pittsburgh Pennsylvania USA, Carnegie Mellon (2006)

[7] Olsson, R., Powers, D.: Machine learning of human language through automatic programming. International Conference on Cognitive Science (2003)

[8] Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach. 2nd edition edn. Prentice-Hall, Englewood Cliffs, NJ (2003)

[9] Mingers, J.: An empirical comparison of pruning methods for decision tree induction. Machine Learning **4**(2) (1989) 227–243

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

[11] Quinlan, J.R.: C4.5: programs for machine learning. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1993)

[12] Quinlan, J.R.: Induction of decision trees. Machine Learning **1**(1) (1986) 81–106

[13] Friedman, M.: The use of ranks to avoid the assumption of normality implicit in the analysis of variance. Journal of the American Statistical Association **32**(200) (1937) 675–701

[14] Friedman, M.: A comparison of alternative tests of significance for the problem of m rankings. The Annals of Mathematical Statistics **11**(1) (1940) 86–92

[15] Iman, R.L., Davenport, J.M.: Approximations of the critical region of the friedman statistic. Communications in Statistics (1980) 571–595

[16] Holm, S.: A simple sequentially rejective multiple test procedure. Scandinavian Journal of Statistics **6**(1) (1979) 65–70

[17] D.J. Newman, S. Hettich, C.B., Merz, C.: UCI repository of machine learning databases (1998)

[18] Freund, Y., Schapire, R.E.: A decision-theoretic generalization of on-line learning and an application to boosting. In: European Conference on Computational Learning Theory. (1995) 23–37