Generating AVTs Using GA for Learning Decision Tree Classifiers with Missing Data

Jinu Joo[1], Jun Zhang[2], Jihoon Yang[1], and Vasant Honavar[2] *

[1] Department of Computer Science, Sogang University
1 Shinsoo-Dong, Mapo-Ku, Seoul 121-742, Korea
jujoo@mllab.sogang.ac.kr, jhyang@ccs.sogang.ac.kr
[2] Artificial Intelligence Research Laboratory, Department of Computer Science
Iowa State University, Ames, IA 50011 USA
{jzhang,honavar}@cs.iastate.edu

Abstract. Attribute value taxonomies (AVTs) have been used to perform AVT-guided decision tree learning on partially or totally missing data. In many cases, user-supplied AVTs are used. We propose an approach to automatically generate an AVT for a given dataset using a genetic algorithm. Experiments on real world datasets demonstrate the feasibility of our approach, generating AVTs which yield comparable performance (in terms of classification accuracy) to that with user supplied AVTs.

1 Introduction

In a typical inductive learning scenario, instances to be classified are represented as ordered tuples of attribute values. However, attribute values can be grouped together to reflect assumed or actual similarities among the values in a domain of interest or in the context of a specific application. Such a hierarchical grouping of attribute values yields an attribute value taxonomy (AVT) [1]. For example, Fig. 1 shows an AVT defined over color. Under *color* node there are more specific values defined which have more specific values defined under each sub node. In one instance the color attribute may be specified as sky blue while in another instance it can be described as just blue.

AVTs have been exploited in learning classifiers from data due to its various advantages [1]. In particular, the AVTs are used in AVT based decision tree learning algorithm (AVT-DTL) [2] which classifies partially missing and totally missing values more efficiently than traditional algorithms such as ID3 [3] and C4.5 [4].

An AVT needs to be specified a priori in a classification task. However, in many domains, AVTs specified by human experts are unavailable. Even when a human-supplied AVT is available, it is interesting to explore whether alternative groupings of attribute values into an AVT might yield more accurate or

* This research was supported in part by grants from the National Science Foundation (grant 021969) and the National Institutes of Health (GM066387) to Vasant Honavar.

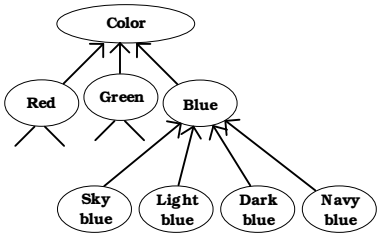

Fig. 1. An AVT on color

more compact classifiers. Against this background, we propose a method that produces an AVT automatically for a given dataset. Due to the huge search space of possible AVTs, we need to consider methods (or algorithms) that finds *near-optimal* solutions. Here we resort to a genetic algorithm to automate the generation of AVTs for AVT-DTL algorithm on datasets with missing values.

2 AVT-guided Decision Tree Learning

AVT-DTL algorithm [2] classifies partially specified data based on a user provided AVT. This algorithm performs a AVT-guided hill climbing search in a decision tree hypothesis space. AVT-DTL works top-down starting at the root of each AVT and builds a decision tree that uses the most abstract attribute values that are sufficiently informative for classifying the training set consisting of the partially specified instances. It is straightforward to extract classification rules from the set of pointing vectors associated with the leaf nodes of the decision tree constructed by AVT-DTL. For example, Fig. 2 shows pointing vectors that points to two high-level attribute values *blue*, *polygon* in the two taxonomies of *Color* and *Shape*. If this pointing vector appears in the leaf node of the decision tree with class label +, the corresponding rule will be: If (*Color=blue Shape=polygon* ...) then Class =+. See [2] for detailed descriptions on AVT-DTL.

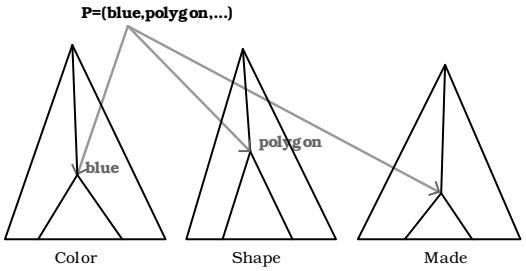

Fig. 2. Pointing vector

3 GA Based AVT Generation

3.1 Genetic Algorithm

Genetic algorithm (GA) [5–7] is an efficient nonlinear heuristic search algorithm, only using simple operators like mutation, crossover, and selection. It is an evolving iterative process working with a solution set called *population*. A population is a set of solutions called *chromosomes*. In each generation different chromosomes form a population, and each population contains better chromosomes than the former generation. By using operators like selection, crossover, and mutation the population set evolves to converge to the optimal solution. During these operations, the fitness function is used to evaluate the *fitness* (or *appropriateness* or *propriety*) value of each individual. Our GA-based approach for generating AVTs proceeds as follows:

GA-AVT

1. Create the initial population of an AVT, where each chromosome is a bit string representation and each attribute is represented by a set of *blocks*. Blocks are binary bits which represent node bindings on their level in the tree. More details on blocks are described in Section 3.2.
2. Evaluate the fitness of each individual in the population set. The fitness function is the classification accuracy estimated by 10-fold cross validation.
3. Enter the population in the roulette wheel selection.
4. Execute the crossover and mutation operation on the roulette wheel selected population.
5. Repeat step 1 through 5 until a certain number of iterations are reached or when the solution does not improve.

3.2 AVT Representation

To modify and generate AVTs in GA, we need to form an appropriate representation for AVTs. We introduce a binary string format to represent an AVT. The difficulty of finding a good representation for an AVT is first, an AVT has a certain number of leaf nodes that never change. Second, the traditional representation of trees would cause a duplication of nodes after genetic operation. Therefore we devised a new type of chromosome in binary string that represents an AVT which satisfies above two constraints.

Fig. 3 is an abstract structure of how a chromosome is formed with *genes* and blocks. Inside a chromosome there are n genes where n is the number of attributes. Each gene represents its attribute's AVT respectively (i.e. in *nursery* dataset, the first gene represents the AVT of the first attribute, *parents occupation*). A gene consists of a set of bit strings called blocks. Again there are m blocks in one gene where m is the logarithm of leaves. In other words, If a_i is the i_{th} attribute and l_{a_i} represents the number of its leaf nodes, then the number

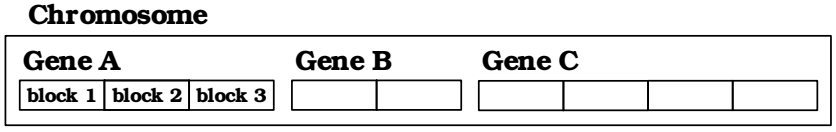

Fig. 3. Structure of a Chromosome

of *blocks* b_{a_i} is $b_{a_i} = \lfloor \log_2 l_{a_i} \rfloor$. Finally, a block contains k binary bits where $k = l_{a_i}$.

Taking this newly defined chromosome, we can build an AVT for a fixed number of leaf nodes. If h_j is the block number of the i_{th} attribute and has a range of $1 \leq h_j \leq b_{a_i}$, then each $block_{h_j}$ represents the information of how the nodes or subtrees should be grouped in step j. Building AVTs with blocks should start from the block with $j = 1$ and end when j reaches b_{a_i}. Bit strings in a block tells how to group nodes and subtrees by comparing a bit from its adjacent bits. For instance , the k_{th} bit will be compared with $(k-1)_{th}$ bit and $(k+1)_{th}$ bit. If the k_{th} bit is the same bit as the $(k+1)_{th}$ bit, the corresponding leaf nodes are considered to be grouped and have the same parent abstract node. If the k_{th} bit has a toggle difference compared to the $(k+1)_{th}$ bit, the two corresponding leaf nodes stay separate. When comparing subtrees, the bit representing the corresponding subtree is the first starting bit of the tree representation. If a subtree corresponds to bits starting from k to $k+p$ (where $1 \leq p \leq l_{a_i} - k$), then bit k will be the representative bit of that subtree to compare with other bits excluding bits k to $k+p$. By executing each step of blocks until h_j reaches b_{a_i} will accomplish an AVT structure.

The example below demonstrates how to group up an AVT with binary bit blocks 000110 - 0XX1X1 corresponding to a certain sorted leaf node set *a, b, c, d, e, f*. The full process is illustrated in the following sequence. (X represents don't care bits which means either 0 or 1 is available in this position, and hyphen distinguish one block from another.):

1. Consider the first block, 000110. The first three nodes *a, b, c* will group to a certain abstract node since the first three bits are all 0's. Hence *d, e* and *f* will group up in a similar way. Let us name the abstract node that groups {*a, b, c*}, *A;* {*d, e*}, *B;* and *f* stands alone. Therefore the nodes considered on the next step will be *A, B, and f* (Fig. 4a).
2. To create the next level of the tree we consider the second block, 0XX1X1. Since nodes *a, b, c* are child nodes to abstract node *A*, and *d, e* are to *B*, which forms a subtree, only first, fourth and sixth bits in *block* 2 are needed to proceed the next step. Because the second block's first and fourth bits turn out to be different, nodes *A* and *B* don't group for now. In a similar manner the fourth and sixth bits are compared, and the two nodes *B* and *f* are combined since the two bits are equal (Fig. 4b).
3. Finally since there are no other *blocks* following, the final remaining nodes *A* and *C* are grouped to the root node which is the attribute name (Fig. 4c).

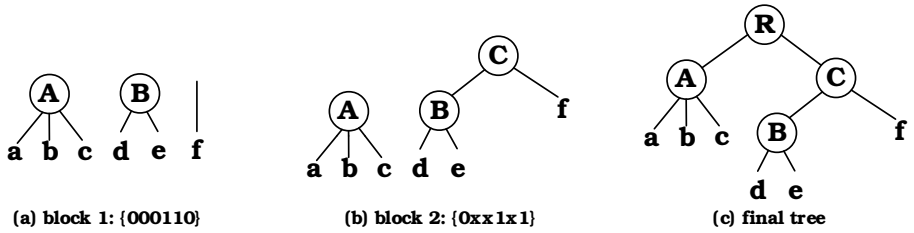

Fig. 4. Decoding AVT

3.3 Genetic Operators for GA

A one point crossover operation is applied to every block respectively. This type of crossover is necessary because each block contains the information of how nodes are to be constructed on the level of the tree. Selection operation is basically a roulette wheel selection. But since classification accuracy (or fitness value) might differ only slightly from individual to individual, adjusting these values are necessary. Here adjustment was performed by first searching the chromosome that has the minimum fitness value in the population. Second, multiply the fitness value difference between the original fitness value and the minimum fitness value by three. This fitness value adjustment is performed to all individuals. Mutation operation is performed with mutation rate 0.01. Once a chromosome is selected to perform a mutation, one bit per every block is randomly chosen to be mutated.

4 Experiment

4.1 Datasets and Experimental Setup

Several experiments were performed to explore the performance of the GA generated AVT (GA-AVT) compared to the predefined AVT given by Zhang and Honavar (*Nursery*) [2] and Taylor *et al.* (*Mushroom*) [8]. In each case, the error rate was estimated by 10-fold cross validation. The experiments compared error rates from the original AVT with GA generated AVT. The GA-AVT was generated with datasets containing totally missing values in this experiment. Experiments were conducted with different pre-specified percentage (0%, 5%, 10%, 20%, 30%, 40%, or 50%) of totally missing attribute values. The GA parameters were set as, population: 20; max generation: 50; mutation rate: 0.01; crossover rate: 0.6; number of elite chosen: 2.

4.2 Experimental Results

We obtained the results shown in (Table 1). In the case of *Nursery* data, the error rate of GA-AVT was somewhat lower than the error rate of the original AVT, especially when missing attribute percentage was high. In general, GA-AVT has

Table 1. Error rate estimation with AVT-DTL and GA-AVT-DTL

Dataset	Method	Missing percentage						
		50%	40%	30%	20%	10%	5%	0%
		% Error rates estimated using 10-fold						
Nursery	AVT-DTL	39.87	33.50	27.82	21.20	12.94	8.37	2.90
	GA-AVT-DTL	37.72	32.98	27.29	21.07	12.52	14.73	3.02
Mushroom	AVT-DTL	6.04	3.87	2.33	1.60	0.64	0.26	0
	GA-AVT-DTL	6.01	3.63	2.39	1.50	0.60	0.19	0

better or similar performance compared with the original AVTs. Because AVT-DTL itself has a long computation time, average of 2 to 3 minutes, running GA AVT-DTL to search for AVTs will give 33 to 34 hours based on our experimental environment of 20 populations and 50 generations. However once the AVT is generated by GA, this AVT will have chances to be reused in different datasets of the same domain. Therefore in the long run using GA to generate AVTs will provide more accurate decision trees than using a user-supplied AVT. Fig. 5 is an example graph of the evolution of AVTs evolving by GA with datasets of missing value at 50%. As we can see in the graph, best solutions were attained within 5 generations while average fitness keeps increasing.

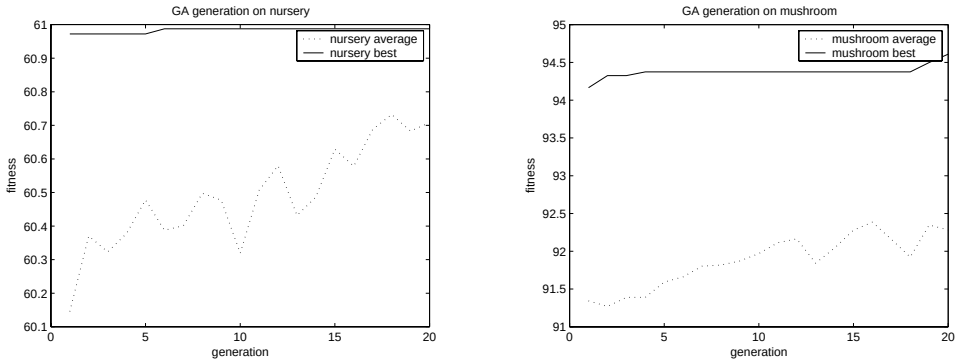

Fig. 5. AVT evolving by GA (datasets of missing value at 50%)

5 Related Work

Cimiano et al. [9, 10] used agglomerative clustering for learning taxonomies from text. Gibson and Kleinberg [11] introduced STIRR, an iterative algorithm based on non-linear dynamic systems for clustering categorical attributes. Ganti et al. [12] designed CACTUS, an algorithm that uses intra-attribute summaries to cluster attribute values. However both of them didn't make taxonomies and used

the generated for improving classification tasks. Pereira et al. [13] describe distributional clustering for grouping words based on class distributions associated with the words in text classification. Yamazaki et al. [14] described an algorithm for extracting hierarchical groupings from rules learning by FOCL (an inductive learning algorithm) [15] and report improved performance on learning translation rules from examples in a natural language processing task. Slonim and Tishby [16, 17] described a technique (called the agglomerative information bottleneck method) which extends the distributional clustering approach described by Pereira et al. [13], using Jensen-Shannon divergence for measuring distance between document class distributions associated with words and applied it to a text classification task. Baker and McCallum [18] report improved performance on text classification using a technique similar to distributional clustering and a distance measure, which upon closer examination, can be shown to be equivalent to Jensen-Shannon divergence [16]. Zhang and Honavar recently proposed an algorithm (AVT-Learner) for automated construction of AVTs from data that uses Hierarchical Agglomerative Clustering (HAC) to cluster attribute values based on the distribution of classes that co-occur with the values [1].

6 Summary and Future Work

Through this paper we have presented the need for automatic generation of AVTs, and introduced a GA based approach. We have performed two experiments with *Nursery* and *Mushroom* datasets to generate optimal AVTs and compared them with the original AVTs. Experimental results demonstrate the feasibility of our approach in terms of the classification accuracy. In particular, our GA-based automatic induction of AVTs obviate the need for ad hoc construction of AVTs that are possibly inefficient and inaccurate.

Some directions for future work include: First, improve the representation of AVTs and corresponding genetic operators to make a more effective GA model in generating AVTs. Second, adjust the AVT-DTL algorithm to achieve better performance and running time. Third, conduct well-designed experiments (possibly with more real-world data), and compare the performance (e.g. classification accuracy, tree size, etc.) of GA-based approach with that of other methods (e.g. AVT-Learner). Fourth, develop GA-AVT combined with a variety of machine learning algorithms (e.g. Naive Bayes classifier).

References

1. Kang, D.K., Silvescu, A., Zhang, J., Honavar, V.: Generation of attribute value taxonomies from data and their use in data driven construction of accurate and compact naive bayes classification. In: Proceedings of the ECML/PKDD Workshop on Knowledge Discovery and Ontologies. (2004)
2. Zhang, J., Honavar, V.: Learning decision tree classifiers from attribute value taxonomies and partially specified data. In: Proceedings of the Twentieth International Conference on Machine Learning (ICML 2003). (2003)

3. Quinlan, R.: Induction of decision trees. Machine Learning **1** (1986) 81–106
4. Quinlan, R.: C4.5: Programs for Machine Learning. Morgan Kaufmann, San Mateo, CA (1992)
5. Mitchell, M.: An Introduction to Genetic algorithms. MIT Press, Cambridge, MA (1996)
6. Goldberg, D.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, New York (1989)
7. Yang, J., Honavar, V.: Feature subset selection using a genetic algorithm. IEEE Intelligent Systems **13** (1998) 44–49
8. Taylor, M., Stoffel, K., Hendler, J.: Ontology-based induction of high level classification rules. In: SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery. (1997)
9. Cimiano, P., Staab, S., Tane, J.: Automated acquisition of taxonomies from text: Fca meets nlp. In: Proceedings of the ECML/PKDD Workshop on Adaptive Text Extraction and Mining, Cavtat-Dubrovnik, Croatia. (2003) 10–17
10. Cimiano, P., Hotho, A., Staab, S.: Comparing conceptual, partitional and agglomerative clustering for learning taxonomies from text. In: Proceedings of the European Conference on Artificial Intelligence(ECAI'04). (2004)
11. Gibson, D., Kleinberg, J., Raghavan, P.: Clustering categorical data: An approach based on dynamical systems. VLDB Journal:Very Large Data Bases **8** (2000) 222–236
12. Ganti, V., Gehrke, J., Ramakrishnan, R.: Cactus - clustering categorical data using summaries. In: Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining, ACM press (1999) 73–83
13. Pereira, F., Tishby, N., Lee, L.: Distributional clustering of english words. In: 31st Annual Meeting of the ACL. (1993) 183–190
14. Yamazaki, T., Pazzani, M., Merz, C.: Learning hierarchies from ambiguous natural language data. In: International Conference on Machine Learning. (1995) 575–583
15. Pazzani, M., Kibler, D.: The role of prior knowledge in inductive learning. Machine Learning **9** (1992) 54–97
16. Slonim, N., Tishby, N.: Agglomerative information bottleneck. In: NIPS-12. (1999)
17. Slonim, N., Tishby, N.: Document clustering using word clusters via the information bottleneck method. In: Proceedings of the 23rd annual international ACM SIGIR conference on Research and Development in Information Retrieval, ACM press (2000) 208–215
18. Baker, L.D., McCallum, A.K.: Distributional clustering of words for text classification. In: Proceedings of the 21rd annual international ACM SIGIR conference on Research and Development in Information Retrieval, ACM press (1998) 96–103