AUTOMATED FEATURE EXTRACTION FOR LINEAR PATTERN RECOGNITION

Peter J. Schubert[1], Douglas Sanqunetti[2], and Bill Chen[1],
[1]Delphi Delco Electronics Systems, Kokomo, IN, USA
[2] Electronic Data Systems, Plano, TX, USA

ABSTRACT

Pattern recognition in linear sensor arrays is facilitated by the extraction of algebraic features. In our automotive application, the data size is large, and requires high accuracy, while simultaneously requiring fast execution, and a minimum of code storage. To accomplish these goals, while also minimizing the total computation time, we use strongly-typed genetic programming (GP). The functions are least-squares-fit polynomials up to second order, with lengths distributed with exponentially-decaying probability. Variables are used instead of random constants for setting thresholds. This greatly reduces the search space for the GP, but requires an optimization to evaluate fitness. We use the new technique of Particle Swarm Optimization (PSO) to evaluate fitness for each member of every population. This produces a pattern recognition algorithm that is parsimonious, generalizes well, and has good accuracy. We present results of code size, accuracy, and robustness, and compare with decision tree, and neural network results.

INTRODUCTION

Cost requirements for automotive pattern recognition applications drive the selection of inexpensive hardware. However, performance is expected to be excellent, even across wide ranges of environmental conditions, manufacturing tolerances, sensor degradation, and target characteristics. These requirements demand a very robust algorithm, which generalizes well, and has a high tolerance for missing or noisy data. In our application, a linear array of 16 ranging sensors must analyze between 7 major target classes, with up to 20 sub-classes possible in each class. Each sub-class is designated 'go' or 'no go' by the final OEM customer. The decision algorithm must be trained on 10,000 or more data readings, must execute in 5 milliseconds, and be implemented in a low-cost, small-memory microprocessor.

Methods for solving this problem include: Artificial Neural Networks (ANN) [1] and their derivatives, such as PROBART [2], or decision tree analysis [3] with its derivatives [4]. In this application, such techniques tend to generalize poorly. Their reliance on individual sensor readings makes them "brittle", or overly-sensitive to variations in the input data. Over-training is an important concern, where highly specific rules or relations are developed for a small fraction of the training data. Several practices, such as windowing or boosting [5] can be employed to partially overcome this limitation. Another difficulty is that a particular feature may appear in several places within the linear array, so that these training methods must recognize the presence of that feature in each location, thus increasing the complexity of the final algorithm.

The Genetic Programming (GP) [6] paradigm appears well-suited to this type of problem. Because target features are known to exist, but are not easy to define, an evolved solution can be pursued to extract features. One method for feature extraction is the use of automatically defined functions (ADF) [7], where re-usable sub-programs are evolved simultaneously with the main algorithm. An alternative method is to provide the GP with pre-defined generic functions which have a reasonable chance to capture these features. This latter is the method employed in this paper. A further sophistication of GP, when incorporating user-defined functions of different type (e.g. Boolean, real-valued, conditional) is to use Strongly Typed GP (STGP)[8,9], where only constructions of valid programs are permitted at initialization and under genetic operation. Finally, typical GP work uses Ephemeral Random Constants (ERC) to set thresholds, conditions or operands. However, an alternative approach, pioneered in this work, is to use variables instead, and to optimize the values of each variable for every member of every population to determine fitness. This method of optimizing as you go ensures the survival of valuable program architecture, independent of whether the ERCs that happen to be assigned are worthwhile.

Following a detailed description of the architecture of our GP solution in the METHODS section, we present RESULTS, and compare them to decision tree and neural network results on the same training and validation data. The relative merits of these algorithm technologies are analyzed in the DISCUSSION. The CONCLUSION session reviews the work performed against the requirements outlined here, and shows the future direction for this work.

METHODS

We elected to use strong typing in our GP application. In the conservative environment of some automotive manufacturers, explainability of our final algorithm is extremely important. Therefore, the primitive functions are relatively complex, while the architectures that we allow to evolve are rather strictly limited. While this reduces the search space for solutions, we gain advantages in computation speed, memory requirements, and less parsing of introns. A further refinement we employ is to omit random constants from the terminal set for the GP. Instead, we provide variables to serve as the terminal set. In STGP, one normally employs one variable type for each data class. However, we have grouped all floating point functions into one variable class. Then, each function type processes this floating point number in a unique way, appropriate to its operation. To determine the optimum value for these variables, each member of each generation of the entire GP population is run through a function optimizer. We have elected to use the relatively new technique of Particle Swarm Optimization (PSO)) [10], since tests show [10] it to have faster convergence than Genetic Algorithms for the initial stages of optimization. During the GP evolution, the PSO is allowed a fixed, and relatively small, number of iterations. This provides an approximately optimal solution, balancing execution time for accuracy. When the final solution is selected, then we run PSO for more iterations to determine the final settings for the variables. These methods are illustrated in more detail below.

Target features are known to contain line segments and simple curves. Polynomials of order three or higher are much less common, and can be influenced by noisy or missing data. So our primitive functions were selected to be point, averages, slopes, and quadratic curves. However, our available data often contains repeated tests where bi-stable data readings can easily lead to brittleness of the algorithm. To make our solutions less sensitive to these cases, we employ least-square fitting of the averages, slopes and curves using singular value decomposition [11]. Features may range in length from one to two or three sensors, or more. However, the shorter the feature, the more likely it will occur in our target. Therefore, the number of points over which any such function will be extracted is given a probabilistic likelihood determined randomly, and exponentially decaying with increasing length. This is implemented with a probability p, according to equation (1).

$$p(L) = \frac{A\exp(-B*L)}{\sum_{i=1}^{15} A\exp(-B*i)} \qquad (1)$$

In practice, the function is selected first, and then the length is determined according the following algorithm. A random number in the range [0,1] is first selected, and then mapped onto L according to its position within bins defined by equation (2).

$$L = j \mid RAND[0,1] \in \left[\frac{\sum_{i=1}^{j-1} A \exp(-B * i)}{\sum_{i=1}^{15} A \exp(-B * i)}, \frac{\sum_{i=1}^{j} A \exp(-B * i)}{\sum_{i=1}^{15} A \exp(-B * i)} \right] \qquad (2)$$

If the length is shorter then the minimum required for that function, for example a second order curve selected with L = 1 or 2, or a first order slope for L=1, then the function is down-graded to be of order one less than the number of points. These exponential probabilities then give a distribution of feature lengths and complexities that favor short segments. There are two reasons for doing this. First, we know our targets are predominantly made of segments, which can take a number of relative orientations. Therefore, longer segments will tend to be less able to generalize to conditions not present in the training set. Second, our prior experience with decision trees, that are based solely on combinations of thresholds applied to single point readings, shows this method to be of some value in solving the problem. Therefore, our method is intended to be an extension of these results, to incorporate features which will tend to be more robust than single point readings. In the trivial case of L=1 for our method, it decomposes to be nearly the same as decision trees.

The GP architecture we wish to evolve contains a Boolean operation at the root, reflecting the GO/NO-GO decision. The strong-typing is used to allow branching from the root which is predominantly Boolean. This allows the generation of several branches, to provide needed complexity, before the GP trees begin to terminate. The next level of typing includes comparison operators (GT, LTE). These operators have a Boolean output, but floating inputs, and are analogous to simple rules-based algorithms with constant thresholds. The lowest level of functions are our primitives of average, slope and curve, plus the terminal variables. Both primitives and terminals are floating point numbers. By placing features at the leaves of our tree, the first sub-decisions made on the way to GO/NO-GO are based on realistic, known and explainable characteristics of our data. The composite architecture can be explained from data to features, from features to comparisons, from comparisons to combinations, and from combinations to a final classification.

Our objective is to create an general feature extraction algorithm that can be calibrated to a variety of specific applications. Therefore, the architecture of the final algorithm must be as simple and robust as possible. Also, the algorithm must be amenable to automated calibration. We have elected to incorporate this automatic calibration into the GP process itself. To our knowledge, this is the first implementation of this concept. Our *a priori* knowledge of our application tells us that we will have a substantial number of thresholds in our algorithm. Rather than requiring the GP to evolve the best values for these thresholds, we instead use an optimization program to determine the threshold values which bring the best fitness for a given GP parse tree. By calibrating as we go, the GP parse tree can achieve high fitness without complex structure. Specifically, we wish to avoid features from becoming thresholds, since their values will change with every target. Of course, this requires additional computing power, and longer times to converge.

Within the strongly-typed GP, each floating point-type terminal of each function of every member of each population is collected and sorted. The parse tree is kept in program memory, referenced by pointers, and optimized as a “black box”, with the terminals as inputs and the accuracy percentage as the output. The PSO optimization technique is set up to run a fixed and limited number of optimization steps. We further restrict the values of the variables to a range useful to the functions in the parse tree. In the GP outer loop, we eventually keep the top 5 solutions discovered, to cover the case where truly better solutions did not optimize as quickly as other solutions. In the final step, the validation data set is analyzed using the stored parse tree, to evaluate the ability of the algorithm to generalize to new data.

RESULTS

The training set used is a small subset of our true application, yet still represents the data set fairly well. This was done to give reasonable convergence times for the GP/PSO, for comparison with other techniques. Because of the sparse nature of the training set, and the relatively greater representation of difficult test cases within it, the accuracy values obtained are lower than we expect to get on the complete data set. Two evaluation sets were selected in the same pseudo-random manner, which are used to evaluate the ability of the algorithm to generalize. No changes to the threshold values were made between training on the training set and validation on the evaluation set. Also note that fine-tuning of the parameters was not tested at this point, so that better results than those reported below may be possible for.

Results were also generated for three other methods of classification. Decision tree learning was done with the C4.5 program [3], using a limited set of rules. Because decision trees can achieve very high scores if allowed to generate large number of rules, but will generalize less well, we restricted the decision tree solutions to either 12 or 19 rules (the latter with 1 boost) for the training set of 130 cases. A Neural Network was also used to train on this data. As with Decision Trees, an arbitrarily large number of hidden neurons can be used to generate very high accuracy, but again, when applied to an evaluation set containing new and generally different data, will suffer a significant loss of accuracy. We elected to run two NNs with hidden layers of 5 and 10 nodes. The third method was Bayesian classification, using the program RoC [13]. This provides a theoretically optimum classification using maximum *a posteriori* (M.A.P) probability. Bayesian classification is based on discrete attributes, so that our data values had to be made discrete. After some experimentation with the number of discrete levels, we selected the results which gave the best accuracies. This provides an estimation of the maximum accuracy possible, given the data provided.

The accuracy reported is the number of correct classifications. The decrease in correctly classified cases between the training set and the evaluation set indicates the robustness of the algorithm, or its ability to generalize. Robustness is thus defined as the training accuracy percentage subtracted from the averaged evaluation accuracy percentage. Robustness is thus a negative number, in general, with more positive numbers representing better ability of the algorithm to generalize to data not included in the training set. Table 1 shows the decision complexity of the final algorithm, the accuracy for the two evaluation sets, and the Robustness measure as defined above. The code size for GP does not include parsing of introns (code which does not contribute to the operation or accuracy of the algorithm). For purposes of understanding the GP process better, we present results for standard GP to compare against the GP with PSO. The GP-only solution uses the more traditional ephemeral random constants instead of real-valued variables. The modulo operation is applied to the inputs of each function, to achieve uniform distribution of the variables across their useful range, and prevent overflows. We further included addition and multiplication functions in the GP-only study, to provide a means for obtaining optimized values for threshold variables.

Table 1. Results of Classifier Algorithms for Linear Pattern Recognition

Method	Decision Complexity	Eval. Set 1	Eval. Set 2	Robustness
Bayesian	Full M.A.P., 9 discrete levels for continuous variables.	86.5 %	85.4 %	-12.1 %
GP only	11 levels, 158 nodes	83.3	81.0	- 4.8
GP with PSO	11 levels, 92 nodes	83.3	82.5	- 1.7
Neural Network	10-neuron hidden layer	85.5	83.3	-12.3
Neural Network	5-neuron hidden layer	84.1	82.9	-11.1
Decision Tree	19 Rules	84.1	83.9	-11.2
Decision Tree	12 Rules	82.5	83.4	-14.8

DISCUSSION

Field operation of a linear array pattern recognition system in automobiles will encounter situations and environments well beyond what can be reasonably tested or simulated. Therefore, the decision algorithm must possess robustness, as demonstrated by minimal degradation from training data set to validation data set. Yet the accuracy of the validation data set must also be as high as possible. In our comparisons, we have used fairly straightforward embodiments of GP, decision trees, neural networks, and Bayesian classifiers We compare these against our new method; an extension of GP to include optimization-as-you-go. We do not intend these results to be absolute and conclusive, but more simply to indicate the potential of a method which promises to offer greater explainability and potentially faster execution.

If we consider the Bayesian classifier as an estimate of the maximum accuracy possible, then 86% averaged over the two evaluation sets would represent a nearly-ideal classifier. This goal is approached fairly equally among the methods evaluated. However, for Robustness, the GP-only and GP/PSO solutions are significantly improved over the Bayesian, NN, or decision tree methods. The reason for this lies partly in the constraints placed on the solution space. Both the levels and numbers of nodes are given upper limits in the GP. In any unrestrained training method, the training accuracy can be increased to 100%, assuming infinite precision in the data values. However, such high accuracies require an increasingly large number of rules, to the limit where each data point can generate a rule of its own. Obviously, a system trained to this degree will be over-specialized, and would perform poorly on new data. The number of rules for decision trees, and the number of hidden layer neurons for the NN were selected specifically to prevent this type of over-training. In the GP programs, the primitive functions were selected to force the algorithm to locate features in a balance with individual data points. This practice discourages over-training, and resulted in improved robustness.

CONCLUSIONS

This paper introduces a new method for using GP plus PSO to obtain algorithms which: recognize features, are parsimonious, and generalize well. By using variables instead of constants within the strongly-typed GP, and optimizing each member of the population at each generation, the resulting algorithm appears to possess the desired qualities. Training time is significantly longer with this approach, yet test set accuracy achieves the estimated maximum accuracy. The greater benefit lies in the improved Robustness, as demonstrated by a lesser reduction in accuracy from training to test data. In our automotive application the improved Robustness alone justifies the additional expense of the long training times. For further evaluation of our method, a Beowulf-class cluster of computers will be used on the complete 10,000 plus training data set. Performance on noisy data also needs to be tested.

REFERENCES

[1] Pao, Y.-H., "Adaptive Pattern Recognition and Neural Networks", 1989.

[2] Srinivasa, N., "Learning and Generalization of Noisy Mappings using a Modified PROBART Neural Network," IEEE Trans. Signal Proc, Vol. 45, No. 10, October 1997.

[3] Quinlan, J.R., "C4.5 : Programs for Machine Learning", Morgan Kaufman, 1977.

[4] Janikow, C.Z., "Fuzzy Decision Trees: Issues and Methods," IEEE Trans. Sys. Man, Cyber. Part B., vol.28, no.1, Feb. 1998.

[5] Freund, Y., Schapire, R.E., "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting," JCSS vol. 55, no. 1, 1997.

[6] Koza, J., "Genetic Programming: On the Programming of Computers by Means of Natural Selection," MIT Press, 1992.

[7] Koza, J., "Genetic Programming II: Automatic Discovery of Reusable Programs," MIT Press, 1994.

[8] Montana, D.J., "Strongly Typed Genetic Programming," Evolutionary Computation, vol. 3, no.2, pp 19-230, 1995.

[9] Kennedy, J., Eberhart, R., "Particle Swarm Optimization," Proc. IEEE Int'l. Conf. On Neural Networks, (Perth, Australia), 1995.

[10] Luke, S., "Genetic Programming Produced Competitive Soccer Softbot Teams for RoboCup97," Proc. Third Ann. Genetic Programming Conf. (GP98), pp. 204-222, Morgan Kaufman, 1998.

[11] Eberhart, R.C., Shi., Y., "Comparison between Genetic Algorithms and Particle Swarm Optimization," Proc. 1998 Annual Conf. Evolutionary Programming, San Diego, March 1998.

[12] Vetterling, W.T., et. al., "Numerical Recipes in C: The Art of Scientific Computing/Disk V 2.02," Cambridge University Press, 1997.

[13] Ramoni, M., Sebastiani, P., "RoC Robust Bayesian Classifier: Users Manual, version 0.1B. © 1999, The Open University.