Proceedings of ICCC 2004, Băile Felix Spa-Oradea, Romania
pp. 321-325, 2004

A Comparison of Genetic Programming and Statistical Methods for Solving Prediction Problems

Mihai Oltean, Laura Sas

Abstract: A comparison of Genetic Programming and Statistical Methods for solving prediction problems is presented in this paper. Several numerical experiments using difficult real-world problems taken from PROBEN1 have been performed. Numerical results show that Genetic Programming performs better than the Statistical Methods.

1 Introduction

Evolutionary Algorithms (EA) [2] are new and powerful tools that may be used for solving difficult, real-world problems. Genetic Programming (GP) [1, 3, 4] is an evolutionary algorithm used for breeding a population of computer programs.

In this paper a comparison of Genetic Programming and classical Statistical Methods [7, 9] is presented. Several difficult real-world problems taken from PROBEN1 [6] are used for assessing the performance of the compared techniques.

Numerical results show that Genetic Programming performs better than the statistical methods in most of the cases.

The paper is organized as follows. Genetic Programming technique is described in section 2. The Statistical Methods used for comparison are briefly described in section 3. Data sets used for assessing the performance of the compared technique are described in section 4. Several numerical experiments are performed in section 5.

2 Genetic Programming

Genetic Programming (GP) [3, 4] is an evolutionary technique used for breeding a population of computer programs. GP individuals are represented as nonlinear structures (usually trees), which are modified using specific genetic operators.

For efficiency reasons, each GP program tree is stored as a vector using the Polish form (see [4] chapter 63). In Figure 1 is depicted a mathematical expression in Infix and Polish notation and the corresponding GP program tree.

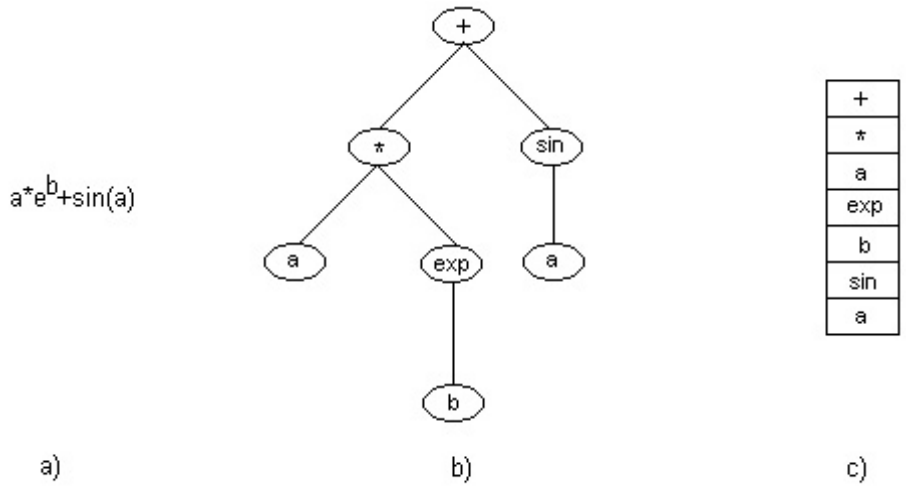

Figure 1: A mathematical expression in Infix form (a), Polish form (c) and the corresponding program tree (b).

Each element in this vector contains a function or terminal symbol. Since each function has a unique arity we can unambiguously interpret each vector that stores an expression in Polish notation. In this notation, a sub-tree of a program tree corresponds to a particular contiguous subsequence of the vector. When applying the crossover or the mutation operator, the subsequences that are exchanged or changed can easily be identified and manipulate.

3 Statistical Methods

For a better forecast of the three output variables we will use the training set for establish the model. We verify the model based on the available data, follow by the validation of the model. For this we will use the information given by the correlation coefficient.

If this coefficient is a value that approximate to one then the link between dependent variable (output) and independent variable (input) is stronger. In this case this kind of variable is considered an important data for the model.

We use the Last Square Method and The Maximum Plausibly Method [5, 7] to calculate the values of the parameters for each line of regression (these two methods provide the best estimation for the coefficients of a linear regression model).

Estimation is then completed by running a regression using the weighted dependent and independent variables to minimize the sum-of-squared residuals:

Y=XA+e,

min(Σe^2)=min(Σ(Y-XAest)^2),

with respect to the k-dimensional vector of parameters. In matrix notation, let W be a diagonal matrix containing the scaled W along the diagonal and zeroes elsewhere, and let Y and X be the usual matrices associated with the left and right-hand side variables. The weighted least squares estimator is computed using the formula:

b= (X'W'WX)^{-1}X'W'Wy

and the estimated covariance matrix is computed as $s^2(X$'W'WX).

In order to test if the values obtained are valid, we use the Wald test [9].

We must test the hypothesis regarding the errors (null mean, deviation standard constant – homoskedasticity, autocorrelation and correlation with dependent variable) in order to be able to say that the models are well specified.

For checking if the errors follow a normal distribution we use the Jarque-Bera test for a probability equal to 96.84%).

Durbin-Watson's test checks for the hypothesis of autocorrelation of errors. The models are well specified if the errors do not verify this hypothesis.

White homoskedasticity test proved that the errors have the same standard deviation. Because these hypotheses are verified our models are valid.

4 Data sets

Test problems used in the numerical experiments are taken from PROBEN1 (which have been adapted from UCI Machine Learning Repository [10]). Each problem taken from PROBEN1 has three versions. The first one reflects the task formulation as it was provided by the collectors, and the other two are random permutations of the examples, simplifying the problem to one of interpolation.

Building

The purpose of this problem is to predict the electric power consumption in a building. The problem was originally designed for predicting the hourly consumption of hot and cold water and electrical energy, based on the date, time of day, outdoor temperature, outdoor air, humidity, solar radiation, and wind speed. In this paper, the original problem was split into three subproblems (predicting the consumption of electrical power, hot water and cold water) because the GP technique is not designed for handling multiple outputs of a problem. The *Building* problem has 14 inputs restricted to the interval [0, 1]. The data set contains 4208 examples.

In PROBEN1 three different variants of each data set are given concerning the order of the examples. This increases the confidence that results are not depending on a certain distribution of the data into training, validation and test set. The original problem is called *Building1* and the other two versions are called *Building2* and *Building3*.

5 Numerical experiments

Several numerical experiments with Genetic Programming and Statistical Methods are performed in this section.

The test problems considered in this experiment are the problems *Building* taken from PROBEN1 [6]. Each dataset is divided into three subsets (training set, 50%; validation set, 25%; and test set, 25%) [6]. A method called *early stopping* is used to avoid overfitting of the population individuals to the particular training examples used [6]. This method consists of computing the test set performance for that chromosome having the smallest validation error during the search process. Using early stopping will increase the generalization performance.

The error function reported in this experiment is:

$$E = 100\frac{1}{N}\sum_{i=1}^{N}|e_i - o_i|.$$

where e_i is the expected output value (the value that must be predicted), o_i is the obtained value (the value obtained by the best individual) for the i^{th} example, and $N$is the number of examples.

The standard deviation of the obtained values (over 100 runs) is also computed in conjunction with the error. The error value of the best individual over all runs is also reported.

GP parameters are given in Table 5. The models used by the statistical methods for computing is given in Table 5. The parameters for the formulas in Table 5 are given in Table 5. Results are presented in Tables 5 (training set), 5 (validation set) and 5 (test set).

Table 1: GP parameters for the numerical experiment.

Parameter	**Value**
Population Size	50
Maximum Tree Depth	7
Number of Generations	51
Crossover rate	0.9
Mutation	1 mutation/chromosome
Function set	F = {+, -, *, %, *sin*, *exp*}
Terminal set	*Problem inputs*

Table 2: The models used by the statistical methods for training set. O_1, O_2 and O_3 are the values for the outputs. $i_1 \ldots i_{14}$ are the inputs and a_i, b_i and c_i are the parameters computed in Table 5.

$O_1 = a_1$+a_2*i_{10}+a_3*i_{11}+a_4*i_{12}+a_5+i_{13}+a_6*i_{14}+a_7*i_9+a_8*i_8+a_9*(i_1+i_2+i_3+i_4+i_5+i_6+i_7).
$O_2 = b_1$+b_2*(i4+i_5+i_6)+b_3*i_8+b_4*i_9+b_5*i_{10}+b_6*i_{11}+b_7*i_{12}.
$O_3 = c_1$+c_2*(i_4+i_5+i_6+i_7)+c_4*i_{10}+c_5*i_{11}+c_6*i_{12}+c_7*i_{13}.

Table 3: The parameters for the formulas in Table 5.

i	a_i	b_i	c_i
1	0.40416800	0.39696500	0.24642500
2	-0.01421000	0.01974700	-0.02202600
3	0.03843600	-0.00000001	0.02706700
4	-0.00740700	-0.00000003	-0.02538700
5	0.19680600	-0.03063700	-0.13264600
6	0.03507500	0.01030200	-0.01328500
7	-0.00000001	0.26968300	0.02282200
8	-0.00000001	0.06155900	
9	-0.00000005	-0.01129200	

The most important values are those obtained for the test set. In 6 cases (out of 9) the average error obtained by GP is better than the error obtained by the statistical methods. The best GP individual (obtained over 100 independent runs) is better than the statistical methods in 8 (out of 9) cases.

6 Conclusions and Further Work

Genetic Programming and Statistical Methods have been compared by using several real-world test data. Numerical experiments have shown that Genetic Programming performs better than the Statistical Methods for most of the considered examples.

There are some problems related to the Statistical Methods:

Table 4: The error for the training set. *Mean* stands for the mean of errors over 100 independent runs. *StdDev* stands for the standard deviation. *Best* is the error for the best individual in 100 runs.

Problem	**Genetic Programming**			**Statistical Method**	
	Mean	StdDev	Best	Error	Stddev
Building1-cw	4.5362	1.21274	3.0776	8.13	7.74
Building1-ep	7.93995	0.756772	6.20253	3.78	3.12
Building1-hw	8.82403	1.49875	2.99479	7.85	6.63
Building2-cw	5.52052	1.74698	3.29473	7.22	5.81
Building2-ep	8.38538	0.941862	6.50617	17.16	11.81
Building2-hw	12.0679	2.53784	5.51406	6.82	6.08
Building3-cw	5.16152	1.65589	3.23013	6.99	5.63
Building3-ep	8.19767	1.03619	6.16832	16.79	11.77
Building3-hw	11.8607	2.91395	5.60048	6.72	6.03

Table 5: The error for the validation set. *Mean* stands for the mean of errors over 100 independent runs. *StdDev* stands for the standard deviation. *Best* is the error for the best individual in 100 runs.

Problem	**Genetic Programming**			**Statistical Method**	
	Mean	StdDev	Best	Error	StdDev
Building1-cw	5.18751	1.41297	3.62963	8.17	6.39
Building1-ep	8.96849	0.951958	7.92709	4.15	4.90
Building1-hw	24.0678	4.1754	7.57291	10.37	11.15
Building2-cw	5.47665	1.7107	3.24047	7.13	5.77
Building2-ep	8.19103	0.966667	6.49338	16.67	11.69
Building2-hw	12.2773	2.6947	5.79551	12.36	10.07
Building3-cw	5.24555	1.76305	3.20515	7.12	5.64
Building3-ep	8.01835	1.09919	5.81517	17.07	11.94
Building3-hw	11.9865	2.90187	5.67449	11.30	10.35

Table 6: The error for the test set. *Mean* stands for the mean of errors over 100 independent runs. *StdDev* stands for the standard deviation. *Best* is the error for the best individual in 100 runs.

Problem	**Genetic Programming**			**Statistical Method**	
	Mean	StdDev	Best	Error	StdDev
Building1-cw	4.96382	1.77075	3.02192	7.25	7.16
Building1-ep	7.76524	0.950159	5.66175	3.82	4.13
Building1-hw	18.5429	3.38277	4.25391	11.30	12.11
Building2-cw	5.45819	1.72957	3.36457	6.89	6.03
Building2-ep	8.17698	0.929255	6.24531	16.47	12.04
Building2-hw	12.2586	2.74854	5.63794	12.97	11.62
Building3-cw	5.40887	1.75398	3.3291	7.38	6.03
Building3-ep	8.35785	1.03525	6.30203	16.88	11.88
Building3-hw	11.9338	2.88831	5.57017	10.32	11.72

- we have to establish the model (which is difficult because of the significance of the input variables);
- The binomial inputs in the model lead to a not singular variation and co variation matrix, leading to the exclusion of data from the model and conducting to more estimation errors;
- model uses codes for the real data. Some of this data's influence could be revealed through the use of *logit* and *probit* models.

There are some problems related to the Genetic Programming:

- we do not have a general method for selecting the optimal parameters for the method (i.e. the number of generation, the population size etc);
- The function set could have a major influence on results obtained by GP.

Further numerical experiments will be focused on comparing other evolutionary algorithms and classical methods for solving difficult real-world problems.

References

[1] Banzhaf, W., Langdon, W. B., *Some Considerations on the Reason for Bloat*, Genetic Programming and Evolvable Machines, Vol. 3, pp. 81-91, 2002.

[2] Goldberg, D. E., *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Reading, MA, 1989.

[3] Koza, J. R., *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, Cambridge, MA: MIT Press, 1992.

[4] Koza, J. R., Bennett III, F.H., Andre, D., Keane, M.A., *Genetic Programming III: Darwinian Invention and Problem Solving*, Morgan Kaufmann, San Francisco, California, 1999.

[5] Lehman, E. L.,*Testing Statistical Hypothesis*, Springer, New-York-Berlin, 1997

[6] Prechelt, L., *PROBEN1 – A Set of Neural Network Problems and Benchmarking Rules*, Technical Report 21/94, University of Karlsruhe, 1994.

[7] Stapleton, J. H., *Linear Statistical Models*, John Wiley&Sons, New-York – Chichester – Brisbane

[8] Syswerda, G., *Uniform Crossover in Genetic Algorithms*, *Proceeding of the* 3^{rd} *International Conference on Genetic Algorithms*, J.D. Schaffer (Ed.), Morgan Kaufmann Publishers, San Mateo, CA, pp. 2-9, 1989

[9] Tassi, P., *Methodes Statistiques*, Economica, Paris, 1989.

[10] *UCI Machine Learning Repository*, Available from www.ics.uci.edu/∼mlearn/MLRepository.html

Babeş-Bolyai University, Romania
Faculty of Mathematics and Computer Science
Kogălniceanu 1, Cluj-Napoca, 3400, Romania
E-mail: moltean@.cs.ubbcluj.ro, sas_laura@yahoo.com