International Journal of Scientific Engineering and Technology (ISSN : 2277-1581)
Volume 2 Issue 5, pp : 311-316 1 May 2013

Computational Hybrids Towards Software Defect Predictions

Manu Banga
Department of Computer Sciences and Engineering,
Maharishi Maharkandeshwar University Solan, HP, India, Email : manubanga@gmail.com

Abstract

In this paper, new computational intelligence sequential hybrid architectures involving Genetic Programming (GP) and Group* Method of Data Handling (GMDH) *viz. GP-GMDH. Three linear ensembles based on (i) arithmetic mean (ii) geometric mean and (iii) harmonic mean are also developed. We also performed GP based feature selection. The efficacy of Multiple Linear Regression (MLR), Polynomial Regression, Support Vector Regression (SVR), Classification and Regression Tree (CART), Multivariate Adaptive Regression Splines (MARS), Multilayer FeedForward Neural Network (MLFF), Radial Basis Function Neural Network (RBF), Counter Propagation Neural Network (CPNN), Dynamic Evolving Neuro–Fuzzy Inference System (DENFIS), TreeNet, Group Method of Data Handling and Genetic Programming is tested on the NASA dataset. Ten-fold cross validation and t-test are performed to see if the performances of the hybrids developed are statistically significant.

Keywords:- MLR, SVR, CART, MARS, MPFF, RBF

1. Introduction

The software defect prediction is one of the most critical problems in software engineering. Software cost development is related to how long and how many people are required to complete a software project. Software development has become an essential question [1] because many projects are still not completed on schedule, with under or over estimation of efforts leading to their own particular problems [2]. Therefore, in order to manage budget and schedule of software projects [3], various software cost estimation models have been developed. A major problem of the software cost estimation is first obtaining an accurate size estimate of the software to be developed [4] because size is the most important single cost driver [5].

The main objective of the present work is to propose new computational intelligence based hybrid models that estimate the software defects accurately. The rest of the paper is organized as follows. Section 2 reviews the research done in the field of software defect prediction. Section 3 overviews the techniques applied in this study. Section 4 describes briefly the NASA dataset that is analyzed by proposed Hybrid Intelligent Systems. Section 5 presents proposed Hybrid Intelligent Systems developed in this study. Section 6 presents the results and discussions. Finally, Section 7 concludes the paper.

2. Literature Review

Various software development effort estimation models have been developed over the last four decades. The most commonly used methods for predicting software development efforts are Function Point Analysis and Constructive Cost Model (COCOMO) [4]. Function Point Analysis was developed first by Albrecht (1979) (www.IFPUG.Org). Function point analysis is a method of quantifying the size and complexity of a software system in terms of the functions that the system delivers to the user [9]. The function does not depend on the programming languages or tools used to develop a software project [1]. COCOMO is developed by Boehm [2]. It is based on linear-least-squares regression. Using line of code (LOC) as the unit of measure for software size itself contains so many problems [10]. These methods failed to deal with the implicit non-linearity and interactions between the characteristics of the project and effort

In recent years, a number of alternative modeling techniques have been proposed. They include artificial neural networks, analogy-based reasoning, and fuzzy system. In analogy-based cost estimation, similarity measures between a pair of projects play a critical role. This type of model calculates distance between the software project being estimated and each of the historical software projects and then retrieves the most similar project for generating an effort estimate. Later, Vinaykumar et al. [6] used wavelet neural networks for the prediction of software defects. Unfortunately the accuracy of these models is not satisfactory so there is always a scope for new software cost estimation techniques.

Tosun et al. proposed feature weighting heuristics for analogy-based effort estimation models using principal components analysis (PCA). Mittas and Angelis proposed statistical simulation procedures involving permutation tests and bootstrap techniques in order to test the significance of the difference between the accuracy of two prediction

methods: the estimation by analogy and the regression analysis.

3. Overview of the techniques employed

In the following, we now present an overview of the techniques applied in this paper.

3.1 Group Method of Data Handling (GMDH)

The group method of data handling (GMDH) was introduced by Ivakhnenko [27] in 1966 as an inductive learning algorithm for modeling of complex systems. It is a self-organizing approach based on sorting-out of gradually complicated models and evaluation of them using some external criterion on separate parts of the data sample [28]. The GMDH was partly inspired by research in Perceptrons and Learning Filters. GMDH has influenced the development of several techniques for synthesizing (or "self-organizing") networks of polynomial nodes. The GMDH attempts a hierarchic solution, by trying many simple models, retaining the best, and building on them iteratively, to obtain a composition (or feed- forward network) of functions as the model. The building blocks of GMDH, or polynomial nodes, usually have the quadratic form: The GMDH neural network develops on a data set. The data set including independent variables (x_1, x_2,... , x_n) and one dependent variable y is split into a training and testing set. During the process of learning a forward multilayer neural network is developed by observing the following steps:

1. In the input layer of the network n units with an elementary transfer function y = xi are constructed. These are used to provide values of independent variables from the learning set to the successive layers of the network.
2. When constructing a hidden layer an initial population of units is generated. Each unit corresponds to the Ivakhnenko polynomial form:
   $y = a + bx_1 + cx_2 + dx_{12} + ex_1x_2 + fx_{22}$ or
   $y = a + bx_1 + cx_2 + dx_1x_2$
   Where y is an output variable; x_1, x_2 are two input variables and a, b, f are parameters.
3. Parameters of all units in the layer are estimated using the learning set.
4. The mean square error between the dependent variable y and the response of each unit is computed for the testing set.
5. Units are sorted out by the mean square error and just a few units with minimal error survive. The rest of the units are deleted. This step guaranties that only units with a good ability for approximation are chosen.
6. Next the hidden layers are constructed while the mean square error of the best unit decreases.
7. Output of the network is considered as the response of the best unit in the layer with the minimal error.

The GMDH network learns in an inductive way and tries to build a function (called a polynomial model), which would result in the minimum error between the predicted value and expected output. The majority of GMDH networks use regression analysis for solving the problem. The first step is to decide the type of polynomial that the regression should find. The initial layer is simply the input layer. The first layer created is made by computing regressions of the input variables and then choosing the best ones. The second layer is created by computing regressions of the values in the first layer along with the input variables. This means that the algorithm essentially builds polynomials of polynomials. Again, only the best are chosen by the algorithm. These are called survivors. This process continues until a pre-specified selection criterion is met.

3.2 Genetic Programming (GP)

Genetic programming (GP) is an extension of genetic algorithms (GA). It is a search methodology belonging to the family of evolutionary computation (EC). GP mainly involve functions and terminals. GP randomly generates an initial population of solutions. Then, the initial population is manipulated using various genetic operators to produce new populations. These operators include reproduction, crossover, mutation, dropping condition, etc. The whole process of evolving from one population to the next population is called a generation. A high -level description of GP algorithm can be divided into a number of sequential steps :

- Create a random population of programs, or rules, using the symbolic expressions provided as the initial population.
- Evaluate each program or rule by assigning a fitness value according to a predefined fitness function that can measure the capability of the rule or program to solve the problem.
- Use reproduction operator to copy existing programs into the new generation.
- Generate the new population with crossover, mutation, or other operators from a randomly chosen set of parents.
- Repeat steps 2 onwards for the new population until a predefined termination criterion has been satisfied, or a fixed number of generations has been completed.
- The solution to the problem is the genetic program with the best fitness within all the generations.

In GP, crossover operation is achieved first by reproduction of two parent trees. Two crossover points are then randomly selected in the two offspring trees. Exchanging sub -trees, which are selected according to the crossover point in the parent trees, generates the final offspring trees. The obtained offspring trees are usually different from their parents in size and shape. Then, mutation operation is also considered in GP. A single parental tree is first reproduced. Then a mutation point is randomly selected from the reproduction, which can be either a leaf node or a sub- tree. Finally, the leaf node or the sub-tree is replaced by a new leaf node or sub-tree generated randomly. Fitness functions ensure that the evolution goes toward optimization by calculating the fitness value for each individual in the population. The fitness value evaluates the performance of each individual in the population.

GP is guided by the fitness function to search for the most efficient computer program to solve a given problem. A simple measure of fitness [31] is adopted for the binary classification problem which is given as follows.

$$\text{Fitness (T)} = \frac{\text{no of samples classified correctly}}{\text{no of samples for training during evaluation}}$$

We used the GP implementation available at http://www.rmltech.com

3.3 Counter Propagation Neural Network (CPNN)

The counter propagation network is a competitive network. The uni-directional PNN has three layers. The main layers include an input buffer layer, a self-organizing Kohonen layer and an utput layer which uses the Delta Rule to modify its incoming connection weights. Sometimes this layer is called a Grossberg Outstar layer. The forward-only counter propagation network architecture, consists of three slabs: an input layer (layer 1) containing n fan out units that multiplex the input signals x_1, x_2,….., x_n, (and m units that supply the correct output signal values y_1,y_2 ,.., y_m to the output layer), a middle layer (layer 2 or Kohonen layer) with N processing elements that have output signals z_I, z_2,..., z_N, and a final layer (layer 3) within processing elements having output signals y_1', y_2',…, y_m'. The outputs of layer 3 represent approximations to the components y_1, y_2,…, y_m of $y = f(x)$ The input layer in CPNN performs the mapping of the multidimensional input data into lower dimensional array. The mapping is performed by use of competitive learning, which employs winner-takes-it-all strategy . The training process of the CPNN is partly similar to that of Kohonen self-organizing maps. The Grossberg layer performs supervised learning. The network got its name from this counter-posing flow of information through its structure.

3.4 Support Vector Regression (SVR)

The SVR is a powerful learning system that uses a hypothesis space of linear functions in a high-dimensional space, trained with a learning algorithm from optimization theory that implements a learning bias derived from statistical learning theory. SVR uses a linear model to implement non-linear class boundaries by mapping input vectors non-linearly into a high dimensional feature space using kernels. The training examples that are closest to the maximum margin hyperplane are called support vectors. All other training examples are irrelevant for defining the binary class boundaries. The support vectors are then used to construct an optimal linear separating hyperplane (in case of pattern recognition) or a linear regression function (in case of regression) in this feature space. The support vectors are conventionally determined by solving a quadratic programming (QP) problem.

3.5 Classification and Regression Tree (CART)

CART was introduced by Breiman et al. [36] can solve both classification and regression problems (http://salford-systems.com). Decision tree algorithms induce a binary tree on a given training data, resulting in a set of 'if–then' rules. These rules can be used to solve the classification or regression problem. The key elements of a CART analysis [36] are a set of rules for: (i) splitting each node in a tree, (ii) deciding when a tree is complete; and (iii) assigning each terminal node to a class outcome (or predicted value for regression). We used the CART implementation available at http://salford-systems.com.

Multivariate adaptive regression splines (MARS) is an innovative and flexible modeling tool that automates the building of accurate predictive models for continuous and binary dependent variables. It excels at finding optimal variable transformations and interactions, the complex data structure that often hides high-dimensional data. This approach to regression modeling effectively uncovers important data patterns and relationships that are difficult, if not impossible, for other methods to reveal.

3.6 Tree Net

DENFIS was introduced by Kasabov and Song [38]. DENFIS evolve through incremental, hybrid (supervised/unsupervised) learning, and accommodate new input data, including new features, new classes, etc., through local element tuning. New fuzzy rules are created and

updated during the operation of the system. At each level, the output of DENFIS is calculated through a fuzzy inference system based on most activated fuzzy rules, which are dynamically chosen from a fuzzy rule set. A set of fuzzy rules can be inserted into DENFIS before or during its learning process. Fuzzy rules can also be extracted during or after the learning process. It makes use of a new concept of 'ultra slow learning' in which layers of information are gradually peeled off to reveal structure in data. TreeNet models are typically composed of hundreds of small trees, each of which contributes just a tiny adjustment to the overall model. TreeNet is insensitive to data errors and needs no time-consuming data preprocessing or imputation of missing values. Tree Net is resistant to overtraining and is faster than a neural net.

3.7 Regressions Analysis:

It is supervised learning type where we are using Multiple Linear Regressions with ordinary least squares technique. As a consequence, the weights of the connections between the kernel layer and the output layer are determined..

4. Data Description and Data Preprocessing

constructing ensemble system we have chosen the three best techniques viz., GMDH, GP and CPNN from stand-alone mode. These three techniques have yielded the best RMSE values in the 10 -fold cross validation method of testing. We constructed ensembles using three methods. They are Arithmetic Mean (AM), Harmonic Mean (HM) and Geometric Mean (GM). The proposed Ensemble system is depicted in Figure 1.

The NASA data is obtained from Promise Repository. (Entire dataset contains information about 4109 projects. The dataset consist of 18 attributes. These attributes are also divided into sub-attributes, thereby making the total number of attributes 105. The first cleaning step was to remove the projects having null values for the attribute named Summary of Work Effort. Secondly regarding summary of work effort only 1538 project values are given for the five attributes *viz.* Input count, Output, Enquiry, File and Interface. If we consider more attributes then we get only a few projects which are not sufficient for machine learning techniques. Hence, finally, we considered 1538 projects values with five attributes to do train and test several intelligent models. Finally, we normalized the data set. The effectiveness of our proposed hybrid intelligent systems is tested on this normalized dataset.

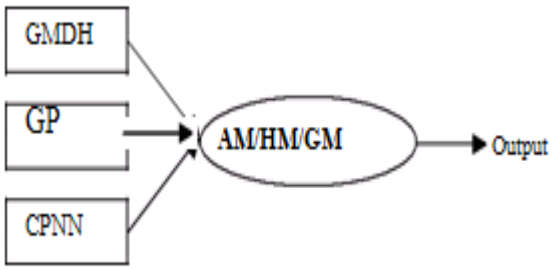

Figure.1 Ensemble System

5. Proposed Hybrid Intelligent Systems

The fundamental assumption in computational intelligence paradigm is that hybrid intelligent techniques tend to outperform the stand-alone techniques. We proposed 6 new hybrid architectures for software cost estimation and compare their performances based on RMSE values

.

5.1 Ensemble System

We first implemented linear ensemble systems. Ensemble systems exploit the unique strengths of each constituent model and combine them in same way.

5.2 Recurrent Genetic Programming (RGP) architecture

The second proposed architecture is a recurrent architecture for Genetic Programming (RGP) in which output of the GP is fed as an input to the GP. This is analogous to recurrent neural networks having feedback loop where output can be fed back as input [41]. However, the difference is that we wait until GP converges and yileds the predictions. These predictions along with the original input variables are fed as inputs to another GP afresh. The flow diagram of the recurrent architecture for Genetic Programming (RGP) is depicted in Figure 2. The idea here is to investigate if the recurrent nature of the hybrid would improve the RMSE of the first GP.

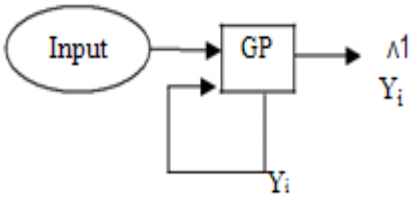

Figure.2 Recurrent architecture for GP (RGP)

5.3 GP-GP hybrid

It is observed that there are some features in the dataset that are contributing negatively to the prediction accuracy of all

the models. Hence, we resorted to feature selection (F.S). We used GP for feature selection. Using GP based feature selection we selected four most important variables for training. Accordingly, in the proposed hybrid first important features are selected using GP and then those are fed to GP for predictions resulting in GP- GP hybrid. The architecture of proposed GP-GP hybrid is depicted in Figure 3.

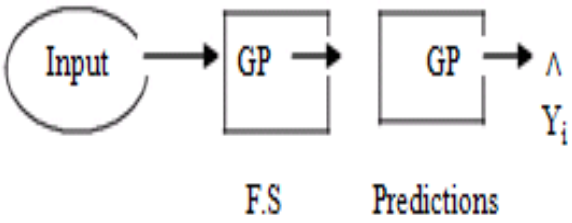

Figure.3 GP-GP Hybrid Architecture

5.4 GMDH-GP hybrid

As an extension to this work, it is worth investigating the boosting of well performing techniques with each other. Accordingly, we proposed a new sequential hybrid in which the predictions of GMDH along with input variables are fed as input to GP for predictions, resulting in GMDH-GP hybrid. The architecture of GP-GMDH hybrid is depicted in Figure 4.

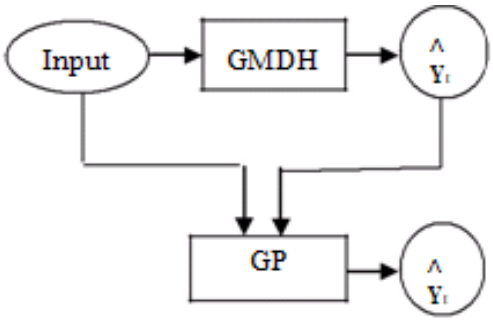

Figure.4 GMDH-GP Sequential hybrid

5.5 GP-GMDH hybrid

We also proposed another sequential hybrid to explore the boosting power of GP with GMDH. In this new hybrid, the predictions of GP along with input variables are fed as input to GMDH for predictions. The architecture of GP-GMDH hybrid is depicted in Figure 5.

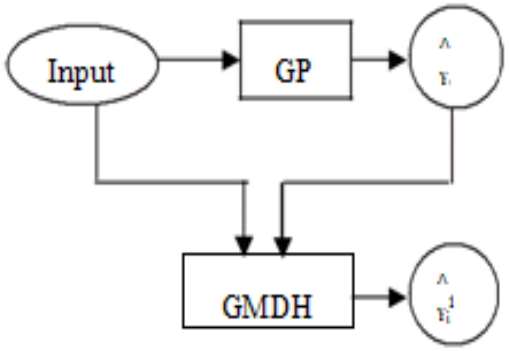

Figure.5 GP-GMDH Sequential hybrid

6. Results and discussion

We used ISBSG data set, which contains 1538 projects and five independent variables and one dependent variable.

We employed GMDH, GP, CPNN, MLR, Polynomial Regression, SVR, CART, MARS, MLFF, RBF, DENFIS, and TreeNet. We performed 10-fold cross validation throughout the study and the average results are presented in Table 1.

Table 1: Average RMSE of 10 fold cross validation

SN	METHOD	RMSE (TEST)
1	**GMDH**	**0.03784**
2	**GP**	**0.03794**
3	CPNN	0.04499
4	CART	0.04561
5	TREENET	0.04565
6	MLP	0.04817
7	MLR	0.04833
8	DENFIS	0.04837
9	MARS	0.0487
10	SVR	0.0492
11	RBF	0.05167
12	Polynomial Regression	0.05327

It is observed that GMDH performed the best with least RMSE value of 0.03784 and GP stood a close second with an average RMSE value of 0.03794 among all stand alone techniques tested followed by CPNN, CART, TREENET, MLP, MLR, DENFIS, MARS, SVR, RBF and Polynomial Regression in that order.

We also implemented three linear ensemble systems using AM, GM and HM to exploit the unique strengths of the best performed stand alone techniques GMDH, GP and CPNN. We notice that AM based ensemble system has outperformed GM

based and HM based ensemble techniques. The results are presented in Table 2. However, they are not so spectacular when compared to the best performing stand- alone methods. This is evident from the nature of the AM, Gm and HM.

Table 2: Average RMSE of ensemble models

SN	METHOD	RMSE (Test data)
1	AM	**0.0421**
2	GM	0.04403
3	HM	0.0455

The results of the hybrids are presented in Table 3. Here we observe that all the proposed hybrid models RGP, GP-RGP, GP- GP, GMDH-GP and GP-GMDH outperformed all other stand-alone techniques due to the synergy that took place in hybridizing them. From the results of GP-GP and GP-RGP it is inferred that the features selected by GP helped to boost the performance of GP and RGP.

We further explored the boosting power of GP with another well performing technique GMDH and boosting power of GMDH with GP. We observed that GP-GMDH yielded least average RMSE value of 0.02833 and GMDH-GP stood second with an average RMSE value of 0.03098 among all the hybrids tested. They are followed by RGP, GP-RGP and GP-GP in that order.

We also performed t-test to test whether the difference in RMSEs obtained by the top five methods *viz.*, GP-GMDH, GMDH-GP, RGP, GP-RGP and GP-GP is statistically significant or not. Thus, the t-statistic values computed for those hybrids are presented in Table 3. The calculated t-statistic values are compared with 2.1, which is the tabulated t-statistic value at $n_1+n_2-2=10+10-2=18$ degrees of freedom at 5 % level of significance. That means, if the computed t-statistic value between two methods is more than 2.1, then we can say that the difference between the techniques is statistically significant. The t-statistic value between GP-GMDH and RGP is 2.47694 whereas that between GP-GMDH and GP-RGP is 2.83543 and in the case of GP-GMDH and GP-GP it is 4.27465. Considering GP-GMDH as best performer, it is observed that the difference between RGP, GP-RGP and GP-GP is statistically significant. The t-statistic value between GP-GMDH and GMDH-GP is 1.6972 which is less than 2.1 and hence the difference between GP -GMDH and GMDH-GP is statistically insignificant.

7. Conclusions

This paper presents new computational intelligence sequential hybrids involving GP and GMDH for software cost estimation. Throughout the study 10-fold cross validation is performed. Besides GP and GMDH, we tested a host of techniques on the ISBSG dataset. The proposed GP- GMDH and GMDH-GP hybrids outperformed all other stand-alone and hybrid techniques. Hence, we conclude that the GP-GMDH or GMDH-GP model is the best model among all other techniques for software cost estimation.

Table 3: Average RMSE of hybrids models

SN	METHOD	RMSE (Test data)	t-test value
1	RGP	0.03275	2.47694
2	GP-RGP	0.03345	2.83543
3	GP-GP	0.03676	4.27465
4	GMDH-GP	**0.03098**	1.6972
5	GP-GMDH	**0.02833**	-

References

I. A.J. Albrecht and J.E. Gaffney, "Software function, source lines of code, and development effort prediction: a software science validation", *IEEE Transactions on Software Engineering*, 1983, 9(6), pp. 639–647.

II. B.W. Boehm, "Software Engineering Economics", Prentice-Hall, Englewood Cliffs, NJ, USA, 1981.

III. L.H. Putnam, "A general empirical solution to the macro software sizing and estimation problem", *IEEE Transactions on Software Engineering*, 1978, 4(4), pp. 345–361.

IV. B. Kitchenham, L.M. Pickard, S. Linkman and P.W. Jones, "Modeling software bidding risks", *IEEE Transactions on Software Engineering*, 2003, 29(6), pp. 542–554.

V. J. Verner and G. Tate, "A Software Size Model", *IEEE Transactions on Software Engineering*, 1992, 18(4), pp. 265-278.

VI. K. Vinaykumar, V. Ravi, M. Carr and N. Rajkiran, "Software development cost estimation using wavelet neural networks", *Journal of Systems and Software*, 2008, 81(11), pp. 1853-1867.

VII. T. Foss, E. Stensrud, B. Kitchenham and I. Myrtveit, "A simulation study of the model evaluation criterion MMRE", *IEEE Transactions on Software Engineering*, 2003, 29(11), pp. 985–995.

VIII. Z. Xu and T.M. Khoshgoftaar "Identification of fuzzy models of cost estimation", *Fuzzy Sets and Systems*, 2004, 145(11), pp. 141-163.

IX. J.E. Matson, B.E Barrett and J.M. Mellichamp, "Software development cost estimation using function points", *IEEE Transactions on Software Engineering*, 1994, 20(4), pp. 275–287.

X. A.Idri, T.M. Khosgoftaar and A. Abran, "Can neural networks be easily interpreted in software cost.