

BETTER SOFTWARE DEFECT PREDICTION USING EQUALIZED LEARNING WITH MACHINE LEARNERS

Kim Kaminsky, Gary D. Boetticher
University of Houston - Clear Lake
2700 Bay Area Boulevard
Houston, TX 77058 USA
kaminsky@cl.uh.edu boetticher@cl.uh.edu

ABSTRACT

Many software organizations frequently do not allocate enough resources for software quality. As a consequence, an automated process for predicting software defects is an important research issue to the software engineering community. Researchers seek to build accurate and reliable predictors based upon legacy data. This is a very difficult task when one considers that software quality defect data is an “implicitly data-starved” domain due to the skewed distribution of the data. The severe lack of data problem is compounded by the fact that the interesting data (software modules with moderate to high defect rates) occurs on a relatively infrequent basis. One potential solution seeks to “balance” the data in order to compensate for the skewness. This is accomplished by replicating those instances of interest in order to emphasize their contribution to the modeling process.

To assess the feasibility of this technique, a series of Genetic Programming experiments are conducted using multiple data sets from various NASA-based repositories of defect data. The results demonstrate the feasibility of balancing the data.

KEYWORDS

Defect prediction, equalized learning, machine learning, genetic programs, software engineering, software quality

1. Introduction

Defect prediction enables software practitioners to detect, track and resolve product anomalies that might affect human safety and lives. Defect prediction identifies problems earlier in the life-cycle process, thus lowering software costs by as much as 100 times [1] and improving customer satisfaction.

Performing defect prediction is a difficult task. The Capability Maturity Model[®] Integration (CMMi) model identifies defect prevention as a General Practice that occurs in level 5 organizations [2]. This implies that defect prevention is reserved for organizations that have a very advanced software process. In September 2003, SEI [3] also reported that it had appraised 1342 organizations

from 1999 to 2003. The SEI found that only 9.2 percent of these organizations are at level 5.

Thus, defect prevention, a very important component of the software lifecycle, does not receive the attention (resources) it deserves. One reason is that many organizations tend to under-appropriate resources to their software quality group. This raises the question, *How to find more software defects with less effort?*

One solution incorporates an automated process for detecting defects within the life-cycle process where legacy quality data serves as the basis for building models. A major obstacle in successfully achieving this initiative is the severe lack of data. Operating in a data-starved domain makes it difficult to formulate reliable models.

Data-starvation assumes two forms: explicit or implicit. Explicit data-starvation means there are few instances of data. This phenomenon occurs frequently in the Software Engineering domain for several reasons. Many corporations are reluctant to share their proprietary information. Also, since software projects take such a long time to complete, establishing benchmarks within an organization is a difficult task.

Implicit data-starvation occurs when a data distribution is highly skewed so those instances of interest occur on an infrequent basis. In the software defect domain there might be a 100 to 1 ratio between the number of instances and the range of defect values. Furthermore, in the case of “interesting values” (e.g. software components with 5 or more defects), this ratio might be even greater. It is not unusual for up 50 percent of the components with few defects to be concentrated at the lowest 5 percent of a distribution curve.

Implicit and explicit data starvation forces researchers to consider innovative solutions to predicting software defects. One approach uses machine learners to predict defects.

Machine learners, such as genetic programs (GPs), artificial neural networks (ANNs), case-based reasoning (CBR), and rule induction (RI), are ideally suited to this situation because they handle noisy and incomplete data

well. At the core of these learners is induction, a process of inferring a generalization based upon a sample set. Among the possible machine learners available, Genetic Programs (GP) are a good choice for the following reasons:

- It is able to produce a human-readable solution in the form of a polynomial equation.
- It requires minimum human intervention with very little previous domain knowledge.
- Relevant attributes receive greater emphasis.
- GPs tend to scale well with problem size.
- GP modeling does not introduce human bias (quantity/quality of rules) in solution formation.
- GPs allow better abstract representation [4].
- GPs can learn on small data samples.

Despite all the reasons for using GPs, if the training data is skewed toward lower-end defect values, the GP's solution will be oriented towards classifying test data with smaller defect values.

To address the implicit data starvation issue, a data pre-processing approach is proposed, called equalized learning. Equalized learning considers the instance distribution of a training set and replicates sparse instances so that the training data contains an equal distribution of instances. In the context of software defect detection, equalized learning means replicating instances with mid to high defect values.

NASA, recognizing the importance of software defect prediction, has established the Independent Verification and Validation (IV&V) Facility in 1993 as part of a strategy to provide the highest achievable levels of safety and cost-effectiveness for mission critical software. As part of this initiative, NASA's IV&V has established a Metrics Data Program (MDP) Data Repository [5]. This repository contains metrics and associated defect rates at the function/method level.

This paper describes a set of experiments in which various training data sets are equalized by replicating higher valued defect instances prior to training a GP. The goal is to determine whether a better GP-based model can be constructed as a consequence of equalized learning. These experiments use four different data sets from the NASA repository: KC1, KC2, PC1, and CM1. Two sets of experiments are conducted on each data set. One sub-experiment trains on the original skewed distributed data and a second trains on an equalized data set. Performing these experiments determines whether equalization is a feasible approach in solving the implicit data starvation issue.

The paper is organized as follows. Section 2 discusses related research in terms of machine learners applied to defect detection and data preparation. Section 3 describes the NASA project data sets. Section 4 offers an overview of equalized learning. Section 5 describes a series of Genetic Programming experiments. The results are discussed in section 6. Finally, sections 7 and 8 present a conclusion and future directions.

2. Related Work

Data Defect Prediction and Machine Learners

Most statistical and software metric-based models struggle with formulating accurate defect predictions [6]. Fenton [6] considers poor quality data as a major cause for the problem. One approach that addresses this problem is the application of expert systems and machine learners [6..11] in formulating a predictor. Such an approach is plausible since expert systems and machine learners handle noisy data and uncertainty rather well.

Fenton [6] proposes a Bayesian Belief Network called AID (Assess, Improve and Decide) for predicting defects. The tool was tested on 28 projects from the Philips Software Centre. Though the results of this study were promising, to properly train the network a great deal of data was required [12].

Applying Neural Networks to detect defects has received a great deal of attention. Neural Networks have been applied to predict defects in a chemical processing plant. The results were 10 to 20 times better than the application of traditional methods [13]. Hochmann extends the use of Neural Networks to software defects [9]. In his first study, a Genetic Algorithm develops optimal back-propagation networks to detect software defects [9]. In a second study, an Evolutionary Neural Network, (ENN), is compared to Discriminate Analysis. The error rates for Discriminate Analysis were much higher than for ENNs. A z-test proved the statistical significance of these findings [9].

Inductive Learning Programs have been explored as a tool for predicting fault density in C++ classes [7]. Further, Genetic Programs have been applied to the problem of defect detection [8].

Machine Learners and Data Preparation

Recognizing the problems with Neural Networks, when applied to software defects and the multicollinearity of the data involved, Neumann [10] developed an enhanced technique for risk categorization called PCA-ANN. This approach combines statistics, pattern recognition and Neural Networks. PCA-ANN enhances the Neural Network by orthogonalizing and normalizing the input data. PCA-ANN performed significantly better than a pure Neural Network [9]. Hochmann, recognizing the importance of data preparation, used Principal Component Analysis to prepare the data set [9]. Finally, Mizuno proposes data equalization to improve an Artificial Neural Network in technical analysis of the stock market [14].

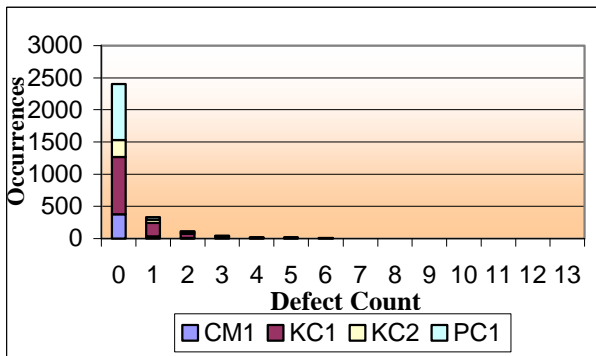
3. NASA Data Sets

The data sets, KC1, KC2, CM1 and PC1, for the machine learning experiments come from a variety of NASA projects. KC1 contains 43,000 lines of C++ code. After eliminating redundant data, there are 1204 functions. KC2 contains over 43,000 lines of C++ code with 379 functions. CM1 has 20,000 lines of C code and 498 functions. PC1 consists of 40,000 lines of C code with 941 functions. For the experiments, each function (module) will correspond to an instance of data.

Each instance contains 21 software metrics based on the product’s size, complexity and vocabulary. Size metrics include *lines of code*, *executable lines of code*, *comment lines*, *blank lines*, *number of lines with both code and comments*, and *branch count*. Metrics based on product complexity includes *cyclomatic complexity*, *essential complexity*, and *module design complexity*. The other 12 metrics are vocabulary metrics that include *Halstead length*, *volume*, *level*, *difficulty*, *intelligent content*, *programming effort*, *error estimate*, *programming time*, *unique operators*, *unique operands*, *total operators*, and *total operands*. The final column for each instance contains the defect count for each function (module).

All four data sets exhibit the data starvation pattern. Figure 1 shows all the instances from all the NASA data sets. Each stacked bar shows the number of instances that had the corresponding number of defects. Thus, there are 2407 total instances that had 0 defects. These 2407 functions account for eighty-two percent of all the instances. Furthermore, 11 percent of all instances had one defect, and 4 percent of the instances had two defects. The remaining 3 percent had 3 or more defects. Thus, 97 percent of the instances (those with 0, 1, or 2 defects) had defects within only 21.4 percent of the defect range. Twenty-one percent is 0..2, or 3, divided by 0..13, or 14.

Figure 1: Data Distribution for NASA Data Sets



4. Equalized Learning

Equalized learning is a process of balancing instance frequency within a data set. It involves three steps. The first step calculates a histogram by counting the number of instances of training data for each defect value. The second step uses the distribution information for replicating the instances so each instance value occurs on an equal basis. The third step involves shuffling the samples. [14].

Mathematically, equalized learning can be represented with the following equation:

$$Num. of Duplicates = Floor(Max Inst. / Attrib. Inst.) \quad Eq. 1$$

where

Maximum Instances refers to the attribute with the highest instance count;

Attrib. Inst. is the instance count of a particular attribute; and

Floor is a mathematical floor function.

This approach gives every discrete value equal weighting. The reason is that it is equally important to predict a software function which has 13 defects, as it is to predict zero defects. Misclassification raises type-I and type-II statistical errors. As seen in Figure 1, the majority of instances for the NASA data sets are skewed towards the lower defect values. Thus, there is a tendency for any machine learner to underestimate the actual defect value for a particular function (module).

5. Machine Learning Experiments

Overview of Genetic Programs

Genetic Programs (GPs) solve problems by genetically breeding a population of individuals, or chromosomes, over several generations. Inspired by theories of evolution, Genetic Programs use the analogy of evolutionary operators on chromosomes to optimize a fitness function. A fitness function assesses the goodness of a *chromosome*. The goodness of a chromosome serves as the basis for propagation decisions.

An implementation of a GP starts with a population of individuals, usually represented as tree structures. Each tree, or chromosome, represents a potential solution to a problem. Collectively, the set of chromosomes is known as a *population*. This population ‘reproduces’ to create the next generation. The act of reproduction calculates a fitness value for each chromosome. The fitness value determines which chromosomes will “mate” and propagate into the next generation. The “most fit” individuals are more likely to survive into future generation through either reproduction or cloning. “Least fit” individuals tend to be eliminated from the population. This process continues until an acceptable solution is found or a specific number of generations are reached.

This research uses the following fitness function in the experimentation process:

$$Fitness = 1 + e^{(7*(1-n-k)/(n-1) * Se^2 / Sy^2)} \quad Eq. 2$$

where

k corresponds to the number of inputs;

n represents the number of valid results;

Se is the standard error; and

Sy equals the standard deviation.

The fitness function ranges from 1 through 1067. All values are scaled to range from 1 through 1000.

Once two individuals within a population are identified, reproductive, or *recombination* may occur. This involves two steps: *crossover* and *mutation*. Crossover takes two trees, chooses a random branch, and then swaps the corresponding sub-trees.

Mutation alters the genetic material of a chromosome by slightly modifying a portion of a chromosome. It promotes diversity within a population by randomly adding in these variations. The approach prevents a GP solution from getting locked within a local minima or maxima, which is a problem experienced by most optimization algorithms [15, 16].

GPs solve optimization and induction problems particularly well. While machine learners, in general, excel at these types of problems, Genetic Programs provide the additional advantage of a symbolic high-level representation of the problem solution. A high-level representation facilitates a faster and easier interpretation of the results [17].

Experiment Description

To verify the accuracy of the equalized learning approach, four independent GP experiments are conducted on four different NASA data sets (CM1, KC1, KC2, and PC1 respectively). Each experiment consists of two independent sub-experiments. The first sub-experiment builds a GP model based on the original, unequalized data. The second sub-experiment is constructed on equalized data.

Each sub-experiment consists of twenty trials. Repeating each experiment twenty times insures that there are enough result samples for comparing results between the two sub-experiments.

In order to maintain experimental consistency, all sub-experiments are conducted with the following GP settings: selection is based on rank; a maximum of 50 generations per trial; initial chromosome length can not exceed 2000 characters; and a population size of 1000 chromosomes.

Experimental Results

There is a significant difference between the two sub-experiments for the CM1 data set. The unequalized data set experiments produce an average fitness value of 2, while the equalized data experiments have an average fitness value of 55.7. Applying a t-test to the 20 trials for each sub-experiment generates a value of 1.72 percent. Since it is less than 5 percent, the null hypothesis that the two means are equal is rejected. Thus, it is verified that there is a significant statistical difference between the two groups for this sub-experiment. Table 1 shows these results.

Table 1: Experimental t-Test Results for CM1 data
t-Test: Paired Two Sample for Means

	Original	Equalized
Mean	2	55.7
Variance	0	925.4842105
Observations	20	20
Pearson Correlation	#DIV/0!	
Hypothesized Mean Difference	0	
df	19	
t Stat	-7.894139136	
P(T<=t) one-tail	1.02073E-07	
t Critical one-tail	1.729131327	
P(T<=t) two-tail	2.04146E-07	
t Critical two-tail	2.093024705	

The sub-experiments on the KC1 data set show the same pattern. The average fitness values for the unequalized and equalized data sets are 2 and 8.75 respectively. The t-test result is 1.72 percent. Once again, the null-hypothesis,

that the two means are equal, is rejected. Table 2 depicts these results.

Table 2: Experimental t-Test Results for KC1 data
t-Test: Paired Two Sample for Means

	Original	Equalized
Mean	2	8.75
Variance	0	6.407894737
Observations	20	20
Pearson Correlation	#DIV/0!	
Hypothesized Mean Difference	0	
df	19	
t Stat	-11.9250741	
P(T<=t) one-tail	1.43989E-10	
t Critical one-tail	1.729131327	
P(T<=t) two-tail	2.87979E-10	
t Critical two-tail	2.093024705	

The results from the KC2 data set sub-experiments are consistent with the previous two experiments. The average fitness value for the unequalized data is 12, while the equalized data produces an average fitness value of 81.4.

As seen in Table 3, performing a t-test yields a t-value (1.969E-09) significantly less than 1 percent. Thus, the null hypothesis, that the two means are equal, is rejected.

Table 3: Experimental t-Test Results for KC2 data
t-Test: Paired Two Sample for Means

	Original	Equalized
Mean	12	81.4
Variance	10.42105263	1061.094737
Observations	20	20
Pearson Correlation	0.676190201	
Hypothesized Mean Difference	0	
df	19	
t Stat	-10.18111329	
P(T<=t) one-tail	1.96923E-09	
t Critical one-tail	1.729131327	
P(T<=t) two-tail	3.93846E-09	
t Critical two-tail	2.093024705	

The results from the PC1 data set sub-experiments support the findings from the first three experiments. The unequalized data set achieves an average fitness value of 3.75, while the equalized data achieves an average fitness value of 61.25.

As seen in Table 4, the t-test yields a t-value less than 1 percent. Thus, the null hypothesis, that the two means are equal, is rejected.

Table 4: Experimental t-Test Results for PC1 data

t-Test: Paired Two Sample for Means		
	Original	Equalized
Mean	3.75	61.25
Variance	0.407895	3089.671
Observations	20	20
Pearson Correlation	-0.22202	
Hypothesized Mean Difference	0	
df	19	
t Stat	-4.61417	
P(T<=t) one-tail	9.46E-05	
t Critical one-tail	1.729131	
P(T<=t) two-tail	0.000189	
t Critical two-tail	2.093025	

6. Discussion

These four experiments demonstrate that it is possible to compensate for data starved instances in a data set. Furthermore, applying equalization is a valuable endeavor for improving the modeling process.

Machine learners typically have problems compensating for underrepresented values when formulating a solution. Thus, equalizing data prior to building a machine learner is very appropriate.

Boehm and Basili [1] claim that 80 percent of the defects occur in 20 percent of the code. The equalizing technique is extremely useful in extending the visibility of those highly defective modules which occur on a relatively infrequent basis.

One drawback of the data equalization technique is that it can dramatically increase the size of the training set. GPs, along with other machine learners, can take a long time to train. Adding up to ten-fold more data slows down the training process. Performance needs to be considered prior to equalizing a training set. For the experiments described in this paper, the largest training data set contained 7847 instances. A trial for this large training data set would take one hour to run.

The consistency of the results across all four experiments shows that applying data equalization is indeed a valuable approach.

7. Conclusion

There are numerous papers which apply Machine Learners to the Software Engineering-based repositories. The focus, and contribution of this paper, is in applying data equalization as a means of producing better models.

These experiments show that data equalization is a very good approach for an implicitly starved data, such as software defect prediction domain, when modeling with a Machine Learner. Lack of interesting data does not necessarily have to impede the ability to generate a reasonable solution for a significant Software Engineering problem.

Using empirical data from NASA's IV&V facility, four independent GP-based experiments are conducted. Each experiment builds 20 defect prediction models using the data in its native format followed by the building of 20 additional models based upon equalizing the training data

set. T-tests results from all four experiments clearly indicate the superiority of equalizing data prior to building a Machine Learner. Thus, equalizing data helps compensate for under-represented instances in the data set and improving the performance of machine learners in implicitly data-starved environments.

Data equalization is not free. In certain cases equalization can expand a training set by ten-fold. However, once a predictor model is built, testing is very fast.

8. Future Directions

The process of equalizing data in an implicitly starved environment proves invaluable when applied to a Software Engineering domain. Other research has found it useful when applied to neural networks in the financial forecasting domain [14].

However, the amount of work done remains limited to a few studies. It seems logical that this methodology could be equally useful when applied to other machine learners, as well as other domains. More research done in this area would help validate the conclusions drawn in this paper.

Another possible area of research involves the effects of data equalization on the performance of the machine learner. It would be useful to know when this technique is infeasible due to the size of the data.

Finally, it would be useful to apply to the results from one NASA defect data set to other NASA defect data sets to determine if the solution is transferable. Also, a comparison can be made with the solution obtained from the original data set and from the equalized data set when applied to a different data set.

References:

- [1] B. Boehm, V. Basili, Software Defect Reduction Top 10 List, *Computer*, 34(1), 2001, 135-137.
- [2] Capability Maturity Model[®] Integration (CMMISM), Version 1.1, CMMISM for Software Engineering (CMMI-SW, V1.1), Continuous Representation, *Technical Report CMU/SEI-2002-TR-028/ESC-TR-2002-028*, SEI Institute, Pittsburgh, Pennsylvania, 2002.
- [3] Process Maturity Profile, Software CMM[®] CBA IPI and SPA Appraisal Results 2003 Mid-Year Update, SEI Institute, CMU, September, 2003.
- [4] Goldberg, David, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, USA, 1987.
- [5] Metrics Data Program, NASA IV&V facility. Information Data sets are available at: <http://mdp.ivv.nasa.gov/about.html>
- [6] N.E. Fenton, A Critique of Software Defect Prediction Models, *IEEE Transactions on Software Engineering*, 25(5), 1999, 675-689.
- [7] W.W. Cohen, P. Devanbu, A Comparative Study of Inductive Logic Programming Methods for Software Fault Prediction, *Proc. 14th International Conference on Machine Learning*, Nashville, TN, 1997, 66-74.
- [8] M. Evett, T. Khoshgoftar, GP-based Software Quality Prediction, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, Madison, WI, 1998, 60-65.

- [9] J.P. Hochmann, Hudepohl, E.B., Allen, T.M., T. Khoshgoftar, Evolutionary Neural Networks: A Robust Approach to Software Reliability, Eighth International Symposium on Software Reliability Engineering (ISSRE '97), Albuquerque, NM, 1997, 13-26.
- [10] D.E. Neumann, An Enhanced Neural Network Technique for Software Risk Analysis, *IEEE Transactions on Software Engineering*, 28(9), 2002, 904-912.
- [11] D. Zhang, Applying Machine Learning Algorithms in Software Development, *Proceedings of the 2000 Monterey Workshop on Modeling Software System Structures in a Fastly Moving Scenario*, Santa Margherita Ligure, Italy, 2000, 275-285. This paper is available at: http://www.disi.unige.it/person/ReggioG/PROCEEDING_S/zhang.pdf
- [12] N.E. Fenton, Krause, P., N. Martin, A Probabilistic Model for Software Defect Prediction, under revision for *IEEE Transactions in Software Engineering*, 2001.
- [13] R.L. Stites, Ward, B., R.V. Walters, Defect Prediction with Neural Networks, *Proceedings of the conference on Analysis of neural network applications*, pages, Fairfax, VA, 1991, 199 – 206.
- [14] Mizuno, et al., Application of Neural Network To Technical Analysis of Stock Market Prediction, *Studies in Informatics and Control (With Emphasis on Useful Applications of Advanced Technology)*, 7(3), 1998, 111-120.
- [15] J.J. Grefenstette, Incorporating Problem Specific Knowledge into Genetic Algorithms, In L. Davis, editor, *Genetic Algorithms and Simulated Annealing*, Morgan Kaufmann Publishers, Inc., 1987, 42-60.
- [16] D. Whitley, A Genetic Algorithm Tutorial, *Technical Report CS-93-103*, November 10, 1993. Department of Computer Science, Colorado State University
- [17] J. Koza, *Genetic programming: on the programming of computers by means of natural selection* (Cambridge, Massachusetts, MIT Press, 1992).