Applying Variant Variable Regularized Logistic Regression for Modeling Software Defect Predictor

Gabriel Kofi Armah, *Member, IACSIT*, Guanchun Luo, Ke Qin, and Angolo Shem Mbandu, *Member, IACSIT*

***Abstract*—Empirical studies on software defect prediction models have come up with various predictors. In this study we examined variable regularized factors in conjunction with Logistic regression. Our work was built on eight public NASA datasets commonly used in this field. We used one of the datasets for our learning classification out of which we selected the regularization factor with the best predictor model; we then used the same regularization factor to classify the other seven datasets. Our proposed algorithm Variant Variable Regularized Logistic Regression (VVRLR) and modified VVRLR; were then used in the following metrics to measure the effectiveness of our predictor model: *accuracy, precision, recall and F-Measure for each dataset.* We measured above metrics using three Weka models, namely: BayesianLogisticRegression, NaiveBayes and Simple Logistic and then compared these results with VVRLR. VRLR and modified VVRLR outperformed the weka algorithms per our metric measurements. The VVRLR produced the best accuracy of 100.00%, and an average accuracy of 91.65 %; we had an individual highest precision of 100.00%, highest individual recall of 100.00% and F-measure of 100.00% as the overall best with an average value of 76.41% was recorded by VVRLR for some datasets used in our experiments. Our proposed modified VVRLR and variant VVRLR algorithms for F-measures outperformed the three weka algorithms.**

***Index Terms*—F-measure, precision, recall, variant variable regularized logistic regression.**

I. Introduction

Detection of software defective modules in an early stage of its life cycle is very valuable and also saves cost. This can be appreciated in the case of telecommunication and military systems [1]-[3], identifying defects at a later stage may lead to paying an expensive price. Software usually comprises of a great number of impartially independent units termed modules which execute certain functions [4]. A software model can be viewed as an empirical tool using a definite algorithm to determine the type of modules [1].

Diversity of software defect prediction techniques are available and these include, machine learning, parametric, statistical and mixed model techniques[5].Current studies has shown that many researchers use machine learning for software quality prediction. Classification and clustering are some approaches in machine learning where classification is extensively used [6], [7].

Many researchers have proposed different defect predictors for classifying defective modules such as; discriminant analysis by [8]; Linear regression as was proposed by [9], and [10] worked on Naïve.

Bayes classification, for the aforementioned predictors [10] in their work stated that, Naïve Bayes performance is significantly better than the other methods. Nonetheless our proposed model performed better than Naïve Bayes in addition to two other algorithms used in our experiments. Although the algorithms used differ, they all employ complex metrics as an input predictor, same as in our work, and a prediction of fault-prone or non-fault- prone as an output response variable and aim at reducing the cost of the misclassification [11].

In this paper we applied variable regularized logistic regression with four regularization factors for predicting defective software and selected the best regularization as fixed regularized logistics factor for our proposed model. We increased the number of attributes to a polynomial of maximum degree of four from a public dataset; and compared the efficiency of VVRLR with three weka classification algorithms; our algorithms performed better in most areas such as accuracy, precision, recall and F-measure both on individual and average basis.

A. Specifically Our Work Comprises of

1) Converting a single *attribute relationship file format* (*.arff*) into two separate *text file* format in which the True/False attributes were converted to 1/0 based on the target datasets (Defect/Non_defect).
2) We applied Logistic regression for our classification, taking into account a regularizing factor to handle high variance problem.
3) We proposed an efficient classification algorithm (predictor) to predict defective and non-defective modules by studying four regularization factors.
4) Our proposed algorithms are: A Variant of Variable Regularization Logistic Regression (VVRLR) and a Modified Variant of Variable Regularization Logistic Regression (MVVRLR).
5) Compared our proposed algorithm with some selected weka algorithms, our algorithm's average accuracy, precision, recall and F-measure performed better with VVRLR and Our modified F-measure.
6) Compare our VVRLR and modified VVRLR: precision,

Manuscript received August 6, 2014; revised January 5, 2015. This research (work) was partially supported by the grant (No. 2012FZ0063) from the Ministry of Science and Technology of Sichuan Province.

G. K. Armah is with University of Electronic Science of China (UESTC), China. He is also with University for Development Studies, Navrongo, Ghana (e-mail: gabrielarmah1@com).

G. Luo and Ke Qin are with School of Computer Science and Engineering, University of Electronic Science and Technology of China/Computer Science, Chengdu, China (e-mail: gcluo@uestc.edu.cn, qinke@uestc.edu.cn).

Angolo Shem Mbandu is with UESTC/Computer Science, China.

recall and F- measure formulae with the corresponding standard formulae respectively which were very close.

7) Our combined contributions are VVRLR and modified VVRLR which were compared with the three weka algorithms.

The rest of the paper is organized as follows: In Section II, we give detailed related work; Section III we present our statistical model, Section IV is based on NASA datasets, Section V detailed experiments and results and Section VI is conclusion and future work.

II. Related Work

Software defect prediction can be modeled as a data mining problem with the categorization of software modules as defective or non-defective with the usage of historical data. Application of data mining and knowledge discovery(DMKD) in software reliability management has made notable progress as in [12] using methods, algorithms, and techniques(procedures) from many disciplines; databases, statistics, machine learning, pattern recognition, artificial intelligence, data visualization, and optimization [13].

Software defect prediction has been ongoing area in software engineering field for some time now. A lot of related studies and approaches have been experimented to come out with the right defect prediction model. Defect need not to be confused with error, mistake or failure. Defect is said to have taken place if in the event of performance the software or system fails to perform its desired function [14]. Defect can also be observed as the deviation from the software's specification [15] as well as any defectiveness associated to software itself and its allied work product [16]. Predicting defects is proactive process of characterizing many types of defects found in software's content, design and codes in producing high quality product [17]. In their work [15] proposed that the size and complexity metrics are among the earlier methods to defect prediction. Lines of code (LOC) and McCabe's cyclomatic complexity were used to predict defects in software. Simple Bayesian Network was another approach used for defect prediction in a form known as Defect type Model (DTM) that predicts defect based on severity minor, major and minor [18]. Ref. [19] proposed a logistic regression classifier for differing training and test distributions; Multivariate Linear regression was used by [20] to come out with defect inflow prediction for large software projects either short-term defect inflow prediction or long-term defect inflow prediction.

Software metric is a simple quantifiable measure obtained from any attribute emanating from software life cycle. This makes it possible for software engineers to measure and predict software process. Software metric is a measure of some property of software and/or specification. Several data mining methods have been suggested for defect analysis in the past years [21]-[24]. Several Researchers have utilized static attributes as a guide for software quality predictions [25]-[30]. To date researchers have not been able to come up with single set of metrics that could act as a unanimously best defect predictor.

For the purpose of easy comparison most of the fault prone prediction techniques depend solely on historical data. Experimental observation may suggest that a module presently undergoing development is said to be fault-prone if it has comparable properties which are measured as a result of software metrics on the basis of similar module that has been developed or released earlier in the same environment [31]. Thus, historical information helps us predict fault-proneness. As earlier mentioned several modeling techniques have been proposed for and applied to software quality prediction. We have techniques such as; logistic regression [32] it purposes to use domain-specific knowledge to establish the input (i.e. software metrics) and output (software fault-proneness) relationship. Other techniques are ; classification trees [33], [34], neural networks [35], and also genetic algorithms [36], all these techniques try to examine the available large-size datasets to come up with or recognize patterns and form generalizations. A wide range of classification algorithms has been applied to different data sets. Different experimental setups result in a limited ability to comprehend algorithm's strengths and weaknesses. A modeling methodology is good if it is able to perform well on all data sets, or at least most of them. Recently, several software engineering data repositories have become publicly available (Metrics data program NASA IV and V facility. Consequently, these datasets can be used to validate and compare the proposed predictive models. In order to identify reliable classification algorithms, [4] recommended trying a set of different predictive models, [37] suggested building a toolbox for software quality engineers which includes "good" prediction performers.

III. Statistical Model Used for Our Classification

In this work our aim is to apply Logistic regression to learn a suitable regularization factor to model a predictor for the classification of eight static datasets. We then compared our model VVRLR and our modified VVRLR performance with some well -known weka algorithms. In this section we will talk briefly on some of the basic terms and also come up with our model.

A. The Logistic Model Formula

The Logistic model formula computes the probability of the selected response as a function of the values of the predictor variables. Normally if a predictor value is a categorical variable with two values, then one of the values is assigned the value 1(defective) and the other is assigned the value 0(non-defective). In summary, the logistic formula has a continuous predictor variable, each dichotomous predictor variable with a value of 0 or 1. Logistic regression is used in supervised machine learning techniques which are mainly used for solving classification problems [38].

Let us consider a classification task with l training instances {(*x* (*i*), *y* (*i*)), with *i*=1, 2, 3, …, l} each *x* (*i*) ϵ Rk which is *k* dimensional variable feature vector and *y* (*i*) ϵ {1,0} is a class label; y is given a feature vector x as in eqn. (1)

$$p(y=1 \mid x,\omega) = h_{\omega}(x) = g(\omega^{T} x) = \frac{1}{1+\exp^{(-(\omega_0 x_0 + \omega_1 x_1 + \omega_2 x_{2+ \ldots +} \omega_k x_k))}} \quad (1)$$

Definitions: is the hypothesis, function; is known as the logistic(sigmoid) function and constitutes a vector learning parameters of the logistic regression model.

$x_0 = 1$, ω_0 is a constant and ω_k are coefficients of the predictor variables.

Assumptions:

$$P\left(y=1 \mid x,\omega\right) = h_\omega(x)$$
$$P\left(y=0 \mid x,\omega\right) = 1-h_\omega(x) \tag{2}$$

$$p(y \mid x,\omega) = (h_\omega(x))^y (1-h_\omega(x))^{1-y} \tag{3}$$

Another assumption is that the l training examples are supposed to be generated independently. The likelihood of the parameters is also expressed as shown below;

$$L(\omega) = \prod_{i=1}^{l} p\left(y^{(i)}/x^{(i)},\omega\right)$$
$$= \prod_{i=1}^{l}\left((h_\omega(x^{(i)}))^{y(i)}(1-h_\omega(x^{(i)}))^{1-y(i)}\right) \tag{4}$$

The probabilistic model relates $y^{(i)}$ s and $x^{(i)}$'s in order to optimize the parameters ω of the logistic regression model. We derive the log likelihood as in (5) to maximize result:

$$l(\omega) = \log L(\omega) = \sum_{i=1}^{l} y^{(i)} \log h_\omega(x^{(i)}) + (1-y^{(i)}) * \log(1-h_\omega(x^{(i)})) \tag{5}$$

B. *Regularized Logistic Regression*

Since the problems considered have datasets which have small sample size, we choose to add a regularization term to (5) so that it caters for the bias – variance trade off [39]. The regularization log likelihood function is in (6), adding the minus sign turns it into minimization.

$$l(\omega) = -\log L(\omega) = -\sum_{i=1}^{l} y^{(i)} \log h_\omega(x^{(i)}) + (1-y^{(i)}) * \log(1-h_\omega(x^{(i)}) + \frac{\lambda}{2l}\sum_{i=1}^{l}\omega_j^2; \tag{6}$$

λ is positive value; which is also the regularization factor, λ was learned using one of the dataset used in this experiment specifically *Ar1*; we used the following range of lambda : $10^{-6} \le \lambda \le 10^{6}$ this gave us an optimal λ value of 10^{-4} with a corresponding optimal ω value at the optimization point, which converges at the $J(I)^{\text{th}}$ iteration using the Newton's method for convergence as in (7) .

$$J(I) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\log(h\omega(x^{(i)})) + (1-y^{(i)})\log(1-h_\omega(x^{(i)}))] + \frac{\lambda}{2m}\sum_{j=1}^{k}\varpi_j^2 \tag{7}$$

In vector notation updates of ω according to Newton's method is shown by (8):

$$\omega^{(t+1)} = \omega^{(t)} - H^{-1}\nabla_\omega J \tag{8}$$

$\nabla_\omega J$, gradient and H the Hessian is the second partial derivative of $J(\omega)$. The formulas used for computing the two values are shown in Equations (9) and (10) respectively.

$$\nabla_\omega J = \begin{bmatrix} \frac{1}{m}\sum_{i=1}^{l}(h_\omega(x^{(i)}) - y^{(i)})x_0^{(i)} \\ \frac{1}{m}\sum_{i=1}^{l}(h_\omega(x^{(i)}) - y^{(i)})x_1^{(i)} + \frac{\lambda}{m}\omega_1 \\ \frac{1}{m}\sum_{i=1}^{l}(h_\omega(x^{(i)}) - y^{(i)})x_2^{(i)} + \frac{\lambda}{m}\omega_2 \\ . \\ . \\ . \\ \frac{1}{m}\sum_{i=1}^{l}(h_\omega(x^{(i)}) - y^{(i)})x_k^{(i)} + \frac{\lambda}{m}\omega_k \end{bmatrix} \tag{9}$$

$$H = \frac{1}{m}\left[\sum_{i=1}^{m} h_\omega(x^{(i)})(1-h_\omega(x^{(i)}))x^{(i)}\left(x^{(i)}\right)^T\right] + \frac{\lambda}{m}\begin{bmatrix} 0 & & & & & & \\ & 1 & & & & & \\ & & 1 & & & & \\ & & & . & & & \\ & & & & . & & \\ & & & & & . & \\ & & & & & & 1 \end{bmatrix} \tag{10}$$

Computation of ω, $\vec{y}$ and X :

$$\omega = \left(X^T X + \frac{\lambda}{m}\begin{bmatrix} 0 & & & & & & \\ & 1 & & & & & \\ & & 1 & & & & \\ & & & . & & & \\ & & & & . & & \\ & & & & & . & \\ & & & & & & 1 \end{bmatrix} \right)^{-1} X^T \vec{y}$$

$\lambda = 10^{-4}$,

$$\vec{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ . \\ . \\ . \\ y^{(m)} \end{bmatrix} \quad X = \begin{bmatrix} -\left(x^{(1)}\right)^T - \\ -\left(x^{(2)}\right)^T - \\ . \\ . \\ . \\ -\left(x^{(l)}\right)^T - \end{bmatrix}$$

The matrix following $\frac{\lambda}{m}$ is an **(k+1)** by **(k+1)** diagonal matrix with a zero in the upper left and ones down the other diagonal entries; **k** is the number of features, without the intercept term($\omega_0 X_0$). $x^{(i)}$ is k x*1* feature vector; $\nabla_\omega J$ is k x *1* vector; $x^{(i)}\left(x^{(i)}\right)^T$ and H are k x k matrices; $y^{(i)}$ and $h_\omega(x^{(i)})$ are *scalars.*

IV. Datasets Used for the Model

This section looks at few characteristics features of the datasets used for our work; the datasets used are publicly available for use in this field *The* data emanated from the NASA IV and V Metrics data program (MDP)/ promise repository and soft This section looks at few characteristics features of the datasets used for our work; the datasets used are publicly available for use in this field. The data emanated from the NASA IV and V Metrics data program (MDP)/ promise repository and softlab.

A. Analysis of Promise NASA Datasets

Each of the dataset used for this work comprises of several software modules, together with their number of defects and characteristics code attributes. Apart from the line of code (LOC) counts; the NASA promise datasets include several Halstead attributes as well as the McCabe complexity measures. The former; estimate reading complexity by counting operators and operands in a module, while the latter is derived from a module of flow graph.

B. Analysis of Datasets in Tabular Form

The analysis of this paper applies static code from 8 projects tabulated in Table I**,** which are downloaded from the PROMISE repository [40], which also shows static code features. An advantage of static code features is that they can be quickly and automatically be collected from source code, even if no other information is available. Summary of the datasets used by [40], [41] in their work as well as individual features per data set, along with some general statistics was used with our own additional features.

TABLE I: Characteristics of Projects from ASA/Softlab

NASA/Softlab dataset	**Ar1**	**Ar4**	**Ar6**	**CM1**	**KC2**	**KC3**	**MC2**	**MW1**
branccount	X	X	X	X	X	X	X	X
codeandcommentloc	X	X	X	X	X	X	X	X
commentloc	X	X	X	X	X	X	X	X
cyclomaticcomplexity	X	X	X	X	X	X	X	X
designcomplexity	X	X	X	X	X	X	X	X
halsteaddifficulty	X	X	X	X	X	X	X	X
halsteadeffort	X	X	X	X	X	X	X	X
halsteaderror	X	X	X	X	X	X	X	X
halsteadlength	X	X	X	X	X	X	X	X
halsteadtime	X	X	X	X	X	X	X	X
halsteadvolume	X	X	X	X	X	X	X	X
totaloperands	X	X	X	X	X	X	X	X
totaloperators	X	X	X	X	X	X	X	X
uniqueoperands	X	X	X	X	X	X	X	X
uniqueoperators	X	X	X	X	X	X	X	X
executalbeloc	X	X	X	X	X	X	X	X
totalloc	X	X	X	X	X	X	X	X
halsteadcontent				X	X	X	X	X
essentialcomplexity				X	X	X	X	X
halsteadvocabulary	X	X	X	X	X			
blankloc	X	X	X	X	X	X	X	X
callpairs	X	X	X			X	X	X
conditioncount	X	X	X			X	X	X
cyclomaticdensity	X	X	X			X	X	X
decisioncount	X	X	X			X	X	X
decisiondensity	X	X	X			X	X	X
halsteadlevel	X	X	X			X	X	X
multipleconditioncount	X	X	X			X	X	X
designdensity	X	X	X			X	X	X
Normcyclomaticcomp.	X	X	X			X	X	X
formalparameters	X	X	X			X	X	X
modifiedconditioncount						X	X	X
maintenanceseverity						X	X	X
edgecount						X	X	X
nodecount						X	X	X
essentialdensity						X	X	X
globaldatacomplexity						X	X	
globaldatadensity						X	X	
percentcomment						X	X	X
numberoflines						X	X	X
Num. of code attributes	**29**	**29**	**29**	**21**	**21**	**39**	**39**	**37**
Number of modules	**121**	**101**	**107**	**498**	**522**	**458**	**162**	**403**
Percentage defectives(%)	**7.4**	**14.9**	**18.7**	**9.8**	**20.1**	**9.4**	**32.3**	**7.6**
Language of dataset	**C**	**C**	**C**	**C**	**C++**	**Java**	**C++**	**C++**

Table I describes eight software projects used in this work, the top row labeled "NASA/Softlab" depicts datasets from NASA aerospace projects and "SOFTLAB" come from a Turkish software company that develop applications for domestic appliances. In the table cells marked **"X"** represents the presence of that feature for the dataset, the total number of features per dataset used for our work are: 22 attributes for KC2,MC2 and MW1; CM1 64 features ; KC3 79 features and all the datasets from "SOFTLAB" – AR1,AR4 and AR6 in each we used 88 features, all the features actually used for our experiments were reviewed upwards; and also at the bottom of the table we have Number of modules, Percentage defective and Language of dataset to write each application.

V. Detailed Experiments and Outcomes

In this study we used MATLAB version R2011b [42], and Waikato Environment for Knowledge Analysis (Weka) version 3.6.7; a popular suit of machine learning software written in Java [43].

A. Statistical Characteristics of Data

All defects frequently exhibit non-normality characteristics; like skewness, unstable variances, collinearity, and excessive outliers. The following are some of the characteristics of the software data sets considered in our analysis. Features are independent of each other; input features are continuous while output features are discrete.

B. Indicators for Our Assessment

Binary classifiers are characteristically assessed by counting the number of correctly predicted modules over hold-out data. This procedure has four possible outcomes: True positives (TP) are modules classified correctly as defective modules. False positives (FP) refer to non- defective modules incorrectly labeled as defective. True negatives (TN) correspond to correctly classified non-defective modules.

Finally, False negatives (FN) are defective modules incorrectly classified as non-defective. These can be put in a 2 × 2 matrix called confusion table. The most commonly used criterions are precision, recall and F-measure defined by (11) - (14). verall accuracy: Accuracy is the percentage of correctly classified modules [7]. It is one of the most widely used classification performance metrics.

$$OverallAccuracy = \frac{TP+TN}{TP+FP+FN+TN} \tag{11}$$

Precision: This is the number of defective modules which are actually defective modules [44].

$$Precision = \frac{TP}{TP+FP} \tag{12}$$

Recall: This is the percentage of defective modules that are correctly classified.

$$Recall = \frac{TP}{TP+FN} \tag{13}$$

F-Measure: It is the harmonic mean of precision and recall. F-Measure has been widely used in information retrieval.

$$F-Measure = \frac{2\times Precision \times Recall}{Precision + Recall} \tag{14}$$

C. Our Proposed Precision, Recall and F-Measure Formulas

In our intuitive formula for Precision for imbalanced data, but with a high performance, we applied the original precision formula in (12): We then suppress FP to approximately one {1} using the False Positive Rate, by rewriting FP as the power of its rate, FP^{FPR}.

$$\Rightarrow FP = FP^{FPR} \tag{15}$$

And

$$TP = TP^{TPR} \tag{16}$$

Now substituting (15) and (16) into (12) gives us (17), our proposed precision (AR) formula:

$$Precision(AR) = \left(\frac{TP^{(TPR)}}{TP^{(TPR)} + FP^{(FPR)}} \right) \tag{17}$$

which is our proposed variant precision formula for an imbalanced dataset. Applying the same principle as in equation (17) we can formulate the Recall and F-Measure formulae by applying the respective rates, as follows:

$$Recall(AR) = \left(\frac{TP^{(TPR)}}{FN^{1-TPR} + TP^{(TPR)}} \right) \tag{18}$$

We now compute F_ Measure using the original formula in (14) in unification with the TPR, FPR and FNR to obtain our proposed F_measure by substituting (17) and (18) into (14) to obtain (19):

$$F-measure(AR) = 2 \times \left\{ \frac{\left(\frac{TP^{(TPR)}}{TP^{(TPR)} + FP^{(FPR)}} \times \frac{TP^{(TPR)}}{FN^{(1-TPR)} + TP^{(TPR)}} \right)}{\left(\frac{TP^{(TPR)}}{TP^{(TPR)} + FP^{(FPR)}} + \frac{TP^{(TPR)}}{FN^{(1-TPR)} + TP^{(TPR)}} \right)} \right\}$$

$$F-measure(AR) = 2 \times \left\{ \frac{\left(\frac{TP^{(TPR)}}{TP^{(TPR)} + FP^{(FPR)}} \right) \times 1}{\left(\frac{TP^{(TPR)}}{TP^{(TPR)} + FP^{(FPR)}} \right) + 1} \right\} \tag{19}$$

Our proposed Algorithm 1 and 2 are used to compute the hypothesis function and evaluators. In particular, Algorithm 1 computes hypothesis function while Algorithm 2 compares h and Y values and then finally used to compute evaluators.

Algorithm 1 A Variant Variable Regularization Logistic Regression (VVRLR)

```
Input: Files X and Y < split *.arff into two text files and convert True to 1 and False to 0>
Output: < J;   Omega (norm); h >

1.  add extra values of ones in the first column, to increase the number of attributes by 1, (n+1)
2.  Mu(i) =  Mean(X); Std(X) = Standard Deviation(X);
3.  for   i ϵ (2,3,…,n) do
4.      $X(:,i) = \frac{X(:,i) - Mu(i)}{Std(i)}$;
5.  end for
6.  $g = \frac{\exp(Z)}{1+\exp(Z)}$ / Application of Sigmoid to compute individual values
7.  for i=1: Max-Iteration do
8.      Omega = Initialize the size of X (I, : ) to zeros;
9.      Z = X * Theta;
10.         H = g(Z); / Computing hypothesis function
11.       J(i) = log( Logistic function) + ( lambda'(2*m) * norm( Omega[2:end]))^2; /  Computing J __ for convergence
12.         G = gradient; H = Hessian matrix;
13.         Theta = Theta - $\frac{H}{G}$; / Update for Omega
14.   end for
15.   Return J;   Omega (norm); h;
```

Algorithm 2 Evaluator's Computation

```
Input:  < h and Y , Omega >
Output: < P >

1.  h1 = zeros(size(h));
2.  for i= 1:num1 (h) do
3.      if h(i) >= 0.45 then
4.          h(i)= 1;
5.      end if
6.  end for
7.  Initialization of : FP, TP, TN, FN ; / Computation of confusion matrix values
8.  for l=1: num1 do
9.      if Y (l) ==1 and h1(l) ==1 then
10.         TP = TP + 1;
11.         else if Y(l)== 1 and   h(l) == 0 then
12.             FN = FN +1 ;
13.             else if Y==0 and h1(l) == 1 then
14.                 FP = FP + 1 ;
15.                 else TN =   TN + 1;
16.             end if
17.         end if
18.     end if
19. end for
20. Compute Evaluators (P) as follows:-
21. Total = TP+ FP+TN+FN;
22. Accuracy = $\frac{TP+TN}{Total}$;
23. $Precision(AR) = \left( \frac{TP^{(TPR)}}{TP^{(TPR)} + FP^{(FPR)}} \right)$;
24. $Recall(AR) = \left( \frac{TP^{(TPR)}}{FN^{1-TPR} + TP^{(TPR)}} \right)$;
25. $F-measure(AR) = 2 \times \left\{ \frac{\left( \frac{TP^{(TPR)}}{TP^{(TPR)} + FP^{(FPR)}} \right) \times 1}{\left( \frac{TP^{(TPR)}}{TP^{(TPR)} + FP^{(FPR)}} \right) + 1} \right\}$;
26. MAE = $\frac{FN+FP}{Total}$;
27. Return P;
```

D. Experimental Setup

The datasets were loaded in turn into our model using threshold values of 0.45; once when parameters ω_j are established, classification model can be employed according to (1). LR gives us probabilities interpretation of class membership in the range [0-1], thus at this point decision threshold needs to be defined. Every value obtained from (1) which is greater than 0.45 is treated as "1" i.e. instance belongs to class and below threshold values is treated as "0" i.e. instance does not belong to class. This process was done for each of the four regularized values resulting in an initial 64 experiments.

After each run of the experiment we read the following metric values: TP, FP, TN, FN, from our algorithms: Algorithm 1 and Algorithm 2. We then proceeded to compute Accuracy (Accy), Precision (Preci), Mean Average Error (MAE), Recall, F-Measures (F-meas), and ω (the norm of the parameters ω) for each of the eight datasets by the application of (11) - (14) and recomputed precision, recall and F-measure with our proposed formulas (17) – (19).

We then compare our proposed model with three predictors from the Weka algorithm suite. The eight datasets with variable features used to carry out the empirical experiments are shown in Table I**;** the three weka classifiers are; BayesianLogisticRegression, Naïve Bayes and Simple Logistic. Our proposed classifier model (algorithm) was modeled using these omega (ω); 10^-4, 10^-6, 10^3 and 10^6 out of which we picked the one which yielded the best results.

E. Outcomes of Experiments

In Table II we recorded the metric measures for our model variable regularized logistic regression using the best values for λ = 10^-4 for the eight datasets: our main discussion was on how best our model performed in terms of F-measure, the harmonic mean of precision and recall which determines the tradeoff between precision and recall; the 0.45 threshold gave similar values of 94.97% and 100% as compared to the 0.5 threshold value for Ar1, Ar4 and Ar6 respectively and gave 100% recall for all three datasets at the 0.45 and 0.5 thresholds. While the datasets; Cm1, Kc2, Kc3, Mc2 and Mw1 produced the following f-measure values from 50.00% to 79.45% for λ =10^-4.

Table III gives a summary of the performance of VVRLR and Modified VVRLR with three weka algorithms applied in this experiment in terms of accuracy and the F-measure metrics. The summary shows that VVRLR performance surpassed the three algorithms from Weka. Table III compares our F-measure results with the three weka results; Fig. 1 is the graphical representation of Table III; thus our evaluator is said to have a higher performance than the three weka algorithms. Fig. 1 depicts the weka algorithms, VVRLR and modified VVRLR the last two which happens to be our algorithms performed better compared to the three weka algorithms.

TABLE II: CLASSIFICATION MODEL RESULTS FOR ʎ =10^-4

Algorithm Dataset	Accuracy in %				F-Measure in %				
	Simple logistic	Na ïve Bayes	Bayesian Logistic Regression	VRLR	Simple logistic	Na ïve Bayes	Bayesian Logistic Regression	VRLR	OUR VRLR
Ar1	92.56	85.95	92.56	99.17	0.00	32.00	0.00	94.74	90.00
Ar4	90.65	85.98	43.93	89.72	67.00	55.00	38.00	100	95.24
Ar6	89.11	85.15	87.13	95.05	42.00	44.00	24.00	100	93.75
Cm1	90.16	84.94	90.16	91.37	0.00	29.00	0.00	51.95	44.79
Kc2	84.48	83.72	20.50	84.29	52.00	50.00	34.00	55.91	60.03
Kc3	90.83	84.93	90.61	93.23	22.00	34.00	0.00	79.45	85.22
Mc2	77.64	73.91	49.07	86.34	57.00	46.00	53.00	79.25	93.44
Mw1	93.30	83.62	92.31	94.04	27.00	33.00	0.00	50.00	42.43
Mean value	88.59	83.53	70.78	91.65	33.38	40.38	18.63	76.41	75.61

TABLE III: CLASSIFICATION MODEL RESULTS FOR: ʎ =10^-4

Data-set	Thresh/ Meas.	ʎ =10^-4									
		TP	FP	TN	FN	Accy	MAE	Preci	Recall	Fmeas	Norm-Θ
Ar1	**0.45**	**9**	**1**	**111**	**0**	0.9917	**0.0083**	**0.9**	**1**	**0.9474**	99.5445
	0.5	9	1	111	0	0.9917	0.0083	0.9	1	0.9474	99.5445
Ar4	**0.45**	**20**	**0**	**87**	**0**	**1**	**0**	**1**	**1**	**1**	153.4152
	0.5	20	0	87	0	1	0	1	1	1	153.4152
Ar6	**0.45**	**15**	**0**	**86**	**0**	**1**	**0**	**1**	**1**	**1**	**141.3118**
	0.5	15	0	86	0	1	0	1	1	1	141.3118
Cm1	**0.45**	20	8	441	29	0.9257	0.0743	0.7143	0.4082	0.5195	173.3848
	0.5	19	4	445	30	0.9317	0.0683	0.8261	0.3878	0.5278	173.3848
Kc2	**0.45**	**52**	**27**	**388**	**55**	**0.8429**	**0.1571**	**0.6582**	**0.486**	**0.5591**	**31.36**
	0.5	50	21	394	57	0.8506	0.1494	0.7042	0.4673	0.5618	31.3647
Kc3	**0.45**	**29**	**1**	**414**	**14**	**0.9672**	**0.0328**	**0.9667**	**0.6744**	**0.7945**	**187.5752**
	0.5	27	0	415	16	0.9651	0.0699	1	0.6279	0.7714	187.5752
Mc2	**0.45**	**42**	**12**	**97**	**10**	**0.8634**	**0.1366**	**0.7778**	**0.8077**	**0.7925**	**34.53**
	0.5	37	9	100	15	0.8509	0.1491	0.8043	0.7115	0.7551	34.5312
Mw1	**0.45**	**12**	**5**	**367**	**19**	**0.9404**	**0.0596**	**0.7059**	**0.3871**	**0.5**	**14.18**
	0.5	10	4	368	21	0.938	0.062	0.7143	0.3226	0.4444	14.1812

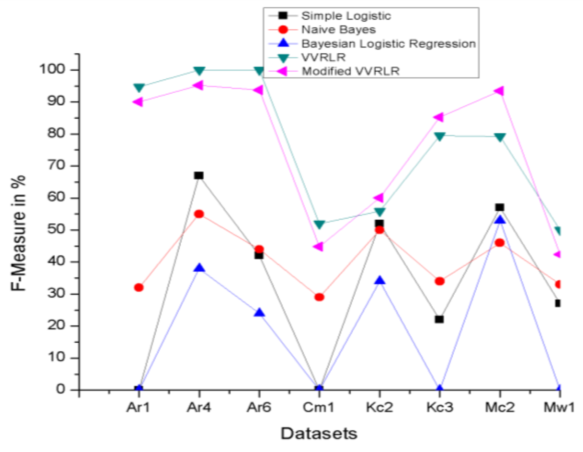

Fig. 1. Graphical representation of VVRLR, MVVRLR and weka F-measures.

VI. CONCLUSION AND FUTURE WORK

In this work we used logistic regression to model our classifier VVRLR with original formulas and our proposed formulas for precision, recall and F-measure, by using variable regularization factors; out of which we obtained the one which gave the best evaluator with a factor of $\lambda = 10^{-4}$ We then subjected the same datasets to three algorithms from Weka to check the performance of VVRLR and modified VVRLR. Our experiments were performed using eight NASA datasets from PROMISE repository. The following measures were used to evaluate the performance of VVRLR: *accuracy, recall and F-Measure*.

The experimental results showed that VVRLR and our proposed modified VVRLR F-measure algorithm outclassed the other three algorithms from weka algorithms.

In this work we considered small training/test datasets, which gave us excellent results. Future research will consider very large datasets in the thousand range set with more and selected instances to achieve better results and consider receiver operating characteristics curves using different threshold.

REFERENCES

[1] T. M. Khoshgoftaar, D. L. Lanning, and A. S. Pandya, "A comparative study of pattern recognition techniques for quality evaluation of

telecommunications software," *IEEE Journals of Selected Areas in Communications*, vol. 12, no. 2, pp. 279-291, Feb. 1994.
[2] L. C. Briand, V. R. Basili, and C. J. Hetmanski, "Developing interpretable models with optimized set reduction for identifying high-risk software components," *IEEE. Transactions on Software Engineering*, vol. 19, no. 11, pp. 1028-1044, Nov. 1993.
[3] J. C. Munson and T. M. Khoshgftaar, "The detection of fault-prone programs," *IEEE. Transactions on Software Engineering*, vol. 18, pp. 423-433, May 1992.
[4] J. C. Munson, *A Handbook of Software Reliability Engineering*, IEEE Computer Society Press and McGraw-Hill Book Company, 1999.
[5] V. U. B. Challagulla, B. Farokh, I.-L. Yen, and A. P. Raymond, "Empirical assessment of machine learning based software defect prediction techniques," in *Proc. the 10th International Work Shop on Object – Oriented Real-Time Dependable Systems*, 2005, pp. 263-270.
[6] J. R. Quilan, *C4.5: Programs for Machine Learning*, SanMateo, CA: Morgan Kaufmann Publishers, 1993.
[7] J. Han, and M. Kamber, *Data Mining Concepts and Techniques*, 2nd edition, San Francisco: Morgan Kaufmann Publishers, 2006.
[8] J. Munson, Zander, and T. M. Khoshgoftaar, "The detection of fault-prone programs," in *Proc. IEEE Transactions on Software Engineering*, vol. 18, no. 5, 1992, pp. 423-433.
[9] J. Munson, Zander, and T. M. Khoshgoftaar, "Regression modeling of software quality: empirical investigation," *Journal of Electronic Materials*, vol. 19, no. 6, pp. 106-114, June 1990.
[10] T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors," *IEEE Transactions on Software Engineering*, vol. 33, no. 1, pp. 2-13, 2007.
[11] T. M. Khoshgoftaar and E. B. Allen, "The impact of costs of misclassification on software quality modeling," in *Proc. Fourth International Software Metrics Symposium*, 1997, pp. 54–62.
[12] Y. Peng, G. Kou, G. Wang, H. Wang, and F. Kou, "Empirical evaluation of classifiers for software risk management," *International journal of Information technology and Decision Making*, vol. 8, issue 4, pp. 749-768, 1990.
[13] Y. Peng, G. Kou, Y. Shi, and Z. Chen, "A descriptive framework for the field of data mining and knowledge discovery," *International Journal of Information technology and Decision Making*, vol. 7, issue 4, pp. 639-682, Dec. 2008.
[14] G. Graham, E. V. Veenendaal, I. Evans, and R. Black, "Foundation of software testing: ISTQB certification," *Thompson Learning*, United Kingdom, 2007.
[15] N. E. Fenton and M. Neil, "A critique of software defect prediction models," *IEEE Transactions on Software Engineering*, vol. 25, no. 5, pp. 675-689, 1999.
[16] B. Clark and D. Zubrow, "How good is the software: A review of defect prediction techniques," Carnegie Mellon University, USA, 2001.
[17] V. Nayak and D. Naidya, "Defect estimation strategies," Patni Computer Systems Limited, Mumbai, 2003.
[18] L. RadliRski, "Predicting defective type in software projects," *Polish Journal of Environmental Studies*, vol. 18, no. 3B, pp. 311-315, 2009.
[19] S. Bickel, M. Bruckner, and T. Scheffer, "Discriminative learning for differing training and test distributions," in *Proc. the 24th International Conference on Machine Learning*, 2007, pp. 81–88.
[20] M. Staron and W. Meding, "Defect inflow prediction in large software projects," *E-Informatica Software Engineering Journal*, vol. 4, no. 1, pp. 1-23, 2010.
[21] W. Tang and T. M. Khoshgoftaar, "Noise identification with the kmeans algorithm," in *Proc. International 2004 Conf. Tools with Artificial Intelligence (ICTAI)*, 2004, pp. 373-378.
[22] S. Zhong, T. M. Khoshgoftaar, and N. Seliya, "Analyzing software measurement data with clustering techniques," *IEEE Intelligent Systems, Special Issue on Data and Information Cleaning and Pre-processing*, vol. 2, pp. 20-27, 2004.
[23] N. Fenton and M. Neil "A critique of software defect prediction models," *IEEE Transactions on Software Engineering*, vol. 25, no. 5, pp. 675-689, 1999.
[24] T. M. Khoshgoftaar and M. Seliya "Tree-based software quality estimation models for faul prediction," in *Proc. 2002 the 8th IEEE International conference on Software Metrics*, 2002, pp. 203-215.
[25] M. Halstead, *Elements of Software Science*, Elsevier, 1997.
[26] T. McCabe, "A complexity measure," *IEEE Trans. Software Eng*, vol. 2, no. 4, pp. 308-320, 1976.
[27] T. Menzies, J. Destefano, A. Orrego, and R. Chapman, "Assessing predictors of software defects," in *Proc. 2004 Workshop Predictive Software Models*, 2004.
[28] P. Singh, "Comparing the effectiveness of machine learning algorithms for defect prediction," *International Journal of Information Technology and Knowledge Management*, vol. 2, no. 2, pp. 481-483, 2009.
[29] G. Hall and J. Munson, "Software evolution: Code delta and code churn," *Journal of Systems and Software*, p. 111, 2000.
[30] A. P. Nikora and J. Munson, "Developing fault predictors for evolving software systems," in *Proc. IEEE Ninth Int'l Software Metrics Symposium*, pp. 338-350, 2003.
[31] T. M. Khoshgoftaar, E. B. Allen, F. D. Ross, R. Munikoti, N. Geol, and A. Nandi, "Predicting fault-prone modules with case-based reasoning," in *Proc. the 8th International Symposium of Engineering (ISSRE'07), IEEE Computer Society*, pp. 27-35, 1997.
[32] V. R. Basili, L. C. Brainde, and W. L. Melo, "A validation of object oriented design metrics as quality indicators," *IEEE Trans Softw. Eng*, vol. 22, no. 10, pp. 751-761, 1996.
[33] T. M. Khoshgoftaar and N. Seliya, "Tree-based software quality estimation models for fault prediction," in *Proc. the 8th IEEE Symposium on Software Metrics,* IEEE Computer Society, pp. 203-214, 2002.
[34] S. S. Gokhals and M. R. Lyu, "Regression tree modeling for the prediction of software quality," in *Proc. the third ISSAT International Conference on Rehabilitee and Quality in Design Anaheim*, pp. 31-36, 1997.
[35] T. M. Khoshgoftaar and D. L. Lanning, "A neural network approach for early detection of program modules having high risk in the maintenance phase," *J Syst. Softw,* vol. 29, no. 1, pp. 85-91, 1995.
[36] D. Azar, D. Precup, S. Bouktif, B. Kegl, and H. Sahraoui, "Combining and adapting software quality predictive Models by genetic algorithms," in *Proc. 17th IEEE International conference on Automated Software engineering,* 2002.
[37] L. Guo, Y. Ma, B. Cukic, and H. Singh, "Robust prediction of fault-proneness random forests," in *Proc. the 15th IEEE International Symposium on Software Reliability Engineering (ISSRE)*, IEEE Press, 2004.
[38] D. Hosmer and S. Lemeshow, *Applied Logistic Regression*, 2nd ed., John Wiley & Sons, 2000.
[39] C. Bieza, V. Robles, and P. Larranaga, "Regularized logistic regression without a penalty term: An application to cancer classification with microarray data," *Expert Systems with Applications*, vol. 38, pp. 5110-5118, 2011.
[40] M. Chapman, P. Callis, and W. Jackson. (2004). Metrics data rogram. NASA IV and V Facility. [Online]. Available: http://mdp.ivv.nasa.gov//(http://promise.site.uottowa.ca/SERepository)
[41] T. M. Khoshgoftaar and E. B. Allen, "Logistic regression modeling of software quality," *International Journal of Reliability, Quality and Safety Engineering,* vol. 6, pp. 303-317, 1996.
[42] MathWorks, "MATLAB: The language of technical computing," *Desktop Tools and Development Environment*, version 7, vol. 9, MathWorks, 2005.
[43] E. Frank, M. Hall, L. Trigg, G. Holmes, and I. H. Witten, "Data mining in bioinformatics using Weka," *Bioinformatics,* vol. 20, pp. 2479-2481, 2004.
[44] M. Božić, M. Stojanović, Z. Stajić, and D. Vukić, "Power transformer fault diagnosis based on dissolved gas analysis with logistic regression," *Przegląd Elektrotechniczny,* vol. 89, 2013.

Gabriel Kofi Armah received his BSc. in computer science in 1997 from KNUST in Ghana and master degree in MIS from University of Ghana in 2003. Currently he is doing his PhD in software engineering at UESTC in China.

His major field of study should be done a two year compulsory national service in Ghana. He is a computer science lecturer at the University for Development Studies, Ghana. He has some publication to his credit. His research interests include software engineering, algorithms, data mining using machine learning and weka.

Guang Chun Luo received his Ph.D. degree in computer science from the UESTC in 2004. He is currently a full professor of School of Graduate at the UESTC in China. He has over seventy publications to his name. His research interests include software engineering, mobile networks and network security.

Ke Qin received his ME and Ph.D degrees from the UESTC in 2006 and 2010 respectively. He was also a visiting scholar at Carleton University, Ottawa, Canada in 2008. He is currently an associate professor at the UESTC, China. He has over forty publications to his credit His research interests include chaos, chaotic neural networks and nonlinear systems.

Angolo Shem Mbandu received his bed technology degree from Moi University, Eldoret, Kenya in 1997. He received his MSc information systems degree from University of Nairobi, Nairobi, Kenya in 2009. Currently, he is a PhD candidate in the Department of Computer Science and Engineering at the University of Electronic Science and Technology of China, Chengdu, PRC. He is also a member of IACSIT (Membership No. 80337496). His research interests are information security and cryptography its applications.