Software defect prediction using Binary Particle Swarm Optimization with Binary Cross Entropy as the fitness function

R Malhotra¹,², A Shakya¹,³, R Ranjan¹,⁴ and R Banshi¹,⁵

¹Department of Software Engineering, Delhi Technological University, Delhi, India

Email: ¹ruchikamalhotra@dtu.ac.in, ³abhishek.shakya112014@gmail.com, ⁴ranjan.ranjan19750@gmail.com, ⁵rishabh.banshi@gmail.com

Abstract. Software is a consequential asset because concrete software is needed in virtually every industry, in every business, and for every function. It becomes more paramount as time goes on – if something breaks within your application portfolio and expeditious, efficient, and efficacious fine-tune needs to transpire as anon as possible. Therefore, recognizing the faults in the early phase of the software development lifecycle is essential for both, diminishing the cost in terms of efforts and money Similary, It is important to find out features which could be redundant or features which are highly correlated with each other, as it could largely affect the model's learning process. This analysis refers to the use of crossover Artificial Neural Network (ANN) and Binary Particle Swarm Optimization (BPSO) with Binary Cross-Entropy (BCE) loss for the fitness function. The conclusion of proposed paper is to provide the significance and opportunity of using Binary Cross-Entropy (BCE) in Binary Particle Swarm Optimization (BPSO) for the feature reduction process in order to minimize the developer's effort and costs for software development as well as for its maintenance.

1. Introduction
In the last few decades, the demand for software quality has increased exponentially. Currently, software plays an important role in various fields. Therefore, testing of software is considered a fundamental task. However, as software industries are developing, the size of software as well as its time and effort in software development is also increasing at an exponential rate. Since humans are strongly dependent on software systems on daily basis, defects in software must be handled properly.

The number of defects in software directly reflects its quality. Today, several data sets exist, which could be mined to gain useful knowledge concerning the defects that transpired during the rapid growth in software development. Researchers have tried and implemented many methods of machine learning and data mining to predict the defects in software. However, methods in which concepts of machine learning are directly applied may not be précised enough.

In this paper, we have analyzed the results of software defects using different classifiers to predict fault proneness for used datasets, i.e., PC1, JM1, KC1, and KC2, and also states that why approach BPSO-ANN with BCE loss is better prediction performance than another compared classifier. This evaluation analyses the prediction performance of software defects by applying the concept of BPSO-ANN with the given historical data set (NASA MDP Data Set).

This paper is organized as follows. Section 2 includes related work that summarizes the previously published studies in this field of dimension reduction and software defect prediction respectively. Data Pre-processing is explained in section 3 which includes a description of data, data-set normalization, and proposed feature selection technique. The software defect prediction approach algorithm is
explained in section 4. Section 5 contains a description of the performance measure used. Experiment results, comparisons, and analyses are discussed in section 6. Section 7 concludes this study.

2. Related work

Particle Swarm Optimization (PSO) is a technique based on population optimization which was introduced by Eberhart and Kennedy in 1995 [10], in which particles can be represented as a swarm of birds, which individually adjust their velocity and position during each iteration according to the previous experience of all the neighboring particles or birds to get the optimized value of fitness function. After every iteration, each particle updates its search space according to the previously found best solution and adjusts the overall best solution using each particle’s personal best solution available.

PSO was developed for optimization problems having real value search space. So, Eberhart and Kennedy developed Binary Particle Swarm Optimization (BPSO) [6] for optimization problems that have discrete binary search space. In BPSO, search space is represented by a binary number, i.e., 0 and 1, where 0 represents a particular feature is absent and 1 represents the presence of that feature. According to this binary search space particle updates, their individual best solution and using those solutions overall best solution is obtained. Due to the presence of a discrete search space, BPSO is used for the feature selection of the dataset.

Lately, various researches have been done in the field of dimension reduction using PSO and BPSO. BPSO is used with a support vector machine (SVM) published by Vieira et al. [4]. A distinct category binary particle swarm optimization [6] used BPSO for feature selection introducing improved BPSO. Researchers have introduced various papers regarding improved binary PSO (IBPSO) for better performance and results. This paper contains modifications in BPSO inspired by the research of Agarwal et al. [5] in IBPSO.

Some extensively good researches were also published in the field of software defect predictions, such as Jin et al. benchmarked proposed Quantum Particle Swarm Optimization (QPSO) [3] and benchmarked various classification models. Ma et al. build classifiers using transfer learning for software defect prediction [2]. Can et al. [1] came with the combination of PSO with SVM for defect prediction.

3. Preprocessing

3.1. Data description

The NASA dataset is obtained from https://github.com/klainfo/NASADefectDataset. The datasets used in this paper are JM1, PC1, KC1, and KC3. The dataset used is written in procedural C and object-oriented C++ for PC1, JM1, and KC1, KC3 respectively. From each dataset 22, software metrics are selected as described in TABLE 1.

| Metric Type   | Software metrics | Description                                      |
|---------------|------------------|-------------------------------------------------|
| McCabe        | Mc_LOC           | Total executable lines                          |
|               | Mc_CC            | Cyclomatic complexity                           |
|               | Mc_EC            | Essential complexity                            |
|               | Mc_DC            | Design complexity                               |
| Derived Halstead | DH_con       | Intelligence present in the program             |
|               | DH_vol           | The volume of the program in bits               |
|               | DH_len           | Total number of operator and operands           |
|               | DH_diff          | Amount of difficulty to maintain program        |
|               | DH_lev           | Intelligence content of the program             |
|               | DH_eff           | Efforts required for writing code               |
|               | DH_err           | Error estimation                                |
|               | DH_prg           | Time estimation for program implementation      |
3.2. Dataset normalization
While training a Neural Network, it is best practice to normalize the data to obtain the mean close to 0. Data Normalization generally speeds up learning and leads to faster loss convergence and ensures equal weightage to every input.

Since the upper bound of all the features is different in their value range. It is important to normalize the attributes in the same range. In these datasets, maximum and minimum values can be easily obtained.

For the \( j \)-th software, metric have \( \min(k^{(i)}) \) and \( \max(k^{(i)}) \) as the minimum and maximum values of \( j \)-th attribute of the dataset, the normalized value is

\[
\chi^{(i)} = \frac{k^{(i)} - \min(k^{(i)})}{\max(k^{(i)}) - \min(k^{(i)})}
\]

Hence, all software metrics value mapped in the interval \([0,1]\).

3.3. Feature Selection

3.3.1. Binary Particle Swarm Optimization
Binary Particle Swarm Optimization (BPSO) [6] is PSO with binary search space. The fact that BPSO searches in the binary space set it apart from standard PSO. Therefore, 0 or 1 are the only assigned values to the position vector of particles. Particles update position and velocity using a for the objective of maximization or minimization fitness function.

In this paper, each particle’s fitness is calculated based on the classification Binary Cross-Entropy (BCE) loss, of the 2-NN classifier using stratified k-fold cross-validation. The best fitness value of an individual particle is named as personal best fitness so far i.e. (pbest) and the corresponding position of particles in their search space (i.e. 0 or 1) is stored. The velocity of a particle at a certain position can be calculated as the resultant of its current velocity, its current position respective to its personal best position, and its current position respective to the overall best position [2]. The best among all individual particle’s fitness values and positions are called as global best fitness (gbest) and position. Particle position and velocity are calculated using these equations.

\[
v_{pd}^{\text{new}} = w \cdot v_{pd}^{\text{old}} + c_1 \cdot \text{randn}_1 (pbest_{pd} - x_{pd}^{\text{old}}) + c_2 \cdot \text{randn}_2 (gbest_d - x_{pd}^{\text{old}})
\]

\[
\text{Sigmoid}(v_{pd}^{\text{new}}) = \frac{1}{1 + e^{-v_{pd}^{\text{new}}}}
\]

\[
\text{if} \left( \text{rand} < \text{Sigmoid}(v_{pd}^{\text{new}}) \right) \text{then} \ x_{pd}^{\text{new}} = 0 \ \text{else} \ x_{pd}^{\text{new}} = 1
\]
Where \( v_{pd}^{new} \) is the particle \( p \) velocity with dimension \( d \), \( w \) is the inertia, \( c_1 \) and \( c_2 \) are positive constants, \( randn_1 \) and \( randn_2 \) are normally distributed random numbers in the range \([0,1]\), \( rand \) is uniformly distributed random number. \( p_{best}_{pd} \) and \( p_{best}_{q} \) are the previously calculated best position of \( i^{th} \) particle and the best value among all particle positions, respectively.

And if velocity overflows or underflows given bounds \((V_{min}, V_{max})\) then,

\[
\begin{align*}
\text{if} \ (v_{pd}^{new} < V_{min}) \ & \text{then} \ v_{pd}^{new} = V_{min} \\
\text{if} \ (v_{pd}^{new} > V_{max}) \ & \text{then} \ v_{pd}^{new} = V_{max}
\end{align*}
\]

For maximizing fitness function, \( p_{best}_{p} \) of particle \( p \) and \( g_{best} \) can be calculated as:

\[
\begin{align*}
\text{if} \ (f(1(x_{p}) > f(1(p_{best}_{p}))) \ & \text{then} \ p_{best}_{p} = x_{p} \\
\text{if} \ (f(1(p_{best}_{p}) > f(1(g_{best})) \ & \text{then} \ g_{best} = p_{best}_{p} \\
\text{if} \ ((f(1(p_{best}_{p}) == f(1(g_{best})) \ and \ (f(2(p_{best}_{p}) < f(2(g_{best})))) \ & \text{then} \ g_{best} = p_{best}_{p}
\end{align*}
\]

where \( f(1(x) \) is the fitness value at position \( x \) and \( f(2(x) \) is the number of selected features at position \( x \), respectively. The termination criterion of BPSO is that, if the fitness value is 0, i.e., 0% loss, then stop else stop when all iterations have completed.

3.3.2. Binary Cross-Entropy loss as the Fitness function

Fitness function is a type of objective function that is used to achieve minimum or maximum solution of set aims. The objective of the fitness function is to determine the quality of a particle’s position. In this paper, binary cross-entropy loss of classifier is used as the fitness function and the main objective is to minimize the BCE loss using the BPSO optimization algorithm. The input of the fitness function is selected features, i.e., values marked with 1 is considered as selected while 0 marked as rejected. The output of fitness function is the BCE loss achieved by 2-NN classifier trained on selected features where BCE loss is defined as:

\[
BCE \text{ loss} = -{1 \over \text{output size}} \sum_{k=1}^{\text{output size}} p_k \log \hat{p}_k + (1 - p_k) \log (1 - \hat{p}_k)
\]

Where \( y_k \) is the ground truth of \( i^{th} \) output and \( \hat{p}_k \) is the prediction probability of the \( k^{th} \) output of being class 1. In this paper, 10-cross stratified cross-validation along with 2-Nearest Neighbors for calculating BCE loss. In each iteration of BPSO, the entire search space is passed to the fitness function for each particle, and the fitness of search space is calculated in the form of BCE loss. The main objective is to minimize the loss generated using Algorithm 1.

3.3.3 BPSO Algorithm

The pseudocode of the BPSO algorithm is shown as follows.

**Algorithm 1.** Binary Particle swarm optimization

1. Initialize the positions, velocity, and \( p_{best} \) positions of all particles
2. Do
3. for all particle \( i = 1 \) to \( p \)
4. calculate fitness value of particle \( i \) using equation (10)
5. update \( p_{best} \), using equation (7)
6. update \( g_{best} \) using equations (8) and (9)
7. for all particle \( i = 1 \) to \( p \)
Once the final solution of the BPSO algorithm [8] obtained, \( g_{best} \) contains the best features that should be selected, i.e., \( g_{best} \) has dimension \( d \) which contains binary values (0 or 1) where 1 represents a feature that is selected, while 0 represents those features which are not selected.

### 4. Software defect prediction approach

In this paper simplified 2-layer neural network is used in which the number of selected features is equal to the number of input neurons, the first hidden layer has 8 neurons, the second layer has 4 neurons, and the number of output neurons is 1, i.e., the confidence score of defects being 0 (absent) or 1 (present).

**Algorithm steps are followed:**

**Step 1** Normalize the dataset in the range [0,1].

**Step 2** Dimensionality reduction of the dataset using BPSO with BCE loss as the fitness function.

**Step 3** Developed and trained simplified ANN model on each dataset with a reduced dimension.

**Step 4** Used trained model to predict the module fault process for each dataset respectively.

**Step 5** Drew ROC curves of the output of the module for performance evaluation corresponding to every dataset.

### 5. Performance measures

Generally, the confusion matrix [9] is used for performance evaluation of two class-problems as shown in Table 2.

**Table 2. Confusion matrix [9]**

| Ground truth | Prediction |
|--------------|------------|
| False(N)     | True Negatives (TN) | False Positives (FP) |
| True(T)      | False Negatives (FN) | True Positives (TP) |

#### 5.1. True positive rate (TPR)

True positive rate describes how good a model is to predict a positive class when the actual class is positive. It can be calculated as:

\[
TPR = \frac{TP}{TP+FN}
\]
5.2. False positive rate (FPR)
The false-positive rate describes that up to how much extend, the model predicts a negative class when the actual class is positive. It can be calculated as:

\[ \text{FPR} = \frac{\text{FP}}{\text{TN} + \text{FP}} \]

5.3. Receiver Operating Characteristic Curve (ROC)
ROC curve measures the degree of separability. It represents the capability of the model to differentiate between classes present in the dataset. ROC curve is plotted TPR against FPR. ROC curve (AUC) is used in this paper for the evaluation of the performance of the proposed software defect prediction approach. The higher the AUC (area under the curve), better the model is at classification. ROC curves compare models for different thresholds.

6. Experiment Results

6.1. Dimensions reduction
In this paper, we used k-fold stratified cross-validation with 10 folds for training and validation of the model for each dataset, i.e., JM1, PC1, KC1, KC3. With parameters TABLE 3, we trained our model using python 3.6.7 on google colab. Table 4 shows the results of BPSO with BCE as its fitness function over all four datasets.

| Table 3. BPSO specifications |
|-----------------------------|
| Parameters          | Values |
| Total particles      | 60     |
| \( V_{\min} \)       | -1     |
| \( V_{\max} \)       | 1      |
| Number of iterations | 100    |
| \( c_1, c_2 \)       | 2      |
| \( w \)              | 0.4    |

Software datasets initially had 21 software metrics, after passing all datasets through BPSO, features reduced to Table 4.

| Table 4. BPSO results |
|-----------------------|
| Dataset | # of selected features | Selected featured          |
|---------|------------------------|-----------------------------|
| PC1     | 7                      | BH_cac, Mc_DC, H_dif, H_eff, H_vol, BH_nopt, BH_loc |
| KC1     | 4                      | BH_cac, Mc_DC, H_con, H_len |
| KC3     | 5                      | H_con, H_dif, H_vol, BH_nopd, BH_uopt |
| JM1     | 6                      | BH_cacMc_LOC, H_eff, BH_uopd, BH_nopt, BH_loc |

6.2. Prediction results
For every fold of k-fold cross-validation, a new ANN model is obtained for training and performance evaluation. We used Pytorch for implementing ANN. Matplotlib and Sklearn are used to plot and calculate ROC curves and AUC respectively.

Table 5 shows the best AUC values achieved when a new ANN model is run independently for each dataset, i.e., PC1, JM1, KC1, KC2. The highest AUC values achieved during 10 independent runs are shown in Fig-1-4 and TABLE 5 for all four datasets respectively.
Table 5. Prediction results

| Datasets | AUC value |
|----------|-----------|
| PC1      | 0.9297    |
| KC1      | 0.8487    |
| KC2      | 0.8823    |
| JM1      | 0.7390    |

Fig 1-4 shows the ROC curves with the highest AUC value achieved during experimentation with a new ANN for each different dataset.

![Figure 1. ROC curve for PC1](image1.png)

![Figure 2. ROC curve for KC1](image2.png)

![Figure 3. ROC curve for KC3](image3.png)

![Figure 4. ROC curve for JM1](image4.png)

From the results shown in Fig 1-4, it can be observed that AUC values for every dataset are greater than 0.73, which indicates that the proposed method used in this paper is effective for software defect prediction. Since we have used k-fold stratified cross-validation, this implies that training and testing sets were different from the previous run during the whole k-fold validation process.

6.3. Comparative results and analysis

Many results were published for software defect prediction on used datasets, i.e., PC1, JM1, KC1, and KC3, using multiple machine learning techniques [11]. To compare our results, AUC values of previous studies are listed in Tables 6.1 and Table 6.2 respectively. Besides this, our results are also
mentioned in Table 6 for comparison. This shows that our proposed method performed better than the other comparable approaches.

Table 6.1. Comparison with other models

| Classifier | LDA* | QDA* | LogReg* | NB* | Bayes Net* | LARS* | RVM* | k-NN* | K** | MLP-1* | MLP-2* | RBF net* |
|------------|-----|------|---------|-----|-----------|-------|------|-------|-----|--------|--------|----------|
| JM1        | 0.730 | 0.700 | 0.730 | 0.690 | 0.730 | 0.720 | 0.720 | 0.710 | 0.690 | 0.730 | 0.730 | 0.690 |
| KC1        | 0.780 | 0.780 | 0.760 | 0.760 | 0.750 | 0.750 | 0.760 | 0.70 | 0.680 | 0.770 | 0.770 | 0.760 |
| KC3        | 0.620 | 0.740 | 0.610 | 0.830 | 0.830 | 0.800 | 0.740 | 0.820 | 0.710 | 0.790 | 0.830 | 0.680 |
| PC1        | 0.820 | 0.700 | 0.820 | 0.790 | 0.840 | 0.700 | 0.840 | 0.820 | 0.720 | 0.890 | 0.910 | 0.640 |

* AUC values are referred from [11]

Table 6.2 Comparison with other models

| Classifier | SVM* | L-SVM* | LS-SVM* | LP* | VP* | C4.5* | CART* | ADT* | RnsFor* | LMT* | ANN | BPSO (acc) + ANN | BPSO (BCE) + ANN |
|------------|------|--------|---------|-----|-----|-------|-------|------|---------|------|-----|----------------|-----------------|
| JM1        | 0.720 | 0.730 | 0.740 | 0.720 | 0.540 | 0.720 | 0.610 | 0.750 | 0.720 | 0.700 | 0.720 | 0.730 | 0.730    | 0.739            |
| KC1        | 0.760 | 0.760 | 0.770 | 0.750 | 0.760 | 0.710 | 0.670 | 0.690 | 0.780 | 0.760 | 0.790 | 0.813 | 0.8487   |
| KC3        | 0.860 | 0.820 | 0.830 | 0.740 | 0.740 | 0.810 | 0.620 | 0.740 | 0.860 | 0.780 | 0.800 | 0.890 | 0.882  |
| PC1        | 0.800 | 0.860 | 0.900 | 0.730 | 0.750 | 0.900 | 0.700 | 0.850 | 0.900 | 0.860 | 0.870 | 0.890 | 0.9297     |

* AUC values are referred from [11]

From Table 6, it can be observed that the proposed prediction approach i.e. BPSO-BCE-ANN has better prediction performance over another compared classifier. Parameters and models of AUC values with superscript ‘a’ are discussed in [11]. It can be easily found that the model with BPSO having BCE as its fitness function achieved the highest AUC value in three out of four datasets, i.e., PC1, KC1, and KC3. Even for JM1 dataset, our model achieved close to the highest value among other classifiers.

All AUC values achieved are greater than 0.73 this implies that BPSO with BCE as fitness function can be used as software defect prediction. Since BPSO removed dimensions of software metrics from 21 to 7,4,5 and 6 for PC1, KC1, KC3, and JM1 respectively, this shows that there was a high correlation between those software metrics. Hence, it was necessary to remove those highly correlated metrics. It also shows that although accuracy is used as a fitness function but from the results obtained it can be concluded that BCE also can be used as an alternative of accuracy for the fitness function.

7. Conclusion

The purpose of the paper is to develop the approach for software defect prediction and dimensionality reduction of the software metrics using BCE + BPSO + ANN. The proposed method is found to be effective for all of the four data set i.e. PC1, KC1, JM1, and KC3. The combination of BPSO with BCE as its fitness function successfully reduced the attributes from the selected 22 attributes to 7, 4, 5, and 6 for PC1, KC1, KC3, and JM1 respectively. This proves that our proposed method is quite effective for removing highly co-related features. Attained AUC values for each dataset used are above 70% which implies that the suggested approach is convincingly good for software fault proneness prediction.
8. References

[1] He C, Xing J, Zhu R, Li J, Yang Q and Xie L 2013 A New Model for Software Defect prediction using Particle Swarm Optimization and Support Vector Machine 25th Chinese Control and Decision Conference (CCDC) pp 4107-08

[2] Ying M, Guangchun L, Xue Z and Aiguo C 2012 Transfer learning for cross-company software defect prediction Information and Software Technology pp 249-51

[3] Cong J and Shu W J 2015 Prediction approach of software fault-proneness based on hybrid artificial neural network and quantum particle swarm optimization Applied Soft Computing 35, pp.717-25.

[4] Susana M V, Luís F M, Goncalo J F and João M C S 2013 Modified binary PSO for feature selection using SVM applied to mortality prediction of septic patients Applied Soft Computing pp 3496-7

[5] Shikha A and Rajesh R 2019 Some Modifications in Binary Particle Swarm Optimization for Dimensionality Reduction Proceedings of the 18th Online World Conference on Soft Computing in Industrial Applications (WSC18). WSC 2014. Advances in Intelligent Systems and Computing p 206

[6] James K and Russell C E 1997 A Discrete Binary version of the Particle Swarm Algorithm IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation pp 4104-6

[7] Muhammd I M, MinRui F, Ling W and Xiping F 2011 A Novel Hybrid Binary PSO Algorithm Advances in Swarm Intelligence, ICSI 2011. Lecture Notes in Computer Science, vol 6728. Springer, Berlin, Heidelberg p 94

[8] Shikha A and Rajesh R 2019 Some Modifications in Binary Particle Swarm Optimization for Dimensionality Reduction Proceedings of the 18th Online World Conference on Soft Computing in Industrial Applications (WSC18). WSC 2014. Advances in Intelligent Systems and Computing p 208

[9] Russell E 1995 A New Optimizer Using Particle Swarm Theory MHS’95. Proceedings of the Sixth International Symposium on Micro Machine and Human Science p 40

[10] Stefan L, Bart B, Christophe M and Swantje P 2008 Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings IEEE Transactions on Software Engineering vol 34 pp 485-96