International Journal of Applied Engineering Research ISSN 0973-4562 Volume 13, Number 19 (2018) pp. 14291-14297


# Software Defect Prediction using Ant Colony Optimization

**Kiran Kumar B., Dr. Jayadev Gyani, Dr. Narsimha G**
*Department. of Information Technology (IT), Kakatiya Institute of Technology and Science (KITS), Warangal, Telangana, India.*
*Department. of Computer Science, CCIS, Majmaah University, Saudi Arabia.*
*Department of Computer Science Engineering, JNTUH College of Engineering, Sultanpur, India.*
*Email: kiran_b_kumar@yahoo.com, jayadevgyani@yahoo.com, narsimha06@gmail.com*

**Abstract**

A defect or bug in the software project may arise due the poor design or poor coding of the software modules. When a bug occurs in the project, it produces incorrect results. Occurrences of bugs in the software increase the total estimated cost of the project. This cost can be reduced by predicting the bugs in the software before delivery of the product. In this paper, we implemented Ant Colony Optimization (ACO) method on eight different open source datasets and compared with Logistic Regression (LR), k-Nearest Neighbors, and SVM algorithms. The results show that ACO gives better performance on other prediction methods.

***Keywords:*** Defect prediction, Ant Colony Optimization, Logistic Regression, k-Nearest Neighbors, Support Vector Machine.

## INTRODUCTION

Defect prediction in programming is seen as a standout amongst the most helpful and cost efficient operation. It is extremely hard to build up a software system without any defects [1-6]. Hence, most software development processes have been endeavoring to detect the defects in the early stages and to correct the defects, however, many as could be expected under the circumstances to enhance the procedure and venture execution before they discharge

their software products [7, 8]. Various prediction techniques were introduced for prediction of defective components in the software projects. These methodologies commonly utilize different highlights, e.g., process metrics, previous-defect metrics, source code metrics, and so forth, to characterize a class/file/module and utilize a characterization calculation to predict if a class/file/module is inadequate or not [7]. The diverse expectation procedures are: cross-validation prediction, cross-version prediction, cross-project prediction [8].

Cross-Project Defect Prediction (CPDP) has as of late turned out to be exceptionally main stream in the field of software defect prediction. It was by and large regarded as a binary classification problem or a regression problem in the greater part of the past examinations [9, 10]. Here, the prediction models are worked by taking information from various ventures as the training set and a test set got from the local project as the target project data [11, 12]. To handle this, cases of source information like target information are chosen to fabricate classifiers. In programming datasets the proportion of defective class to clean class is far lower and is called as class imbalance problem. It for the most part brings down the execution of classifiers [13, 14]. Cross-project defect prediction is exceptionally engaging on the grounds that (i) it permits foreseeing deserts in ventures for which the accessibility of information is restricted, and (ii) it permits creating generalizable expectation models [15-18]. Nonetheless, existing examination recommends that cross-project prediction is especially testing and, because of heterogeneity of projects, prediction exactness is not generally great [19, 20]. To foresee the cross product defect, the analysts utilized different sorts of methodologies; they are; multi-objective cross-project defect prediction, HISNN (Hybrid Instance Selection Using Nearest-Neighbor), Multi-Objective (MO) Learning techniques, ROCPDP (Ranking Oriented CPDP) method, HYDRA (Hybrid Model Reconstruction Approach) etc.

A novel, multi-objective approach for cross-project defect prediction, in light of a multi-Objective logistic regression display fabricated utilizing a hereditary calculation. The multi-objective approach permits software engineers to pick predictors accomplishing a trade-off between number of likely defect-prone artifacts (effectiveness) and LOC to be dissected/tried instead of furnishing the software engineer with a single predictive model, [1, 21]. HISNN technique uses a hybrid classification selectively by learning local knowledge by kNN and global knowledge by naive Bayes. Examples having solid nearby information are distinguished by means of nearest-neighbors with the same class label. Be that as it may, it has low probability of detection or high probability of false alarm which is unrealistic to utilize. [4]. An effective Multi-Objective Improved Teaching– Learning Based Optimization (MO-ITLBO) algorithm utilizes a grid-based approach with a specific end goal to keep decent variety in the outer document. This calculation is productive and has focused execution over the MO problems. In any case, these calculations have lacked in improving a portion of the multi-objective problems [1, 22]. A HYbrid model reconstruction approach (HYDRA) for cross-project defect prediction contains two phases: Genetic Algorithm (GA) phase and Ensemble Learning (EL) phase. These two stages make a massive composition of classifiers [23]. Overview of Ant Colony Optimization and its applications with more classical techniques from artificial intelligence and operations research are discussed [27]. Ant Colony Optimization Technique (ACOT) for prediction of reliability of software and optimizing the accuracy of software reliability predictive models when used with raw data is proposed in [28]. AntMiner+ is used in building of internal rating systems for customers to identify their credit risk [29].

To predict the software cross-project defect, we used a multi objective ACO algorithm. At first, the source data and the target data are preprocessed utilizing the pre-processing technique to standardize the both data sets. The probability of defect, probability of false alarm and the misclassification cost is the three primary multi objective defect prediction problems. These issues are addressed by the ACO calculation.

The remainder of the paper is depicted in the section underneath. In section 2, the background of the research work is depicted. The proposed work is depicted in section 3. In section 4, the experimentation results are discussed.

## RELATED WORK

G. Canfora *et al*. [2] have formalized the defect-prediction issue as a multi objective optimization issue. In particular, they have introduced an approach, authored as multi objective defect predictor (MODEP), in light of multi objective types of machine learning techniques—logistic regression and decision trees specifically—trained utilizing a genetic algorithm. The multi objective approach permits software engineers to pick predictors accomplishing a particular trade off between the quantity of likely defect-prone classes or the number of defects that the examination would likely discover (effectiveness), and lines of code to be investigated/tried (which can be considered as an intermediary of the cost of code assessment). After effects of an empirical evaluation on 10 datasets from the PROMISE repository demonstrate the quantitative prevalence of MODEP with deference over single-objective predictors and as for trivial by measure in climbing or diving request. Likewise, MODEP beats an option approach for cross-project prediction, in view of local heaps of comparative classes.

T. Khoshgoftaar and Y. Liu [3] have displayed a hereditary programming-based choice tree model which encourages a multi-objective optimization with regards to the software quality classification problem. The primary target was to limit the "Modified Expected Cost of Misclassification". The second goal was to advance the number of predicted fault-prone modules with the end goal that, which was equivalent to the quantity of modules which could be examined by the dispensed assets. Some usually utilized classification techniques, for example, logistic regression, decision trees, and analogy-based reasoning is not suited for directly optimizing multi-objective criteria. In contrast, genetic programming was especially suited for the multi-objective optimization issue. An empirical contextual analysis of a real-world industrial software system shows the promising outcomes and the handiness of their model.

D. Ryu *et al*. [4] have proposed a Hybrid Instance Selection Using Nearest-Neighbor (HISNN) technique that plays out a crossover order specifically learning local knowledge (via k-nearest neighbor) and global knowledge (via naive Bayes). Occasions having solid nearby information are recognized via nearest-neighbors with a similar class name. Past investigations demonstrated low PD (probability of detection) or high PF (probability of false alarm) which was unfeasible to utilize. Their test comes about demonstrated that HISNN creates high general execution and also high PD and low PF.

D. Ryu and J. Baik [5] have focused to distinguish compelling multi-objective learning strategies under Cross-Project (CP) situations. Three targets are concocted considering the class imbalance context. The primary target was to augment the probability of detection (PD). The second target was to limit the probability of false alarm (PF). The third target was to expand the general execution (e.g., balance). They have exhibited novel MO naive Bayes learning systems demonstrated by a harmony search meta-heuristic calculation. Their methodologies are contrasted and single-objective models, other existing MO models and within-project defect prediction models. The test comes about demonstrated that their methodologies are promising. Therefore, they can be viably connected to fulfill different forecast needs under CP settings.

G. You *et al*. [6] have talked about Cross-Project Defect Prediction (CPDP) has as of late turned out to be exceptionally well known in the field of software defect prediction. In their paper, CPDP was defined as a ranking problem. Enlivened by the possibility of the guide savvy approach toward figuring out how to rank, they have exhibited a ranking-oriented CPDP approach called ROCPDP. A contextual investigation led to the datasets gathered from AEEEM and PROMISE demonstrated that ROCPDP outflanks the eight benchmark techniques in two CPDP situations, to be specific One-To-One and Many-To-One. Additionally, in the Many-To One situation, ROCPDP was, all things considered, equivalent to the best baseline strategy performed in a particular inside undertaking defect prediction situation.

X. Xia *et al.* [7] have proposed a Hybrid Model Reconstruction Approach (HYDRA) for cross-project defect prediction, which incorporates two phases: genetic algorithm (GA) phase and ensemble learning (EL) phase. These two phases make a massive composition of classifiers. To analyze the advantages of HYDRA, they have performed investigates 29 datasets from the promise repository which contains a sum of 11,196 cases (i.e., java classes) named as defective or clean. They have contrasted their approach and the most as of late proposed cross-project defect prediction approaches. Their outcomes demonstrated that HYDRA accomplishes an average F1-score of 0.544. By and large, over the 29 data sets, these outcomes compare to a change in the F1-scores of 26.22%, 34.99%, 47.43%, 28.61%, and 30.14% over TCA+, Peters Filter, GP, MO, AND CODEP, individually.

O. Choi *et al*. [8] have talked about; the software defect prediction was a standout amongst the most essential tasks for software quality change. They have examined if the class imbalance learning could be advantageous for CPDP. In their approach, the asymmetric misclassification cost and the similarity weights got from distributional qualities are nearly related to control the proper resampling system. They have played out the impact estimate A-statistics test to assess the size of the change. For the statistical significant test, they utilized Wilcoxon rank-sum test. The exploratory outcomes demonstrated that their approach could give higher prediction execution wander randomly. When an ant finds a source of

food, than both the current CPDP procedure and the current class imbalance method.

## PROPOSED WORK

### Ant Colony Optimization:

The ant colony algorithm simulates the behavior of the ants in searching of the food.

In the process of searching for the food, different ants chooses different paths to reach the food source from their nest and back to their nest by leaving some sort of chemical (pheromone) on their path. T he ants walked through the shortest route may reach the nest quickly compared to the ants walked through the longest path. As the time increases, the chemical deposited on the path may be evaporated which results in more pheromone deposited on shortest path than the pheromone deposited on the longest path.

All other ants started from the nest follow the path which contains more pheromone deposited by depositing their chemical on the path. So the pheromone deposited on the longest path may disappear after certain time and hence no ants choose that path. In this way the behavior of the ants can be used to find the optimal or shortest path. As the ant-colony works on a very dynamic system, we can simulate the behavior of ants in finding the solutions for optimization problems.In this work, we simulated behavior of ants in finding the optimum solution for software defect prediction. We considered Jureczko datasets [25] obtained from PROMISE repository [26] shown in Table 1 for experimentation.

**Table 1.** Sample datasets

| Name of the Dataset | Number of attributes | Number of records |
|---|---|---|
| CM1 | 38 | 369 |
| KC1 | 95 | 145 |
| KC2 | 22 | 522 |
| KC3 | 40 | 194 |
| MC2 | 40 | 125 |
| PC1 | 38 | 705 |
| PC3 | 38 | 1077 |
| PC4 | 38 | 1458 |

### Data preprocessing:

In preprocessing step, we selected nine independent attributes which are common in all these datasets and one class label attribute. Different datasets contain different values for class label attribute like Y/N, 1/0, T/F. These values are converted to 1/0 for all the datasets. Each dataset is divided randomly into training set (70% of records) and test set (30% of records). Training set is used to build the model and test set is used predict the class label.

## EXPERIMENTATION AND RESULTS

The proposed software defect prediction module contains true prediction and false prediction. True prediction implies true positive or true negative, it refers the quantity of software which effectively anticipated as non defective or defective software. False prediction implies false positive or false negative, it eludes the quantity of software which are incorrect as defective or non defective software. The present execution of software defect is assessed by utilizing geometric mean.

**Table 2.** Confusion Matrix

| | | Real class | |
|---|---|---|---|
| | | Defective | Non defective |
| Predicted class | Defective | TP (True Positive) | FP (False Positive) |
| | Non-defective | FN (False Negative) | TN (True Negative) |

Table 2 demonstrates the confusion matrix, used to assess the predictive performance. True Positive (TP) is various defect class occurrence predicted accurately as defective. False Positive (FP) is the quantity of non defective class case predicted as defective. False Negative (FN) is the quantity of defect class occurrence predicted as non-negative. True Negative (TN) is number of non-defect class case predicted as non-defective.

We computed sensitivity, specificity, geometric mean, precision, F-measure and accuracy by using the following formulas.

$$Sensitivity = \text{Re}call = \frac{TP}{TP + FN}$$

$$Specificity = \frac{TN}{TN + FP}$$

$$Geometric \quad Mean = \sqrt{sensitivity \times specificity}$$

$$\text{Pr}ecision = \frac{TP}{TP + FP}$$

$$F - Measure = \frac{2 * Precision * Recall}{Precision + Recall}$$

$$Accuracy = \frac{(TP + TN)}{(FP + FN + TP + TN)}$$

The performance of the proposed software defect predictor is compared with existing approaches like LR (Logistic Regression), K-NN (K-Nearest Neighbor) and SVM (Support Vector Machine).

The flowing tables depict the comparison of proposed ACO algorithm with existing algorithms.

**Data Set: CM1**

| Algorithm | GM | Sensitivity | Specificity | Precision | Fmeasure | Accuracy |
|---|---|---|---|---|---|---|
| LR | 0.73 | 0.54 | 1.0 | 1.0 | 0.70 | 0.86 |
| KNN | 0.74 | 0.57 | 0.98 | 0.92 | 0.70 | 0.87 |
| SVM | 0.75 | 0.56 | 1.0 | 1.0 | 0.72 | 0.87 |
| **ACO** | **0.80** | **0.65** | **1.0** | **1.0** | **0.78** | **0.91** |

D**ata Set: KC1**

| Algorithm | GM | Sensitivity | Specificity | Precision | Fmeasure | Accuracy |
|---|---|---|---|---|---|---|
| LR | 0.75 | 0.57 | 1.0 | 1.0 | 0.72 | 0.83 |
| KNN | 0.71 | 0.53 | 0.95 | 0.87 | 0.66 | 0.81 |
| SVM | 0.81 | 0.66 | 1.0 | 1.0 | 0.8 | 0.89 |
| **ACO** | **0.85** | **0.72** | **1.0** | **1.0** | **0.84** | **0.91** |

**Data Set: KC2**

| Algorithm | GM | Sensitivity | Specificity | Precision | Fmeasure | Accuracy |
|---|---|---|---|---|---|---|
| LR | 0.92 | 0.93 | 0.92 | 0.98 | 0.95 | 0.93 |
| KNN | 0.93 | 0.95 | 0.92 | 0.98 | 0.96 | 0.94 |
| SVM | 0.96 | 0.97 | 0.96 | 0.99 | 0.98 | 0.96 |
| **ACO** | **0.99** | **0.98** | **1.0** | **1.0** | **0.99** | **0.98** |

**Data Set: KC3**

| Algorithm | GM | Sensitivity | Specificity | Precision | Fmeasure | Accuracy |
|---|---|---|---|---|---|---|
| L R | 0.66 | 0.45 | 0.97 | 0.83 | 0.58 | 0.85 |
| KNN | 0.73 | 0.54 | 1.0 | 1.0 | 0.70 | 0.89 |
| SVM | 0.70 | 0.5 | 1.0 | 1.0 | 0.66 | 0.87 |
| **ACO** | **0.76** | **0.58** | **1.0** | **1.0** | **0.73** | **0.92** |

**Data Set: MC2**

| Algorithm | GM | Sensitivity | Specificity | Precision | Fmeasure | Accuracy |
|---|---|---|---|---|---|---|
| LR | 0.93 | 0.90 | 0.95 | 0.90 | 0.90 | 0.93 |
| KNN | 0.85 | 0.76 | 0.94 | 0.90 | 0.83 | 0.87 |
| SVM | 0.82 | 0.75 | 0.9 | 0.81 | 0.78 | 0.84 |
| **ACO** | **0.91** | **0.84** | **1.0** | **1.0** | **0.91** | **0.93** |

**Data Set: PC1**

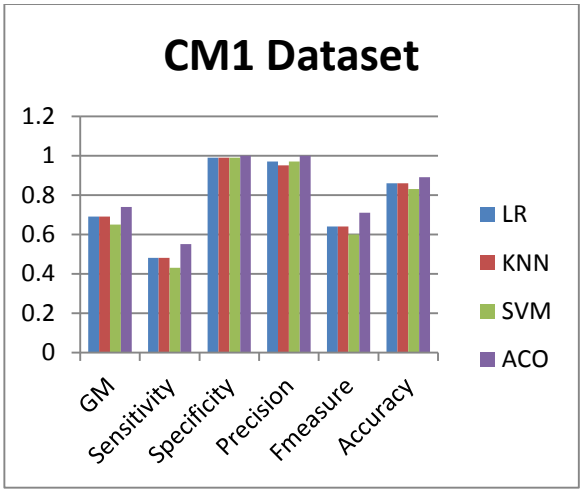
| Algorithm | GM | Sensitivity | Specificity | Precision | Fmeasure | Accuracy |
|---|---|---|---|---|---|---|
| LR | 0.57 | 0.33 | 1.0 | 1.0 | 0.5 | 0.83 |
| KNN | 0.56 | 0.32 | 0.99 | 0.93 | 0.47 | 0.82 |
| SVM | 0.62 | 0.38 | 1.0 | 1.0 | 0.55 | 0.86 |
| **ACO** | **0.70** | **0.5** | **1.0** | **1.0** | **0.66** | **0.91** |

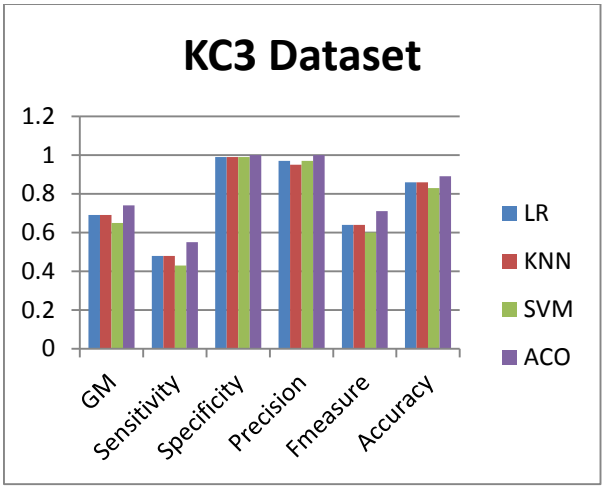
**Data Set: PC3**

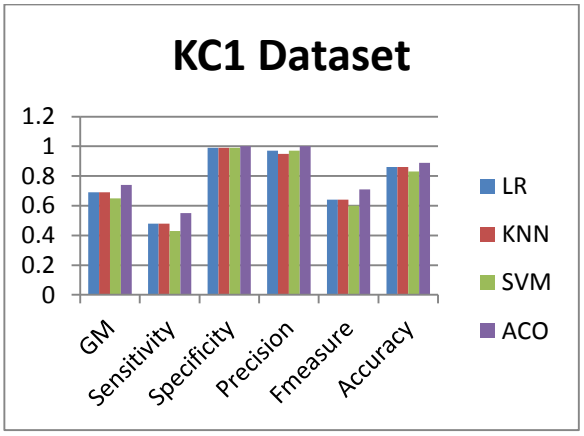
| Algorithm | GM | Sensitivity | Specificity | Precision | Fmeasure | Accuracy |
|---|---|---|---|---|---|---|
| LR | 0.61 | 0.37 | 1.0 | 1.0 | 0.54 | 0.82 |
| KNN | 0.64 | 0.41 | 0.99 | 0.97 | 0.58 | 0.84 |
| SVM | 0.64 | 0.41 | 1.0 | 1.0 | 0.58 | 0.85 |
| **ACO** | **0.74** | **0.56** | **1.0** | **1.0** | **0.71** | **0.91** |

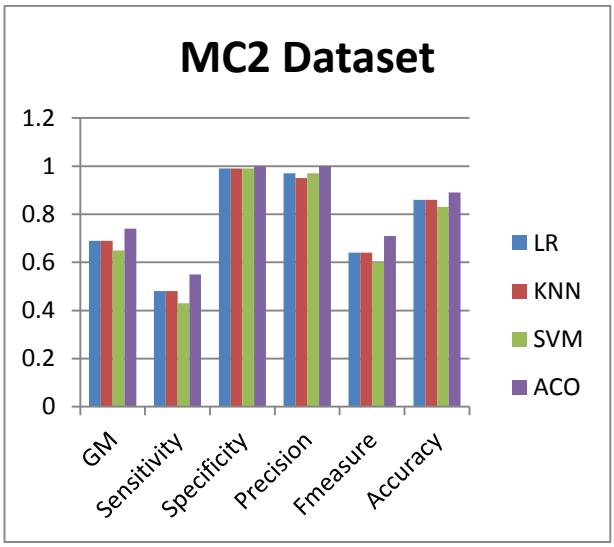
**Data Set: PC4**

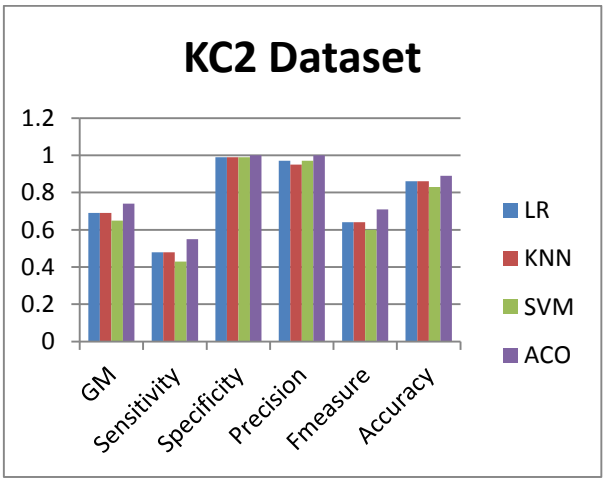
| Algorithm | GM | Sensitivity | Specificity | Precision | Fmeasure | Accuracy |
|---|---|---|---|---|---|---|
| LR | 0.69 | 0.48 | 0.99 | 0.97 | 0.64 | 0.86 |
| KNN | 0.69 | 0.48 | 0.99 | 0.95 | 0.64 | 0.86 |
| SVM | 0.65 | 0.43 | 0.99 | 0.97 | 0.60 | 0.83 |
| **ACO** | **0.74** | **0.55** | **1.0** | **1.0** | **0.71** | **0.89** |

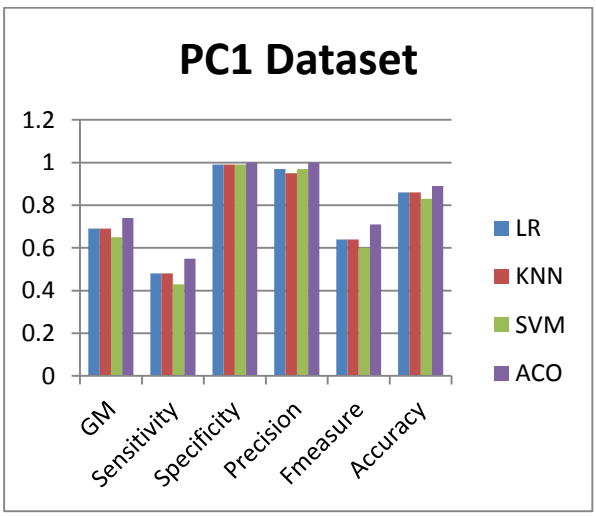

CM1 Dataset
1.2
1
0.8
0.6
0.4
0.2
0
GM
Sensitivity
Specificity
Precision
Fmeasure
Accuracy
LR
KNN
SVM
ACO

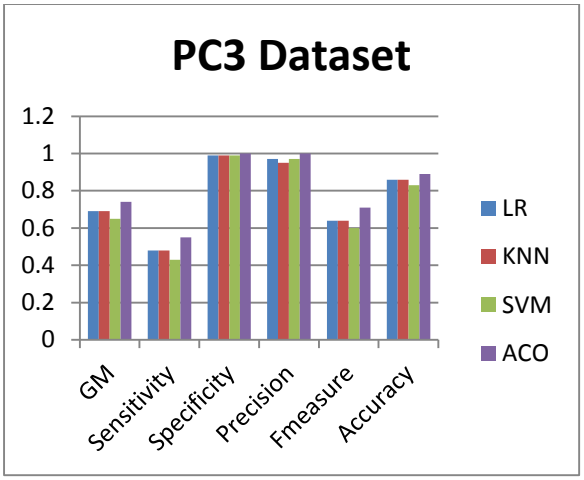

KC3 Dataset
1.2
1
0.8
0.6
0.4
0.2
0
GM
Sensitivity
Specificity
Precision
Fmeasure
Accuracy
LR
KNN
SVM
ACO

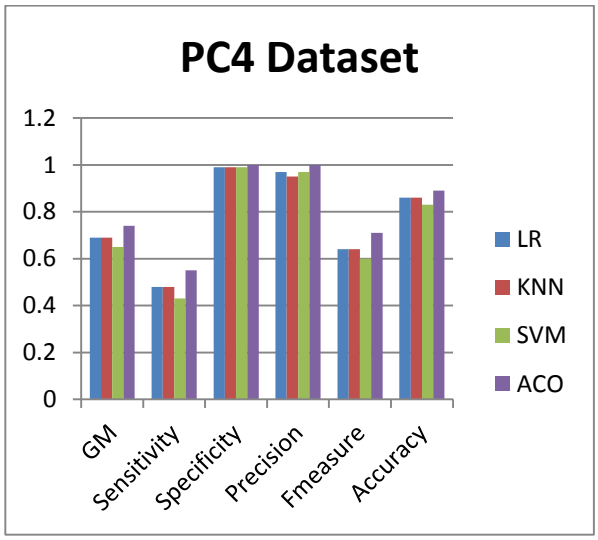

KC1 Dataset
1.2
1
0.8
0.6
0.4
0.2
0
GM
Sensitivity
Specificity
Precision
Fmeasure
Accuracy
LR
KNN
SVM
ACO


MC2 Dataset
1.2
1
0.8
0.6
0.4
0.2
0
GM
Sensitivity
Specificity
Precision
Fmeasure
Accuracy
LR
KNN
SVM
ACO


KC2 Dataset
1.2
1
0.8
0.6
0.4
0.2
0
GM
Sensitivity
Specificity
Precision
Fmeasure
Accuracy
LR
KNN
SVM
ACO


PC1 Dataset
1.2
1
0.8
0.6
0.4
0.2
0
GM
Sensitivity
Specificity
Precision
Fmeasure
Accuracy
LR
KNN
SVM
ACO




## CONCLUSIONS

In this paper, we implemented ant colony optimization technique to predict the software defects by considering different datasets. The results show that the proposed approach delivers the promising outcomes.

## REFERENCES

[1] G. Canfora, A. De Lucia, M. Di Penta, R. Oliveto, A. Panichella and S. Panichella, "Multi-objective Cross-Project Defect Prediction", 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation, 2013.

[2] G. Canfora, A. Lucia, M. Penta, R. Oliveto, A. Panichella and S. Panichella, "Defect prediction as a multi objective optimization problem", Software Testing, Verification and Reliability, vol. 25, no. 4, pp. 426-459, 2015.

[3] T. Khoshgoftaar and Y. Liu, "A Multi-Objective Software Quality Classification Model Using Genetic Programming", IEEE Transactions on Reliability, vol. 56, no. 2, pp. 237-245, 2007.

[4] D. Ryu, J. Jang and J. Baik, "A Hybrid Instance Selection Using Nearest-Neighbor for Cross-Project Defect Prediction", Journal of Computer Science and Technology, vol. 30, no. 5, pp. 969-980, 2015.

[5] D. Ryu and J. Baik, "Effective multi-objective naïve Bayes learning for cross-project defect prediction", Applied Soft Computing, vol. 49, pp. 1062-1077, 2016.

[6] G. You, F. Wang and Y. Ma, "An Empirical Study of Ranking-Oriented Cross-Project Software Defect Prediction", International Journal of Software Engineering and Knowledge Engineering, vol. 26, no. 0910, pp. 1511-1538, 2016.

[7] X. Xia, D. Lo, S. Pan, N. Nagappan and X. Wang, "HYDRA: Massively Compositional Model for Cross-Project Defect Prediction", IEEE Transactions on Software Engineering, vol. 42, no. 10, pp. 977-998, 2016.

[8] D. Ryu, O. Choi and J. Baik, "Value-cognitive boosting with a support vector machine for cross-project defect prediction", Empirical Software Engineering, vol. 21, no. 1, pp. 43-71, 2014.

[9] C. Hu, X. Xue, L. Huang, H. Lyu, H. Wang, X. Li, H. Liu, M. Sun and W. Sun, "Decision-Level Defect Prediction Based on Double Focuses", Chinese Journal of Electronics, vol. 26, no. 2, pp. 256-262, 2017.

[10] Y. Shi, M. Li, S. Arndt and C. Smidts, "Metric-based software reliability prediction approach and its application", Empirical Software Engineering, 2016.

[11] F. Porto and A. Simao, "Feature Subset Selection and Instance Filtering for Cross-project Defect Prediction - Classification and Ranking", CLEI electronic journal, vol. 19, no. 3, pp. 4:1-4:17, 2016.

[12] X. Yang, D. Lo, X. Xia and J. Sun, "TLEL: A two-layer ensemble learning approach for just-in-time defect prediction", Information and Software Technology, vol. 87, pp. 206-220, 2017.

[13] T. Lee, J. Nam, D. Han, S. Kim and H. Peter In, "Developer Micro Interaction Metrics for Software Defect Prediction", IEEE Transactions on Software Engineering, vol. 42, no. 11, pp. 1015-1035, 2016.

[14] M. Bisi and N. Goyal, "An ANN-PSO-based model to predict fault-prone modules in software", International Journal of Reliability and Safety, vol. 10, no. 3, p. 243, 2016.

[15] W. Li, Z. Huang and Q. Li, "Three-way decisions based software defect prediction", Knowledge-Based Systems, vol. 91, pp. 263-274, 2016.

[16] T. Lee, J. Nam, D. Han, S. Kim and H. Peter In, "Developer Micro Interaction Metrics for Software Defect Prediction", IEEE Transactions on Software

Engineering, vol. 42, no. 11, pp. 1015-1035, 2016.

[17] R. Rana, M. Staron, C. Berger, J. Hansson, M. Nilsson and W. Meding, "Analyzing defect inflow distribution and applying Bayesian inference method for software defect prediction in large software projects", Journal of Systems and Software, vol. 117, pp. 229-244, 2016.

[18] F. Zhang, A. Mockus, I. Keivanloo and Y. Zou, "Towards building a universal defect prediction model with rank transformed predictors", Empirical Software Engineering, vol. 21, no. 5, pp. 2107-2145, 2015.

[19] D. Ryu, J. Jang and J. Baik, "A transfer cost-sensitive boosting approach for cross-project defect prediction", Software Quality Journal, vol. 25, no. 1, pp. 235-272, 2015.

[20] M. Cheng, G. Wu, H. Wan, G. You, M. Yuan and M. Jiang, "Exploiting Correlation Subspace to Predict Heterogeneous Cross-Project Defects", International Journal of Software Engineering and Knowledge Engineering, vol. 26, no. 0910, pp. 1571-1580, 2016.

[21] X. Jing, F. Wu, X. Dong and B. Xu, "An Improved SDA Based Defect Prediction Framework for Both Within-Project and Cross-Project Class-Imbalance Problems", IEEE Transactions on Software Engineering, vol. 43, no. 4, pp. 321-339, 2017.

[22] Z. Zhu, J. Xiao, S. He, Z. Ji and Y. Sun, "A multi-objective memetic algorithm based on locality-sensitive hashing for one-to-many-to-one dynamic pickup-and-delivery problem", Information Sciences, vol. 329, pp. 73-89, 2016.

[23] P. Kumar and S. K., "Defect Prediction Model for AOP-based Software Development using Hybrid Fuzzy C-Means with Genetic Algorithm and K-Nearest Neighbors Classifier", International Journal of Applied Information Systems, vol. 11, no. 2, pp. 26-30, 2016.

[24] J. Luo, Q. Liu, Y. Yang, X. Li, M. Chen and W. Cao, "An artificial bee colony algorithm for multi-objective optimization", Applied Soft Computing, vol. 50, pp. 235-251, 2017.

[25] M. Jureczko and L. Madeyski, "Towards identifying software project clusters with regard to defect prediction", Proceedings of the 6th International Conference on Predictive Models in Software Engineering - PROMISE '10, 2010.

[26] T. Menzies, B. Cagayan, Z. He, E. Kocaguneli, J. Krall, F. Peters, et al., The PROMISE Repository of empirical software engineering data, 2012 http:// openscience.us/repo/.

[27] Christian Blum, “Ant colony optimization: Introduction and recent trends”, Elsevier, Physics of Life Reviews 2 pp. 353–373, 2005

[28] Ramakanta Mohanthy, Venkatshwarlu Naik, Azmath Mubeen, “Predicting Software Reliability Using Ant Colony OptimizationTechnique” 2014 Fourth International Conference on Communication Systems and Network Technologies, pp. 496-500, 2014

[29] D Martens, T Van Gestel, M De Backer, R Haesen, J Vanthienen and B Baesens, “Credit rating prediction using Ant Colony Optimization” Journal of the Operational Research Society (2010) 61, pp. 561-573, 2010