International Journal of Software Engineering and Its Application
Vol.12, No.1 (2018), pp.21-32
http://dx.doi.org/10.21742/ijseia.2018.12.1.02

Review: Software Defect Prediction for Class Imbalance Datasets

K. Nitalakshewara Rao and Ch. Satyananda Reddy

Andhra University, Visakhapatnam
nitla_kolukula@yahoo.co.in, satyanandau@yahoo.com

Abstract

In today's growing world, the quality of software developed is very essential for critical and huge applicable areas. The undetected bug in the software leads to an avalanche breakdown of services in the total network. On the other hand, the complete defect analysis in each and every module is again a huge task with lot of man-hours. A trade of between these two is established by following Software Defect Prediction (SDP) techniques or algorithms. Software defect Prediction is the process of finding the defects in the specific modules of the software. This paper aims to review the Software Defect Predicting (SDP) techniques and algorithms proposed in the literature to alleviate in the area of class imbalance software defect.

Keywords: *software defects analysis, classification, decision tree, class imbalance learning, Software Defect Prediction (SDP)*

1. Introduction

In Machine Learning community, and in Data Mining works, Classification has its own importance. Classification is an important part and the research application field in the data mining [1]. With ever-growing volumes of operational data, many organizations have started to apply data-mining techniques to mine their data for novel, valuable information that can be used to support their decision making [2]. Decision tree learning is one of the most widely used and practical methods for inductive inference [3].

Software Defect Prediction:

A learning algorithm in software testing which aims to locate and analyze which part of software is more likely to contain defects is known as software defect predicting (SDP) algorithm. When the project budget is limited or the whole software system is too large to be tested completely, a good defect classifier can guide software engineers to focus the testing on defect-prone parts of software.

SDP data feature: collected training data contains much more non-defective modules (majority) than defective ones (minority), as shown in the Table 1. The rare defective examples are more costly and important. Class imbalanced distribution is harmful for classification performance, especially the minority class. Existing methods to tackle class imbalance in SDP problems are under sampling non-defective examples [4-6], over sampling defective examples [7-8], cost-sensitive: setting a higher misclassification cost for the defect class [9-10].

They were compared to the methods without applying any class imbalance techniques, and showed usefulness. However, the following issues have not been answered:

1. In which aspect and to what extent class imbalance learning can benefit SDP problems? (*e. g.*, more defects are detected or fewer false alarms?).

Received (May 31, 2017), Review Result (July 2, 2017), Accepted (July 7, 2017)

ISSN: 1738-9984 IJSEIA

2. Which class imbalance learning methods are more effective? Such information would help us to understand the potential of class imbalance learning methods in SDP and develop better solutions.

Software Engineering Datasets:

The software engineering datasets which are publicly available for software defect prediction analysis are given below in Table 1. The Table 1 contains S.no, System, Features, Total number of modules (examples) present in the software engineering project, percentage of defective modules and the imbalance ratio of the dataset, which can give the idea of the level of class imbalance in the dataset.

Table 1. Details of the PROMISE Data Sets of Software Engineering Projects

S.no.	System	Features	#Module	%defective	IR
1.	ar1	30	121	10.89	12.44
2.	ar3	30	63	5.04	6.87
3.	ar4	30	107	21.4	4.35
4.	ar5	30	36	2.28	3.5
5.	ar6	30	101	15.15	5.73
6.	CM1	38	327	137.34	6.78
7.	CM11	38	344	144.48	7.19
8.	DATATRIEVE	9	130	14.3	10.81
9.	JM1	22	7782	130115	3.65
10.	JM11	22	9593	168740	4.45
11.	KC2	22	522	558.5	3.89
12.	KC3	40	194	69.84	4.38
13.	KC31	40	200	79.2	4.55
14.	MC1	39	1988	914.48	42.21
15.	MC2	40	125	55	1.84
16.	MC11 39	9277	6308	135.4	
17.	Mozilla4	6	15545	794038	2.04
18	MW1	38	1253	68.31	8.37
19.	PC1	38	705	430.05	10.55
20.	PC2	37	745	119.2	45.56
21.	PC3	38	1077	1443.18	7.03
22.	pc4	38	1458	2595.24	7.19
23.	pc5	38	1054	1402.2	7.01

Experimental Validation Framework:

The Experimental validation methodology used for training and testing the classifier is 10 fold cross validation. The main steps in the 10 fold cross validation is shown in the below Figure 1.

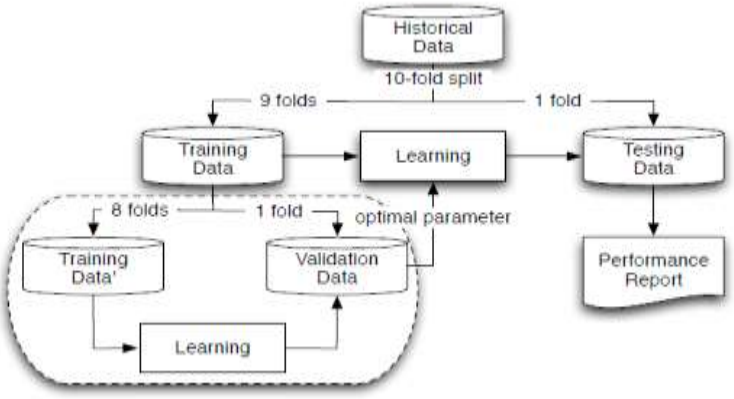

Figure 1. The Experimental Framework used for 10 fold Cross Validation for Training and Testing the Classifier

2. Data Mining

Data Mining is the analysis of (often large) observational data sets to find unsuspected relationships and to summarize the data in novel ways that are both understandable and useful to the owner [11]. There are many different data mining functionalities. A brief definition of each of these functionalities is now presented. The definitions are directly collated from [12]. Data characterization is the summarization of the general characteristics or features of a target class of data.

Association analysis is the discovery of association rules showing attribute value conditions that occur frequently together in a given set of data. Classification is an important application area for data mining. Classification is the process of finding a set of models (or functions) that describe and distinguish data classes or concepts, for the purpose of being able to use the model to predict the class of objects whose class label is unknown. The derived model can be represented in various forms, such as classification rules, decision trees, mathematical formulae, or neural networks. Unlike classification and prediction, which analyze class-labeled data objects, clustering analyzes data objects without consulting a known class label.

Outlier Analysis attempts to find outliers or anomalies in data. A detailed discussion of these various functionalities can be found in [12]. Even an overview of the representative algorithms developed for knowledge discovery is beyond the scope of this paper. The interested person is directed to the many books which amply cover this in detail [11], [12].

The Classification Task

Learning how to classify objects to one of a pre-specified set of categories or classes is a characteristic of intelligence that has been of keen interest to researchers in psychology and computer science. Identifying the common ―core characteristics of a set of objects that are representative of their class is of enormous use in focusing the attention of a person or computer program. For example, to determine whether an animal is a zebra, people know to look for stripes rather than examine its tail or ears. Thus, stripes figure strongly in our *concept* (generalization) of zebras. Of course stripes alone are not sufficient to form a class description for zebras as tigers have them also, but they are certainly one of the important characteristics. The ability to perform classification and to be able to *learn* to classify gives people and computer programs the power to make decisions. The efficacy of these decisions is affected by performance on the classification task.

In machine learning, the classification task described above is commonly referred to as *supervised learning*. In supervised learning there is a specified set of classes, and example objects are labeled with the appropriate class (using the example above, the program is told what a zebra is and what is not). The goal is to generalize (form class descriptions) from the training objects that will enable novel objects to be identified as belonging to one of the classes. In contrast to supervise learning is *unsupervised learning*. In this case the program is not told which objects are zebras. Often the goal in unsupervised learning is to decide which objects should be grouped together—in other words, the learner forms the classes itself. Of course, the success of classification learning is heavily dependent on the quality of the data provided for training—a learner has only the input to learn from. If the data is inadequate or irrelevant then the concept descriptions will reflect this and misclassification will result when they are applied to new data. The popular approach of classification examples are C4.5 [13], CART [14], REP [15]and Random Forest [16].

3. Problem of Imbalanced Datasets

A dataset is class imbalanced if the classification categories are not approximately equally represented. The level of imbalance (ratio of size of the majority class to minority class) can be as huge as 1:99. It is noteworthy that class imbalance is emerging as an important issue in designing classifiers. Furthermore, the class with the lowest number of instances is usually the class of interest from the point of view of the learning task.

4. Data Balancing Techniques

Whenever a class in a classification task is underrepresented (*i.e.*, has a lower prior probability) compared to other classes, we consider the data as imbalanced. The main problem in imbalanced data is that the majority classes that are represented by large numbers of patterns rule the classifier decision boundaries at the expense of the minority classes that are represented by small numbers of patterns. This leads to high and low accuracies in classifying the majority and minority classes, respectively, which do not necessarily reflect the true difficulty in classifying these classes. Most common solutions to this problem balance the number of patterns in the minority or majority classes.

A general issue encountered in data mining is dealing with imbalance datasets, in which one class is predominantly outnumbers the other class. This issue results in high accuracy for the instances of majority class i.e. instances belonging to the predominant class and less accuracy for the instances of minority class. Therefore, when dealing with class imbalance datasets a specific strategy has to be implemented for efficient knowledge discovery from the datasets. There are different type of approaches exists in the literature to handle the problem of class imbalance nature, to name a few are oversampling, under sampling, subset approaches, cost sensitive learning, algorithm level implementations and hybrid techniques which combine more than one approaches.

In oversampling, the instances in the minority subset are oversampled by following different strategies. In under sampling, the instances in the majority subset are reduced by several techniques. In subset approaches, the dataset is split into different subsets to reduce the imbalance nature. In cost sensitive learning, the instances are assigned with cost values and the reshuffling of the dataset is performed by considering the cost values. In algorithmic level approaches, the base algorithm applied to the class imbalance data is modified to suit with the imbalance data learning. In hybrid level implementation, more than one above said approaches are applied to solve the problem of class imbalance learning. Table 2 presents the summary of the strategies in literature.

Table 2. Balancing Strategies Proposed in Literature

S.No	Algorithm	Advantages	Disadvantages
1.	AdaBoost.NC[17]	Improve prediction accuracy of minority	Ignore overall performance of classifier
2.	RUSBoost [18]	Simple, faster and less complex than SMOTE Boost	Unable to solve Multiclass imbalance algorithm problem
3.	Infinitely imbalanced logistic regression [19]	Mostly used for binary classification	Performance is depends on number of outlier in data.
4.	Linear Proximal support vector machines [20]	Handle dynamic class imbalance problem	No consideration for distribution of sample
5.	Boosting SVM [21]	Improved the performance of SVM classifier for prediction minority sample	Ignore imbalance class distribution.

5. Evaluation Criteria's for Class Imbalance Learning

5.1. Evaluation Criteria

To assess the classification results we count the number of true positive (TP), true negative (TN), false positive (FP) (actually negative, but classified as positive) and false negative (FN) (actually positive, but classified as negative) examples. It is now well known that error rate is not an appropriate evaluation criterion when there isclass imbalance or unequal costs. In this paper, we use AUC, Precision, F-measure, TP Rate and TN Rate as performance evaluation measures.

Let us define a few well known and widely used measures for C4.5 [14] as the baseline classifier with the most popular software defect datasets. Apart from these simple metrics, it is possible to encounter several more complex evaluation measures that have been used in different practical domains. One of the most popular techniques for the evaluation of classifiers in imbalanced problems is the Receiver Operating Characteristic (ROC) curve, which is a tool for visualizing, organizing and selecting classifiers based on their tradeoffs between benefits (true positives) and costs (false positives).

The most commonly used empirical measure; accuracy does not distinguish between the numbers of correct labels of different classes, which in the framework of imbalanced problems may lead to erroneous conclusions. For example, a classifier that obtains an accuracy of 90% in a dataset with a degree of imbalance 9:1, might not be accurate if it does not cover correctly any minority class instance.

$$ACC = \frac{TP + TN}{TP + FN + FP + FN}$$

Because of this, instead of using accuracy, more correct metrics are considered. A quantitative representation of a ROC curve is the area under it, which is known as AUC. When only one run is available from a classifier, the AUC can be computed as the arithmetic mean (macro-average) of TP rate and TN rate:

The Area under Curve (AUC) measure is computed by,

$$AUC = \frac{1 + TP_{RATE} - FP_{RATE}}{2}$$

Or

$$AUC = \frac{TP_{RATE} + TN_{RATE}}{2}$$

On the other hand, in several problems we are especially interested in obtaining high performance on only one class. For example, in the diagnosis of a rare disease, one of the most important things is to know how reliable a positive diagnosis is. For such problems, the precision (or purity) metric is often adopted, which can be defined as the percentage of examples that are correctly labeled as positive:

The Precision measure is computed by,

$$\Pr ecision = \frac{TP}{(TP)+(FP)}$$

The F-measure Value is computed by,

$$F-measure = \frac{2 \times \Pr ecision \times \mathrm{Re} call}{\Pr ecision + \mathrm{Re} call}$$

To deal with class imbalance, sensitivity (or recall) and specificity have usually been adopted to monitor the classification performance on each class separately. Note that sensitivity (also called true positive rate, TP rate) is the percentage of positive examples that are correctly classified, while specificity (also referred to as true negative rate, TN rate) is defined as the proportion of negative examples that are correctly classified:

The True Positive Rate measure is computed by,

$$TruePositiveRate = \frac{TP}{(TP)+(FN)}$$

The True Negative Rate measure is computed by,

$$TrueNegativeRate = \frac{TN}{(TN)+(FP)}$$

6. Recent Advances on Class Imbalance Learning: Software Defect Prediction

Currently, the trends of research in software defect analysis with class imbalance learning methods are presented in this section. The recent research directions for software defect analysis are as follows:

Yu Song *et al.*, [22] have applied principal component analysis (PCA) to analyze the factors which lead to defects in software projects, and determine the important factors to improve the developing process. Muhsien M. Yazid *et al.*, [23] have developed an approach involving coercivity versus the nucleus Volume for identified defect regions using Raman spectroscopy as an oxide in character rather than metallic for magnetic force microscopy (MFM) to investigate the domain structure of sintered Nd–Fe–B magnet in a thermally demagnetized state. Yan Xiaobo *et al.*, [24] have studied the fault propagation from the perspective of data signal and control signal. They have constructed signal-component fault propagation model based on signal component graph for embedded software.

Chakkrit Tantithamthavorn *et al.*, [25] have investigated the bias and variance of model validation techniques in the domain of defect prediction using single repetition holdout validation and out-of-sample bootstrap. After empirical experimental validation they have recommend that future defect prediction studies avoid single-repetition holdout validation, and instead, use out-of-sample bootstrap validation. Feng Zhang *et al.*, [26] have investigated how different aggregation schemes impact defect prediction models in software defect analysis. Patrick Rempel *et al.*, [27] have studied the effect on the four main requirements implementation supporting activities that utilize traceability on expected defect rate in the developed software.

Raymond A. Paul *et al.*, [28] have investigated on the Orthogonal Defect Classification (ODC) method, which uses data gathered from several projects to track the reliability of a new program. Combining ODC with root-cause analysis can be useful in many applications where it is important to know the reliability of a program for a specific type of a fault. Ramanath Subramanyam *et al.*, [29] have collected evidence supporting the role of OO design complexity metrics, specifically a subset of the Chidamber and Kemerer (CK) suite, in determining software defects.

Marco D Ambros *et al.*, [30] have proposed an approach which focuses on historical dependencies and defect information to learn about a software system and detect potential problems in the source code. Zude Li *et al.*, [31] have proposed an empirical study of six releases of a large legacy software system (of approx. size 20 million physical lines of code) to analyze PMCDs with respect to: (1) the complexity of fixing such defects and (2) the persistence of defect-prone components across phases and releases. They have given the overall hypothesis that PMCDs inflict a greater negative impact than do other defects on defect correction efficacy.

Chen Qixiang *et al.*, [32] have discussed the role defect analysis in the software testing, defect data collection and the specific methods of defect data. Marıa Perez-Ortiz *et al.*, [33] have developed a specific ordinal over-sampling method which uses ordinal information by approaching over-sampling from a graph-based perspective for improve the performance of machine learning classifiers. Xiao-Yuan Jing *et al.*, [34] have provided effective solutions for both within-project and cross-project class imbalance problems using Subclass Discriminant Analysis (SDA) learning method using features with more powerful classification ability from original metrics.

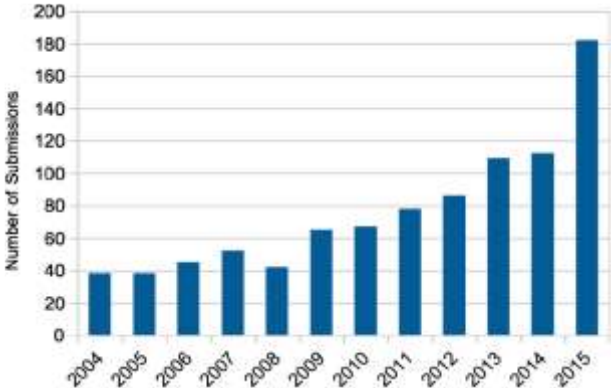

Figure 2. Number of Paper Submissions to the Working Conference on Mining Software Repositories since its First Chapter in 2004 until its most Recent Chapter in 2015

The Working Conference on Mining Software Repositories (MSR) has experienced an increase in the number of paper submissions since its first chapter in 2004 (see Figure 2) [3].

Software Defect (bug) Prediction

The main techniques which is used in the software development life cycle in order to reduce the amount of defects:

- Defect prevention takes aim to reduce the number of defects introduce while producing the software in the software development life cycle. This is done directly in any software engineering activity.
- Defect removal is to detect defects by software verification or software inspection. The main goal is to eliminate introduced defects. Strategies to achieve this may be dynamic analysis, or formal inspections of code.
- Defect tolerance is to provide continuous software service which satisfies given requirements despite a defect having occurred in the software.
- Defect forecasting is to estimate where new defects are likely to emerge in the software.

In 2002 IEEE Metric Panel, a group of noted researchers have agreed that fixing defects in a software product after being delivered to the customer is up to 100 times more expensive than finding and fixing them during the requirements and design phases [36 - 37]. They have also argued that up to 50 % of effort is spent on avoidable work, 80 % of which comes from a small number of defects (*i.e.*, 20%) in the system. The bottom-line is that software testing is a costly challenge and practitioners seek the knowledge of where the defects might exist before they start testing. In this respect, defect predictors are data mining applications to help prioritising the list of software modules to be tested, to allocate limited testing resources effectively and to detect as many defects as possible with minimum effort.

Software defect prediction has been a popular area of software quality research that has drawn the attention of significant organisations including, but not limited to, Microsoft, NASA and AT&T [38 - 40]. Basically, software defect prediction models require a set of features to characterize the problem and to give estimation on the defect proneness of the system. In software quality, these attributes are referred to as software metrics and numerous previous studies demonstrated defect predictors learned from product [38] (*e.g.*, size, complexity) and process [40 - 41] (*e.g.*, code churn) metrics. Once relevant data are available, a variety of data mining algorithms can be applied to learn defect predictors, please see Hall *et al.*, for a systematic literature review [42].

Most defect prediction studies formulate the problem as a supervised learning problem, where the outcomes of a defect predictor model depend on historical data used for training. They can be either labels indicating that a software module is or is not likely to contain defects, or the predictive number of defects expected to be present in the software module, or a ranking of software modules according to their defect proneness. Majority of research focus on the algorithmic models and report simulation results of defect predictors that are trained on a project and tested on a reserved portion of the same project, *i.e.*, retrospective analyses, or the application of defect predictors to the newer versions of the same project in terms of longitudinal case studies [43,41]. These attempts for defect prediction modelling assume the availability of local project data (*i.e.*, within project predictors). In other words, building data mining models requires a project to have a historical data repository, where project metrics and defect information from past are stored. However, this is rarely the case in reality.

To address this issue, recently a branch of defect prediction research emerged that makes use of transfer learning and deals with cross-project predictors, where the goal is to learn a predictor model from a project and then to apply the model to another project [44]. Cross-project defect prediction is a challenge with important practical aspects. One such practical aspect is that cross-project predictions may enable practitioners to use the

available open-source project data for defect prediction [45], without making big changes in or investments to their existing processes for data collection, and process improvement activities. Existing studies provide empirical evidence over a wide range of software systems, advocating that cross-project defect predictors can be effective. Considering that the idea behind cross-project prediction is to make estimates of faulty locations in projects with no history, it is a viable stop-gap choice [46] in data starving project environments.

Data Collection

Software engineering experts also play an important role in creating and providing data that can be used by data mining approaches. They have useful knowledge regarding data quality, which can be provided for data mining experts to decide how to best process the data before applying data mining approaches. As explained by Bener *et al.*, [47], data mining professionals can decide whether or not to include certain parts of the data in the training set based on software engineering experts' knowledge. Given that poor data quality is likely to result in poor predictive models, software engineering experts may also have the key knowledge to identify the reasons for possibly poorly performing predictive models.

7. Conclusion

In this paper, the state of the art methodologies to deal with software defect prediction in the context of class imbalance problem has been reviewed. In recent years, several methodologies integrating solutions to enhance the induced classifiers in the presence of class imbalance by the usage of evolutionary techniques have been presented.

Acknowledgments

We would like to thank our anonymous reviewers for their insightful detailed comments and suggestions on the paper, as these comments led us to an improvement of the work.

References

[1] J. Hu, J. Deng and M. Sui, "A New Approach for Decision Tree Based on Principal Component Analysis", Proceedings of Conference on Computational Intelligence and Software Engineering, **(2009)**, pp. 1-4.
[2] H. Zhao and A. P. Sinha, "An Efficient Algorithm for Generating Generalized Decision Forests", IEEE Transactions on Systems, Man, and Cybernetics -Part A: Systems and Humans, vol. 35, no. 5, **(2005)** September, pp. 287-299.
[3] M. Mitchell, "Machine Learning", McGraw Hill, New York, **(1997)**.
[4] 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, **(2007)**, pp. 2-13.
[5] A. A. Shanab, T. M. Khoshgoftaar, R. Wald and J. V. Hulse, "Comparison of approaches to alleviate problems with high-dimensional and class imbalanced data", In IEEE International Conference on Information Reuse and Integration (IRI), pp. 234-239.
[6] K. Gao, T. Khoshgoftaar and A. Napolitano, "A hybrid approach to coping with high dimensionality and class imbalance for software defect prediction", In 11th International Conference on Machine Learning and Applications (ICMLA), **(2012)**, pp. 281-288.
[7] L. Pelayo and S. Dick, "Evaluating stratification alternatives to improve software defect prediction", IEEE Transactions on Reliability, vol. 61, no. 2, **(2012)**, pp. 516-525.
[8] R. Shatnawi, "Improving software fault-prediction for imbalanced data", In International Conference on Innovations in Information Technology (IIT), **(2012)**, pp. 54-59.
[9] J. Zheng, "Cost-sensitive boosting neural networks for software defect prediction", Expert Systems with Applications, vol. 37, no. 6, **(2010)**, pp. 4537-4543.
[10] T. M. Khoshgoftaar, E. Geleyn, L. Nguyen and L. Bullard, "Cost-sensitive boosting in software quality modelling", In Proceedings of 7th IEEE International Symposium on High Assurance Systems Engineering, pp. 51-60.
[11] D. Hand, H. Mannila and P. Smyth, "Principles of Data Mining", MIT Press, **(2001)** August.
[12] J. Han and M. Kamber, "Data Mining: Concepts and Techniques", Morgan Kaufmann, **(2000)** April.

[13] L. Breiman, J. Friedman, R. Olshen and C. Stone, "Classification and Regression Trees", Belmont, CA: Wadsworth, **(1984)**.
[14] J. Quinlan, "C4.5 Programs for Machine Learning", San Mateo, CA: Morgan Kaufmann, **(1993)**.
[15] J. Quinlan, "Induction of decision trees", Machine Learning, vol. 1, **(1986)**, pp. 81C106.
[16] L. Breiman, "Random Forests", Machine Learning, vol. 45, no. 1, **(2001)**, pp. 5-32.
[17] S. Wang and X. Yao, "Multiclass ImbalanceProblems: Analysis and Potential Solutions", IEEE Transactions On Systems, Man, And Cybernetics -Part B: Cybernetics, vol. 42, no. 4, **(2012)** August.
[18] C. Seiffert, T. M. Khoshgoftaar, J. Van Hulse and A. Napolitano, "RUSBoost: A Hybrid Approach to Alleviating Class Imbalance", IEEE Transactions On Systems, Man, And Cybernetics -Part A: Systems and Humans, vol. 40, no. 1, **(2010)** January.
[19] R. Batuwita and V. Palade, "Fuzzy Support Vector Machines for Class Imbalance Learning", IEEE Transactions on Fuzzy Systems, vol. 18, no. 3, **(2010)** June.
[20] L. Zhu, S. Pang, G. Chen and A. Sarrafzadeh, "Class Imbalance Robust Incremental LPSVM for Data Streams Learning", WCCI 2012 IEEE World Congress on Computational Intelligence, Australia, **(2012)** June 10-15.
[21] B. X. Wang and N. Japkowicz, "BoostingSupport Vector Machines for Imbalanced Data Sets", Proceedings of the 20th International Conference on Machine Learning-2009.
[22] Y. Song and X. Wang, "Research on Application of Software Defect Analysis based on PCA", 3rd International Conference on Advanced Computer Theory and Engineering (ICACTE), **(2010)**.
[23] M. M. Yazid, S. H. Olsen and G. J. Atkinson, "MFM Study of a Sintered Nd–Fe–B Magnet: Analyzing Domain Structure and Measuring Defect Size in 3-D View", IEEE Transactions on Magnetics, vol. 52, no. 6, **(2016)** June.
[24] Y. Xiaobo, W. Yichen, L. Bin and L. Jianxing, "A Fault Propagation Model for Embedded Software system", IEEE, **(2016)**.
[25] C. Tantithamthavorn, S. McIntosh, A. E. Hassan and K. Matsumoto, "An Empirical Comparison of Model Validation Techniques for Defect Prediction Models", IEEE Transactions On Software Engineering.
[26] F. Zhang, A. E. Hassan, S. McIntosh and Y. Zou, "The Use of Summation to Aggregate Software Metrics Hinders the Performance of Defect Prediction Models", IEEE Transactions On Software Engineering, vol. 43, no. 5, **(2016)**, pp. 476-491.
[27] P. Rempel and P. Mader, "Preventing Defects: The Impact of Requirements Traceability Completeness on Software Quality", DOI 10.1109/TSE.2016.2622264, IEEE Transactions on Software Engineering.
[28] R. A. Paul, F. Bastani, L. Yen and V. U. B. Challagulla, "Defect-Based Reliability Analysis for Mission-Critical Software", **(2000)**, IEEE.
[29] R. Subramanyam and M. S. Krishnan, "Empirical Analysis of CK Metricsfor Object-Oriented Design Complexity: Implications for Software Defects", IEEE Transactions On Software Engineering, vol. 29, no. 4, **(2003)** April.
[30] M. D. Ambros, "Supporting Software Evolution Analysis with Historical Dependencies and Defect Information", IEEE, **(2008)**.
[31] Z. Li, M. Gittens S. Shariyar Murtaza, N. H. Madhavji, A. V. Miranskyy, D. Godwin and E. Cialini, "Analysis of Pervasive Multiple-Component Defects in a Large Software System", IEEE, **(2009)**.
[32] C. Qixiang, M. Minsheng and Z. Qian, "The analysis and research on defect results of software localization testing", IEEE, **(2010)**.
[33] M. Perez-Ortiz, P. Antonio Gutierrez, C. Herv_as-Martınez and X. Yao, "Graph-Based Approaches for Over-Sampling in the Context of Ordinal Regression", IEEE Transactions On Knowledge and Data Engineering, vol. 27, no. 5, **(2015)**, May.
[34] X.-Y. 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, **(2016)**.
[35] L. L. Minku, E. Mendes and B. Turhan, "Data mining for software engineering and humans in the loop", Prog ArtifIntell, vol. 5, **(2016)**, pp. 307-314, DOI 10.1007/s13748-016-0092-2.
[36] B. W. Boehm and V. R. Basili, "Software defect reduction top 10 list", IEEE Comput., vol. 34, no. 1, **(2001)**, pp. 135-137.
[37] F. Shull, V. Basili, B. Boehm, A. W. Brown, P. Costa, M. Lindvall, D. Port, I. Rus, R. Tesoriero and M. Zelkowitz, "What we have learned about fighting defects", In: VIII International Symposium on Software Metrics, IEEE Computer Society, Washigton, DC, **(2002)**, pp. 249-258, doi:10.1109/METRIC.2002.1011343.
[38] T. Menzies, J. Greenwald and A. Frank, "Data mining static codeattributes to learn defect predictors", IEEE Trans. Softw. Eng., vol. 33, no. 1, **(2007)**, pp. 2-13.
[39] N. Nagappan and T. Ball, "Use of relative code churn measures topredict system defect density", Proceedings of the International Conference on Software Engineering, **(2005)**, pp. 284–292, doi:10.1145/1062455.1062514.
[40] T. J. HOstrand, E. J. Weyuker and R. M. Bell, "Automating algorithms for the identification of fault-prone files", D.S. Rosenblum, S.G. Elbaum (eds.) Proceedings of the International Symposium on Software Testing and Analysis, ACM, **(2007)**, pp. 219-227.

[41] E. J. Weyuker, T. J. Ostrand and R. M. Bell, "Do too many cooks spoil the broth? using the number of developers to enhance defect prediction models", Empirical Softw. Eng., vol. 13, no. 5, **(2008)**, pp. 539-559.
[42] T. Hall, S. Beecham, D. Bowes, D. Gray and S. Counsell, "Asystematicliterature review on fault prediction performance in softwareengineering", IEEE Trans. Softw. Eng., vol. 38, no. 6, **(2012)**, pp. 1276-1304.
[43] A. Tosun, A. B. Bener, B. Turhan and T. Menzies, "Practical considerations in deploying statistical methods for defect prediction: a case study within the turkish telecommunications industry", Inform. Softw. Technol., vol. 52, no. 11, **(2010)**, pp. 1242-1257.
[44] L. C. Briand, W. L. Melo and J. Wst, "Assessing the applicability of fault-proneness models across object-oriented software projects", IEEE Trans. Softw. Eng., vol. 28, no. 7, **(2002)**, pp. 706-720.
[45] T. Zimmermann, N. Nagappan, H. C. Gall, E. Giger and B. Murphy, "Cross-project defect prediction: a large scale experiment on data vs. domain vs. process", In: van Vliet, H., Issarny, V. (eds.) Proceedings of the the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, ACM, **(2009)**, pp. 91-100.
[46] B. Turhan, T. Menzies, A. Bener and J. Di Stefano, "On the relative value of cross-company and within-company data for defect prediction", Empirical Softw. Eng., vol. 14, no. 5, **(2009)**, pp. 540-578.
[47] A. Bener, A. Misirli, B. Caglayan, E. Kocaguneli and G. Calikli, "The Art and Science of Analyzing Software Data: Analysis Patterns, chap", Morgan Kaufmann, Lessons Learned For Software Analytics in Practice, **(2015)**.

Authors

K. Nitalaksheswara Rao is a Ph.D candidate in Computer Science and Systems Engineering at Andhra University. He received his Master's degree in Computer Science and engineering in 2009. Now, he is an Assistant Professor in department of Computer Science and engineering, Narasaraopet engineering college, Guntur. His current research interest includes Software Engineering, Data Engineering and Quality Assurance.

Dr. Ch. Satyananda Reddy has a Ph.D in Computer Science Engineering from the Faculty of Engineering, Andhra University, INDIA and working now as Associate Professor in the Department of Computer Science and Systems Engineering, College of Engineering, Andhra University, Visakhapatnam.

Dr. Reddy is mainly interested in the fields of Software Engineering i.e. Software Engineering, Software Project and Process Management, Software Estimation, Software Metrics, Software Project Scheduling, Software Quality Assurance, Human Computer Interaction, Software Requirements Engineering, Software Architecture, Software Testing, Data Engineering, Documents Summarization etc. He is a reviewer for several International Journals and Program Committee member for several International Conferences. He is a Member of IAENG, IDES, ISTE, ACEEE. He had published 40 papers in the field of Computer Science and Software Engineering.

For full works visit: http://www.andhrauniversity.edu.in/engg/csse/faculty.html

