PREFACE

Software testing is an essential phase of the software development life cycle (SDLC); it requires approximately 40% of resources in software development. Software defect causes a discrepancy in actual output and expected output, which leads to system failure. A software defect is a condition in a software product that does not meet the software requirement (as stated in the requirement specifications) or end-user expectation (which may not be specified but is reasonable). Software Defect Prediction (SDP) is a method to predict the fault-prone module of a software system. After detecting the fault-prone module, the project leader can optimally allocate testing resources; more tester team members to more fault-prone software modules. It reduces testing effort and leads to a decrease in the cost of software development. Many software practitioners have proposed several SDP methodologies using statistical techniques, machine learning methods, or deep learning architecture. There are many hindrances for detecting faulty modules in software fault prediction systems, such as missing values or samples, data redundancy, irrelevance features, and correlation. Many researchers have built various Software Bug Prediction (SBP) models, which classifies buggy and clean modules associated with software metrics. There are several challenges in the SDP domain, such as skew distribution in the public datasets results in a class imbalance problem, overfitting, quality of a public dataset. This thesis's main objective is to reveal the favorable result by feature selection, machine learning, or deep learning methods to detect defective and non-defective software modules, estimating the number of a module in the target project, and predicting data associated with the next version of a software project. Along with addressing class imbalance and overfitting problems in the SDP domain and produces unbiased results. Further, normally predicting based on one or more similar projects may start showing deficient prediction for newer projects; to counter this possibility of learning from across multiple projects may be useful. In this thesis, we also proposed a cross-project defect prediction method. Additionally, an attempt has been made to include the prediction of certain software metrics apart from classical software bug prediction. The proposed work in the thesis is classified into four sections first that are based on a classification-based approach to identify the faulty or clean module. Then we will discuss how to identify the number of defects in every module in the upcoming version of a software project; it is one step ahead of just identifying a module is faulty or not. Further, the cross-project defect

number prediction and prediction of the entire software project are discussed. These approaches help in optimally allocating development and testing resources and leads to produce high-quality software products.
First, we investigate various existing classification-based defect prediction methods and propose a rudimentary classification-based framework, Bug Prediction using Deep representation and Ensemble learning Technique (BPDET) for SBP. It combinedly applies by ensemble learning and deep representation. The software metrics which are used for SBP are mostly conventional. Staked denoising auto-encoder (SDA) is used for the deep representation of software metrics, which is a robust feature learning method. The proposed model is mainly divided into two stages: the deep learning stage and two layers of the EL stage (TEL). The extraction of features from SDA in the very first step of the model then applied TEL in the second stage. TEL is also dealing with the class imbalance problem. The experiment mainly performed NASA (12) datasets to reveal the efficiency of DR, SDA, and TEL. The performance is analyzed in terms of Mathew co-relation coefficient (MCC), the area under the curve (AUC), precision-recall area (PRC), F-measure, and time. Out of 12 dataset MCC values over 11 datasets, ROC values over 6 datasets, PRC values overall 12 datasets, and F-measure over 8 datasets surpass the existing state-of-the-art bug prediction methods. We have tested BPDET using Wilcoxon rank-sum test, which rejects the null hypothesis at $\alpha = 0.025$. We have also tested the stability of the model over 5, 8, 10, 12, and 15 fold cross-validation and got similar results. Finally, we conclude that BPDET is stable and outperformed on most of the datasets compared with existing EL-based methods and other benchmark techniques.

In the next work, we investigate various existing and baseline works over software number fault prediction. Predicting the number of bugs in a software system intensifies the software quality and turns down the testing effort required in software development. It reduces the overall cost of software development. The evolution of hardware, platform, and user requirements leads to develop the next version of a software system. In this chapter, we formulate a problem statement and its novel solution, i.e., we are considering the prediction of the bug count vector of a successive version of a software system. After predicting the bug count vector in the next version of a software, the developer team leader can adequately allocate the developers in respective fault dense modules; in a more faulty dense module, more developers are required. We have conducted our experiment over seven PROMISE repository datasets of different versions. We build metadata using a concatenation of various versions of the same software system for conducting experiments. We proposed a novel architecture using deep learning called BCV-Predictor. BCV-Predictor predicts the bug count vector of the next version software system; it is trained using metadata (collection of different dataset). To the best of our knowledge, no such work has been done in these aspects. We also address overfitting and class imbalance problems using the random oversampling method and dropout regularization technique. We conclude that BCV-Predictor is conducive to predicting the bug count vector of the next version of the software project. We found five out of seven meta datasets reach more than 80% accuracy. In all seven meta datasets, Mean Squared Error (MSE) lies from 0.71 to 4.715, Mean Absolute Error (MAE) lies from 0.22 to 1.679, MSE and MAE over validation set lie between 0.84 to 4.865, and 0.22 to 1.709 respectively. We also compared the performance of the BCV-Predictor with eleven baseline techniques and found the proposed approach statistically outperforms on most of the dataset.

In the next work, we revisited the software fault number estimation and along with existing work of cross-project number prediction. Both techniques help in allocating resources before the testing phase more optimally. Due to a lack of an adequate dataset, defects can be predicted by employing data from different projects to train the classifier called cross-project defect prediction (CPDP). Cross-project defect number prediction (CPDNP) is one step ahead of CPDP, in which we can also estimate the number of defects in each module of a software system; we contemplate it as a regression problem. This work dealt with the CPDNP mechanism and suggested a CPDNP architecture by employing a deep neural network and attention layer called DNNAttention. We synthesis substantial data named cross-heap by utilizing an amalgamation of 44 projects from the PROMISE data repository. We feed the cross-heap into DNNAttention to train and evaluate the performance over 44 datasets by applying transfer learning. We have also address class imbalance (CI) and overfitting problems by employing multi-label random oversampling and dropout regularization, respectively. We compared the performance of DNNAttention using MSE, MAE, and accuracy over eight baseline methods. We found out of 44 projects, 19 and 20 have minimum MSE and MAE, respectively, and in 19 projects, accuracy yields by the proposed model surpasses exiting techniques. Moreover, we found the improvement of DNNAttention over other baseline methods in terms of MAE, MSE, and accuracy by inspecting 20% lines of code is substantial. In most situations, the improvements are significant, and it has a large effect size across all 44 projects.

The overall cost of software development is detrimental to determining the utility of any software. However, the uncertain nature of this cost increases the risk associated with the software. Hybrid regression analysis investigates the various regression-based approaches and estimates all software features, including a number of bugs in every module of the software system. Predicting the next version of the software system helps reduce this cost, allowing a better allocation of developers and testers for software development. In this chapter, we develop a novel approach for the prediction of the next version of a software system. Our method predicts the bug count and the metric values for each module in the software using a computational framework consisting of two phases, the data augmentation phase and the next version prediction phase. We perform our experimentation on 8 public datasets from the PROMISE repository using all existing versions. To achieve unbiased results, we conducted each experiment 50 times and took a mean value of it. The proposed methodology has an accuracy of over 60% on 5 out of eight datasets, while MSE and MAE lie between 45.34% to 185.5% and 30.59% to 131.21%, respectively. We have also compared the proposed model's performance with eight baseline learning methods and found the proposed model significantly outperforms these techniques.