Chapter 8

Conclusion and Future Direction

"A conclusion is simply the place where you got tired of thinking."- Dan Chaon

The list of main contributions of this thesis are summarized below, followed by possible directions for future work.

8.1 Conclusion

Software defect prediction is an emerging and fast-growing field of research. However, we also considered possible aspects of many defect prediction directions. Several works have been done in classification-based defect prediction methods. We found that software features extracted from deep neural networks (deep features) can be more useful in defect prediction and demonstrate better defect prediction approaches; some initial work has been done in this direction. This thesis is devoted to developing a better **classification based SDP approach**, **cross-version defect count vector prediction**, **cross-project defect number prediction** and **hybrid regression analysis**. We have investigated and studied three main tasks: software defect prediction, software defect count estimation (cross-version, cross-project), and hybrid regression analysis (entire software prediction). The scope of this thesis is limited to the following defect prediction issues: class imbalance, overfitting, cost-effectiveness, the stability of a model, and training cost. We have tried to address these challenges using various techniques. The list of the conclusion of the thesis is given below:

(i) SDP plays an important role in ensuring high-quality software products with reliability. We proposed a novel SDP model (BPDET) by utilizing deep learning to extract deep features and heterogeneous ensemble learning techniques. The proposed architecture has two stages, i.e., deep learning and ensemble learning phases. Stacked denoising autoencoder employed in deep learning phase to extract a deep representation of features over traditional matrices. We employed 12 different NASA projects for the experiments. We also found that deep learning and ensemble learning phases meaningfully address the class imbalance and overfitting problems. We also found that performance metrics (MCC, ROC, PRC, and F-Measure) on most of the datasets produced by BPDET model are significantly higher than existing baseline techniques. The stability of the model over k-fold cross-validation to measure the stability is also tested, and found the proposed model is adequately stable till the 15-fold cross-validation. Based on earlier understanding and our observation, it can safely be concluded that deep learning is helpful to extract a deep representation of software metrics and also able to avoid class imbalance and overfitting problems. Deep representation combined with ensemble learning builds a better SBP model concerning baseline and traditional techniques.

(ii) Estimating the number of bugs in every module of a software system and predicting the bug count vector of the upcoming version of a software project helps the software project manager to suitably allocate developers and testers in the software development process. Cross-version defect count vector prediction is a possible way to estimate the bug count vector for the upcoming version of a software project. We amalgamate different versions of the same software to create a more extensive dataset and proposed LSTM based architecture to estimate the bug count vector for the successive version of a software project the technique called BCV-Predictor. The different versions of the software are used as time steps in the architecture. Results concluded that deep learning-based architecture is effective over existing machine learning-based approaches and also found that dropout regularization and multi-label random oversampling are more suitable to conquer overfitting and class imbalance problems in such aspects. Out of seven PROMISE datasets, we found that five meta-datasets have more than 80% accuracy, and six & five out of seven projects have MAE less than 0.9 and MSE less than 3, respectively. We also discovered that BCV-Predictor significantly outperforms baseline methods in terms of MAE, MSE,

and accuracy. We conclude that BCV-Predictor is more efficient for an extensive software system; it takes lesser computational cost than baseline methods and gives better results.

(iii) Cross-project defect number estimation is the process of estimating the number of defects in every module of a software system by training the model using a diverse, distributed set of projects. It is one step ahead of cross-project defect prediction; it is a combination of two SDP domains, i.e., cross-project defect prediction and software fault number prediction. The proposed architecture includes the unique amalgamation of various software projects using 44 different PROMISE repository projects. It employed LSTM and attention layer in the learning algorithm and named it DNNAttention. We train the model using the amalgamated dataset and test over the target project using transfer learning. After comparing existing baseline methods, we found that the proposed model significantly outperforms bigger and moderate-sized projects. Random oversampling and dropout regularization are applied in the preprocessing step, results conclude that DNNAttention overcomes class imbalance and overfitting problems. The results indicate that the proposed model can more precisely generalize software properties over new projects. We found the performance of DNNAttention is significantly better than existing baseline methods over most of the project. We found that the proposed architecture is more suitable for large and moderate-sized projects, whereas it gets a high loss for small-sized projects. Moreover, we also discovered the training time of the DNNAttention is high compared with baseline methods because of various matrix operations at different layers. Due to a lack of adequate training data, DNNAttention underperformed over small-sized projects. While inspecting 20% of the lines of code in each target project, we found significant improvement; it has large effect sizes across all 44 datasets.

(iv) Cross-project or cross-version bug prediction only predicts or estimates the bugs in every module. But in the entire version prediction method, we estimate the number of defects and the metrics values of every module in the upcoming version of a software system. The proposed approach employed two different deep learning architectures and utilized eight public PROMISE datasets in experiments. We found the proposed approach obtained more than 60% accuracy over five projects. The results obtained by the proposed model are significantly

better than other deep learning or machine learning regression methods. We found this method can enormous reduces the software development cost.

8.2 Future Direction

This section presents the possible directions for exploration based on the consequent work done. The following are the future directions that are identifiable.

BPDET: Classification Based Software Defect Prediction Model

Deep learning can also be applied in the ensemble learning phase, leading to a better result and addressing the class imbalance problem. Other suitable optimization techniques, vectorization, and broadcasting method can be applied to reduce training costs. A possible application for different deep learning methods may also be intuitive and exciting. The BPDET model's architecture can be used for a number of fault count problems in software reliability.

BCV-Predictor: Cross-Version Defect Count Vector Predictor

In the future, the possible extensions of the research work are possible. First, the applicability of a similar model in cross-project bug count prediction in the version of a software project. It can be effective in a cross-project defect prediction scenario. Second, adding an attention layer along with LSTM to enhance the performance. Third, it can be helpful to estimate the testing effort for a module by identifying the size of a bug. Integrating testing methodologies over design metrics. Fourth, similar experiments can be conducted on other open-source software systems with substantial data instances and several versions. Fifth, hyperparameters tuning can lead to much better results. Sixth, use of transfer learning over design metrics to predict a number of bugs in the successive version of a software system. Organizations should avail their datasets for research purposes to make a much better prediction model. To achieve more robust results, different sequence models with optimal hyperparameter settings can also be applied to this problem.

DNNAttention: Cross-Project Defect Number Prediction

The following possibilities are identifiable to extend our work. First, DNNAttenion can be applied over other domain software projects and adding more data instances in the cross-heap. We are also planning to use the DNNAttention in software defect number prediction and software cost estimation. Hyperparameters tuning is always a scope of improvement in the deep learning model, and we are planning to re-tune

the model for more optimal results. N-gram and skip-gram techniques can be added to the proposed model; it can enhance the performance of the DNNAttention model.

Predicting the next version of the Software Systems: A Hybrid Regression Analysis

A possible future direction could be involving more extensive training data and combining software datasets from multiple industries. We also extend the experiments to improved accuracy in the data augmentation phase. To achieve better results, we can employ transfer learning and other complex deep learning architectures in the next version prediction phase. The proposed model can be used to generate sequence data by employing existing sequences. Hyperparameter tuning always provides the scope of better results.