Software Defect Prediction Using Deep Learning

Achmad Iqbal Al Faizin[1, a)], Edy Suharto[1, b)], Aris Puji Widodo[1, c)], Hendinur Faizal[1, d)], Muhammad Naufal Pratama[1, e)], Raihan Mufadhal[1, f)], and Rafli Azra Virendra Azhari[1, g)]

Author Affiliations
[1]*Department of Computer Science/Informatics, Diponegoro University, Semarang, Indonesia*

Author Emails
[a)] *iqbalrouzle@students.undip.ac.id*
[b)] *edys@lecturer.undip.ac.id*
[c)] *arispw@gmail.com*
[d)] *hendinurfaizal@students.undip.ac.id*
[e)] *muhammadnaufalp@students.undip.ac.id*
[f)] *rmfdl@students.undip.ac.id*
[g)] *rafliazra@students.undip.ac.id*

Abstract. In the process of software development, a defect is often present but hard to detect if the software in development is complex. This could be a problem for customers or the end user. To increase the reliability of the software a plethora of methods can be used to detect these defects, one of them is using deep learning. The use of deep learning can help us to extract features automatically using a neural network instead of extracting them manually. The dataset used for this paper is taken from the public PROMISE repository. The result of this research will be gauged against preceding methods on its effectiveness.

KEYWORDS

Software defects, machine learning, deep learning.

INTRODUCTION

Software defect is a condition where a software product is not up to the standards of softwares or when the software is not working as expected of the end user. In other words, a defect is an error in the coding or logic that causes the program to not function properly or have outcomes that are wrong or unexpected.

Software defect is also defined as an error happening to the software because of a faulty code, documentation, or to the design that causes a failure in the software's performance. By making predictions on software defects, the expected outcome is to decrease the chances of a software to experience failure mostly during production so that it will be easier when testing occurs.

To detect the defects in the softwares, we are using two approaches, first using machine learning and secondly using deep learning algorithms. Using these methods, we will compare which model from the two approaches that will hold the better outcome. The dataset that will be used in this research are from the PROMISE public dataset repository which is comprised of datas about softwares that have defects, also attributes that has been determined based on McCabe's Metrics, which includes essential complexity, cyclomatic complexity, design complexity, and LOC (lines of code). We are also using Halstead's metrics, which also includes base measures, derived measures, and LOC measures.

To detect the defects in the softwares, we are using two approaches, first using machine learning and secondly using deep learning algorithms. Using these methods, we will compare which model from the two approaches that will hold the better outcome. The dataset that will be used in this research are from the PROMISE public dataset repository which is comprised of datas about softwares that have defects, also attributes that has been determined based on McCabe's Metrics, which includes essential complexity, cyclomatic complexity, design complexity, and LOC (lines of code). We are also using Halstead's metrics, which also includes base measures, derived measures, and LOC measures.

APPROACH

The approach that is used for software defect detection in this research has an emphasis on the preprocessing of the dataset given so that it will hold the best result. The dataset used is from the PROMISE public dataset repository which is composed of datas of defect softwares and those that are not, also with the defect parameters. The datasets itself are of different programming languages, like C/C++. The preprocessing will be done with various methods like oversampling and undersampling before going to the next step, that is the classification with a model. The classification models that will be used are the random forest model, CNN, and LSTM. Following that, the three models used will be compared to find out which model will hold the better result. To make it easier for the process of comparing the models, performance metrics are used as the parameter. As an outline, our approach are divided into a couple of steps: 1) preprocessing using the available methods, 2) Classification using machine learning and deep learning models to determine a defective software, 3) evaluating the performance using performance metrics so that it can be used to compare which model is better at detecting defects in a software.

PREPROCESSING

In this section, we will describe how to preprocess the PROMISE dataset especially when dealing with imbalance data. However, before going on about it, we will explain the structure of the dataset itself.

Datasets

As previously mentioned, this study will use the PROMISE repository as the dataset. From 15 datasets, there are about 9 datasets that give better results than other datasets. From the 9 datasets, elimination is taken for datasets with less than 1000 rows of data and taking the datasets with the best accuracy results when run with the random forest model using oversampling. The result is 3 datasets, they are *mc1*, *pc2*, and *pc4*, and we also combine several datasets into one dataset called *'combined'*. Table 1 describes the structure of the four selected datasets.

TABLE 1

Dataset	Total Instances	Defective Instances	Non-Defective Instances
mc1	9466	68	9398
pc2	5589	23	5566
pc4	1458	178	1280
combined	18237	481	17756

It can be seen that the amount of defective software is much less than the defective one. For this reason, it is necessary to handle this imbalance data using the techniques that we will describe in the next part. None of the existing datasets have missing values. Even so, most of the existing features have a small (close to 0) correlation value that does not significantly take a role in the defect detection process. Therefore, we perform automatic feature extraction by removing features with correlation values in range $-0{,}1 < corr < 0.1$. Keep in mind that not all features will be used later. Only features that have a value greater than 0.1 or lower than -0.1 will be used.

Sampling Method

The first option to deal with the imbalanced datasets is to use an oversampling method. Oversampling means duplicating data from the minority class [3] until the number is the same as the majority class or in this case duplication is carried out on the defects class.

The second way is to use an undersampling method. Undersampling is the opposite of oversampling. Instead of duplicating the few data classes, the large data classes are reduced in number so that they are equal to the number of minority data classes. However, this method has the potential to mess up the information from existing data [3].

After handling the imbalanced dataset, the next step is splitting the dataset to 50:30:20, each for train data, test data, and validation data, respectively, before being included in the models for classification.

MODEL BUILDING & TRAINING

In this section we describe the details of the models we use for classifying defects in software, which consists of random forest as the machine learning model and also CNN and LSTM as the deep learning models.

Convolutional Neural Network (CNN)

In a CNN model, there are 6 layers consisting of an input layer, a 2D convolutional layer with 64 nodes, a 2D convolutional layer with 32 nodes, one flatten layer, one dense layer consisting of 16 nodes, and the last layer which is the output layer. The input shapes used in this CNN model are the number of rows, the number of columns, and the channel for each row of data in the dataset. The use of CNN which only has 2 layers is considered sufficient for the existing dataset, and the node configuration has been adjusted to the existing dataset. The flatten layer between the convolutional layer and the density layer used to make the input from 2d vector to n-d vector.

The *ReLu* activation function is used in every layer except for the flatten and output layers because of their better and faster performance for this kind of classification. On the output layer the *sigmoid* is chosen as the activation function due to the fact that sigmoid is able to be used for classification of two classes or binary classification. In the compilation process, the Adam optimizer is used because of its ease of use (not much tuning needed). The number of epochs are determined from the result of loss and validation of each dataset, which is calculated using binary cross-entropy.

When training the model, the number of epochs used are determined by the *loss* and *val_loss* that occurred during training, if an indication of overfitting is shown, then the epoch will be determined before that happens. The CNN model training will be done in 300 epochs.

Long Short-Term Memory (LSTM)

The LSTM model has the following structure, an input layer, three hidden layers consisting of 64, 32, and 32 dimensions of output, also an output layer. Each LSTM layer has a dropout value of 60%-65% to prevent overfitting of a model. The LSTM Input layer has a number of timesteps followed by the number of features in the datasets. All return sequences of the LSTM layers are activated, except for the last hidden layer which is false, so that there is only one output from the last LSTM layer. Finally, there is an output layer that uses *sigmoid*, whose value ranges from 0 to 1, as its activation function. The use of *sigmoid* is intended because the expected result of this model is binary classification or only two classes, defect or not defect. The optimizer and loss function used are still the same as in the CNN model, which are Adam and binary cross-entropy. This training model is carried out with 300 epochs.

Random Forest

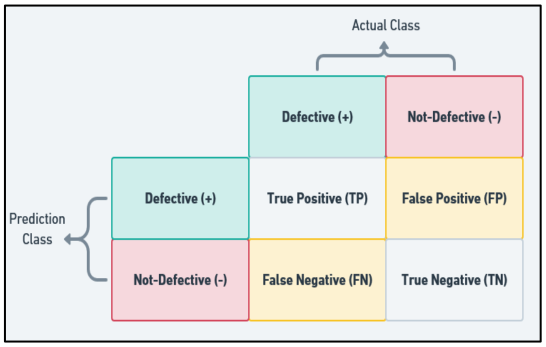
For the machine learning model, we use a Random forest classifier with a max depth of 4 trees. The tree splitting algorithm used is the default from Scikit-Learn [1], which is Gini impurity. The use of Gini is also more common than other tree node splitting techniques, such as information gain. The training is conducted with those four datasets and two different sampling methods.

EVALUATION

This section will explain the results of the three models after the training and testing using the four datasets. In addition, we also provide pictures to help explain the results of the sampling method used.

Performance Measures

Performance measures used to measure the performance of a prediction model across the two classes. Our evaluation is focused on detecting the defective class. A confusion matrix is used to evaluate the decisions made by a prediction or classifier model. For instance, if a file is classified as defective and it is truly defective, the classification is a true positive (tp). If the file is classified as defective when it is actually clean, then the classification is a false positive (fp). If the file is classified as clean but it is in fact defective, then the classification is a false negative (fn). Finally, if the file is classified as clean and it is in fact clean, then the classification is true negative (tn). The values



given by the confusion matrix are used to compute the widely-used Accuracy, Precision, and Recall.

FIGURE 1. Confusion matrix

- Accuracy: The ratio of total files classified correctly. It is calculated as:

$$acc = \frac{TP+TN}{TP+TN+FP+FN} \tag{1}$$

- Precision: The ratio of correctly predicted defective files over the files that were predicted as defective. It is calculated as:

$$pr = \frac{TP}{TP+FP} \tag{2}$$

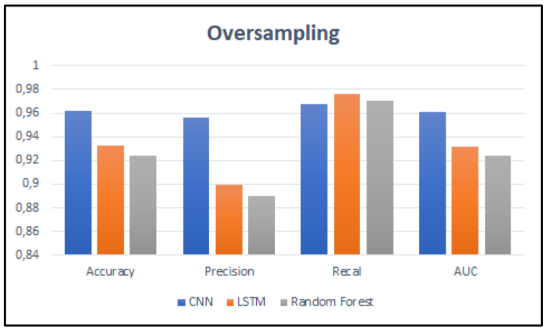
- Recall: The ratio of correctly predicted defective files over the files that were classified correctly. It is calculated as:

$$re = \frac{TP}{TP+FN} \tag{3}$$

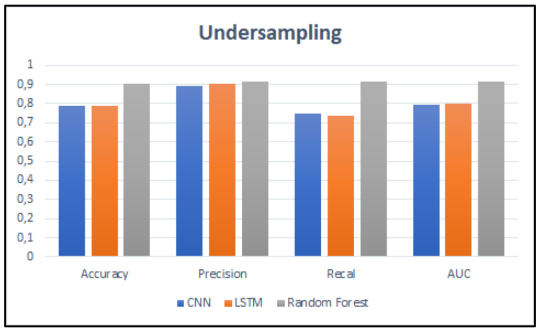
- Area Under the ROC Curve (AUC) is used to evaluate the degree of separability achieved by the model. The value of AUC is ranged from 0 to 1. Model with higher AUC indicates a better performance [2].

Results

Those three models that have been created before will use *mc1*, *pc2*, *pc4*, and combined data as the datasets for the training and testing. As we explained previously, to handle the imbalanced dataset we use two well known methods i.e. oversampling and undersampling. The following figures illustrate the results of the three classification models using oversampling and undersampling.



(a)



(b)

FIGURE 2. Results of training and testing the classifier models. (a) Result when using oversampling method, (b) Result when using undersampling method.

The values of accuracy, precision, recall, and AUC in the above figures are the average results from the four datasets used in training and testing. During the experiment, mc1 became the dataset with the highest accuracy level, around 94%-95% for all three models. However, things should be considered that the average amount of data available when using undersampling is roughly only 180 data. This happens because undersampling tries to reduce the amount of majorities data so that it is equal to the number of the minority data. This number is considered as very small and cannot provide a true picture of a real world situation, where software defects are still common. Therefore, the results we emphasized in this study are the results when using an oversampling method instead of undersampling. Of the four performance metrics used, CNN almost dominates all of them by providing the best results on accuracy, precision, and AUC. Meanwhile on the recall, the LSTM model is slightly better than the other two models.

CONCLUSION

In this study, we attempt to predict software defects using machine learning and deep learning classifier models. The dataset used in this study is a PROMISE dataset with additional preprocessing techniques which includes feature extraction and handling imbalanced datasets using oversampling and undersampling. The classification models used in this study were Random forest, CNN, and LSTM, all of which were compared to see which model gave better results than other models in detecting software defects. The results give us information that in terms of classifying defects on a software, CNN gives best results on accuracy, precision, and AUC performance metrics while LSTM leads on the recall and random forest is under both of them.

ACKNOWLEDGMENTS

The authors gratefully acknowledge support from faculty of science and mathematics Diponegoro University and Dr. Eng. Adi Wibowo, S.Si. and M.Kom Edy Suharto, S.T who have guided us in writing this paper.

REFERENCES

1. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V. and Vanderplas, J., 2011. Scikit-learn: Machine learning in Python. In *Journal of Machine Learning Research*, 12, pp.2825-2830.
2. Dam, H.K., Pham, T., Ng, S.W., Tran, T., Grundy, J., Ghose, A., Kim, T. and Kim, C.J., 2018. A deep tree-based model for software defect prediction. *arXiv preprint arXiv:1802.00921*.
3. Li, J., He, P., Zhu, J. and Lyu, M.R., 2017, July. Software defect prediction via convolutional neural network. In *2017 IEEE International Conference on Software Quality, Reliability and Security (QRS)* (pp. 318-328). IEEE.

4. Sayyad Shirabad, J., and Menzies, T.. (2005). The PROMISE Repository of Software Engineering Databases.
5. Chollet, F., and others. (2015). Keras. https://keras.io.