A Replication Study: Just-In-Time Defect Prediction with Ensemble Learning

Steven Young
Data Science Laboratory
Ryerson University
Toronto, ON, Canada
steven.young@ryerson.ca

Tamer Abdou
Arish University
Ryerson University
Toronto, ON, Canada
tamer.abdou@ryerson.ca

Ayse Bener
Data Science Laboratory
Ryerson University
Toronto, ON, Canada
ayse.bener@ryerson.ca

ABSTRACT

Just-in-time defect prediction, which is also known as change-level defect prediction, can be used to efficiently allocate resources and manage project schedules in the software testing and debugging process. Just-in-time defect prediction can reduce the amount of code to review and simplify the assignment of developers to bug fixes. This paper reports a replicated experiment and an extension comparing the prediction of defect-prone changes using traditional machine learning techniques and ensemble learning. Using datasets from six open source projects, namely Bugzilla, Columba, JDT, Platform, Mozilla, and PostgreSQL we replicate the original approach to verify the results of the original experiment and use them as a basis for comparison for alternatives in the approach. Our results from the replicated experiment are consistent with the original. The original approach uses a combination of data preprocessing and a two-layer ensemble of decision trees. The first layer uses bagging to form multiple random forests. The second layer stacks the forests together with equal weights. Generalizing the approach to allow the use of any arbitrary set of classifiers in the ensemble, optimizing the weights of the classifiers, and allowing additional layers, we apply a new deep ensemble approach, called deep super learner, to test the depth of the original study. The deep super learner achieves statistically significantly better results than the original approach on five of the six projects in predicting defects as measured by F_1 *score*.

CCS CONCEPTS

• **Computing methodologies → Supervised learning by classification**; **Machine learning approaches**; **Ensemble methods**; • **Software and its engineering → Software defect analysis**;

KEYWORDS

Deep Learning, Defect Prediction

ACM Reference Format:
Steven Young, Tamer Abdou, and Ayse Bener. 2018. A Replication Study: Just-In-Time Defect Prediction with Ensemble Learning. In *Proceedings of 6th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE'2018).* ACM, New York, NY, USA, Article 4, 6 pages. https://doi.org/10.475/123_4

RAISE'2018, May 2018, Gothenburg, Sweden

ACM ISBN 123-4567-24-567/08/06…$15.00
https://doi.org/10.475/123_4

1 INTRODUCTION

This paper reports on the replication and extension of empirical research using the proposed guidelines for experimental replications [2] to verify the results of the original study and investigate whether alternatives, such as diversifying the set of classifiers, optimizing the weights of the classifiers when combining them, and additional layers in the ensemble, can enhance performance in predicting defects. The original experiment [5] investigated whether hybrid ensembles of machine learning methods could improve the performance of just-in-time defect prediction. Accurate defect prediction at change-level can assist with ensuring software quality during the development process and assist developers in finding and fixing defects in a timely manner [3]. Following on from the original study, we tested another ensemble called deep super learner [7], and compared the results to the original approach[1].

2 INFORMATION ABOUT THE ORIGINAL STUDY

The research questions from the original study are summarized in Table 1. The proposed methodology employs a two-layer ensemble (TLEL). The inner layer combines decision trees with bagging to build random forests. The outer layer uses random under-sampling of the majority class to train multiple random forests and stacks them together in an equally weighted manner. Pseudo-code of TLEL is provided in Algorithm 1.

```
for iteration in 1 to number of learners do
    Random sub-sample from majority class to balance classes;
    Train random forest classifier;
end
class = 0
for iteration in 1 to number of learners do
    Predict test instance with trained model;
    if predicted class is buggy then
        class ← class +1
    end
end
if class ≥ (number of learners)/2 then
    final predicted class ← buggy
end
```

Algorithm 1: A Pseudo-code of TLEL

Datasets used come from six open source projects: Bugzilla, Columba, JDT, Platform, Mozilla, and PostgreSQL [3]. Combined,

[1] The base algorithm used in replicating the original study has been developed as part of the Capstone Project Course of the Certificate in Data Analytics, Big Data, and Predictive Analytics at Ryerson University.

they contain data on 137,417 changes. The changes are labeled as either buggy or not buggy. However, the classes are unbalanced with the portion of buggy changes ranging from 5% to 36% within the datasets.

Two metrics are used to evaluate performance: cost effectiveness and F_1 *score*. Cost effectiveness is the percentage of buggy changes found when reviewing a specific percentage of the lines of code. The F_1 *score* is commonly-used to evaluate classification performance. F_1 *score* is an aggregate of precision and recall that evaluates if an increase in precision or recall outweighs a reduction in the other. Precision is the ratio of true positive (TP) instances to the number of predicted positive instances, both true positive and false positive (FP), as shown in Equation 1.

$$Precision = \frac{TP}{TP + FP} \tag{1}$$

Recall is the ratio of the number of true positive instances to the number of actual positive instances, both true positive and false negative (FN), as shown in Equation 2.

$$Recall = \frac{TP}{TP + FN} \tag{2}$$

F_1 *score* is a harmonic mean of precision and recall, defined in Equation 3.

$$F_1\ score = \frac{2 * Recall * Precision}{Recall + Precision} \tag{3}$$

These metrics are used to compare the performance of TLEL against previously proposed methodologies for just-in-time defect prediction. One of these methodologies, Deeper [6], was also proposed by the authors of the original study that is being replicated here.

The results of the original study showed that TLEL of decision trees can achieve statistically significantly better results compared to previously tested learning algorithms.

Table 1: Research Questions in the Original Study

RQ_1	How effective are the classification methodologies
RQ_2	How effective are they with different percentages of lines of code inspected
RQ_3	What is the benefit of having multiple layers
RQ_4	What is the effect of varying the amount of training data on their effectiveness
RQ_5	How much time does it take for the methodologies to run
RQ_6	What is the effect of varying the parameter settings

3 INFORMATION ABOUT THE REPLICATION

When constructing an ensemble having diversity among the base learners is essential for performance and a strong generalization ability [8]. Deep learning is a machine learning method that uses layers of processing units where the output of a layer cascades to be the input of the next layer [1]. The multiple layers allow for varying levels of abstraction and the cascade between the layers enables the extraction of features from lower to higher level layers to improve performance [1]. To determine if adding diversity in classifiers and additional depth in layers can further improve performance on these datasets, we are introducing a novel algorithm called deep super learner [7] as an extension to the replication study. We also test the deep super learner (DSL) using the same experimental setup. The DSL can use any arbitrary set of base learners, optimized weights for the classifiers, and an adaptive number of layers depending on the dataset. The DSL here uses five base learners: logistic regression, k-nearest neighbors, random forest, extremely randomized trees, and XGBoost. For each layer, where the entire training set is passed through each of the base learners, weights of the learners are optimized to minimize cross entropy. The algorithm continues adding layers until cross entropy ceases to improve. Pseudo-code of DSL is provided in Algorithm 2.

```
for iteration in 1 to max iterations do
    Split data into k folds each with train and validate sets;
    for each fold in k folds do
        for each learner in ensemble do
            Train learner on train set in fold;
            Get class probabilities from learner on validate set
              in fold;
            Build predictions matrix of class probabilities;
        end
    end
    Get weights to minimize loss function with predictions
      and true labels;
    Get average probabilities across learners by multiplying
      predictions with weights;
    Get loss value of loss function with average probabilities
      and true labels;
    if loss value is less than loss value from previous iteration
      then
        Append average probabilities to original feature data;
    else
        Save iteration;
        Break for;
    end
end
```

Algorithm 2: A Pseudo-code of DSL

3.1 Level of interaction with original experimenters

This replication study was performed externally without involvement from the original researchers.

3.2 Changes to the Original Experiment

The analysis in this replication study uses data from the same six open source projects and the same feature set as the original study. The replication of the experiment, research questions, and design are similar to the original experiment except for the following:

- The datasets in this replication study do not include enough information about the number of lines of code inspected to calculate cost effectiveness. The first research question is evaluated using only F_1 *score*.
- The second research question is omitted due to the same reason as above.

4 COMPARISON OF RESULTS TO ORIGINAL

Our goal is to determine the depth of the performance of TLEL by comparing it to another ensemble employing a diverse set of classifiers, optimized weights, and more layers. The following subsections present the results of the replication using Deeper, TLEL, and DSL. These results are then compared to the results from the original study for each of the research questions.

4.1 Comparing the results to RQ_1

RQ_1 How effective are the methodologies?

Table 2 presents the precision of the original study, the replicated Deeper, TLEL, and DSL. Table 3 presents the recall of the methodologies, and Table 4 presents their F_1 *score*. Wilcoxon signed-rank statistical tests were performed between DSL and replicated Deeper, as well as DSL and replicated TLEL. The p-values with respect to F1-scores are presented in Table 5

Table 2: Precision Percent Values of Deeper, TLEL, and DSL

Project	Deeper Original	Deeper Replicated	TLEL Original	TLEL Replicated	DSL
Bugzilla	57.28	59.17	62.39	**62.24**	62.17
Columba	48.01	48.68	51.22	52.61	**53.60**
JDT	26.02	25.92	29.34	28.68	**30.00**
Mozilla	13.23	12.57	15.79	15.34	**15.52**
Platform	26.31	27.09	31.42	30.92	**31.52**
PostgreSQL	46.93	47.37	49.86	49.64	**50.30**
Average	36.30	36.80	40.00	39.91	**40.52**

Table 3: Recall Percent Values of Deeper, TLEL, and DSL

Project	Deeper Original	Deeper Replicated	TLEL Original	TLEL Replicated	DSL
Bugzilla	69.83	68.49	75.92	73.17	**73.47**
Columba	67.37	67.07	74.33	**71.36**	70.82
JDT	69.06	68.63	73.48	**73.50**	72.02
Mozilla	68.00	69.27	77.75	**77.61**	76.95
Platform	69.84	70.30	77.48	**75.18**	74.26
PostgreSQL	66.71	65.14	76.97	**74.55**	74.22
Average	68.47	68.15	75.99	**74.23**	73.62

Table 4: F_1 *score* of Deeper, TLEL, and DSL

Project	Deeper Original	Deeper Replicated	TLEL Original	TLEL Replicated	DSL
Bugzilla	0.6292	0.6348	0.6850	0.6722	**0.6730**
Columba	0.5606	0.5641	0.6065	0.6050	**0.6090**
JDT	0.3779	0.3762	0.4194	0.4125	**0.4233**
Mozilla	0.2215	0.2127	0.2625	0.2561	**0.2582**
Platform	0.3822	0.3910	0.4471	0.4381	**0.4425**
PostgreSQL	0.5509	0.5485	0.6052	0.5958	**0.5994**
Average	0.4537	0.4546	0.5043	0.4966	**0.5009**

Table 5: p-Values of DSL compared with Deeper and TLEL in terms of F_1 *score*

Project	With Deeper	With TLEL
Bugzilla	< 5.45e-06	0.3864
Columba	< 5.45e-06	0.0433
JDT	< 5.45e-06	< 5.45e-06
Mozilla	< 5.45e-06	< 5.45e-06
Platform	< 5.45e-06	< 5.45e-06
PostgreSQL	< 5.45e-06	0.0024

Consistent Results: Similar to the original study, we report precision, recall, and F_1 *scores* for the methodologies on each of the six datasets and the average across datasets. The F_1 *score* is statistically significantly higher at an alpha of 0.05 for TLEL across all projects compared to Deeper.

Differences in Results: The F_1 *score* is statistically significantly higher for the DSL on five of the six projects compared to TLEL and statistically significantly higher across all projects compared to Deeper. While the difference in results may appear small, the practical gain from DSL relative to TLEL in terms of number of defects detected can be significant when dealing with projects with a large number of changes. There is more evidence of outperformance by DSL on the larger datasets.

4.2 Comparing the results to RQ_3

RQ_3 What is the benefit of having multiple layers?

Table 6 presents the F_1 *score* after applying each layer of the TLEL in the original study and in the replication. Table 7 presents the F_1 *score* after applying each layer of the DSL.

Table 6: Cumulative contribution of each TLEL layer in terms of F_1 *score*

Project	Inner Layer Original	TLEL Original	Inner Layer Replicated	TLEL Replicated
Bugzilla	0.6503	0.6850	0.6312	0.6722
Columba	0.5783	0.6065	0.5637	0.6050
JDT	0.3871	0.4194	0.3874	0.4125
Mozilla	0.2300	0.2625	0.2478	0.2561
Platform	0.4080	0.4471	0.4143	0.4381
PostgreSQL	0.5647	0.6052	0.5666	0.5958
Average	0.4697	0.5043	0.4685	0.4966

Consistent Results: Similar to the original study, we report F_1 *scores* for the layers of the methodologies on each of the six datasets and the average across datasets. The F_1 *scores* of the layers of the original and replicated TLEL are statistically equivalent between the original and replicated experiments.

Differences in Results: The original study and the replicated TLEL show an improvement in F_1 *score* by stacking the base learners together. However, the DSL does not always display improved F_1 *scores* by stacking the base learners, as evidenced by individual

Table 7: Individual contribution of base learners in first layer and cumulative contribution of stacked layers in DSL in terms of F_1 *score*

Project	Logistic Regression	K-Nearest Neighbors	Random Forest	Extremely Randomized Trees	XGBoost	First Stack	Total Stacked Layers	DSL
Bugzilla	0.6037	0.5706	0.6502	0.6451	0.6718	0.6648	4	0.6730
Columba	0.5942	0.5585	0.5863	0.6066	0.5856	0.6084	3	0.6090
JDT	0.3353	0.3664	0.4232	0.4224	0.4164	0.4228	5	0.4233
Mozilla	0.1843	0.2034	0.2529	0.2461	0.2412	0.2555	3	0.2582
Platform	0.3270	0.3633	0.4330	0.4150	0.4402	0.4400	4	0.4425
PostgreSQL	0.5223	0.5311	0.5884	0.5774	0.5739	0.5935	3	0.5994
Average	0.4278	0.4322	0.4890	0.4854	0.4882	0.4975		0.5009

base learners outperforming the stack, nor do subsequent layers significantly improve F_1 *score* values. DSL, as implemented in this study, optimizes cross entropy, which is equivalent to maximizing the log likelihood of observing the data from the model. Since F_1 *score* is a non-convex function, it cannot be maximized with standard convex optimization algorithms. While optimizing cross entropy is not directly maximizing F_1 *score*, optimizing the likelihood maximizes F_1 *score* in expectation [4].

4.3 Comparing the results to RQ_4

> RQ_4 What is the effect of varying the amount of training data on their effectiveness?

Figure 1 presents the F_1 *scores* for 2-folds to 10-folds cross validation. With a smaller number of folds, a smaller portion of the data is used for training.

Consistent Results: Similar to the original study, an upward trend in F_1 *score* is observed with more training data for both TLEL and DSL. The largest variations are less than 2% indicating good robustness across different amounts of training data.

Differences in Results: Exact F_1 *score* values from the original study were not available for comparison. The visual presentation of the data appears very similar.

4.4 Comparing the results to RQ_5

> RQ_5 How much time does it take for the methodologies to run?

Table 8 presents the runtime to train and test Deeper, TLEL, and DSL.

Consistent Results: Similar to the original study, runtime is positively correlated with the size of the dataset. All methodologies run in reasonable amounts of time.

Differences in Results: All algorithms are implemented in Python using the scikit-learn library for all classifiers except XGBoost, which used the XGBoost. Experiments are run on a desktop with an Intel Core i7-7700 with 16 GB of RAM. DSL has a longer runtime due to more classifiers and additional layers being used.

Table 8: Runtime to train and test Deeper, TLEL, and DSL in seconds

Project	Deeper	TLEL	DSL
Bugzilla	8.19	21.26	37.83
Columba	5.78	21.25	37.12
JDT	16.77	21.90	41.53
Mozilla	18.80	22.91	44.94
Platform	24.33	22.50	52.45
PostgreSQL	14.64	21.64	41.12
Average	14.75	21.91	42.50

4.5 Comparing the results to RQ_6

> RQ_6 What is the effect of varying the parameter settings?

Figure 2 presents the effects of varying the parameter NTree or NLearner while keeping the other constant in the TLEL. Table 9 presents the effects of removing one of the base learners at a time while keeping the other learners constant in the DSL.

Table 9: The effect of removing one of the base learners while keeping the others constant on the F1-score of DSL on six datasets

Project	No Logistic Regression	No K-Nearest Neighbors	No Random Forest	No Extremely Randomized Trees	No XGBoost
Bugzilla	0.6617	0.6662	0.6708	0.6670	0.6711
Columba	0.5986	0.6176	0.6007	0.6115	0.6049
JDT	0.4175	0.4182	0.4167	0.4203	0.4107
Mozilla	0.2547	0.2573	0.2568	0.2589	0.2574
Platform	0.4371	0.4447	0.4432	0.4403	0.4395
PostgreSQL	0.5949	0.5980	0.5968	0.5971	0.5954
Average	0.4941	0.5003	0.4975	0.4992	0.4965

Consistent Results: Similar to the original study, an upward trend in F_1 *score* is observed with higher values for both NTree and NLearner with gains tapering off at values of 10. The largest variations are less than 0.05 indicating good stability across different parameter values.

Differences in Results: Exact F_1 *score* values from the original study were not available for comparison. The visual presentation of the data appears very similar. The DSL is robust with different

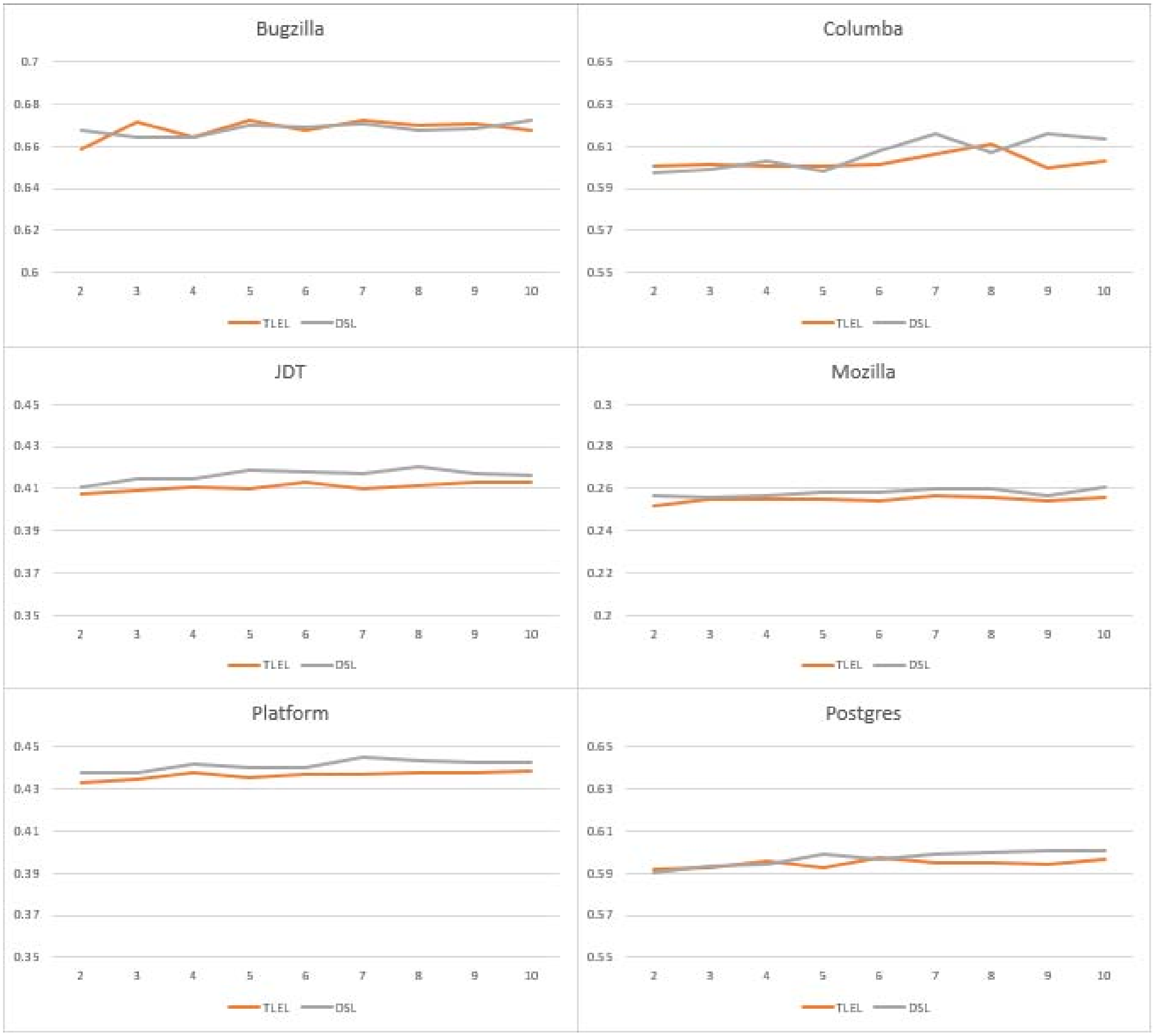

Figure 1: Two-to-ten fold validation results on six datasets.

sets of base learners. This flexibility in the set of base learners to use depending on requirements, performance on the problem, or constraints as dictated by the problem or computational resources, enables further insight into which learners are critical to an accurate prediction, as well as helping organizations in providing at least some of the important methods if other learners are not available.

5 CONCLUSIONS ACROSS STUDIES

We have conducted an empirical validation of the hybrid ensemble methodologies to improve performance in defect prediction relative to previously tested methodologies. In this paper, our objectives are to apply the guidelines for reporting a replication experiment and to analyze our findings for drawing cross study conclusions for both the original and replication studies. We analyze and highlight conclusions about using multiple base learners, optimized weightings, and additional layers with respect to their relationship to classification performance on the tested datasets. Although this study is a replication, we present a new technique with strong generalization abilities that can be used in future research. This generalization ability enables superior performance on larger datasets where defects may have different characteristics that are better identified by different classifiers. Replication of just-in-time defect prediction approaches has enabled us to confirm the external validity of our new prediction approach. Although our outcome may have led to

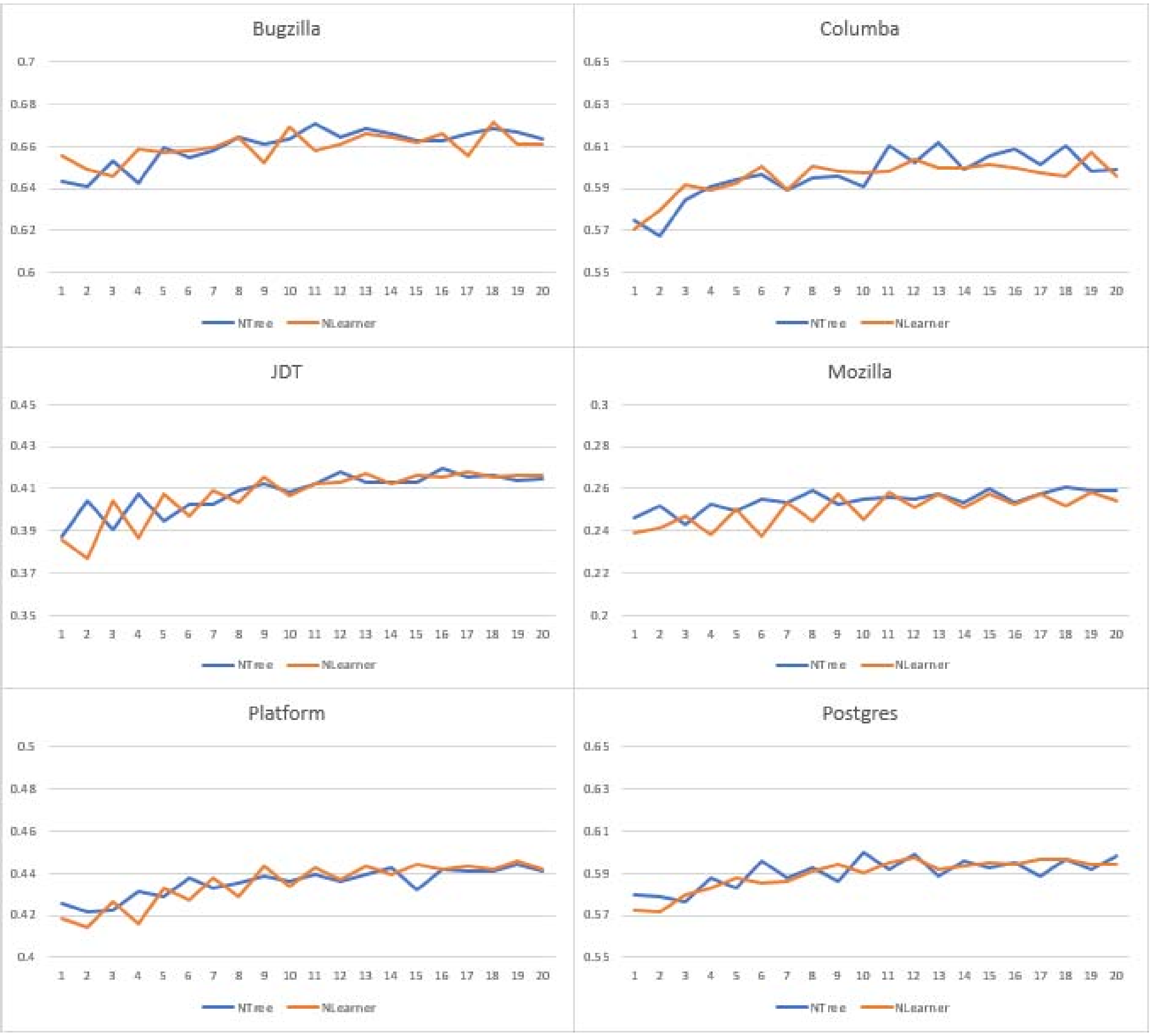

Figure 2: The effect of varying either parameter NTree or NLearner while keeping the other equal to 10 on the F1-score of TLEL on six datasets.

similar performance outcomes of the original studies, conducting this replication makes us more confident that Deep Super Learner is competitive with the best alternatives for just-in-time defect prediction.

REFERENCES

[1] Yoshua Bengio, Aaron Courville, and Pascal Vincent. 2013. Representation Learning: A Review and New Perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35, 8 (2013), 1798–1828.

[2] JC Carver. 2010. Towards Reporting Guidelines for Experimental Replications: A Proposal. In *Proceedings of the 1st International Workshop on Replication in Empirical Software Engineering Research (RESER) [Held during ICSE 2010]*. Cape Town, South Africa, 2–5.

[3] Yasutaka Kamei, Emad Shihab, Bram Adams, Ahmed E. Hassan, Audris Mockus, Anand Sinha, and Naoyasu Ubayashi. 2013. A Large-scale Empirical Study of Just-in-time Quality Assurance. *IEEE Transactions on Software Engineering* 39, 6 (2013), 757–773.

[4] Ye Nan, Kian Ming Chai, Wee Sun Lee, and Hai Leong Chieu. 2012. Optimizing F-measure: A Tale of Two Approaches. In *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*. 289–296.

[5] Xinli Yang, David Lo, Xin Xia, and Jianling Sun. 2017. TLEL: A Two-layer Ensemble Learning Approach for Just-in-time Defect Prediction. In *Information and Software Technology*, Vol. 87. 206–220.

[6] Xinli Yang, David Lo, Xin Xia, Yun Zhang, and Jianling Sun. 2015. Deep Learning for Just-in-Time Defect Prediction. In *Proceedings - 2015 IEEE International Conference on Software Quality, Reliability and Security, QRS 2015*. 17–26.

[7] Steven Young, Tamer Abdou, and Ayse Bener. 2018. Deep Super Learner: A Deep Ensemble for Classification Problems. In *Proceedings of the 31st Canadian Conference on Artificial Intelligence (CanadianAI-31)*.

[8] Zhi-Hua Zhou. 2012. *Ensemble Methods: Foundations and Algorithms.* Chapman & Hall/CRC. Machine Learning & Pattern Recognition Series. 236 pages.