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

ICSE'18

Aayushi Agrawal
NC State University
agrawa@ncsu.edu

ABSTRACT

In this poster, paper[1] has been discussed. Just-in-time defect prediction is used to efficiently allocate resources and manage project schedules in the software testing, debugging process and reduce the amount of code review. The prediction process includes a replicated experiment and an extension comparing the prediction of defect-prone changes using traditional machine learning techniques and ensemble learning. Datasets from six different open source projects, have been used namely Bugzilla, Columba, JDT, Platform, Mozilla, and PostgreSQL where the original approach has been replicated for verification of results and used them as a basis for comparison for alternatives in the approach.

1 INTRODUCTION

Developers often have limited resource and are constrained to perform rigorous and comprehensive testing and debugging efforts on all parts of a code base. 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. The original experiment [2] investigated whether hybrid ensembles of machine learning methods could improve the performance of just-in-time defect prediction. The original experiment uses a combination of data pre-processing 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. In paper[1], the approach has been generalized which allows the use of any arbitrary set of classifiers in the ensemble, optimizing the weights of the classifiers, and allowing additional layers, and to test the depth of the original study,a new deep ensemble approach, called deep super learner is applied.

2 ORIGINAL STUDY

The research questions include:

- **RQ1:** How effective are classification methodologies
- **RQ2:** How effective are they with different percentages of lines of code inspected
- **RQ3:** What is the benefit of having multiple layers
- **RQ4:** What is the effect of varying the amount of training data on their effectiveness
- **RQ5:** What is the run time for methodologies
- **RQ6:** What is the effect of varying parameter settings

Pseudo code for original study is shown in Algorithm 1. Cost effectiveness and F1 score are the two metrics used to evaluate performance. Cost effectiveness is the percentage of buggy changes found when reviewing a specific percentage of the lines of code. The F1 score is to evaluate classification performance. These metrics are used to compare the performance of TLEL against previously proposed methodologies for just-in-time defect prediction.

Algorithm 1: Pseudo Code of TLEL

```
 1: for iteration in 1 to number of learners do
 2:     Random sub-sample from majority class to balance
        classes
 3:     Train random forest classifier
 4: end for
 5: class = 0
 6: for iteration in 1 to number of learners do
 7:     Predict test instance with trained model
 8:     if predicted class is buggy then
 9:         class ← class +1
10:     end if
11: end for
12: if class ≥ (number of learners)/2 then
13:     final predicted class ← buggy
14: end if=0
```

3 REPLICATION STUDY

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 uses five base learners: logistic regression, k-nearest neighbors, random forest, extremely randomized trees, and XGBoost. In each layer the entire training set is passed through each of the base learners, weights of the learners are optimized to minimize cross entropy. Layers are continuously added using algorithm until cross entropy ceases to improve. Pseudo code for original study is shown in Algorithm 2.

4 CHANGES TO ORIGINAL EXPERIMENT

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 F1 score.
- The second research question is also omitted due to the same reason as the above.

5 COMPARISON OF RESULTS

The following subsections present the results of the replication using Deeper, TLEL, and DSL. These results are then

Algorithm 2: Pseudo Code for DSL

```
    for iteration in 1 to max iterations do
2:      Split data into k folds each with train and validate
        sets
        for each fold in k folds do
4:          for each learner in ensemble do
                Train learner on train set in fold;
6:              Get class probabilities from learner on validate
                set in fold;
                Build predictions matrix of class probabilities;
8:          end for
        end for
10:     Get weights to minimize loss function with predictions
        and true labels;
        Get average probabilities across learners by multiply-
        ing predictions with weights;
12:     Get loss value of loss function with average
        probabilities and true labels;
        if loss value is less than loss value from previous
        iteration then
14:         Append average probabilities to original feature
            data;
        else
16:         Save iteration;
            Break for;
18:     end if
    end for=0
```

compared to the results from the original study for each of the research questions. 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 F1 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

Figure 1: Results of RQ3: Cumulative contribution of each TLEL layer in terms of F1 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

Figure 2: Results of RQ3: Individual contribution of base learners in first layer and cumulative contribution of stacked layers in DSL in terms of F1 score

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

Figure 3: Results of RQ5: Runtime to train and test Deeper, TLEL, and DSL in seconds

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**

(a) Precision Values

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

(b) Recall Values

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**

(c) F1 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

(d) p-values in terms of F1

Figure 4: Results of RQ1

6 LIMITATIONS:

There can be few limitations as listed below:

- Approach is only applied to open source projects and not tested with private or enterprise softwares.
- New markets (e.g., mobile applications and energy consumption) have not been considered in the domain of defect predections.

7 CONCLUSION

An empirical validation of the hybrid ensemble methodologies has been conducted to improve performance in defect prediction relative to previously tested methodologies. DSL helps to 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. It is a new technique with strong generalization abilities that can be used in future research. Replication of just-in-time defect prediction approaches has enabled to confirm the external validity of new prediction approach, which makes more confident that Deep Super Learner is competitive with the best alternatives for just-in-time defect prediction.

REFERENCES

[1] Young, Steven, Tamer Abdou, and Ayse Bener. *"A replication study: just-in-time defect prediction with ensemble learning."* Proceedings of the 6th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering. ACM, 2018.

[2] Yang, Xinli, et al. *"TLEL: A two-layer ensemble learning approach for just-in-time defect prediction."* Information and Software Technology 87 (2017): 206-220.

[3] 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.