Software Defect Prediction in the Cloud

Cagatay Catal, Merve Erdogan, Cem Isik
Istanbul Kultur University
Istanbul, Turkey
c.catal, m.erdogan, c.isik@iku.edu.tr

***Abstract*—Software defect prediction is one of the software quality assurance activities that can be applied during the software development life cycle. This activity helps quality assurance groups and project managers to determine risky modules which require more attention and more testing efforts. In this study, we investigated nine classification algorithms on 22 datasets which contain class-level metrics as part of our within-project case study. After this case study, we sorted the datasets regarding their data instances and performed cross-project experiments on the large datasets, namely Apache Xalan, Xerces, and POI projects. We demonstrated that Decision Tree based algorithms are mostly superior to the other classification algorithms for within-project defect prediction and acceptable results can be achieved with Logistic Regression algorithms for cross-project defect prediction even if the data transformation approaches are not applied.**

I. INTRODUCTION

Most of the software defect prediction models utilize from historical defect and metrics data [1]. From machine learning perspective, this prediction problem can be represented as a supervised learning problem and classification algorithms such as Naive Bayes and Random Forests can be applied. However, there are some real-world cases in which there are no previous defect data. For example, the first project of a company in a new domain such as unmanned aerial vehicles (UAV) will not be able to use historical defect data to build the defect prediction model. In this kind of cases, it's possible to use data from the other projects or companies. In literature, this research area is known as Cross-Project Defect Prediction (CPDP). Recently, many researchers proposed novel methods to apply data from the other companies for defect prediction. Herbold [2] published a very recent systematic mapping study on this issue and he accessed 49 CPDP papers published between year 2002 and 2015. He reported that 25 publications applied Logistic Regression technique and summarized the contributions of these 49 papers in detail.

In this study, we aimed to analyze the available classification algorithms in Azure Machine Learning (ML) Studio for within-project defect prediction problem and build cross-project defect prediction models without using any data transformation algorithm. We performed our experiments on the following 22 datasets which exist in the PROMISE repository [3] and prepared by Jureczko and Madeyski [4]: Workflow, Xerces, Xalan, POI, Velocity, Tomcat, Thermoproject, Systemdata, Synapses, Skarbonka, Sherapion, Redactor, Prop, Forest, Ivy, JEdit, Log4j,Lucene, Ant, Arc, Kalkulator, Nieruchomosci.

There are 20 metrics in these datasets. 6 metrics (WMC, DIT, NOC, CBO, RFC, and LCOM) belong to Chidamber-Kemerer metrics suite [5], two metrics (CA and CE) are proposed by Martin [6], three metrics (IC, CBM, AMC) are suggested by Tang et al. [7], five metrics (NPM, DAM, MOA, MFA, CAM) belong to Bansiy and Davis metrics suite [8], one metric (LCOM3) is proposed by Henderson-Sellers [9], two cyclomatic complexity metrics (MaxCC, avgCC) are related with McCabe cyclomatic complexity metric [10], and one metric is the popular lines of code (LOC) metric.

The investigated classification algorithms are Averaged Perceptron, Bayes Point Machine, Boosted Decision Tree, Decision Forest, Decision Jungle, Locally Deep Support, Logistic Regression, Neural Network, and Support Vector Machine. After the performance of these algorithms are calculated based on Area under ROC Curve (AUC) evaluation parameter, top three algorithms were marked in the tables with #1, #2, and #3 labels. Since we did not observe an algorithm which performs best on all the 20 datasets, we decided to count the number of these labels. Based on the count of these labels, we identified the top algorithms for within-project defect prediction. After this first case study, we listed the datasets based on their sizes and selected the large ones for cross-project defect prediction case study. In the second case study, we performed our experiments on Apache Xalan, Xerces and POI projects. Therefore, we created four different analyses:

1) Xalan for training – Xerces for testing
2) Xerces for training – Xalan for testing
3) Xerces for training – POI for testing
4) POI for training – Xerces for testing

After the best algorithm is selected, the best model was transformed into a web service and deployed on Azure cloud platform. In addition, a web-based client application was implemented to consume this web service. The following explains the Related Work. Section III shows the methodology applied in this study. Experimental results are given in section IV. Section V shows the conclusion and future work.

II. RELATED WORK

There are many recent papers on cross-company defect prediction. Traditional software defect prediction models are evaluated under within-company defect prediction research area. Yu et al. [11] analyzed whether the features or instances are more important for cross-project defect prediction and concluded that features are more important than instances

based on their analyses on NASA and PROMISE datasets. Gunarathna [12] identified 30 CPDP studies as part of systematic literature review (SLR) study and reported that models using Nearest Neighbour and Decision Tree provide good performance in CPDP problems. Hosseini et al. [13] proposed a new data selection approach called Genetic Instance Selection for CPDP context and concluded that it is promising for the selection of training data. He et al. [14] proposed a novel data selection method called TDSelector and reported that the prediction models using this method provide better performance than the baseline methods. Wu et al. [15] developed a semi-supervised structured dictionary learning (SSDL) approach for cross-project semi-supervised defect prediction problem. Hosseini et al. [16] demonstrated that search based instance selection and feature selection can improve the performance of CPDP models. Yu et al. [17] designed a feature matching and transfer approach (FMT) and showed that it is effective for CPDP. Yu et al. [18] suggested a data filtering approach based on Agglomerative clustering and showed that it improves the performance of CPDP models. Yu et al. [19] showed that the class imbalance learning method called under-sampling improves the performance of CPDP models. Herbold et al. [20] analyzed the performance of local models in CPDP context and reported that they provide a minor difference compared to global models. You et al. [21] CPDP problem was modelled as a ranking problem and suggested a ranking-oriented CPDP method. As seen in these recent studies, researchers are still developing methods for this challenging problem.

III. METHODOLOGY

Datasets regarding to the software projects were downloaded from the PROMISE repository [3]. Since there were different versions of each project such as Xerces-init, Xerces-1.2, Xerces-1.3, Xerces-1.4, we decided to merge these datasets to get a larger dataset. It's known that a larger dataset mostly simplifies the learning process of machine learning algorithms in machine learning community. This merge operation was performed for the other projects such as JEdit, Ivy, Forrest, POI, and Log4j as well. Therefore, we got only one dataset for each project.

For the evaluation of our fault prediction models, we applied the hold-out validation approach. In this approach, the original dataset is split into two parts, namely training and testing. The performance of the model on the testing set is considered as the generalization capability of the learning model. We used 70% of the dataset for training and 30% of the dataset of testing. This step is performed with the Split Data component in Azure ML Studio platform. The experiment screen regarding to the within-project defect prediction is shown in Figure 1. The dataset component (poi-merge.csv) is inserted into the top of the experiment screen as seen in Figure 1. Two-Class Boosted Decision Tree component, shown at the left hand side of the screen, represents the learning algorithm in this experiment. All the other classification algorithms were used after this algorithm was applied. Split Data component helps us to specify the hold-out evaluation approach. If the dataset is not large enough, K-fold cross-validation might also be considered as an alternative evaluation method in machine learning. To do so, Partition and Sample module must be used instead of Split Data component. Train Model component in the figure uses the training dataset to learn the parameters of the learning algorithm. Once the training model component is inserted into the screen, the class label of the dataset must be specified on the Train Model component with the help of Launch Column Selector button. Score Model component analyzes the performance of the model in the testing set. Evaluate Model component helps to depict the evaluation parameter results. All of these components in italics are shown in Fig. 1.

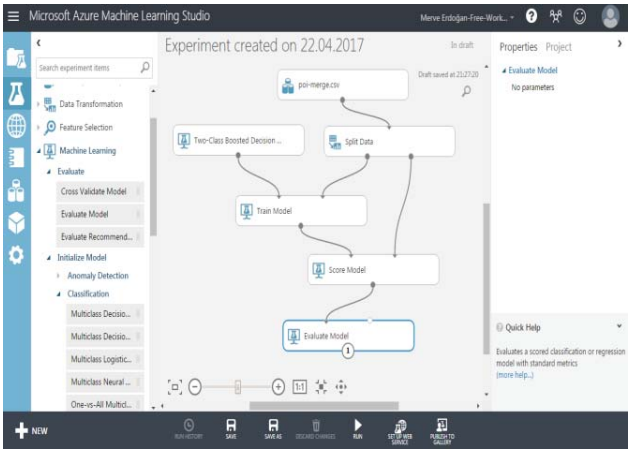

Fig. 1. Experimental design for the case study-I

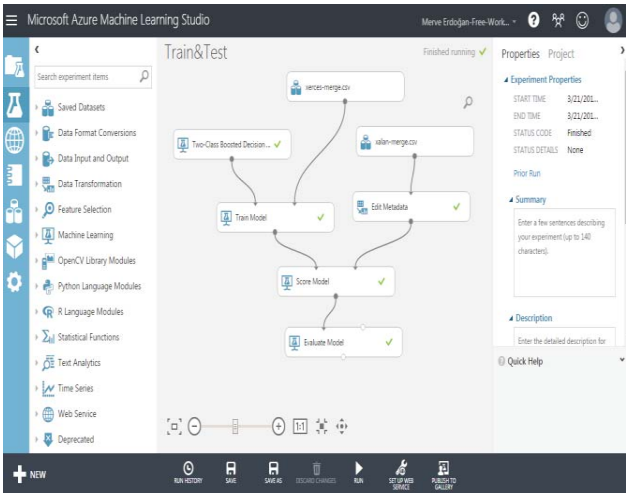

Fig. 2. Experimental design for the case study-II

In Fig. 2, experimental design for the case study II is depicted. In this figure, we have two datasets because while one of them will be used for the training step, the other one will be used for the testing step.

Evaluation results are given based on the AUC (Area under ROC Curve) evaluation parameter which is between 0 and 1. If the value is near to 1, this indicates that the model's performance is perfect. We can state that the higher this value is, the better the performance of the model is. While the x-axis of the ROC (Receiver Operating Characteristics) curve plots the false positive rate, y-axis represents the true positive rate.

This curve passes from the points (0, 0) and (1, 1). Once new points are added to this curve, the integral of this curve provides the area under this curve.

We divide our experiments into two categories:

- Case Study I: Within-Project Defect Prediction
- Case Study II: Cross-Project Defect Prediction

Since a large amount of data will be required to build a high-performance CPDP model, we sorted the datasets based on their sizes. Therefore, Case Study-II was performed on the large datasets instead of all the datasets used in Case Study I. However, Case Study I was performed for all the datasets and all the classification algorithms. Also, all the classification algorithms were investigated for Case Study II.

IV. Experimental Results

Nine classification algorithms were applied for both of the case studies. While 22 datasets were used in the first case study, three datasets were investigated for CPDP analyzes.

A. Case Study I: Within-Project Defect Prediction

In this case study, all the classification algorithms were investigated for the within-project defect prediction. The top three algorithms are indicated with the red colour in Table 1. Red 1 indicates that the corresponding algorithm works best for that dataset. We decided to count the number of these labels (1, 2, and 3) to select the best algorithm for the case study I.

TABLE I. Experimental Results for Case Study-I

Classification	Workflow	xerces-merge	kalan-merge	velocity-merge	tomcat	termoproject	systemdata	synapse-merge
Two Class Averaged Perceptron	0.594	0.823	0.819	0.755	0.790 2	0.639	0.389	0.732
Two Class Bayes Point Machine	0.625	0.674	0.544	0.517	0.517	0.694 3	0.500 3	0.701
Two Class Boosted Decision Tre	0.781 3	0.926 2	0.915 * 1	0.846 * 1	0.748	0.667	0.278	0.825 * 1
Two Class Decision Forest	0.984 * 1	0.931 * 1	0.862 3	0.817 2	0.706	0.764 * 1	0.306	0.717
Two Class Decision Jungle	0.656	0.914 3	0.882 2	0.771	0.746	0.736 2	0.306	0.752 3
Two Class Locally Deep Support	0.531	0.869	0.791	0.715	0.754	0.639	0.444	0.750
Two Class Logistic Regression	0.875 2	0.854	0.861	0.790 3	0.782	0.667	0.778 * 1	0.754 2
Two Class Neural Network	0.563	0.857	0.847	0.756	0.791 * 1	0.639	0.667 2	0.693
Two Class Support Vector Machi	0.656	0.830	0.840	0.735	0.786 3	0.639	0.167	0.683

Classification	skarbonka	serapion	redaktor	prop-merge	poi-merge	ForrestMerge	ivyMERGE	jeditMERGE
Two Class Averaged Perceptron	0.758 2	0.424	0.724 3	0.718	0.816	0.583	0.732	0.879 * 1
Two Class Bayes Point Machine	0.500	0.667	0.500	0.716	0.768	0.806	0.405	0.737
Two Class Boosted Decision Tre	0.636	0.970 * 1	0.722	0.870 * 1	0.888 * 1	0.972 * 1	0.795 3	0.872 3
Two Class Decision Forest	0.485	0.879 2	0.768 2	0.851 2	0.878 2	0.819	0.804 2	0.860
Two Class Decision Jungle	0.470	0.833 3	0.666	0.846 3	0.874 3	0.875 3	0.868 * 1	0.852
Two Class Locally Deep Support	0.697	0.394	0.564	0.659	0.835	0.639	0.743	0.837
Two Class Logistic Regression	0.636	0.788	0.704	0.718	0.853	0.500	0.774	0.862
Two Class Neural Network	0.727 3	0.606	0.796 * 1	0.804	0.826	0.917 2	0.739	0.873 2
Two Class Support Vector Machi	0.818 * 1	0.409	0.719	0.626	0.830	0.389	0.717	0.831

Classification	log4jMERGE	luceneMERGE	ant-1.7	arc	kalkulator	nieruchomosci
Two Class Averaged Perceptron	0.894	0.683	0.839 3	0.753	0.917 *	0.375
Two Class Bayes Point Machine	0.530	0.640	0.554	0.500	0.500	0.625 3
Two Class Boosted Decision Tre	0.923	0.714 3	0.822	0.672	0.500	0.500
Two Class Decision Forest	0.939 * 1	0.676	0.804	0.743	0.833	0.719 2
Two Class Decision Jungle	0.938 2	0.721 * 1	0.821	0.811 * 1	0.833 2	0.750 * 1
Two Class Locally Deep Support	0.923	0.715 2	0.821	0.667	0.917 *	0.375
Two Class Logistic Regression	0.931 3	0.721 *	0.843 2	0.782 3	0.833	0.500
Two Class Neural Network	0.887	0.629	0.845 * 1	0.789 2	0.917 * 1	0.500
Two Class Support Vector Machi	0.921	0.713	0.837	0.537	0.667 3	0.438

In Table 2, we show that how many times an algorithm has been listed in the top three ranks during the experiments. According to the Table 2, it is observed that decision jungle, decision forest, and boosted decision tree algorithms work better than the other classification algorithms when all the datasets are considered. Since all of these three algorithms are based on the decision tree concepts, we can state that the decision tree based algorithms are more appropriate for within-project defect prediction when class-level metrics are used.

TABLE II. Performance Results of Each Algorithm

Classification	# of times in the first three ranks
Two Class Averaged Perceptron	5
Two Class Bayes Point Machine	3
Two Class Boosted Decision Tree	12
Two Class Decision Forest	12
Two Class Decision Jungle	14
Two Class Locally Deep Support	1
Two Class Logistic Regression	7
Two Class Neural Network	9
Two Class Support Vector Machine	3

B. Case Study II: Cross-Project Defect Prediction

Since Xalan, Xerces, and POI datasets had larger amount of data instances, we decided to work with them for cross-project analyzes. According to the experimental results, we observed that Logistic Regression works best for most of the cross-project cases as shown in Table 3. We did not perform any data filtering or data selection method while building our CPDP models. Therefore, we can state that a large amount of cross-project data might be good at predicting the defects in another environment and no complex data selection algorithms are required to build these CPDP models. However, if the reported performance results are not acceptable for a specific domain such as real-time mission critical systems, it's absolutely required to integrate recent data selection and data filtering methods into the machine learning models.

TABLE III. Cross-Project Analysis Results

Classification	xalan train xerces test	xerces train xalan test	xerces train poi test	poi train xerces test
Two Class Averaged Perceptron	0.596	0.503	0.686	0.599
Two Class Bayes Point Machine	0.564	0.510	0.646	0.575
Two Class Boosted Decision Tree	0.548	0.602 *	0.586	0.527
Two Class Decision Forest	0.577	0.506	0.612	0.561
Two Class Decision Jungle	0.571	0.577	0.649	0.557
Two Class Locally Deep Support	0.542	0.530	0.678	0.498
Two Class Logistic Regression	0.604 *	0.534	0.688 *	0.605 *
Two Class Neural Network	0.580	0.493	0.678	0.592
Two Class Support Vector Machine	0.601	0.534	0.659	0.598

After these analyzes were done, the best model for within-project defect prediction was transformed into a web service in Azure ML Studio Platform. The selected dataset for this transformation was Log4j and the algorithm was Decision Jungle since it provides the best performance over all the other classification algorithms.

Web Service Input and Web Service Output components must be integrated into the experiment screen to build the web service. After the web service was deployed into the Azure cloud platform, a web-based client application was implemented using ASP.NET technology. The inputs are received from the user, sent to the web service, and the defect prediction result is returned to the user. Azure ML Studio platform easily lets developers to use the request/response API produced after the web service is produced.

V. Conclusion and Future Work

In this study, several classification algorithms were first investigated for within-project defect prediction and it was shown that decision jungle, decision forest, and boosted decision tree algorithms which are decision tree-based algorithms provide better performance than the other classification algorithms. In the second case study, it was demonstrated that logistic regression based CPDP provides acceptable results even if any data transformation and data filtering method are applied. In addition, we showed that the transformation of the prediction model into a web service is an efficient approach for building software fault prediction systems. Also, it was concluded that acceptable results can be achieved if the cross-project data are large enough to build the training model. In the future, new experiments will be performed on the new datasets and data filtering algorithms will be integrated into these systems to improve the prediction results.

References

[1] C. Catal, "A systematic review of software fault prediction results", *Expert Systems with Applications.*, vol.36, no.4, 2009, pp. 7346-7354.

[2] S. Herbold, "A systematic mapping study on cross-project defect prediction", *Empir Software Eng.*, in press.

[3] PROMISE official website, Web: http://openscience.us/repo.

[4] M. Jureczko, L. Madeyski, "Towards identifying software project clusters with regard to defect prediction", *in Proc. 6th International Conference on Predictive Models in Software Engineering*, 2010, pp. 1-10

[5] S.R. Chidamber, C.F. Kemerer, "A metrics suite for object oriented design", *IEEE Transactions on Software Engineering*, vol. 20, no.6, 1994, pp.476-493.

[6] R. Martin, "OO design quality metrics. An analysis of dependencies", *in Proc. ROAD 1995*, vol. 12, 1994, pp. 151-170.

[7] M.H. Tang, M.H. Kao, M.H. Chen, "An empirical study on object-oriented metrics", *in Proc. Software Metrics Symposium*, 1999, pp. 242-249.

[8] J. Bansiya, C.G. Davis, "A hierarchical model for object-oriented design quality assessment", *IEEE Transactions on Software Engineering*, vol.28, no.1, 2002, pp. 4-17.

[9] B. Henderson-Sellers, *Object-oriented metrics: measures of complexity*. NJ, Prentice-Hall, 1995.

[10] T.J. McCabe, "A complexity measure", *IEEE Transactions on Software Engineering*, vol.4, 1976, pp. 308-320.

[11] Q. Yu, S. Jiang, J. Quian, "Which is more important for cross-project defect prediction: instance or feature?", *In Proc. International Conference on Software Analysis, Testing, and Evolution*, 2016, pp.90-95.

[12] D. Gunarathna, "A systematic literature review on cross-project defect prediction", 2016, University of Oulu, Master Thesis.

[13] S. Hosseini, B. Turhan, M. Mantyla, "Search based training data selection for cross project defect prediction", *In Proc. of the 12th International Conference on Predictive Models and Data Analytics in Software Engineering*", 2016, pp.3.

[14] P. He, Y. Ma, B. Li, "*TDSelector: A training data selection method for cross-project defect prediction*", *CoRR*, in press.

[15] F. Wu, X.Y. Jing, X. Dong, J. Cao, M. Xu, H. Zhang, B. Xu, "Cross-project and within-project semi-supervised software defect prediction problems study using a unified solution", *"In Proceedings of the 39th International Conference on Software Engineering Companion"*, 2017, pp. 195-197.

[16] S. Hosseini, B. Turhan, M. Mäntylä, "A benchmark study on the effectiveness of search-based data selection and feature selection for cross project defect prediction", *Inf. Soft. Tech.* in press.

[17] Q. Yu, S. Jiang, Y. Zhang, "A feature matching and transfer approach for cross-company defect prediction", *Journal of Systems and Software*, vol.132, 2017, pp. 366-378.

[18] X. Yu, J. Zhang, P. Zhou, J. Liu, "A data filtering method based on agglomerative clustering", *In Proc. SEKE 2017*, pp. 392-397.

[19] X. Yu, M. Zhou, X. Chen, L. Deng, L. Wang, "Using class imbalance learning for cross-company defect prediction", *In Proc. SEKE 2017*, pp. 117-122.

[20] S. Herbold, A. Trautsch, J. Grabowski, "Global vs. local models for cross-project defect prediction", *Empirical Software Engineering*, 2016, pp. 1-37.

[21] G. You, F. Wang, Y. Ma, "An empirical study of ranking-oriented cross-project software defect prediction", *International Journal of Software Engineering and Knowledge Engineering*, vol.26, 2016, pp.1511-1538.