Is Bigger Data Better for Defect Prediction: Examining the Impact of Data Size on Supervised and Unsupervised Defect Prediction

Xinyue Liu[1,2] and Yanhui Li[1,2]

1. State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China
2. Department of Computer Science and Technology, Nanjing University, Nanjing, 210023, China

Abstract. Defect prediction could help software practitioners to predict the future occurrence of bugs in the software code regions. In order to improve the accuracy of defect prediction, dozens of supervised and unsupervised methods have been put forward and achieved good results in this field. One limiting factor of defect prediction is that the data size of defect data is not big, which restricts the scope of application with defect prediction models.
In this study, we try to construct *bigger* defect datasets by merging available datasets with the same measurement dimension and check whether bigger data will bring better defect prediction performance with supervised and unsupervised models or not. The results of our experiment reveal that larger-scale dataset doesn't bring improvements of both supervised and unsupervised classifiers.

Keywords: Defect Prediction · Supervised · Classifier · Data Size.

1 Introduction

Fixing software defects is a very difficult and time-consuming job for software practitioners during the development of a large-scale software project [3], at the start of which we need to identify potential locations of these defects. In last decades, software defect prediction has been proposed to locate defect-prone code regions [7, 8, 10, 12, 19, 20], most of which are constructed by supervised classifiers (e.g. trained with label information in the training set and tested on the disjoint test set). Generally, these supervised prediction models have helped a lot in software test and alleviate debug burden of software engineers. In most of previous studies, it has been reported that these prediction models have a promising performance in defect prediction.

Recently, unsupervised prediction defect has drawn more attention from academic fields. Zhou et al. [24] pointed out that a simple unsupervised prediction defect model (simple module size models) has a prediction performance comparable or even superior to most of the existing defect prediction models in cross-project scenario. Yang et al. [22] showed that in effort-aware just in time defect prediction settings, many simple unsupervised models have a better performance compared with the state-of-the-art supervised models.

One limiting factor of defect prediction is that the data size of defect data extracted from the real world software projects is not big, which prevents defect prediction models from employing popular methods of analyzing and mining big data and restricts the scope of application with defect prediction models. In this study, we try to construct *bigger* defect datasets by merging available datasets with the same measurement dimension and check whether bigger data will bring better defect prediction performance with supervised and unsupervised models or not.

In detail, we introduce traditional supervised classifiers and the state-of-the-art supervised classifier (simple module size model [24]) to test their performances on the bigger datasets we collected. Besides, due to the curiosity about if the promotion of the scale of dataset would boost the performances of supervised classifiers, we will organize a comparative experiment between raw smaller datasets and merged bigger datasets. The results of our experiment reveal that larger-scale dataset doesn't bring improvements of supervised classifiers, and simple module size model is also comparable to traditional supervised models on bigger datasets.

The rest of this paper is organized as follows. Section 2 introduces the background of defects predict models and some prior related work. Section 3 describes all the preparations, including studied dataset, introduced classifiers and evaluation measures. Section 4 presents the experimental results in detail. Section 5 is a summarization of all the threats to the validity of our experience. Section 6 concludes the paper.

2 Background and Related Work

There are two kind of defect predict model - supervised and unsupervised. The typical process of supervised prediction model is as follow: firstly, marking each instance's data as buggy or clean according to the bug report; secondly, gathering relevant metrics as prediction features; thirdly, training prediction models using selected classifiers with the bug labels and metrics given; finally, predicting whether an instance given is buggy or not based on trained models. In contrast, unsupervised models could classify objects directly without training process, so it's time-saving and easy to implement. Clustering and simple module size model are two common unsupervised techniques used in software field.

According to the dataset based on, we also could divide defect prediction into two scenarios, one of them is within-project defect prediction (WPDP). In WPDP, the dataset used to train classifiers and the one used for prediction come from a same project. The other scenario is cross-project defect prediction (CPDP). In practice, some companies may find it's hard to collect enough data in their project to train the classifier [25] , so it's essential for them to seek for CPDP solutions, which could utilize data from a different source project.

In this study, we will leverage both supervised and unsupervised models, and pay our attention mainly on WPDP.

3 Experimental Setup

In this section, we will introduce our preparations for the experiment. In the following statement, you will have a primary understanding about how we collect dataset, select classifier and other similar things.

3.1 Collected datasets

We use data from 15 projects of three groups - AEEEM, Eclipse and JURECZKO. The data in one group have unified metrics, so they are easy to be merged up and conduct further testing. Each project has both code metrics and clear defect label. More specifically, each file's defect label is marked as 1 (buggy file) or 0 (clean file).

Table 1 describes the 43 data sets used in Section 5. The first to the third columns respectively list the group name, project name and version number. For each project, the fourth to the sixth columns respectively list the number of modules, the number of metrics, and the percent of defective modules.

In this study, we try to figure out how these defect predict models perform on datasets with larger scale, so all the versions in a project are merged into one file (they have the same measurement dimension). Group AEEEM is an exception, because the projects of it don't process series of versions for us to merge. Hence, we merge all the projects in AEEEM into one file. All these merged files are called **combination** in the following description. In contrast, original raw files are called **individual**. All the merged files are listed in **Table 2**.

3.2 Classifiers

In this study, we leverage three common supervised learning models and a simple module size model. They are DNN (Deep Neural Network), RF (Random Forest), LR (Logistic Regression) and ManualDown. Compared with other frequently-used supervised learning models such as Linear Regression or Decision Tree, the models we chose are more suitable and representative to be applied on our dataset. Notions and implement details are presented below in sequence.

Deep Neural Network Deep Neural Network (DNN) is a classical machine learning technique and is widely used in a variety of learning scenarios. A typical DNN model equips multiple hidden layers between the input and output layer, which imitates the construction of natural neural networks. Through continuous adjustment of neuron parameters in the model, ultimately DNN could turn input into proper output.

In our study, we build DNN model with the help of tensorflow. the input of DNN is heterogeneous metric sets, and we need to use its output to predict

Table 1. Information of collected data sets.

Group	Project	Version	#Modules	#Metrics	%Defective
AEEEM	JDT core	-	997	31	20.66%
	Equinox	-	324	31	39.81%
	Lucene	-	691	31	9.26%
	Mylyn	-	1862	31	13.16%
	PDE	-	1497	31	13.96%
Eclipse	eclipse	2.0	6729	31	14.49%
		2.1	7888	31	10.83%
		3.0	10593	31	14.80%
JURECZKO	ant	1.3	125	20	16.00%
		1.4	178	20	22.47%
		1.5	293	20	10.92%
		1.6	351	20	26.21%
		1.7	745	20	22.28%
	camel	1.0	339	20	3.83%
		1.2	608	20	35.53%
		1.4	872	20	16.63%
		1.6	965	20	19.48%
	jedit	3.2	272	20	33.09%
		4.0	306	20	24.51%
		4.1	312	20	25.32%
		4.2	367	20	13.08%
		4.3	492	20	2.24%
	lucene	2.0	195	20	46.67%
		2.2	247	20	58.30%
		2.4	340	20	59.71%
	poi	1.5	237	20	59.49%
		2.0	314	20	11.78%
		2.5	385	20	64.41%
		3.0	442	20	63.57%
	synapse	1.0	157	20	10.19%
		1.1	222	20	27.03%
		1.2	256	20	33.59%
	velocity	1.4	196	20	75%
		1.5	214	20	66.36%
		1.6	229	20	34.06%
	xalan	2.4	723	20	15.21%
		2.5	803	20	48.19%
		2.6	885	20	46.44%
		2.7	909	20	98.79%
	xerces	1.2	440	20	16.14%
		1.3	453	20	15.23%
		1.4	588	20	74.23%
		init	162	20	47.53%

Table 2. Information of merged datasets.

Merged Project	#Modules	#Metrics	%Defective
AEEEM	5371	31	15.88%
eclipse	17999	31	13.43%
ant	1567	20	21.06%
camel	2784	20	20.19%
jedit	1749	20	17.32%
lucene	782	20	56.01%
poi	1378	20	51.31%
synapse	635	20	25.51%
velocity	639	20	57.43%
xalan	3320	20	54.4%
xerces	1643	20	39.81%

defect-prone instances in the same project. The DNN we use equips 2 hidden layers and 15 neural nodes in each layer. We select AdamOptimizer provided by tensorflow as training algorithm and Cross Entropy as loss function. In the case of a step size of 0.00005, the loss are becoming steading after 10000 training sessions.

Random Forest Random forest (RF) is an ensemble learning method usually used for classification. Its basic idea is to construct a multitude of decision trees during training process, and use mean prediction of each individual trees as output. Compared with single decision tree, random forest avoids overfitting situation, and it could be utilized on high dimensional data.

Nowadays random forest is used more and more frequently in software engineering field study, and have been proved to be effective by former related work [5, 16] . In our experience, we leverage the package in R named **randomForest** to implement the model. The number of trees grown (a parameter in randomForest function) is set 100, which is a apropos value after our testing.

Logistic Regression Logistic regression (LR) is a technique borrowed by machine learning from the field of statistics. As a typical machine learning model, logistic regression is also chosen as one of traditional defect prediction model. we use function **glm** in R to the model, which is used to fit generalized linear models.

ManualDown ManualDown is a simple module size model coming from the idea that we could use module size in a target project to predict bugs' location. Project with larger module is thought to be more defect-prone. Since ManualDown do not need any data from the source projects to build the models, they are free of the challenges on different data metric sets of the source and target project data. In particular, they have small calculation demand and are easy to implement. Furthermore, previous studies show that module size has a intense confounding effect on the associations between code metrics and defect-proneness [2] [23] . We hence include ManualDown as a baseline model for following comparison.

3.3 Prediction setting

In order to decrease test error and estimate the accuracy of a predictive model in practice, we apply one round of 5-fold cross-validation for each test in our study. For example, when we test DNN performance on a data set, firstly we need to partition all the instances (or called module) in the data set into 5 complementary subsets, signed as **p1**, **p2**, **p3**, **p4**, and **p5**. In the first pass, **p1**, **p2**, **p3**, **p4** are chosen as training data set, and **p5** is chosen as testing data set. In the second pass, **p1**, **p2**, **p3**, **p5** are chosen as training data set, and **p4** is chosen as testing data set. The rest can be done in the same manner. Then we could get 5 groups of bug prediction. At last we apply evaluation measures on these predictions to obtain 5 performance indexes, which reflect classifier's performance and constitute box plots exhibited in section 4.

3.4 Evaluation measures

There are numerous evaluation measures in defect prediction field. We mainly consider effort-aware scenarios and select three typical evaluation measures to evaluate our prediction models' performance.

AUC (The area under ROC curve) In Machine Learning, we usually count on AUC for evaluation when it comes to a classification problem. AUC is the most common used evaluation measure for it is capable of measure any classification model's performance. AUC is the area under ROC curve, which is a probability curve plotted with TPR against the FPR (TPR is on y-axis and FPR is on the x-axis). ROC curve reflects classification model's performance variation at various thresholds settings. Hence AUC could fairly tell model's capability of distinguishing between classes. AUC is between 0 and 1 (inclusive). The closer AUC is to 1, the better the model performs to distinguish between instances with buggy and clean.

F1(F1-Score) F1 (F1 score) is the harmonic mean of precision and recall:

$$F_1 = (\frac{recall^{-1} + precision^{-1}}{2})^{-1} = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

By the way of harmonic mean, F1 combines precision and recall to give a comprehensive description of how much model is able to distinguish two classes. If you get a good F1 score, it means that your false positives and low false negatives is low, so you correctly identify real buggy instances. A F1 score is also between 0 and 1 (inclusive), and is considered perfect when its value is close to 1.

CE(Cost-Effectiveness) During defect prediction process, classifier always gives each file their probability of being buggy. In some cases, software practitioners may find there is not enough resources to have a whole-project inspection, so it's more likely for them to check those files with high defect- proneness and small module size. Therefore, we could rank files in descending order using their probability of being buggy. The effort-aware ranking effectiveness of classifiers is usually measured by cost-effectiveness (CE) curve, which is a common evaluation measure in the field of defect prediction. And it's widely used in prior work [13, 17]. In the plot of CE curve, x-axis is the cumulative percentage of LOC (number of lines of code) of all the instances in a file, while y-axis represents the cumulative percentage of buggy ones classified among corresponding instances. Postulate M as a prediction model, firstly we need to sort all the instances in the descending order of LOC, since we regard LOC indicate the predicted probability of an instance being buggy. Then we can draw M's CE curve plot by adding instances in order and calculating current cumulative percentage of LOC as the abscissa and cumulative percentage of buggy instances as the ordinate. Once we get CE curve plot, CE could be calculated by the following formula introduced by Arisholm et al. [1]:

$$CE_\pi = \frac{Area_\pi(M) - Area_\pi * (Random)}{Area_\pi(Optimal) - Area_\pi * (Random)}$$

Where $Area_\pi(M)$ is the area under the curve of model M (likewise to Optimal and Random) for a given π. π is a cut-off varying from 0 to 1, and it indicates the percentage of cumulative LOC we take into account. In optimal model, instances are ranked in descending order of their actual defect densities, while in random model, instances are ranked with no order and buggy ones are randomly distributed among them. A larger CE_π represents a better ranking effectiveness. In this work, we discuss CE_π at $\pi = 0.2$.

3.5 Analysis methods

In order to make the results in section 4 more convincing, we introduce two practical analysis methods to help us analyze in detail.

SK test(Scott-Knott Test) Scott-Knott (SK) test [18] is an analysis method used for group classifiers into statistically distinct ranks. When a plot appears before our eyes, it's hard to identify if there is significant distinction between classifiers, and SK test could help on it. The SK test recursively ranks the given classifiers through hierarchical clustering analysis. It clusters the given classifiers into two group based on evaluation indicators and recursively executes until there is no significant distinct group created [4] . The SK test used in classifiers' comparation can be found in prior works [4, 9, 14]. In this study, we use SK test in 95% confidence level.

Win/Tie/Loss Win/Tie/Loss result is another analysis method which is useful in performance comparation between different techniques and has been widely used in prior works [11, 15]. In our work, we will apply it on the comparation between combination and individuals. For individuals, through one round 5-fold test on each version of a project we could get $5n$ performance evaluation data (n is number of versions within the project). For combination, after one round 5-fold test on a merged file of a project, we could get 5 performance evaluation data. Then Wilcoxon signed-rank test [21] and Cliff's delta δ [6] is conducted to compare their performance. If test on individuals outperforms test on combination based on the Wilcoxon signed-rank test (p <0.05), and there is distinct difference between the two based on Cliff's delta δ ($\delta \geq 0.147$), we mark test in individuals as a 'Win'. In contrast, test in individuals is marked as 'Loss' if p <0.05 and $\delta \leq$ -0.147. Otherwise, the case is marked as a 'Tie'. In the last, we could know how many times test on individuals wins, Losses, or ties against test on combination. The Win/Tie/Loss result shows if one of them outperforms the other one actually in all conditions.

4 Results

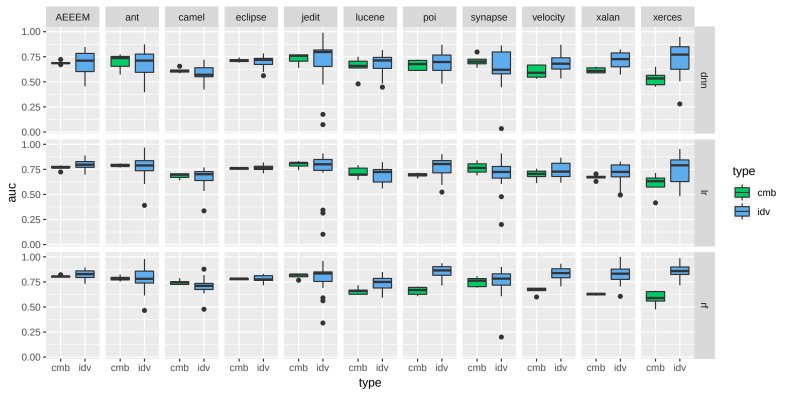
This section will provide a detailed description of our experimental results. We focus on how different defect predict models perform on combination and individual datasets, and answer the following research question:

4.1 *RQ1: Does improvement of the size of dataset promote the effect of defect prediction.?*

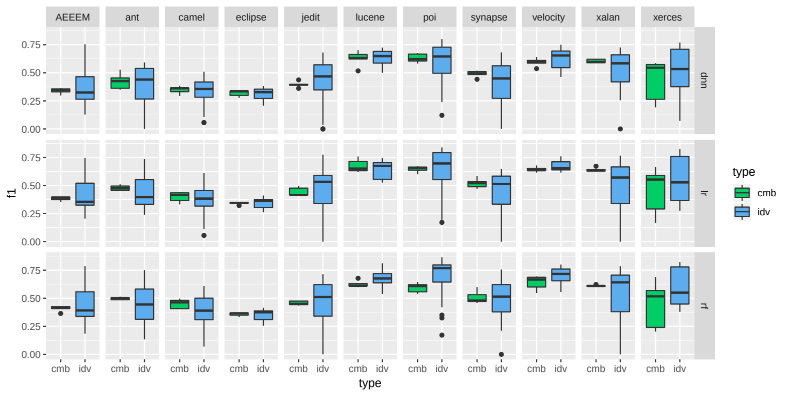
The answer we give is NO. Generally speaking, training on the combination of a project doesn't outperform the one on the individual versions evaluated by AUC, F1 and $CE_{0.2}$. And the result is convincingly supported by Win/Tie/Loss results.

Figure 1 shows an overview of the comparison between prediction on combinations and individuals. The boxplots show the distribution of evaluation measures (AUC, F1 and $CE_{0.2}$) of each classifier in the studied datasets. Green boxes represent the performances on merged files. Blue boxes represent the performances on individual files. Generally speaking, the results of the two don't exist significant discrepancy. But if you observe the plot more carefully, you could find that almost all the blue boxes are slightly higher than green boxes. Take the second plot in **Figure 1** as an example, except the performance on lucene with LR, all the blue boxes are higher than green boxes. This trend means that, surprisingly, the merge operation on the data set slightly reduces the performance of the classifier. This may be due to the hidden differences in different versions of a project.

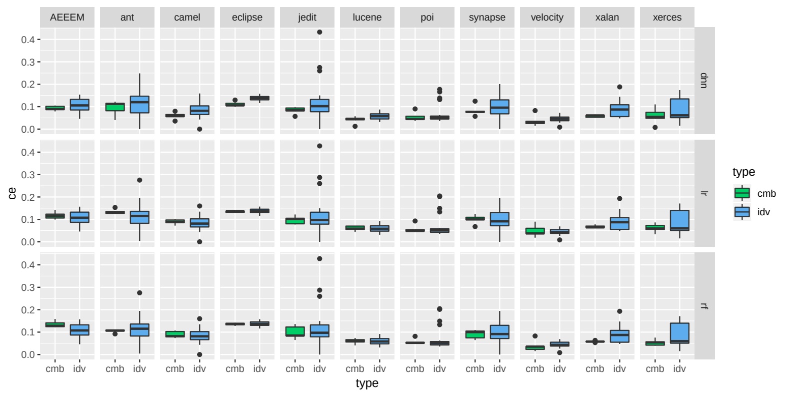
In order to have a more detailed observation, we also apply the Win/Tie/Loss indicator to help analysing. The Win/Tie/Loss result manifests whether combination is significantly better or not when compared with individual. **Table 3** displays the details of Win/Tie/Loss result - it gives the number of Win/Tie/Loss in



(a) AUC Performance.



(b) F1 Performance.



(c) $CE_{0.2}$ Performance.

Fig. 1. Comparision between combinations and individuals on AUC, F1 and $CE_{0.2}$

our test. Take the first row of table for illustration, it shows that performances of test on individuals wins 11 times, losses 1 time and ties 21 times against test on combinations. (11 merged files, 3 classifiers, so 33 competitions in total) This result manifests that our prior observation based on **Figure 1** is correct - individuals' performances slightly surpass the combination ones. In general, combination is not significantly better than individual on all three measure indexes. This result evinces the combination of software engineering dataset does not achieve distinct improvement.

Table 3. Information of Win/Tie/Loss indicator.

Index	win	loss	tie	relation	result
AUC	11	1	21	win + tie >loss	not significantly bad
F1	3	1	29	win + tie >loss	not significantly bad
$CE_{0.2}$	1	1	31	win + tie >loss	not significantly bad

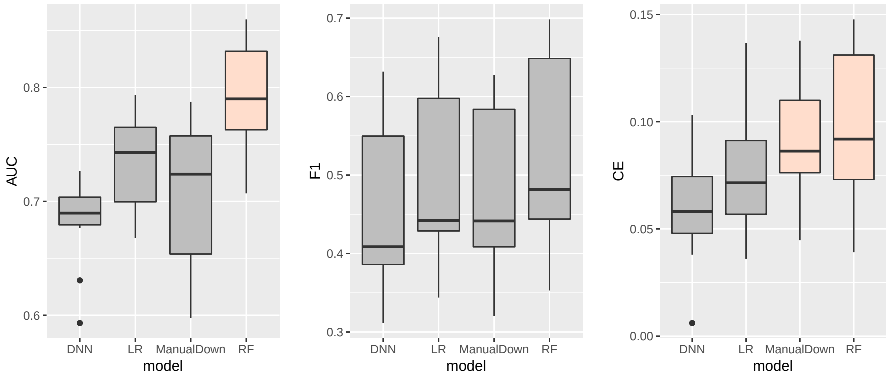
In summary, combination is not significantly better than individual on AUC, F1 and $CE_{0.2}$, which is supported by Win/Tie/Loss evaluation.

4.2 *RQ2: Does ManualDown outperforms other supervised techniques?*

In addition to differences between combinations and individuals, we also want to make a thorough inquiry on different performances of the four defect prediction model - DNN, RF, LR and ManualDown. Our question is if ManualDown could outperform other three typical supervised learning model, and it will be answered on both the merged and independent datasets.

First let's inspect the performance on independent datasets, and the three box plots in **Figure 2** show our effort. The models from left to right on the x-axis are in turn DNN, LR, ManualDown and RF. Three pictures represent the performance on AUC, F1 and $CE_{0.2}$. Our approach is to calculate the evaluation measure values of individual version and put them into box plot, so we can clearly discern the discrepancy between different model.

To explain with more detail, taking picture 1 for example. **ant** is one of the projects in the study, and firstly, we have a 5-fold DNN training and test on each version of **ant** (**ant1.3, ant1.4, ant1.5, ant1.6, ant1.7**) get 5 mean AUC values out of it. Then we calculate the mean of these values from different versions, so we get a synthesis AUC value of **ant**. In our experience we have prepared 11 projects, therefore we could procure 11 synthesis AUC value, which constitute the leftmost box in the plot. Similarly we could complement this box plot by using other models to have 5-fold training and test on the same dataset.



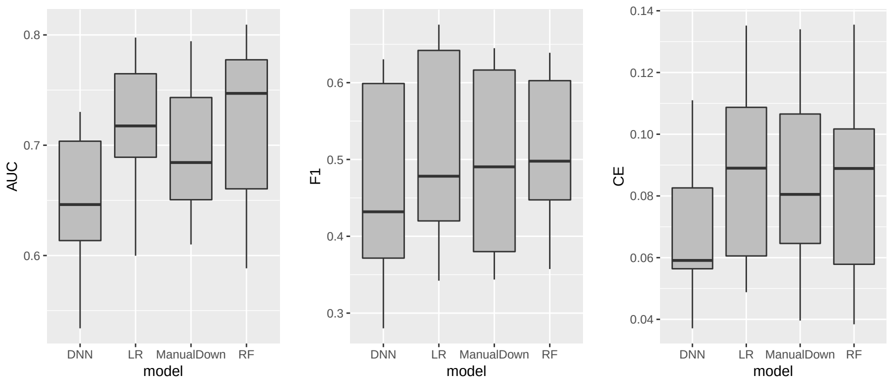
(a) AUC Performance. (b) F1 Performance. (c) $CE_{0.2}$ Performance.

Fig. 2. Comparision between classifiers on individuals

Besides, Scott-Knott (SK) test is also applied here to depict if distinct difference exists among models. In **Figure 4**, clusters with high SK values have been tinted into carnation, while others are staying gray. From the three pictures we can tell that RF outperforms other models. And RF is significantly distinct from DNN, LR and ManualDown on AUC, and from DNN, LR on $CE_{0.2}$ through SK test. As a result, we could draw a conclusion that ManualDown doesn't outperform other supervised techniques on independent datasets.

Now let's pay our attention on the performs on merged datasets, and this is showed on **Figure 3**. Unlike the experience on independent datasets, taking picture 1 for example. Firstly, we combine all the instances in different versions of a project. Hence, we get 11 combined datasets (because we have 11 projects). Then we have a 5-fold DNN training and test on each combined dataset and calculate AUC value of it, which make up the leftmost box in the picture 1 plot. In a similar way it's easy to draw up the performance box plot on $CE_{0.2}$ and F1 on merged datasets.

From **Figure 3** it's conspicuous that there doesn't exist obvious difference among the performance on these defect predict model. On the other hand, along with the increase of the instance's number in one dataset, the performance of model which has week performance on independent datasets makes a measly progress. And this lead to differences between different models is less obvious. In summary, ManualDown doesn't outperform other supervised techniques on merged datasets as well, and ManualDown is not significantly worse than traditional supervised models. Considering the complexity of traditional models, ManualDown is undoubtedly a predict method with more practical significance based on us experience.



(a) AUC Performance. (b) F1 Performance. (c) $CE_{0.2}$ Performance.

Fig. 3. Comparision between classifiers on combinations

In conclusion, ManualDown is also comparable to traditional supervised models on bigger combination datasets.

5 Threats to validity

Project selection In this study, we select 15 open-source projects which have been used in prior works. These projects equip large enough data scale and regular metric information, which is conducive to train classifiers. Whereas limits still exist since these projects only come from 3 groups, thus there may be a different result if more diverse data is introduced in the experiment. So, it's necessary to replicate our study in the future with a wider variety of datasets.

Classifier selection The classifiers we select in this work are DNN, RF and LR. Although these are common investigated in defect prediction literatures, they cannot represent all the classifiers. There would be a chance to get a better prediction performance if some newly proposed effective techniques are used during model training. Replication studies using different classifiers may prove fruitful.

Study replication DNN and ManualDown are implemented in Python. LR and RF are implemented using R packages. The three evaluation measures are implemented in Python. All these open source implementations and datasets can be accessed online at https://github.com/NJUaaron/2019DataTest.

6 Conclusions

Accurate software defect prediction plays an important role in the software industry to alleviate burden of software engineers. Many supervised and unsupervised methods have been proposed in prior works and are proved effective in the literature. However, for supervised methods, there is a limiting factor of defect prediction - the size of data used for training is not big, which restricts the scope of application with defect prediction models in practice.

In this study, we construct bigger defect datasets by merging available datasets with same measurement dimension and check whether the promotion in data size will lift defect prediction performance or not. Meanwhile, ManualDown, one of simple module size models, is introduced as a baseline model to measure supervised models performances.

In the experience, we test DNN, RF and LR on individual files and merged files. Their prediction performances are evaluated by AUC, F1 and $CE_{0.2}$, and the experimental results are analyzed by SK test and Win/Tie/Loss technique. In summary, our conclusions are as follow:

– Performance on larger-scale dataset is not significantly better than performance on raw smaller dataset under AUC, F1 and $CE_{0.2}$. More precisely, the increase in the size of dataset even makes the classifier perform worse, although the degree of deterioration is not distinct.
– There is not significant difference between performances of ManualDown and other supervised techniques. In other words, classical supervised models cannot outperform simple module size model on our merged bigger data.

In the future, we would like to add more advanced supervised models into account. Besides, we will optimize our dataset to make it more multifarious and convincing. In the meantime, we encourage future works to apply newly proposed models to have a comparation against simple module size model in particular cases.

References

1. Arisholm, E., Briand, L.C., Johannessen, E.B.: A systematic and comprehensive investigation of methods to build and evaluate fault prediction models. Journal of Systems & Software **83**(1), 2–17 (2010)
2. Emam, K.E., Benlarbi, S., Goel, N., Rai, S.N.: The confounding effect of class size on the validity of object-oriented metrics. IEEE Transactions on Software Engineering Se **27**(7), 630–650 (1999)
3. Erlikh, L.: Leveraging legacy system dollars for e-business (2000)
4. Ghotra, B., Mcintosh, S., Hassan, A.E.: Revisiting the impact of classification techniques on the performance of defect prediction models. In: International Conference on Software Engineering (2015)
5. Ibrahim, D.R., Ghnemat, R., Hudaib, A.: Software defect prediction using feature selection and random forest algorithm. In: International Conference on New Trends in Computing Sciences (2017)

6. J. Romano, J.D.K., Coraggio, J.: Exploring methods for evaluating group differences on the nsse and other surveys: Are the t-test and cohen's d indices the most appropriate choices? (2006)
7. Jiang, T., Tan, L., Kim, S.: Personalized defect prediction. In: IEEE/ACM International Conference on Automated Software Engineering (2014)
8. Jing, X.Y., Ying, S., Zhang, Z.W., Wu, S.S., Liu, J.: Dictionary learning based software defect prediction (2014)
9. Khalid, H., Nagappan, M., Shihab, E., Hassan, A.E.: Prioritizing the devices to test your app on: a case study of android game apps. In: Acm Sigsoft International Symposium on Foundations of Software Engineering (2014)
10. Kim, S., Zimmermann, T., Jr, E.J.W., Zeller, A.: Predicting faults from cached history. In: International Conference on Software Engineering (2008)
11. Kocaguneli, E., Menzies, T., Keung, J., Cok, D., Madachy, R.: Active learning and effort estimation: Finding the essential content of software effort estimation data. IEEE Transactions on Software Engineering **39**(8), 1040–1053 (2013)
12. Lee, T., Nam, J., Han, D.G., Kim, S., In, H.P.: Micro interaction metrics for defect prediction (2011)
13. Ma, W., Lin, C., Yang, Y., Zhou, Y., Xu, B.: Empirical analysis of network measures for effort-aware fault-proneness prediction. Information & Software Technology **69**(C), 50–70 (2016)
14. Mittas, N., Angelis, L.: Ranking and clustering software cost estimation models through a multiple comparisons algorithm. IEEE Transactions on Software Engineering **39**(4), 537–551 (2013)
15. Nam, J., Fu, W., Kim, S., Menzies, T., Tan, L.: Heterogeneous defect prediction. IEEE Transactions on Software Engineering **PP**(99), 1–1 (2015)
16. Pushphavathi, T.P., Suma, V., Ramaswamy, V.: A novel method for software defect prediction: Hybrid of fcm and random forest. In: International Conference on Electronics & Communication Systems (2014)
17. Rahman, F., Devanbu, P.: How, and why, process metrics are better. In: International Conference on Software Engineering (2013)
18. Scott, A.J., Knott, M.: A cluster analysis method for grouping means in the analysis of variance. Biometrics **30**(3), 507–512 (1974)
19. Wang, J., Shen, B., Chen, Y.: Compressed c4.5 models for software defect prediction. In: International Conference on Quality Software (2012)
20. Wang, S., Liu, T., Tan, L.: Automatically learning semantic features for defect prediction (2016)
21. Wilcoxon, F.: Individual comparisons of grouped data by ranking methods. Journal of Economic Entomology **39**(6), 269 (1946)
22. Yang, Y., Zhou, Y., Liu, J., Zhao, Y., Lu, H., Xu, L., Xu, B., Leung, H.: Effort-aware just-in-time defect prediction: Simple unsupervised models could be better than supervised models. In: Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering. pp. 157–168. FSE 2016, ACM, New York, NY, USA (2016). https://doi.org/10.1145/2950290.2950353, http://doi.acm.org/10.1145/2950290.2950353
23. Zhou, Y., Xu, B., Leung, H., Chen, L.: An in-depth study of the potentially confounding effect of class size in fault prediction. Acm Transactions on Software Engineering & Methodology **23**(1), 1–51 (2014)
24. Zhou, Y., Yang, Y., Lu, H., Chen, L., Li, Y., Zhao, Y., Qian, J., Xu, B.: How far we have progressed in the journey? an examination of cross-project defect prediction. ACM Trans. Softw. Eng. Methodol. **27**(1), 1:1–1:51 (Apr 2018). https://doi.org/10.1145/3183339, http://doi.acm.org/10.1145/3183339

25. Zimmermann, T., Nagappan, N., Gall, H., Giger, E., Murphy, B.: Cross-project defect prediction a large scale experiment on data vs. domain vs. process. In: Proc Joint Meeting of the European Software Engineering Conference & the Acm Sigsoft Symposium on the Foundations of Software Engineering (2009)