The Effects of Learning Highly-Dense Components for Software Defect Prediction!

Hongyu Zhang
School of Software, Tsinghua University
Beijing 100084, China
hongyu@tsinghua.edu.cn

Tim Menzies, Adam Nelson
CS & EE, West Virginia University
Morgantown, WV, USA
tim@menzies.us,
rabituckman@gmail.com

ABSTRACT

[Abstract here.]

1. INTRODUCTION

2. THE PROBLEM OF IMBALANCED CLASSIFICATION

It is widely believed that some internal properties of software (e.g., metrics) have relationship with the external properties (e.g., defects). Many prediction models have been proposed based on software metrics. For example, Khoshgoftaar and Seliya (2004) performed an extensive study on NASA JM1 and KC2 datasets using 25 classification techniques with 21 static code metrics. They observed low prediction performance, and they did not see much improvement by using different classification techniques.

Menzies et al. (2007a) also performed defect predictions for five NASA projects using static code metrics. In their work, Probability of Detection (pd) and Probability of False Alarm (pf) are used to measure the accuracy of a defect prediction model. Their models generate the average results of pd = 71% and pf = 25%, when Naive Bayes classifier is used (with data log-transformed). Zhang and Zhang (2007) pointed out that Menzies' results are not satisfactory when precision is considered. They found that high pd and low pf don't necessarily lead to high precision. The reason is that the percentage of defective modules could be very small. The Zhangs' equation for Precision is defined as follows:

$$Precision = \frac{TP}{TP+FP} = \frac{1}{1+\frac{FP}{TP}} = \frac{1}{1+\frac{NEG*PF}{POS*PD}} \quad (1)$$

, where NEG is the number of negative instances and POS the number of positive instances. From the Equation (1), we can see that even if pd is high and pf is low, the Precision would be low if the number of negative instances (NEG) is much more than the number of positive instances (POS).

In the NASA datasets, the percentage of defective modules in each NASA project (except the KC4 dataset) is very low, ranging from 0.41% to 12.21%. In average the NASA datasets have only 4.4% of defective modules. Therefore the Precision is low even the pd is high and pf is low.

In general, the presence of imbalanced class distribution makes classification learning difficult, leading to low accuracy of software defect prediction (measured in terms of Precision). In addition, the size and other complexity measures at fine granularity level (such as function/method level) are usually small, which makes it difficult for a machine learning technique to distinguish a small number of defective modules from a large number of non-defective modules.

3. THE DISTRIBUTION OF DEFECTS

Our subsequent work (Zhang 2008; 2009) show that in a large software system, the distribution of defects are skewed - that a small number of modules accounts for a large proportion of the defects. For example, in Eclipse 3.0, 20% of the largest packages are responsible for 60.34% of the pre-release defects (defects found six months before the release) and 63.49% of post-release defects (defects found six months after the release). At the file level, 20% of the largest Eclipse 3.0 files are responsible for 62.29% pre-release defects and 60.62% post-release defects (Zhang, 2009). Our results are consistent with those reported by other researchers such as Fenton and Ohlsson (2000) and Andersson and Runeson (2007). Furthermore, we find that the distribution of defects follows the Weibull function, which is one of the most widely used probability distributions in the reliability engineering discipline.

The skewed distribution of defects applied to NASA datasets too. We find that a few NASA modules (at the function/method level) have a large number of defects and a large number of modules have a few defects. As an example, Figure 1 shows the distribution of defects over CM1 and KC1 modules. All modules are ranked by the number of defects they are responsible for. Clearly the distributions are highly skewed ones - a few modules have many defects and most modules have 0 or 1 defect. We also calculate the cumulative percentage of defects over modules. We find that the top 5% "most defective" CM1 modules contain 68.57% defects, the top 10% "most defective" CM1 modules contain 100% defects. For KC1, the top 5% modules contain 55.81% defects and the top 10% modules contain 77.90% defects. We obtained similar results for other NASA projects.

We can also formally express the distribution of NASA defects across modules as the Weibull distribution. The CDF (cumulative density function) of the Weibull distribution can be formally defined as:

.

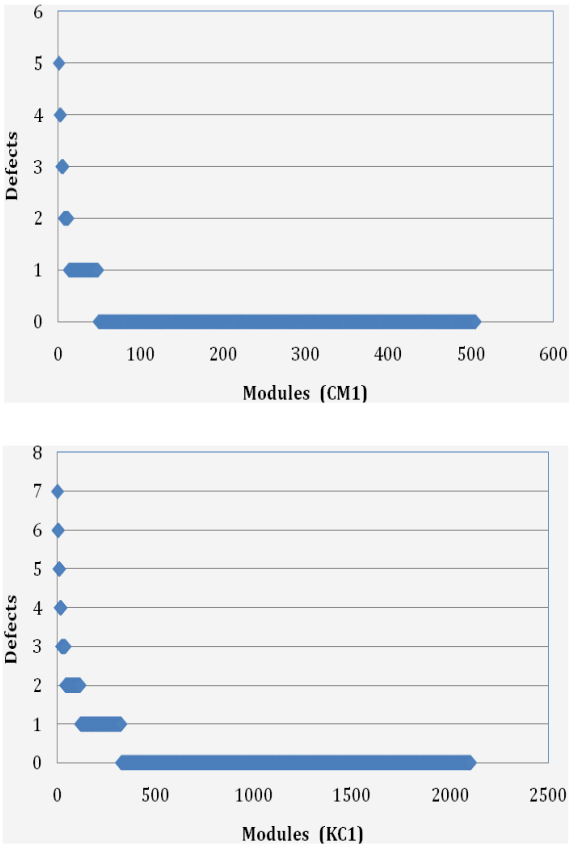

Figure 1: The Distribution of Defects in CM1 and KC1

$$P(x) = 1 - exp\left(-\left(\frac{x}{\gamma}\right)^{\beta}\right) \quad (2)$$

Using statistical packages such as SPSS, we are able to perform non-linear regression analysis and derive the parameters for each distribution. To statistically compare the goodness-of-fit of the Weibull distribution, we compute the coefficient of determination (R^2) and the Standard Error of Estimate (Se). Table 1 summarizes the Weibull parameters and the accuracy measures. The R^2 values ranging from 0.979 to 0.994 and Se values ranging from 0.009 to 0.025, confirming the Weibull distribution of defects.

Table 1: The Weibull distribution of NASA defects across modules

Project	γ	β	R^2	Se
JM1	0.076	0.965	0.979	0.025
KC1	0.067	1.313	0.994	0.013
KC3	0.030	0.950	0.987	0.009
PC4	0.049	0.960	0.986	0.016
PC5	0.011	1.045	0.994	0.006
MC1	0.011	1.736	0.991	0.007

4. THE EXPERIMENT

In order to test the implications of learning using components dense with software defects, an experiment was constructed using five NASA defect data sets (CM1, KC1, MC1, PC1, PC3). These data sets were chosen because they have been studied in the field extensively, and also that they are widely available to the PROMISE community. Five were chosen due to the limited number of data sets containing noteworthy numbers of components.

For each data set, components are extracted (using a unique identifier) containing both defective and non-defective modules (also labeled with a unique value). Thus, each component contains any number of modules having a defect. The count of defective modules per software component is used as follows: if the number of defective modules per component exceeds the median number of defects across *all* components in that data set, it is labeled as a defect-*dense* component. For example, in Figure 2 the bottom horizontal line represents the median number of defects in the KC1 data set. Thus, those components lying under this line are not used in further stages of the experiment, and so are denoted as *sparse* components. The pseudocode in Figure 3 illustrates the remaining setup of the experiment:

Lines 1 and 5 of Figure 3 illustrate the use of the 10 X 10-way cross validation used in the experimental process. The standard 10 X 10-way cross validation operates by selecting 90% of the data randomly for training, and the remaining 10% for testing. This process is then repeated 10 times for consistency. The experiment shown in Figure 3, however, handles this operation in a slightly different manner. Since the objective is to analyze the performance of training on modules in components containing a high number of defects compared to standard methods of training on all components, a minute alteration was made to the cross-validation of the experiment. A "pool" of training data was constructed by focusing on only those instances *within* a dense component, as in line 3 of the psuedocode. The available pool of testing instances, thus, are gathered from the *remaining* components in the data set (line 4). This is employed to prevent training and testing on modules within the same component. Lines 6 and 7 illustrate collecting 90% of the current dense component's instances as the final training set $Train'$, and 10% of the modules from the available instances in components *not* labeled dense as $Test'$.

While this represents a slight modification to the standard pratice of performing a cross-validation, it is within our engineering judgement to apply techniques that best mimick current methods in an area of experimentation still in its infancy. Thus, the recentness of this specific branch of research invites further techniques to be discovered and implemented.

Line 8 of Figure 3 executes the classifier (in this case, Naive Bayes) on the previously created training and testing sets $Train'$ and $Test'$. The Naive Bayes classifier was utilized because of its speed, and also for the fact that it has been shown to perform well on PROMISE defect data against other learners [1].

Determining a possible benefit of training our classifiers using fewer, but more densely-packed components also requires the comparitive analysis of learning on all components (and thus all modules). Comparisons are made between each approach, and the results are shown in the following section.

5. RESULTS

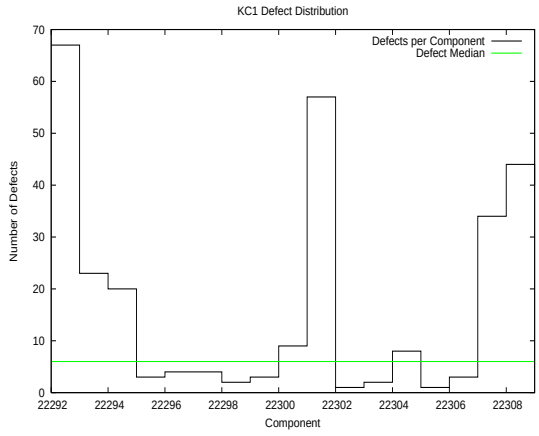

Figure 2: Defect distributions of components found in the KC1 data set. Note that only a small number of components contain a relatively high number of defects.

```
1 For run = 1 to 10
2   For each dense component C in data set D

3    Let Train = C
4    Let Test = All components in D except for C

5     For bin = 1 to 10
6      Train' = Randomly select 90% modules from Train
7      Test' = Randomly select 10% modules from Test

8     Naive Bayes (Train', Test')
9     end bin
10   end component
11 end run
```

Figure 3: Training on dense components versus all components. The experiment performs training on modules residing in dense components, and testing on modules contained in all other components in the data set.

The metrics used in the analysis of comparing results from training on dense components over the traditional method of using all components in the data set are *pd* (Probability of Detection), *pf* (Probability of False Alarm) and *precision*. If A,B,C, and D denote the true negatives, false negatives, false positives and true positives (respectively) found by a classifier, then:

$$pd = Recall = \frac{D}{(B+D)} \tag{3}$$

and

$$pf = \frac{C}{(A+C)} \tag{4}$$

and

$$precision = \frac{D}{(D+C)} \tag{5}$$

Therefore, *pd* and *precision* values are best if maximized, while *pf* results should be minimized.

Figure 5, Figure 6 and Figure 7 show statistical rankings of each treatment, as well as quartile charts displaying the median and variance of each metric for the *combined* data sets, as a whole, used in the experiment. The black circle in the center of each plot denotes the median value, and the line going from left to right from this circle shows the second and third quartile respectively. We prefer quartile charts of performance to other summarization methods for a multitude of studies, as they offer a very succinct summary of a large number of experiments.

It can be seen that training on components containing a higher number of defective modules maintains higher or tied ranks with the traditional method, and yields similar medians; while *precision* and *pd* medians lose 3% and 2% respectively, learning on dense areas provides much better *pf* medians – almost half.

Perhaps more interestingly are the analyses of data sets separately. Figure 4 demonstrates the outcome of each treatment for each data set independently. A "+" denotes a *win* for a particular treatment against the other, per data set. Conversely, a "-" indicates a *loss*. For example, the fourth row in the table of Figure 4 (data set PC1), shows that learning on dense components wins over learning on all for both *pd* and *prec*, but *loses* when considering *pf* scores. A win or a loss is assigned to a treatment by examining its statistical rank *as well* as its median value in comparison to the opposing treatment. If the treatments are statistically different, the method receiving the highest rank is given a "+" for that metric. If there is a tie in the ranks, the highest (or lowest, for *pf*) is used to determine the winner. The last row of the table represents the score of each treatment, given simply as the sum of "+"s for each treatment using the three metrics.

The results from this table demonstrate that learning on only components containing higher numbers of defective modules wins for every data set except for CM1. These results are significant for further defect prediction. By applying an instance filtering strategy by way of *component* selection, substantial increases can be made to the reliability of our quality predictors.

Data set	Measure	All Components	Dense Components
CM1	precision	+	-
	pd	+	-
	pf	-	+
KC1	precision	-	+
	pd	-	+
	pf	-	+
MC1	precision	-	+
	pd	-	+
	pf	-	+
PC1	precision	-	+
	pd	-	+
	pf	+	-
PC3	precision	-	+
	pd	+	-
	pf	-	+
	Score	4	11

Figure 4: Each treatment is assigned a "+" or "-" if it *won* over the other treatment, per metric, per data set. A "+" is assigned to a treatment winning a statistical ranking (based on a Mann-Whitney test at 95% confidence), or the best median per metric.

6. CONCLUSION

In this paper, we have learned that supplying our classifiers with training data selected from only components having a larger number of defective modules, we are benefitted because...

- defect prediction performance is improved significantly
- less data is required during the training phase, meaning faster runtimes and results
- insight is provided for component types; software organizations can make informed decisions about how to approach certain problematic components

...more

Rank	Treatment	pd percentiles 25%	50%	75%	2nd quartile median, 3rd quartile
1	Train on Dense Components	31	69	91	
2	Train on All Components	35	71	93	
					0 50 100

Figure 5: *PD* values for learning on dense components compared to learning on all components across all data sets, sorted by statistical ranking via a Mann-Whitney test at 95% confidence.

7. REFERENCES

[1] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch. Benchmarking classification models for software defect prediction: A proposed framework and novel findings. *IEEE Transactions on Software Engineering*, May 2008. Available from `http://iccle.googlecode.com/svn/trunk/share/pdf/lessmann08.pdf`.

Rank	Treatment	pf percentiles 25%	50%	75%	2nd quartile median, 3rd quartile
1	Train on Dense Components	0	15	52	
1	Train on All Components	0	26	65	
					0 50 100

Figure 6: *PF* values for learning on dense components compared to learning on all components across all data sets, sorted by statistical ranking via a Mann-Whitney test at 95% confidence.

Categories and Subject Descriptors

i.5 [**learning**]: machine learning; d.2.8 [**software engineering**]: product metrics

Keywords

algorithms,experimentation, measurement

Rank	Treatment	precision percentiles 25%	50%	75%	2nd quartile median, 3rd quartile
1	Train on All Components	20	78	95	
1	Train on Dense Components	12	75	96	
					0 50 100

Figure 7: *Precision* values for learning on dense components compared to learning on all components across all data sets, sorted by statistical ranking via a Mann-Whitney test at 95% confidence.