The Effects of Software Defect Prediction Using Highly-Dense Components

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. 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 by first determining both defective and non-defective modules contained in that data set. Once the modules are obtained, those components (named after a unique identifier) containing these modules are selected for further analyses.

After extracting the components and corresponding number of defects, components were retrieved for further analysis from each data set if the number of defective modules per component exceeded the median number of defects across *all* components in that data set. 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. The components selected (at or above the median number of defects) are considered *dense* components. The pseudocode in Figure 1 illustrates the remaining setup of the experiment:

```
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 1: 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.

Lines 1 and 5 of Figure 1 illustrate the use of the 10X10-way cross validation used in the experimental process. The standard 10X10-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 1, 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 while testing on all other components' modules, 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. This is employed to prevent training, and then 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 area of research invites further techniques to be discovered and implemented.

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

As the overall goal is to determine if training our classifiers using fewer, but more densely-packed components is advantageous to the usual practice of learning on a pool of all components (and thus all modules), comparisons are made between each approach, and the results are shown in the following section.

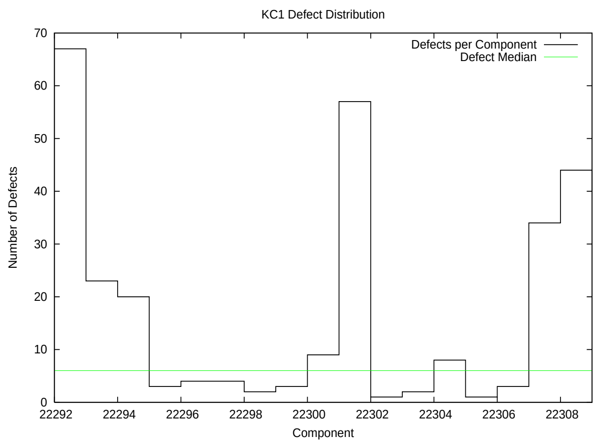


Figure 2: Defect distributions of components found in the KC1 data set. Note that only a small number of components contain a relatively high number of defects.

2. RESULTS

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:

$$pd = Recall = \frac{D}{(B+D)} \tag{1}$$

and

$$pf = \frac{C}{(A+C)} \tag{2}$$

and

$$precision = \frac{D}{(D+C)} \tag{3}$$

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

Figure 4, Figure 5 and Figure 6 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. Note 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 3 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.

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 3: 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.

Conversely, a "-" indicates a *loss*. For example, the fourth row in the table of Figure 3 (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 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 is beneficial 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

Categories and Subject Descriptors

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

Keywords

algorithms,experimentation, measurement

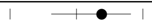
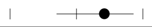
Rank	Treatment	pd percentiles 25%	50%	75%	2nd quartile median, 3rd quartile
1	Train on Dense Components	31	69	91	
2	Train on All Components	35	71	93	
					0 50 100

Figure 4: ***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.**

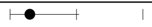
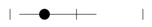
Rank	Treatment	pf percentiles 25%	50%	75%	2nd quartile median, 3rd quartile
1	Train on Dense Components	0	15	52	
1	Train on All Components	0	26	65	
					0 50 100

Figure 5: ***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.**

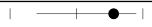
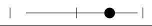
Rank	Treatment	precision percentiles 25%	50%	75%	2nd quartile median, 3rd quartile
1	Train on All Components	20	78	95	
1	Train on Dense Components	12	75	96	
					0 50 100

Figure 6: ***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.**