Preliminary Results of Defect Prediction Experiments

Kel Cecil
Graduate Student, Researcher
325 Pinnacle Height Drive
Morgantown, WV 26505
+1 (304) 952-8208
kelcecil@praisechaos.com

Jesse Taylor
Graduate Student, Researcher
106 Wedgewood Dr Apt 2
Morgantown, WV 26505
+1 (301) 471-0999
JesseTaylor@ieee.org

ABSTRACT
In this paper, we examine the conflicting opinions of two papers written by researchers Menzies et al and Zimmermann et all studying the ability to predict defective modules using cross-project data. We also discuss the effort to design experiments to prove or disprove the possibility of predicting defect modules using cross-project data.

Categories and Subject Descriptors
H.2.8 [**Database Applications**]: Data Mining

D.2.4 [**Software/Program Verification]**: Reliability, Validation

D.2.8 [**Metrics**]: Product Metrics

General Terms
Algorithms, Reliability, Verification

Keywords
Keywords are your own designated keywords.

1. INTRODUCTION
The goal of this project is to empirically prove or disprove the concept of cross-project defect prediction. We are starting with a simplistic set of data mining techniques and building more complicated methods as our theories evolve. This paper outlines the research effort to date, the algorithms currently implemented and in use, the observations at this stage of the project and concludes with areas that future work on this project is likely to explore.

2. BACKGROUND
Cross-company defect prediction is an unresolved issue in the field of Software Engineering. Few studies exist that have delved into this issue. More troubling, however, are the conflicting results of these studies. It is generally accepted that local or within company data is most ideal for producing software defect models. For companies that do not have sufficient defect data or do not track such data, it would be beneficial to be able to apply theories learned on other companies' defect data.

Two such studies [1,2] have been conducted to evaluate the applicability of such theories and made contradictory conclusions. Turhan & Menzies [1] reported that filtered cross-company data provided accurate defect prediction close to, but still less than, a model generated on local data. Zimmermann [2] showed that cross company defect prediction yielded a remarkably low accuracy rate of 3.4%. This discrepancy identifies a need for more testing and evaluation.

3. DATA
The below data sets are taken from software projects for NASA and SOFTLAB. The NASA projects are developed throughout North America and the SOFTLAB projects are developed in Turkey. The static code features in each project are not consistent among the other projects.

Table 1. Software projects used in this paper

Source	Project	Language	Description
NASA	pc1	C++	Flight software for earth orbiting satellite
NASA	kc1	C++	Storage management for ground data
NASA	kc2	C++	Storage management for ground data
NASA	cm1	C++	Spacecraft instrument
NASA	kc3	JAVA	Storage management for ground data
NASA	mw1	C++	A zero gravity experiment related to combustion
NASA	mc2	C++	Video guidance system
SOFTLAB	ar3	C	Embedded controller for white-goods

4. DISCUSSION OF ALGORITHMS
There are a number of important algorithms used in this project. These algorithms have been divided into categories for the purposes of discussion; preprocessors, discretizers, row reducers, feature subset selectors and classifiers. Each set represents a logical division within the learner process.

Conference'04, Month 1–2, 2004, City, State, Country.

4.1 PREPROCESSORS

4.1.1 nvalues

Since many of the datasets have the same column structure, it may be useful to combine several of these datasets to produce various combinations of the data set.

4.1.2 numericvalues

Numericvalues operates strictly on numeric fields in a data set. This function imposes a minimum value of 0.00001 for all numeric data and performs a log (base 10) on each value. This function aims to simultaneously reduce the range of values and spread them more evenly over the range.

4.1.3 bins

The bins algorithm is used for cross-examination of a data set. It takes a data set and splits it into 5 equally sized bins. The class distribution of the original set is preserved in each bin. Furthermore, each bin is split into a "train" set of 80% of the bin and a "test" set of the remaining 20%.

4.1.4 Nearest Neighbor

A nearest neighbor tool was implemented to help determine the closest neighbor instances in a given table. Given an instance, the nearest neighbor tool computes the Euclidean distance between the given instance and every instance in the target data set. A list sorted from shortest to largest distance is returned from the function. This tool serves as the foundation for building a k-nearest neighbor clustering algorithm later and is an important part for the Burak and Super-Burak filters.

4.2 DISCRETIZERS

4.2.1 Equal-width

Equal width discretization works by finding the minimum and maximum value in a numeric range. This range is divided into N bins, where N can be supplied, or calculated via bin logging. Bin logging calculates N using the N=max $(1, \log_2(uniqueValue))$. Unique values are the number of unique values in the range. All results gathered to date have used a default N value of 10.

An interesting observation about the equal width algorithm is that if most of the data is concentrated in one area, but there exists some outlier, the majority of the data may fall into one bin. For the purposes of discretization, such a result is not very useful and loses a level of granularity in the feature.

4.2.2 Equal-frequency

Equal frequency discretization sorts a field and attempts to split the instances equally over the bins. The only exception is that if a value is repeated when a new bin is supposed to be created, all repeating values are kept in the current bin and the first different value marks the start of the next bin. This is to prevent multiple bins that contain the same data.

4.3 ROW REDUCERS

4.3.1 Burak and Super-Burak

For several experiments, the Burak row reducer filter is used to remove instances of the data that could be considered an outlier. In order to use the Burak row reducer, the data must first be split into a training set and a testing set. The ten nearest neighbors in the training set for each instance in the testing set are found before discretization occurs using Euclidean distance as a distance measure.

$$d = \sqrt{a^2 + b^2 + \cdots + n^2}$$

Each of the ten nearest neighbors for each instance is pushed onto a new data set if the neighbor has not previously been added to the new data set. This new dataset is returned as the new training set. In many cases, this algorithm reduces the training set significantly. When Burak used on a random subset of the ar3 data set, the number of instances reduced from 63 instances to around 45 instances.

The Super-Burak filter is similar to the Burak filter in that rows that are considered outliers are removed from the train set. The Super-Burak filter combines any number of D data sets and follows the standard Burak method of removing columns.

4.3.2 Sub-Sampling

The sub-sampling algorithm used in this project finds the class that occurs the least in a data set. The number of times this class occurs is saved as N. A reduced data set is returned that includes only N instances of each class. This preprocessor prevents infrequent classes from getting drowned out by frequent classes.

4.3.3 Micro-Sampling

Similar to sub-sampling; the micro-sampling algorithm takes a user specified N and returns a data set comprising of N instances of each class. Assuming there are insufficient instances of a class to meet this N value, all available instances are taken.

4.4 FEATURE SUBSET SELECTORS

4.4.1 B-squared

The b-squared column reducer was implemented for the first phase of this project. The b-squared algorithm works as follows:

- Replace each class symbol with a utility score
- Sort instances by that score
- Divide the instances into 20% best and 80% rest
- Collect the frequencies of ranges in best & rest
- Sort each range by $b^2/(b+r)$ where
    - b = freq(r in best) / size of best
    - r = freq(r in rest) / size of rest
    - if b < k, b = 0 where k is some pre-determined cutoff value
- Sort each attribute by the median score of it's ranges
- Reject attributes (features) less than X% of the max median score

The goal of this fss (feature subset selector) algorithm is to identify reductions that occur in multiple data sets.

4.4.2 Manual Pruning

This method was added to supplement the information returned by the b-squared feature subset selector. It takes a data set and a list

of columns to keep. It was intended to be used after b-squared had been run on the data sets and common attributes could be identified that could be reduced from all data sets.

4.5 CLUSTERING

4.5.1 K-Nearest Neighbors

In phase 2 of our experiments, we began to utilize a clustering technique known as K-Nearest Neighbors to reduce the training set to the k instances local to the instance being tested. K-nearest neighbors is best used when a class is heavily represented in the space surrounding the target class. Applying a K-nearest neighbors approach disregards a large number of distant points that may sway the learner against the target class.

Our implementation of K-nearest neighbors works by finding the k nearest neighbors in the training set for each test instance. The distance for each numeric attribute is the test attribute less the train attribute. For discrete attributes, we assign a distance of 0 if the train and test attributes are the same value and a distance of 1 if they are different. Our distance is calculated using the Euclidean distance formula presented in the discussion of the Burak row reducer. The k training instances with the lowest distances from the test instance is used as a training set to classify the test instance using a Naïve Bayes classifier. This process is repeated for all test instances.

4.6 CLASSIFIERS

After preprocessing, we pass the newly created data sets to a classifier for testing. We record a matrix of four values for each class that is incremented whenever:

- The classifier correctly decided that the instance does not belong to this class (A),
- The classifier incorrectly decided that the instance does not belong to this class (B),
- The classifier incorrectly decided that the instance does not belong to this class (C),
- The classifier correctly decided that the instance does not belong to this class (D).

We use these values A, B, C, and D to calculate more meaningful statistics such as:

- $precision = D/(C + D)$
- $accuracy = (A + D)/(A + B + C + D)$
- $probability\ of\ detection(pd) = D/(B + D)$
- $probability\ of\ false\ detection(pf) = C/(A + C)$

These statistics help us determine the effectiveness of a combination of pre-processors, row-reducers, column-reducers, and learners.

4.6.1 Naïve Bayes

The Naïve Bayes classifier has been the most frequently used classifier in our experiments to date. Naïve Bayes has a strong assumption of independence, but the classifier works surprisingly well in real-world situations.

Naïve Bayes computes the posterior probability by multiplying the prior probability (the probability we have seen so far) by a likelihood function and dividing by the . This can be expressed as the following formula:

$$P(C_i|x) = \frac{P(C_i) \times P(x|C_i)}{P(x)}$$

For discrete data, this is as simple as counting the number of times an item occurred over the total number of instances. For numeric data, a Gaussian distribution is assumed, and the probability is computed using the probability density function of the Gaussian distribution. It is recommended to discretize data before classification, so the handling of numerics is only useful to establish a baseline to help determine the performance and improvement of a method.

5. EXPERIMENTS

5.1 Phase 1

Three initial experiments were conducted in the first phase of this project. We will discuss those experiments here while leaving the discussion of the results for the appropriate section.

We began by using the bins algorithm discussed earlier to attempt cross-validation of each set. The purpose of this experiment was to see if the results such as accuracy remained consistent across all ten bins. This experiment was repeated using the B-squared algorithm and the Burak filter.

Our next experiment was an attempt to find a size of the train data as a percentage of the full data set that would maximize the accuracy of the classifier. Random training sets from 10% to 95% of the full data set inclusively with 5% intervals were tested to try to find the percentage for each data set that maximized the accuracy of the classifier. This experiment was repeated using the Burak filter as well as the B-squared FSS algorithm.

The final experiment in the first phase was a modification to the scoring function used in the B-squared FSS algorithm. The scoring function originally provided equal weight to both true and false classes. This scoring function did very little to help sort the data into the required best and rest classes. On further consideration, it was decided that the TRUE class may benefit from having a separate score in an effort to bring out the attributes that characterize the TRUE class.

5.2 Phase 2

The second phase of this project attempted to develop some of the theories gathered in the first stage. Among the theories tested were:

- Discretization method effect on b-squared FSS
- Scoring patterns effect on b-squared performance
- Clustering (knn) effect on Naïve Bayes performance
- Manual reduction of columns effect on Naïve Bayes
- Micro-sampling values effect on Naïve Bayes

During phase 1, it was observed that the b-squared algorithm potentially could favor columns that had a very large range and a very small standard deviation. This was an outcome that was resulting from equal width discretization. To counter this effect, we implemented equal frequency discretization which places an equal number of instances into each bin. In phase 2, we analyzed

the learner performance with various combinations of equal width and equal frequency discretization.

In a two class system, the b-squared FSS has to assign greater value to one of the two classes. We had initially given the target class, TRUE, a higher value. The b-squared algorithm will place higher scoring instances in the "Best" set and thus it is likely that the higher scoring classes will have higher pd/pf performance. In phase 2, we evaluated the effect of weighting the FALSE class more.

The b-squared FSS was returning a very small number of columns during our phase 1 tests. Some of these columns were consistent across data sets. We thought it might be interesting to manually pick a subset of the columns that could be help constant across all data sets and manually reduce them. We picked three columns from the NASA data sets to test this theory; LOCCODEANDCOMMENT, BRANCHCOUNT, and IVG. These three columns each appeared at least 3 times during testing of the b-squared FSS.

K-nn clustering was used in conjunction with the b-squared algorithm and sub-sampling. K-nn clustering was not used with the Burak filter. We decided that the K-nn algorithm was performing accomplishing the same goal as the Burak filter without running the filter.

The final theory tested in phase 2 of this project was the effect of varying micro-sampling values on learner performance. This test involved comparing the learner performance with micro-sampling of 50 to micro-sampling of 75 and 100. The goal of this test was to see if a smaller number of samples could be used and achieve comparable or better results to the full data set.

6. RESULTS

6.1 Phase 1

Our attempts to cross-validate the datasets showed that the Burak filter was most likely to improve the overall accuracy of the classifier. The filter tended to increase the probability of detection for the majority class (FALSE) while the probability of detection for the minority class (TRUE) suffered. In some cases, the probability decreased slightly while the decrease was much more significant in other cases. The statistics generated through cross-validation are consistent through each train and test set.

Our experiment using random subsets to find a percentage of data that maximizes the accuracy of the data sets was met with results that we do not entirely understand at this time.

Lastly, the b-squared FSS returned a very small subset of features for some of the data sets. In some cases, only one column was returned other than the class. This could possibly be attributed to the equal width discretizer algorithm which may inflate feature scores in cases where the majority of the data falls in very small number of bins. In the cases of datasets cm1 and kc1 where a handful of columns are left remaining, the probability of detection for the majority class was close to equal the same probability of detection without the use of b-squared. In the case of the minority class, b-squared can significantly decrease the probability of detection and increase the probably of a false detection. At this time, we cannot say that each dataset would have this same effect without a modification to the algorithm that would require a minimum number of columns to be returned.

6.2 Phase 2

We observed that the discretization method had very little effect on the performance of b-squared, and subsequently, the learner. This is in line with the widely held belief that discretization methods have very little impact on learner performance.

Data was collected using b-squared with both a TRUE dominant and a FALSE dominant scoring scheme. After comparing results for both cases, we observed that changing the scoring scheme did swap the class performance for TRUE/FALSE. The b-squared performance was still poor across most datasets, however, so there is no clear benefit in this change.

Manually reducing each of the NASA datasets to three columns showed that the performance did not drastically change across most data sets. This is an important observation because it means that we can achieve comparable learner performance with significantly less data. For example, this used three columns as opposed to nineteen. The three columns, and their selection, are explained above in the Experiments section.

The micro-sampling results were unfortunately too varied and inconsistent to draw conclusions from. There was, however, no clear evidence to support the notion that similar performance could be obtained with significantly less data.

Initial tests with cross-company data with manually reduced columns show promising results. Due to unanticipated bugs in our cross-validation code, we were unable to do an indepth analysis of the datasets, but the results so far suggest that an indepth analysis will yield positive results.

7. CONCLUSIONS AND FUTURE WORK

Our efforts have shown that B-squared does not improve performance on datasets. Reducing rows using Burak resulted in slight performance increases. To date, K-nn with reduced columns has provided the best performance on classification. We will work further with K-NN classifiers. We will keep track of the frequency of which members of the train set are selected and use this information to create a train set.

We have done initial within-company tests after establishing baselines for performance of combinations of row-reducers, clusters, and column and row reducers on the same data set. Since we have established the foundation for comparing cross-company data sets, we will begin to use the best performing combinations on cross-company datasets.

8. REFERENCES

[1] B. Turhan, T. Menzies, A. B. Bener, and J. D. Stefano, "On the relative value of cross-company and within-company data for defect prediction " *Empirical Software Engineering*, DOI: 10.1007/s10664-008-9103-7, 2009.

[2] T. Zimmermann and N. Nagappan and H. Gall and E. Giger and B. Murphy. "Cross-project Defect Prediction: A Large Scale Experiment on Data vs. Domain vs. Process" *FSE'09* 2009.