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

The preprocessor division encompasses the following algorithms: nvalues, numericvalues, bins, knn, Burak, and sub-sampling. Each of these algorithms will be discussed separately.

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 10 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 90% of the bin and a "test" set of the remaining 10%.

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.

4.1.5 DISCRETIZERS

To date, only one discretizer has been implemented in this project and it uses the unsupervised equal width approach. 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 ROW REDUCERS

4.2.1 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.

4.2.2 Subsampling

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 FEATURE SUBSET SELECTORS

4.3.1 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 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.

4.4.1 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.

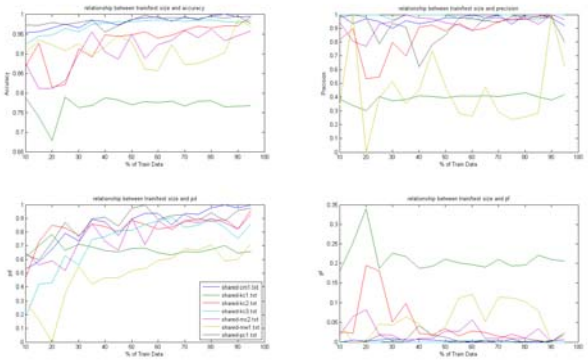

Figure 1 - Comparison of size of train sets test sets using Burak and Naive Bayes

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:

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

4.4.2 Hyperpipes

Due to a newly discovered bug that only affects certain datasets, no experiments using hyperpipes using our combinations of pre-processors, column reducers and row-reducers were performed. However, since the algorithm has already been written, it is worth discussing the algorithm and the possible benefits of it's deployment for the next deadline.

Hyperpipes is a simple learner that provides good performance on sparse datasets. The hyperpipes algorithms steps through each instance of the training set to see the discrete values and numeric ranges and the class associated with each instance. The algorithm stores these values and ranges as being seen. To classify an instance, the algorithm compares the instance to the values for each class that has been seen. A count is incremented for each time a value that is common to the class and the instance is found. The instance is "most contained" in the class with the highest count at the end of the process and is classified to that class.

Although the data sets in this project are not sparse, hyperpipes may still be of use. There may be potential in replacing a discretized symbol from the table (i.e. the most frequently occurring symbol) with an unknown symbol.

5. EXPERIMENTS

Limited time was available for experiments, but we conducted three initial experiments. 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 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.

6. RESULTS

Although this project is still in the very early stages, a number of interesting observations were encountered. It is our hope to pursue these observations as means of developing (or abolishing) interesting theories about cross-company defect prediction.

Our attempts to cross-validate the datasets show that the Burak filter is most cases improve the overall accuracy of the classifier. The filter tends to increase the probability of detection for the majority class (FALSE) while the probability of detection for the minority class (TRUE) suffers (illustrated by figures 2 and 4 as well as figures 3 and 5). In some cases, the probability decreases slightly while the decrease is 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 returns a very small subset of features for some of the data sets. In some cases, only one column is 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 is 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.

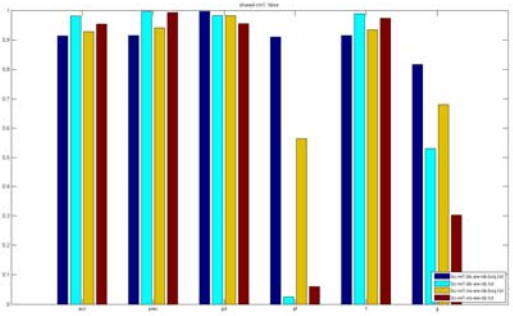

Figure 2 - Statistics from FALSE class for shared-cm1

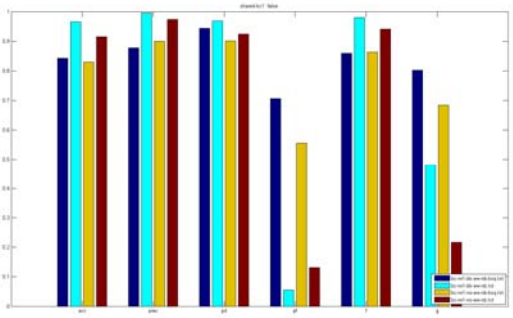

Figure 3 - Statistics from FALSE class for shared-kc1

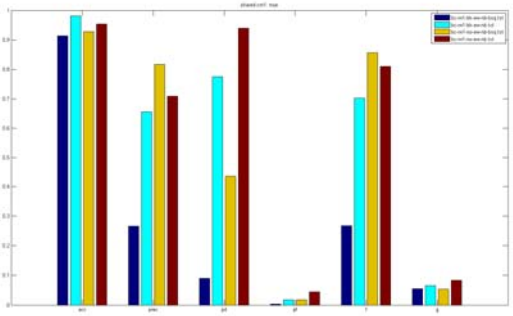

Figure 4 - Statistics from TRUE class for shared-cm1

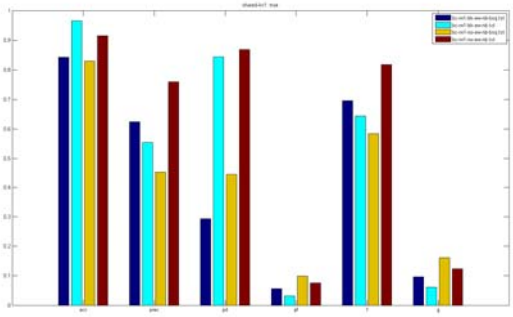

Figure 5 - Statistics from TRUE class for shared-kc1

7. CONCLUSIONS AND FUTURE WORK

The ultimate goal of FSS is to identify columns that can be removed across multiple, or all, data sets. In learner runs thus far, we have witnessed two cases of column reduction. For datasets cm1 and kc1, many columns are left in the table, while b-squared prunes the rest of the datasets leaving only the D attribute. The results datasets in which b-squared reduced to a single column is obvious in the high probably of false detection. Datasets kc1 and cm1 both displayed an intersection of five columns after b-squared was used to reduce columns. We would like to explore this phenomenon by implementing a manual column reducer and enforcing this column set across data sets. The goal of this manual pruning would be to see if column reduction patterns learned on a couple data sets can be applied successfully to other data sets.

The next phase of this project will hopefully explore a different discretization method due to a belief that equal-width is potentially skewing the column scoring algorithm. Equal frequency discretization is a likely candidate for this experiment.

The utility function used in the b-squared algorithm is based upon a belief that some classes are more important than others. In all of the data sets explored in this project, the class range consists of TRUE and FALSE only. Since this Boolean field indicates the presence of a defect, and it would be beneficial to identify defects, we chose to weight the TRUE class twice as heavily. This was merely an intuition and not based upon a mathematical basis. We would like to experiment with different scoring methods and observe the effect on learner performance.

No clusterer was used at this stage of the project due to timing constraints. It could prove a useful data mining technique in later phases of this project, so ideally we will be implementing one.

The classifier of choice in this project to date has been Naïve Bayes. A bug in our implementation of the hyper-pipes algorithm prevented us from running multiple classifiers. Since the entire project data is numeric, we feel it is worth fixing hyperpipes and comparing it to the previously observed performance of Naïve Bayes.

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.