*

General Theories of Software Defect Prediction: A Preliminary Report

William Mensah
WVU, Morgantown, WV
wmensah@csee.wvu.edu

Adam Nelson
WVU, Morgantown, WV
anelson8@csee.wvu.edu

Tomi Prifti
WVU, Morgantown, WV
tprifi@csee.wvu.edu

ABSTRACT

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

1. INTRODUCTION

By predicting defects in software systems *before* the deployment of that software, it is possible to gauge not only the probable quality upon delivery, but also the maintenance effort. Software defect prediction builds models using available company data that can then be applied in order to predict these software faults. But in order to employ these models, a company must have a data repository where information regarding defects from past projects are stored. However, according to Turhan, et. al. [7], few companies are applying this practice. Turhan, et. al. claims (and we agree), that this is most likely due to a lack of local data repositories. When this is the case, companies must use non-local data in order to build defect predictors. Thus, it is not only important to determine how well cross-company data can be used when local data is unavailable, but also to find the presence or absence of a general theory of software defect prediction. In other words, in determining the existence of an empirically-derived correlation between varying defect data sets, a statement can be made in regards to not only the stability of current defect prediction, but also the underlying similarities of cross-company software projects. On the other hand, if no correlation is found to exist, instability of those current predictors may suggest that further research should be conducted in order to provide incite into the variance between projects.

2. BACKGROUND

The ability of an organization being able to use cross-company (CC) data when within-company (WC) data is not available in order to build defect predictors would be advantageous. However, it remains unclear if this practice can yield beneficial results.

Turhan et al. conducted three experiments to rule in favor of CC data obtained from other sites, or WC data gathered locally. The conclusions of those experiments show that CC data, when applied using *relevancy filtering* via a k-nearest neighbor scheme. The idea behind the k-NN filter is simple; by building a training set that is homogeneous with the testing set, it is assumed that a bias in the model will be introduced. The filter works as follows: for each instance in the test set, the k nearest neighbors in the training set are chosen. Then, duplicates are removed and the remaining instances are used as the new training set. This relevancy filtering can lead to defect predictors almost as effective as WC data. Thus, as stated by Gay et. al. [5], "...while local data is the preferred option, it is feasible to use imported data provided that it is selected by a relevancy filter."

Gat et. al. confirmed Turhan et. al.'s results, but instead of implementing a nearest neighbor filter, a locally weighted scheme was used to filter the data via [4]. This experiment was of significance due to the fact that Gay et. al.'s results showed not only that CC data can be used when local data is not available, but also that publicly available data such as the PROMISE [1] data.

On the other hand, [1] shows that CC data cannot be used to build accurate defect predictors. For example, in one experiment conducted by Zimmerman et. al., Firefox and Internet Explorer were used due to their domain relationship (browsers) in order to determine how well one could predict for the other. It was found that while Firefox could predict for Internet Explorer at a precision of 76.47%, Firefox could *not* predict for Internet Explorer approaching the same precision (4.12%). However, this experiment did not utilize any form of relevancy filtering, so it is unknown how the two data sets would react to predicting for one another under these circumstances.

If CC data, when filtered, can be used in order to predict defects, we are left to assume that this is due to the fact that the data sets share some innate similarities of their metrics. But if it is found conclusively that CC data cannot most generally build good defect predictors, we are to conclude that other measures must be taken in either the collection of the WC data, or for more research in the further filtering

*A full version of this paper is available as *Author's Guide to Preparing ACM SIG Proceedings Using LaTeX2ε and BibTeX* at `www.acm.org/eaddress.htm`

[1] http://promisedata.org/

of CC data.

3. FINDING GENERALITY

[This section describes what generality is and how we went about finding it]

4. THE EXPERIMENT

4.1 Preparing the Data

We employed a number of preprocessing techniques to prepare the original data before applying our learner to it. These include:

- Logging
  Replacing all number values N with log(N) when N > 0.0001 or with 0.0001 if otherwise.
- Discretization
  Splitting each data set randomly into *10* bins (by default)
- Feature Subset Selection
  Sample bias via *B-Square* whereby each column in the data set is ranked by some criteria and the top n columns are maintained and used for the experiment because they are more likely to have a stronger influence on the results.

Why are all these preprocessing techniques necessary? Why not use the data in its raw form? When it comes to data mining, real world data is considered by most software engineers as *dirty*. This is because the data could be incomplete in that it could be missing some attributes or attribute values or it could simply consist of only aggregated values. Moreover, the data could be inconsistent, and could also contain errors and outliers. Preprocessing transforms the data into a format that will be more easily and effectively processed.

4.1.1 Discretization techniques

Two discretization method were used for performing the experiments: Equal Interval Width (10 bins) and K-Means. Both methods fall under the unsupervized discretization methods class. Equal bin length is a global techinque which replaces each value by the identifier of the bin. The range of values is divided into k equally sized bins where k is a parameter manually supplied for the purpose of our experiment. It makes no use of the instance class and is thus an unsupervized discretization method. Equal Bin Length is a global discretization method since it produces a mesh over the entire n-dimensional instance space [2]. The k-means on the other hand is a local discretization tool which groups the data into a number of clusters based on some similarity measure. Local discretization methods are applied to local regions of the data.K-Means is grouping the rows in k different clusters using the Eucledian distance as a measure of similarity between the instances. At the beginning the centroids (means) for each cluster are picked at random and then the instances are assigned to the group with the closest centroids to that particular instance. K-Means will stop iteration through the instance set when the centroids for all the clusters do not change anymore or when a stopping criteria is reached. In both cases of discretization the data is preprocessed by taking the log of each numeric value. Since these two method do not take into consideration class lables there might be some loss of classification information as a result of grouping instances that are stronly related but belong to different classes.

4.1.2 Merging Cross-Company Data

[This section describes why we chose to merge the data rather than use them as individual datasets]

4.1.3 Feature Subset Selection

[This section describes why we chose B-Square for FSS as opposed to InfoGain as used in prior experiments]

4.1.4 Slice Generation

[This seciton describes the randomization procedure used and how the slices were split up into train and test sets]

4.2 Learners

4.2.1 PRISM

One of the learners we used in our experiment was *PRISM*. Its algorithm was first introduced by Cendrowska and its aim is to induce molecular classification rules directly from the training set.

Prism uses the '*take the first rule that fires*' conflict resolution strategy when the resulting rules are applied to unseen data. Simply speaking, it identifies a rule that covers many instances in a given class, separates out the covered instances and continues with the remainder of the data.

When working with the Prism learner, we first calculate the probability that class = i for each attribute / value pair. After that we select the pair with the largest probability and create a subset of the training set compromising all the instances with the selected attribute / value combination (for all classifications). We then repeat the previous steps for this subset until a subset is reached that contains only instances of class i. The induced rule is then the conjunction of all the attribute / value pairs selected. Finally, we remove all instances covered by this rule from the training set.

The pseudo-code for the Prism algorithm can be outlined as follows:

For each class C Initialize E to the instance set While E contains instances in class C Create a rule R with an empty left-hand side that predicts class C Until R is perfect (or there are no more attributes to use) do For each attribute A not mentioned in R, and each value v, Consider adding the condition A = v to the left-hand side of R Select A and v to maximize the accuracy p/t (break ties by choosing the condition with the largest p) Add A = v to R Remove the instances covered by R from E

Both the Turkish data and data from NASA comprised of True / False classes, and also had quite a number of attributes in common so we figured Prism was a good option to explore, especially when it comes to cross-project defect prediction. The effect of using Prism on our datasets is analysed and discussed in our *Results* section.

4.2.2 B-SQUARE

B-Square served as the sole irrelevant-column-pruner employed to assist us better model our learners while focusing on the most important attributes from the datasets. The pseudocode for B-Square is outlined as follows:

Replace each class symbol with a utility score: True = 1, False = 0.

Sort instances by that score and divide them using N=20%-chops.

Label all the rows in bin=1 (20%) "best" and the remaining (80%) "rest".

Collect the frequencies of ranges in best/rest.

Sort each range (r) by by br/(b+r) where

b = freq(r in Best) / SizeOfBest.

r = freq(r in Rest) / SizeOfRest.

Sort each attribute by the median score of b2/(b+r)

Reject attributes with less than X% of the maximum median score.

Whenever b < r, we simply set b = 0.

4.3 Experiment 1: Naive Bayes

[The bayesian experiement]

4.4 Experiment 2: Prism

[The prismisian experiment :)]

5. RESULTS

[Left table outline in here for reference]

Table 1: Top Features selected by InfoGain and B-Squared

Feat.	Attribute (Shared)	Attributes (Turkish)
1	LOCOMMENT	EXECUTABLE LOC
2	EV	UNIQUE-OPERATORS
3	V	TOTAL-OPERANDS
4	D	TOTAL-OPERATORS
5	I	HALSTEAD-LENGTH
6	E	HALSTEAD-VOLUME
7	B	HALSTEAD-DIFFICULTY
8	T	HALSTEAD-ERROR
9	LOCODE	BRANCH-COUNT
10	UNQ-OPND	DECISION-COUNT
11	TOTAL-OP	CONDITION-COUNT
12	TOTAL-OPND	CYCLOMATIC-COMPLEXITY

6. RELATED WORK

This project is inspired by the Cross-project Defect Prediction work conducted by Thomas Zimmerman et. al. who claimed that software defect prediction works well whenever there is sufficient data to train any models and that in the case where data is insufficient, cross-project defect prediction suffices[1]. In their experiment, they ran 622 cross-project predictions for 12 real-world applications including Microsoft's Internet Explorer and Mozilla Foundation's Firefox web browser, and their results indicated that simply using models from projects in the same domain or with the same process does not lead to accurate predictions. With respect to the experiments they conducted, they learned that Firefox could predict defects in Internet Explorer but not vice versa and they succumbed to the conclusion that this is so because Firefox has more files than Internet Explorer has binaries and that the probability of a software with more data is more likely to predict defects in software with relatively less amount of data or modules.

Their study led to conclusions that some attributes were less significant than others which is somewhat obvious but also lead to questions such as why defect-prediction is not transivitve. As in, with regards to the experiments they conducted, File system predicted defects in Printing and Printing predicted Clustering but File system did not predict Clustering.

7. CONCLUSIONS

8. FUTURE WORK

[Finding Generality by splitting each slice into train and test sets as opposed to setting aside 10% of the number of slices as the test set and training on the remainder] [Microsampling - pruning both rows and columns and not just the columns - Menzies' mapping curiousity can be answered here]

9. ACKNOWLEDGMENTS

10. REFERENCES

[1] Cross-project defect prediction: A large scale experiment on data vs. domain vs. process.

[2] J. Dougherty, R. Kohavi, and M. Sahami. Supervised and unsupervised discretization of continuous features. *Morgan Kaufmann*, 1995.

[3] N. E. Fenton and M. Neil. A critique of software defect prediction models. In *IEEE Transactions On Software Engineering*, pages 675–687, October 1999.

[4] E. Frank, M. Hall, and B. Pfahringer. Locally weighted naive bayes. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pages 249–256. Morgan Kaufmann, 2003.

[5] G. Gay, T. Menzies, and B. Cukic. How to build repeatable experiments. In *PROMISE '09: Proceedings of the 5th International Conference on Predictor Models in Software Engineering*, pages 1–9, New York, NY, USA, 2009. ACM.

[6] J. F. K. Marvin L., Brown. Data mining and the impact of missing data. In *Industrial Management Data Systms*, pages 611–621. MCB UP Ltd, October 2003.

[7] B. Turhan, T. Menzies, A. Bener, and J. Distefano. On the relative value of cross-company and within-company data for defect prediction. *Empirical Software Engineering*, November 2009.