*

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

This paper provides a sample of a LaTeX document which conforms, somewhat loosely, to the formatting guidelines for ACM SIG Proceedings. It is an *alternate* style which produces a *tighter-looking* paper and was designed in response to concerns expressed, by authors, over page-budgets. It complements the document *Author's (Alternate) Guide to Preparing ACM SIG Proceedings Using LaTeX2$_\epsilon$ and BibTeX*. This source file has been written with the intention of being compiled under LaTeX2$_\epsilon$ and BibTeX.

The developers have tried to include every imaginable sort of "bells and whistles", such as a subtitle, footnotes on title, subtitle and authors, as well as in the text, and every optional component (e.g. Acknowledgments, Additional Authors, Appendices), not to mention examples of equations, theorems, tables and figures.

To make best use of this sample document, run it through LaTeX and BibTeX, and compare this source code with the printed output produced by the dvi file. A compiled PDF version is available on the web page to help you with the 'look and feel'.

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

*A full version of this paper is available as *Author's Guide to Preparing ACM SIG Proceedings Using LaTeX2$_\epsilon$ and BibTeX* at `www.acm.org/eaddress.htm`

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 of CC data.

2.1 Defect Prediction Stability

3. THE EXPERIMENT

3.1 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.

- Binning
  Splitting each data set into *10* bins (by default) and then building train data from 90% of the data and 10% from what is left.

- Clustering
  Partitioning n observations from each data set into k clusters using *k-means* algorithm whereby each observation belongs to the cluster with the nearest mean.

- Feature Subset Selection
  Sample bias via *InfoGain* 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 and why is time wasted them before the learner is applied? 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, that is, be missing some attributes or attribute values or it could simply consist of only aggregated values. In addition to that, the data could be inconsistent, and could also contain errors and outliers.

[1]http://promisedata.org/

3.2 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 supplied by the user. 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. Since our main learning algorithm is Naive-Bayes, both discretization method greatly improve the performance of the learner. Naive-Bayes is used with both k-means clustering and equal interval width bins on each of the datasets.

4. RESULTS

Naive-Bayes learner was applied on all data sets. Three different combinations of preprocesing techniques and feature subset selections were tested on different datasets:

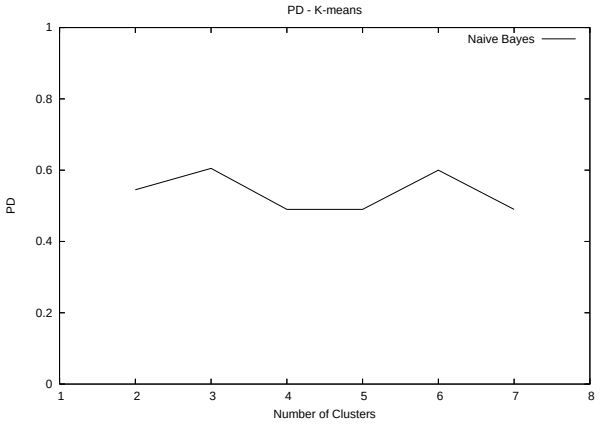
- Naive-Bayes without discretizing. K-means applied on the data
  In this experiment the data was supplied to the learner without performing any discretization. After splitting the data the train set was clustered using k-means. For each instance on the test data set the closest cluster was determined and the instance was classified using Naive-Bayes trained on that specific cluster.The performance of the learner was tested on different number of clusters k ranging from 2 to 7

- Naive-Bayes with discretization (10 bins). InfoGain applied on the data
  After discretizing the data InfoGain was applied on the whole data set to retrieve n number of features (columns) where n is a user specified value ranging from 2 to 14.

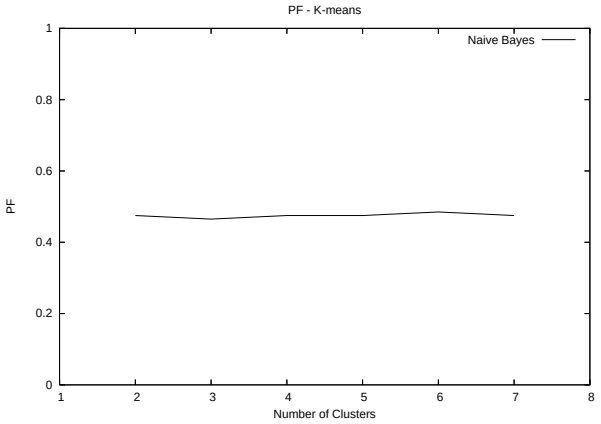
- Filter the features with InfoGain. Apply Naive-Bayes with clustering
  In this experiment InfoGain was applied to select the n best features. After selecting the best features a new dataset was build out of the original dataset containing only the selected features returned by InfoGain. The new dataset is clustered using K-Means. The learner

is tested using different combinations of n-number of features and k-number of clusters

Running the first learner, Naive-Bayes without discretization on different datasets shows clearly that discretiztion effect the performance of Naive-Byaes. Figure 1 and 2 show the PD PF values after running the learner with different k number of clusters.



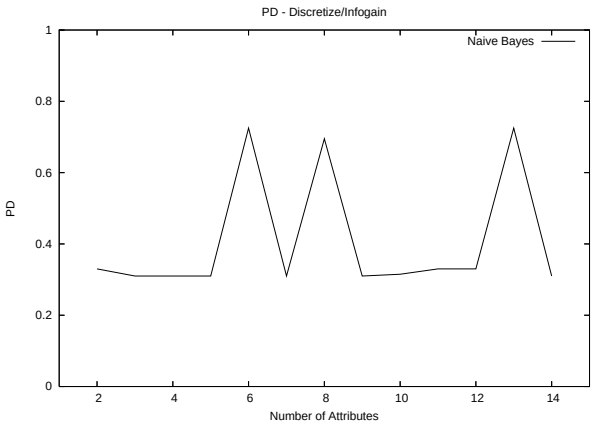
: Figure 1. PD: Naive-Bayes without discretization



: Figure 2. PF: Naive-Bayes without discretization

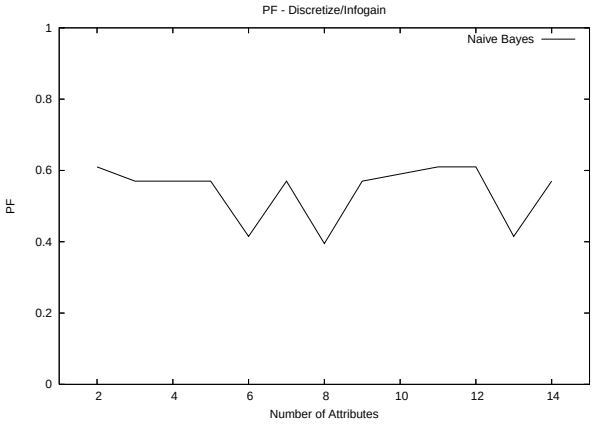
PD and PF are ploted agains the number of clusters k. Without discretization Naive-Bayes does not perform well. PD is no higher than 60 %. The number of clusters computed by k-means does not seem to influence the prediction probability however a relatively better PD are retrieved with 3 and 6 number of clusters. This results also corresponds with [2] which state that discretization influence Naive-Bayes performance.

Applying InfoGain on discretize data outperforms the first learner. Data is discretized and InfoGain is used to extract the best n features (columns) from the data. Than Naive-Bayes is applyed on the closest cluster of each instance in the train set.Figure 3 and 4 show the PD and PF of this learner.



: Figure 3. PD: InfoGain and Naive-Bayes

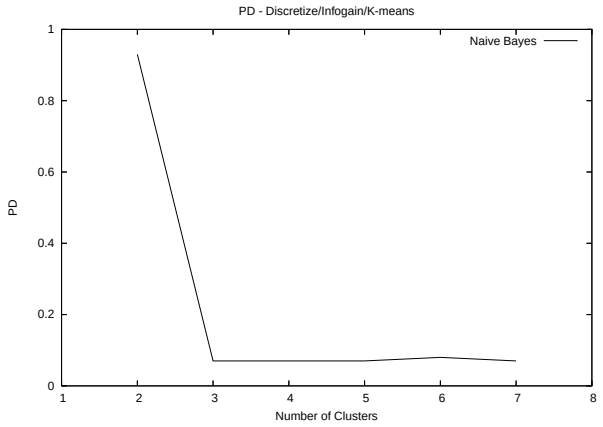
The second learner scores the highest values for PD when the number of attributes selected by InfoGain n is 6 8 and 13. The max PD values are close to 80 %. In this case the advantage of discretizing the data and applying feature subset selection gives better results than applying the clustering on undiscretized data.



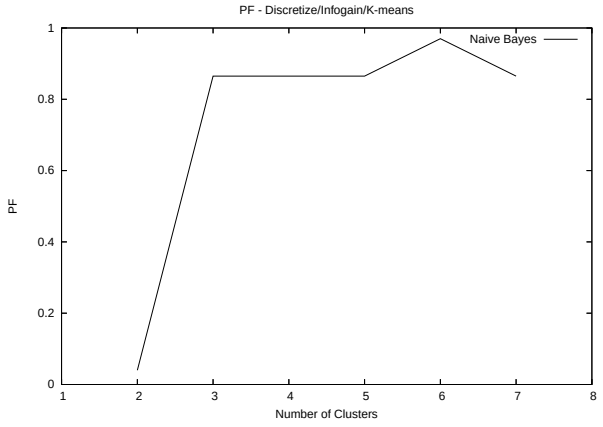
: Figure 4. PF InfoGain and Naive-Bayes

The third learner uses the information supplied by InfoGain to build a new data set containing only the features suggested by InfoGain. Discretization clustering is

performed on the data using k-means and Naive-Bayes is trained on each cluster. This learner is exploring a combination of n-features and k-clusters in order to find the optimal solution. Figures 5 and 6 shows the PD and PF for the third learner.



: Figure 5. PD InfoGain/Clustering and Naive-Bayes



: Figure 6. PF InfoGain/Clustering and Naive-Bayes

The figures show PD and PF plotted against k-number of clusters which varies from 2 to 7. The PD values decrease sharply when 3 or more clusters are used. The PD remain unchanged as the number of clusters increases. The PD values are relatively high (more than 90 %) when using less then 3 clusters. These high results should be confirmed by performing further testing as to verify the validity of this approach.

Table 1: Statistics for each method

	PD	PF	Balance
No-Discretization-NB	0.57	0.47	0.42
Discretize-Infogain-NB	0.86	0.09	0.21
Infogain-Clustering-NB	0.5	0.5	0.41

Table 2: Top Features selected by InfoGain

Feat.	Shared$_P C1$	Shared$_C N1$	Shared$_M W1$
1	LOComment	I	L
2	LOCCodeAndComment	Uniq-Op	Uniq Opnd
3	Uniq-Opnd	Uniq-Opnd	LOC
4	I	L	LOCommen
5	Uniq-Op	LOComment	LOCCodeAndCo
6	Total-Opnd	LOC	I
7	LOC	Total-Op	BranchCoun
8	V	Total-Opnd	V(G)
9	Total-Op	B	V
10	B	V	B
11	LOCode	IV(G)	EV(G)
12	BranchCount	D	LOCode
13	L	BranchCount	Total-Op

Table 1 . Shows the median PD and PF values for the above three approaches

Table 2. Shows the top 13 columns selected by InfoGain

5. 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.

6. CONCLUSIONS

The study undertaken and discussed in this paper involves investigating a criteria for predicting default-prone modules in both small and large-scale software systems. From the experiments performed on each of the data sets we obtained from NASA and SoftLab, we learned that within-company software defect prediction works when the data accumulated for prior versions of the software in question is available. The results from our self-test experiments validates this conclusion. In other words, creating train and test data from the same data set allows us to successfully build models for software defect prediction. This is especially feasible in cases where the company (whose datasets are being used in the experiment) has been around for years, for example NASA, Microsoft or the Mozilla Foundation; or older versions of its software have already been in use for years. The data generated from such older versions can be analyzed, learned on and used to build defect prediction models in order to predict defects in newer versions of the software.

Some researchers indicate that collecting data from case studies and subjecting it to isolated analysis is not enough because statistics on its own does not provide scientific explanations; and that "we need compelling and sophisticated theories that have the power to explain the empirical observations"[3]. However, in our experiments, we ran 10 tests on 10 datasets, each time with a different number of nearest-neighbors and number of columns to be selected via infogain, and despite the fact that these experiments were conducted on randomly selected objects from the different data sets and resulted in inconsistencies in the accuracy of defect predictions, the average probability of detection was relatively high while the probability of false alarm was relatively low. Of course, these values were obtained after intensive analysis on the results from applying learners on our datasets which we pre-processed using a number of machine learning / data mining methods as mentioned in section 3.1.

7. FUTURE WORK

The experiments for this project were conducted using data obtained from NASA and SoftLab for this project simply because they are 2 complete different companies that operate and function in different ways. The National Aeronautics and Space Administration (NASA) is an agency of the United States government that undertakes the nation's projects related to space exploration and SoftLab on the other hand is a research laboratory of computer engineering in Turkey that conducts research on cost and effort estimation, defect estimation/prevention, value based software engineering and software process development and typically writes software controllers for home appliances. It is worthy to note that these 2 companies mentioned have no ties with each other.

We believe that for better results and much better success rates at predicting software defects, regression algorithms could be applied to the datasets to model the data with the least error. That is, apply the learners to the datasets a number of times and discard arguments to algorithms such as k-nearest neighbor and infogain that germinated less desirable results. Also, the learning algorithms used could be altered to analyze additional data instances added to the data already learned on and determine whether the new data would be benefitial to predicting defects or not. Furthermore, this work could be extended to exploit datasets that comprised of more than 2 classes (TRUE, FALSE) and with multiple class columns (at least 2). Also, for any cross-company software defect prediction to be done, datasets from 2 or more different sources but with with similar columns will be required. In the case of our study, the data from SoftLab contained more metrics than that from NASA. This rendered it difficult to build models by learning on the 2 datasets thus experiment on only within-company.

Nonetheless, a number of other rule-based learners could be used rather than *NaiveBayes* as was used for this project. These include, but are not limited to, the *PRISM* algorithm which aims at inducing modular classification rules directly from the training set; *OneR*, *TwoR*, *RIPPER*, and even *HyperPipes*. Due to the constraint on time, we were unable to experiment on such learners but the outcome of applying each of them on the same datasets we used would be of great value to software engineers / data miners.

Finally, because all the data sets we used were complete, the alrgorithms used in our experiments didn't account for missing values but in future implementations, this will have to be addressed since ignoring this problem can introduce bias into the models being evaluated and lead to inaccurate data mining conclusions. This can easily be done by either ignoring the entire record, filling in with a global constant (not recommended since most data mining algorithms will regard it as a normal value), filling in with the attribute's mean or median, filling in with the most likely value (using regression, decisino trees, most similar records, etc) or using other attributes to predict the value of the missing data [6].

8. 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.