Can Data Transformation Help in the Detection of Fault-prone Modules?

Yue Jiang, Bojan Cukic,Tim Menzies
Lane Department of Computer Science and Electrical Engineering
West Virginia University
Morgantown,WV,USA
yue@csee.wvu.edu;bojan.cukic@mail.wvu.edu;tim@menzies.us

ABSTRACT

Data preprocessing (transformation) plays an important role in data mining and machine learning. In this study, we investigate the effect of four different preprocessing methods to fault-proneness prediction using nine datasets from NASA Metrics Data Programs (MDP) and ten classification algorithms. Our experiments indicate that log transformation rarely improves classification performance, but discretization affects the performance of many different algorithms. The impact of different transformations differs. Random forest algorithm, for example, performs better with original and log transformed data set. Boosting and NaiveBayes perform significantly better with discretized data. We conclude that no general benefit can be expected from data transformations. Instead, selected transformation techniques are recommended to boost the performance of specific classification algorithms.

1. INTRODUCTION

Data mining is the process of finding useful information from data sets. Data sets typically contain noise which affects the performance of classification algorithms. Hence, preprocessing (transformation) methods have been proposed to "prepare" the data set for mining. Normalization [7], linear and non-linear transformation [7], feature subset selection [9], principal component analysis [11], and discretization [6] are often conducted before prediction experiments. It has been suggested that transformation can improve the performance of software quality models [12]. This is the research hypothesis we want to investigate. In this study, we compare the impact of four different preprocessing methods on nine NASA MDP datasets across 10 different machine learning algorithms. The four preprocessing methods include the original data (i.e., no transformation, denoted by $none$), log transformed data (log), discretized data (nom), and discretization of the log transformed data ($log\&nom$).

2. EXPERIMENT SETUP

Cross-validation is the statistical practice of partitioning a sample of data into two subsets: training and testing subset. In 10×10 CV 90% of data is randomly assigned to the training subset and the remaining 10% of data is used for testing. The data is randomly divided into 10 fixed bins of equal size. We leave one bin to act as test data and the other 9 bins are used for training. This procedure is repeated 10 times.

Receiver Operating Characteristic (ROC) curves provide an intuitive way to compare the classification performance. An ROC curve is a plot of the Probability of Detection (pd) as a function of the Probability of False alarm (pf) across all the possible experimental threshold settings. AUC is a numeric performance evaluation measure directly associated with an ROC curve. It is very common to use AUC to compare the performance of different classifiers.

In all reported experiments, we use 10×10 cross validation to generate ROC curves for each classification algorithm, using each data set and four transformation methods. Therefore, the results reported in this paper represent $3,600$ fault prediction modeling experiments (10 classification algorithms $\times$ 9 data sets $\times$ 4 transformations $\times$ $10x$ cross validation). The Area Under the ROC curve, referred to as AUC, is used as the performance measurement to evaluate the performance of different models.

A box and whisker diagram (boxplot diagram) graphically depicts numerical data distributions using five first order statistics: the smallest observation, lower quartile (Q1), median, upper quartile (Q3), and the largest observation. The box is constructed based on the interquartile range (IQR) from Q1 to Q3. The line inside the box depicts the median which follows the central tendency. The whiskers indicate the smallest observation and the largest observation. In this study, we use Boxplot diagrams to visually depict the performance of different fault prediction modeling techniques.

2.1 Data sets and Transformation Methods

The nine data sets from Metrics Data Programs (MDP) [2] used in this study are listed in Table 1. The same table shortly describes their characteristics too. In each experiment, our models predict whether a module is fault-prone or not. We do not attempt to predict how many faults a module may contain. As mentioned earlier, we experiment with the following data transformation methods:

- **none:** This is the original data set without applying any preprocessing methods. All the independent variables (metrics) in MDP are continuous attributes.
- **log:** The original continuous variable values are transformed by taking mathematical log operation. To avoid numerical errors with $\ln(0)$, all numbers under 0.000001 are replaced with $\ln(0.000001)$. This transformation method is reported to be effective in [12].
- **nom:** The original continuous variable values are discretized to nominal values using Fayyad and Irani's minimum description length (MDL) method (Weka's default discretization method) [6, 14]. We discuss this method in detail below.

DEFECTS'08, July 20, 2008, Seattle, Washington, USA.
Copyright 2008 ACM 978-1-60558-051-7/08/07 ...$5.00.

Table 1: Datasets used in this study

data	# mod.	# faulty mod.	% faulty	notes	lang.
CM1	505	81	16.04%	Spacecraft instrument	C
KC1	2107	293	13.9%	Storage management for receiving/processing ground data	C++
KC3	458	29	6.3%	Storage management for ground data	Java
KC4	125	60	48%	A ground-based subscription server	Perl
PC1	1107	73	6.59%	Flight software from an earth orbiting satellite	C
PC3	1563	163	10.43%	Flight software for earth orbiting satellite	C
PC4	1458	178	12.21%	Flight software for earth orbiting satellite	C
MW1	403	27	6.7%	A zero gravity experiment related to combustion	C
MC2	161	52	32.30%	A video guidance system	C++

Table 2: The average number of distinct values for attributes.

dataset	none	Log	nom	log&nom	# attrib.
cm1	63.27	63.27	1.81	1.78	37
kc1	68.38	68.38	3.1	3.1	21
kc3	51.46	51.46	1.9	1.9	39
kc4	34.77	34.77	1.69	1.69	13
pc1	69.84	69.84	1.68	1.65	37
pc3	72.54	72.54	2.11	2.11	37
pc4	64.89	64.89	2.22	2.22	37
mw1	53.14	53.14	1.68	1.65	37
mc2	51.85	51.85	1.64	1.62	39
ave.	58.90	58.90	1.98	1.97	33

Table 3: Classification algorithms used in the study.

	learner	Abbrev.
1	Random Forest	rf
2	Bagging	bag
3	Logistic regression	lgi
4	Boosting	bst
5	Naivebayes	nb
6	Jrip	jrip
7	IBk	IBk
8	J48	j48
9	Decorate	dec
10	AODE	aode

- **log&nom:** The log-transformed data is discretized to nominal values using Fayyad and Irani's discretization method.

Fayyad and Irani is the recursive minimal entropy discretization method [8]. It is a supervised, non-parametric discretization method. It uses the class information entropy to partition bin boundaries. The stoping criterion is the minimum description length. For example, assume there is a dataset with S instances, a variable A, and a partition boundary T. There are two classes (S_1, S_2) for the dataset and the class entropy is denoted *E(A,T;S)*:
$E(A, T; S) = \frac{|S_1|}{|S|}E(S_1) + \frac{|S_2|}{|S|}E(S_2)$
The algorithm creates a discretization tree top down. The branch of the partition is recursively discretized and evaluated independently. Thus, in some branches, the continuous values with relatively high entropy will be partitioned very finely, while others that have relatively low entropy will be partitioned coarsely.

Table 2 shows the average number of distinct values for attributes in the MDP dataset. For example, in column $none$ 63.27 means that in CM1 data set, on average, attributes contain 63.27 distinct values. The average number of distinct values for $none$ and log are the same. The average number of distinct values for nom and $log\&nom$ are similar. The average number of distinct values for $none$ and log data are larger than that of nom and $log\&nom$.

Since the goal of this study is to compare the effects of data transformations across different classification algorithms, we decided to use machine learning and classification algorithms available in Weka [14]. These algorithms are often used in demonstrations of software quality modeling. Table 3 lists the machine learners compared in this study. Due to limited space, we omit their description, but interested readers will find it in [14]. All the machine learning algorithms were applied using their default parameters, except for random forest. In Weka, the size of the forrest of classification trees is set to 10, an insufficient number based on our prior experience [10]. We increased the default number of trees in the forest to 500, the value originally proposed as a default by Breiman [3]. 8 of the classifiers accept data from all 4 preprocessing methods. $Decorate$ only runs on *none* and *log* data sets, while AODE can only run on *nom* and *log&nom* data sets.

2.2 Statistical hypothesis tests

10×10 CV experiments result in 10 individual values of AUC. These 10 values are usually similar to each other, given that they come from the same population. But with 10 values it is difficult to say whether they follow the normal distribution (i.e., indicate that they obey the central limit theorem). Therefore, parametric statistical methods which assume a normally distributed population to compare the performance of classifiers may not be the right choice. A prudent approach calls for the use of nonparametric methods. The loss of efficiency caused by using nonparametric tests is typically marginal [4, 13].

In [5], Demsar recommends to use the Wilcoxon signed rank test for the comparison of two classifiers; the Friedman test for the comparison of more than two classifiers and the Nemenyi test of post-hoc comparison (rank ordering) among all classifiers over multiple data sets. The Wilcoxon signed rank test, the Friedman test, and the Nemenyi test are nonparametric counterparts for paired $t - test$, analysis of variance (ANOVA), and Tukey test parametric methods, respectively. The advantage of the recommended tests is that nonparametric procedures make less stringent demands on the data. However, two issues need attention. First, nonparametric tests do not utilize all the information available. The actual data values (in our case, for example, AUCs) are not used in the test procedure. Instead, the signs or ranks of the observations are used. The second point is that the Wilcoxon signed rank test is constructed for the null hypothesis that the difference of the performance measure is symmetrically distributed. For non-symmetric distributions, this test can lead to a wrong conclusion.

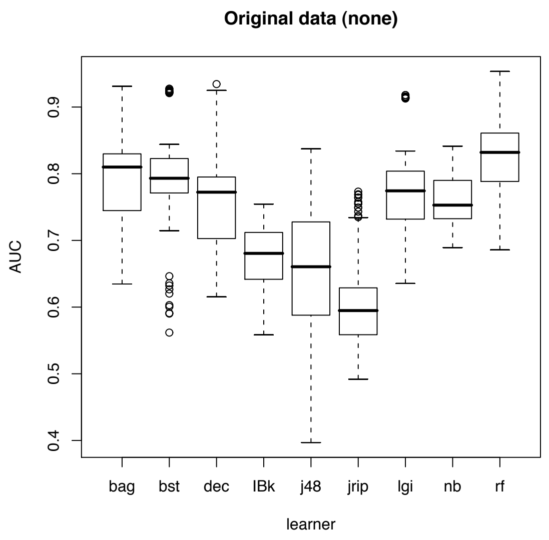

Figure 1: Boxplot diagrams with original data.

The Friedman test, the Nemenyi test and the Wilcoxon test are implemented in a statistical package R [1]. These implementations comply with the assumptions in [5]. Based on Demsar's recommendation, we use the Friedman test to evaluate whether there is a difference in the performance amongst the models developed by the 10 classifiers over the 9 data sets. Provided that the Friedman test indicates statistically significant difference, we use the Nemenyi test to determine which classifier(s) performs the best. The Wilcoxon signed rank test is used to compare the performance of the two specific models, if necessery.

3. EXPERIMENTAL RESULTS

In the following subsections, we show the test results of the Friedman test, the post-hoc Nemenyi test, and the Wilcoxon non-parametric tests applied to compare the classification performance of software engineering fault prediction models.

3.1 Original Data

Figure 1 shows the boxplot diagrams representing AUC values of different classifiers applied across all the data sets. In these experiments, the data sets were unchanged, i.e., no transformation function had been applied. The diagrams graphically depict numerical data distribution using five first order statistics: the smallest observation, lower quartile (Q1), median, upper quartile (Q3) and the largest observation for each classification algorithm.

When applied to the data from Figure 1 the Friedman test confirmed that there exists statistically significant difference in the performance of different fault prediction modeling techniques. The next step in our procedure was to apply the Nemenyi test. Figure 2 ranks the performance of the classifiers from the worst (the leftmost) to the best (the rightmost). CD stands for the critical difference of the Nemenyi test. If the difference between two classifiers is less than the value of CD, there is no significant difference in 95% confidence interval ($p\_value = 0.05$ in this case), otherwise, there is. The classifiers connected with a straight line in Figure 2 are those whose performance does not demonstrate statistically significant difference in 95% confidence interval. The following are further obesrvations from the application of the Nemenyi test:

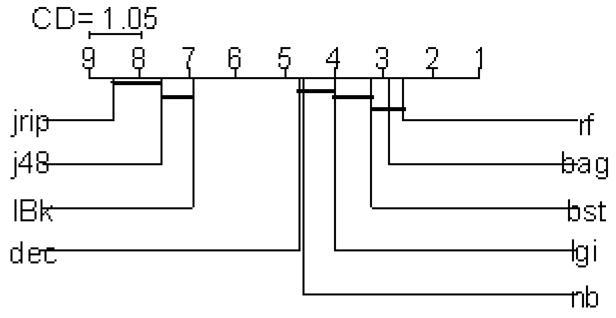

Figure 2: Comparison of all classifiers in the original data domain ($none$) with the Nemenyi test using 95% confidence interval.

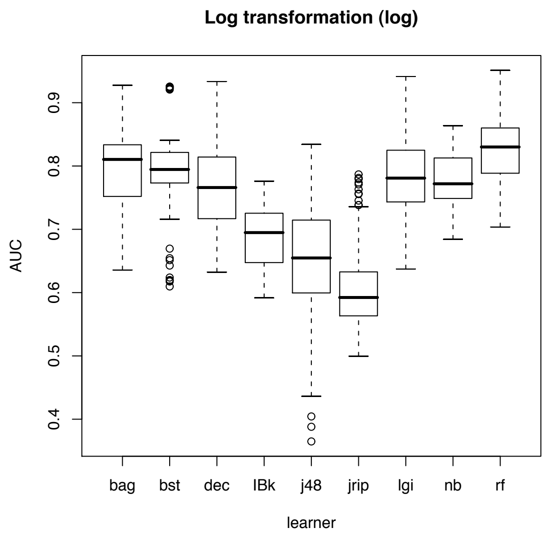

Figure 3: Boxplot diagrams with (log) data.

1. Classifier's performance trend, from the worst to the best, is: jrip, j48, IBk, decorate, Naivebayes, logistic regression, boosting, bagging, and random forest.
2. The classifier with the best performance is random forest. However, the performance of random forest, bagging, and boosting does not demonstrate statistically significant difference.
3. The classifier which offers the worst performance is jrip, but the performance of j48 is statistically indistinguishable.

3.2 Log transformation

Figures 3 and 4 indicate that that the classification performance with (log) transformation in place is similar to that of the original data set. According to our experiments, log transformation does not improve classification performance. To the contrary, the models typically suffer from the increase in variance.

3.3 Discretized data

When a data set is discretized, a large number of values that independent variables may assume is consolidated into a small number

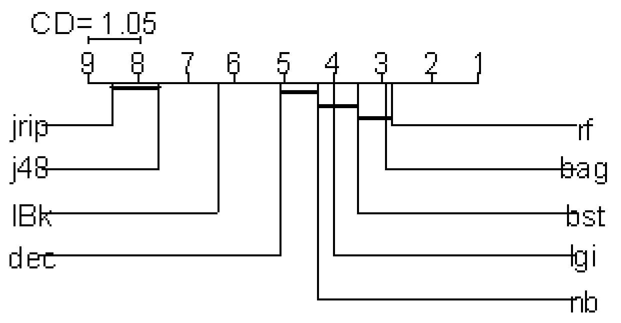

Figure 4: Comparison of all classifiers in the log data domain using the Nemenyi test

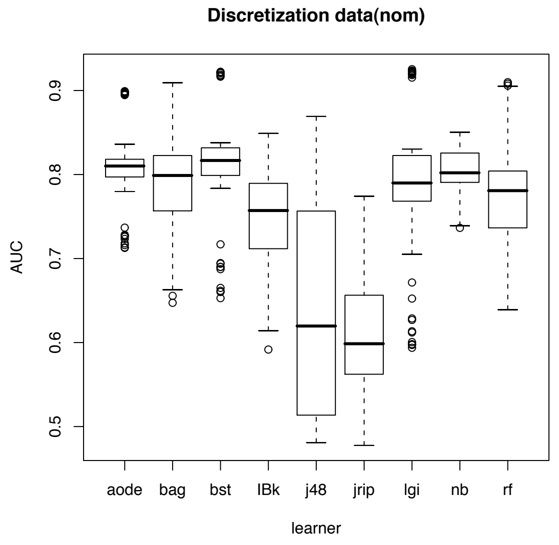

Figure 5: Boxplot with the discretized (nom) domain.

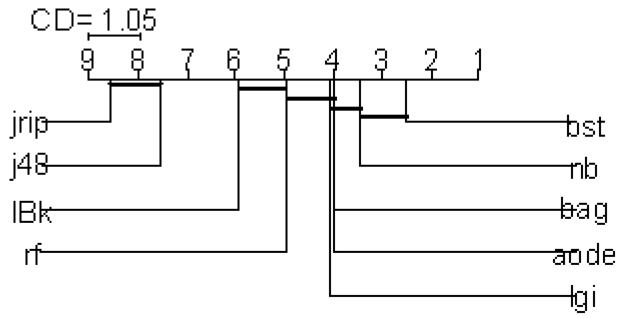

Figure 6: Comparison of all classifiers in the nom data domain with the Nemenyi test

of discrete classes. Such a transformation has an observable effect on the classification results. Figures 5 and 6 support the following observations:

- $Boosting$ offers the best performance on discretized data. Naivebayes is in a statistical performance tie with boosting (using the 95% confidence interval).
- Consistent with [6], Naivebayes performs much better on discretized data in the continuous domain.
- The performance of $randomforest$, the preferred classifier on the original dataset, decreases on discretized data.
- IBk, $j48$, and $jrip$ remain uncompetitive.

3.4 Discretized log data

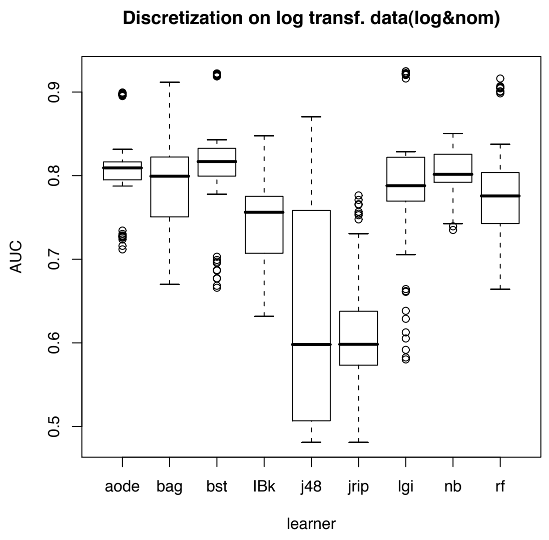

Figure 7: Boxplot in $log\&nom$.

Figures 7 and 8 report the results of statistical analysis of the classification performance of models that use $log\&nom$ transformed data sets. The remarks offered for the discretization transformation hold for $log\&nom$ transformation too.

3.5 Comparison between Random Forest and Boosting

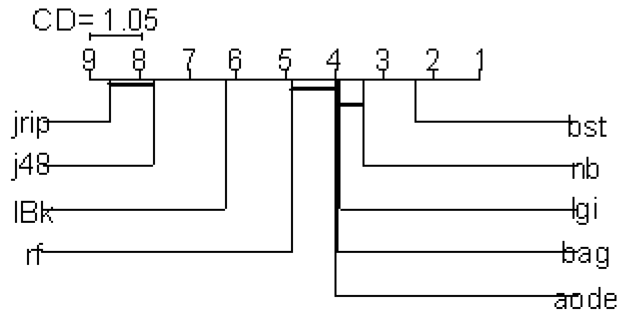

Figure 8: Comparison of all classifiers in the $log\&nom$ data domain with the Nemenyi test

When summarizing the results, we noticed that random forest models outperform others in $none$ and log domains, although they are in statistical tie with bagging and boosting. On the other hand, boosting appears to offer the best performance on nom and $log\&nom$ data sets. We performed the Wilcoxon nonparametric statistical test to compare the performance of these two classifiers in data domains where they offer their best performance: 1) random forest in $none$ and log data sets and 2) boosting in nom and $log\&nom$ data sets. The p value of the null hypothesis is $0.00000001510363 (< 0.05)$, suggesting a significant difference between these two models. The p value of the hypothesis of whether random forest is better than boosting is $0.000000007551813 (< 0.05)$ supporting the claim that the performance of random forest in $none$ and log is better than boosting in nom and $log\&nom$ domains.

3.6 Comparing transformation methods

Table 4: Classifier performance in the four transformation domains.

rf	none=log>nom=log&nom
bag	none=log>nom=log&nom
bst	none=log<nom=log&nom
IBk	none<log&nom; log<nom
nb	none=log<nom=log&nom
lgi	none<log=nom=log&nom
j48	none=log=log&nom=nom
jrip	none=log>nom=log&nom
dec	none=log
aode	nom=log&nom

We also analyzed how the choice of a data domain can guide the choice of the classification algorithm. Table 4 shows the comparison of classifiers based on the outcome of the Friedman test and its post-hoc Nemenyi nonparametric statistical test over the four preprocessing methods. Since $decorate$ and $aode$ only accepts the data from two preprocessing methods, we use the Wilcoxon nonparametric test for their comparison. Not surprisingly, classification algorithms have their “preferred" domains:

- $Boosting$, $NaiveBayes$, IBk, and $logistic$ achieve better performance on discretized data.
- $J48$ has the same (miserable) performance regardless of the transformation.
- $jrip$ has better performance in the original and log transformation domains.
- $Decorate$ performs similarly in the original and log transformation domains.
- $AODE$ offers the same performance in two discretized data domains.
- $Randomforest's$ performance is better in $none$ and log than in (nom and $log\&nom$) domains.
- $Logistic$ offers better performance in log and discretized transformation data domains.

4. CONCLUSIONS

We investigated the effect of four data transformation methods on the prediction of fault-prone modules in several NASA data sets. Our most important finding is that transformations *did not* improve overall classification performance measured by the area under the ROC curve (AUC). Random forest offered the best overall performance in the untransformed data domain. However, discretization improved classification performance of several classification algorithms. In summary:

- Random forest is reliably one of the best classification algorithms overall. Its performance decreased with discretization, thus confirming that its main strength lays in the analysis of noisy data, always the fact of life in software engineering experiments.
- Boosting offers the best models in the discretized data domain. It is also consistently one of the best models with all four preprocessing methods. But, boosting is also one of the most computationally expensive modeling techniques, which may play a role in its selection especially if training requires a large number of samples.
- The performance of NaiveBayes is greatly improved in the discretized domain. This observation confirms results from [6]. Arguably, NaiveBayes is the simplest of classification algorithms. If selected for software quality prediction, discretizing the data prior to its application is desirable.
- IBk, J48, and jrip are consistently the worst classification algorithms and including them in software quality modeling is likely a waste of time.
- Log transformation rarely affects the performance of predictive software quality models (logistic is the only exception).

5. REFERENCES

[1] The R Project for Statistical Computing, available `http://www.r-project.org/`.

[2] Metric data program. NASA Independent Verification and Validation facility, Available from `http://MDP.ivv.nasa.gov`.

[3] L. Breiman. Random forests. *Machine Learning*, 45:5–32, 2001.

[4] W. J. Conover. *Practical Nonparametric Statistics*. John Wiley and Sons, Inc., 1999.

[5] J. Demsar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7, 2006.

[6] J. Dougherty, R. Kohavi, and M. Sahami. Supervised and unsupervised discretization of continuous features. In *International Conference on Machine Learning*, pages 194–202, 1995.

[7] J. J. Faraway. *Practical Regression and Anova using R*. online, July 2002.

[8] U. M. Fayyad and K. B. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. pages 1022–1027, 1993.

[9] I. Guyon, S. Gunn, M. Nikravesh, and L. A. Zadeh. *Feature Extraction: Foundations and Applications*. Springer, 2006.

[10] Y. Jiang, B. Cukic, and T. Menzies. Fault prediction using early lifecycle data. pages 237–246. Software Reliability, 2007. ISSRE '07. The 18th IEEE International Symposium on, Nov. 2007.

[11] I. Jolliffe. *Principal Component Analysis*. Springer,New York, 2002.

[12] T. Menzies, J. Greenwald, and A. Frank. Data mining static code attributes to learn defect predictors. *IEEE Transactions on Software Engineering*, 33(1):2–13, January 2007. Available from `http://menzies.us/pdf/06learnPredict.pdf`.

[13] S. Siegel. *Nonparametric Satistics*. New York: McGraw- Hill Book Company, Inc., 1956.

[14] I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, Los Altos, US, 2005.