Combining Feature Subset Selection and Data Sampling for Coping with Highly Imbalanced Software Data

Kehan Gao
Eastern Connecticut State University
Willimantic, Connecticut 06226
gaok@easternct.edu

Taghi M. Khoshgoftaar & Amri Napolitano
Florida Atlantic University
Boca Raton, Florida 33431
khoshgof@fau.edu, amrifau@gmail.com

***Abstract*—In the software quality modeling process, many practitioners often ignore problems such as high dimensionality and class imbalance that exist in data repositories. They directly use the available set of software metrics to build classification models without regard to the condition of the underlying software measurement data, leading to a decline in prediction performance and extension of training time. In this study, we propose an approach, in which feature selection is combined with data sampling, to overcome these problems. Feature selection is a process of choosing a subset of relevant features so that the quality of prediction models can be maintained or improved. Data sampling seeks a more balanced dataset through the addition or removal of instances. Three different approaches would be produced when combing these two techniques: 1- sampling performed prior to feature selection, but retaining the unsampled data instances; 2- sampling performed prior to feature selection, retaining the sampled data instances; 3- sampling performed after feature selection. The empirical study was carried out on six datasets from a real-world software system. We employed one filter-based (no learning algorithm involved in the selection process) feature subset selection technique called correlation-based feature selection combined with the random undersampling method. The results demonstrate that sampling performed prior to feature selection, but retaining the unsampled data instances (Approach 1) performs better than the other two approaches.**

***Index Terms*—software defect prediction, feature selection, data sampling, subset selection**

I. Introduction

Quality and reliability are the most important factors that determine success or failure of software projects, especially for high-assurance and mission-critical systems. Early detection of faults prior to system deployment and operation can help for reducing development costs and allowing for timely improvement to the software product. Various techniques have been developed for this purpose, and some of them have achieved beneficial results. One such technique is software quality classification, in which a classifier is constructed on historical software data (including software metrics and fault data) collected during the software development process, then that classifier is used to classify new program modules under development as either fault-prone (*fp*) or not-fault-prone (*nfp*) [1]. This prediction can help practitioners to identify potentially problematic modules and assign project resources accordingly. However, two problems, high dimensionality and class imbalance, may affect the classifier's performance.

In the software quality modeling process, high dimensionality occurs when a data repository contains many metrics (features) that are either redundant or irrelevant to the class attribute. Redundant features refer to those having information which is already contained in other features, while irrelevant features are features with no useful information related to the class variable. Several problems may arise due to high dimensionality, including high computational cost and memory usage, a decline in prediction performance, and difficulty of understanding and interpreting the model.

Feature selection is a process of selecting a subset of relevant features for use in model construction, so that prediction performance will be improved or maintained, while learning time is significantly reduced. Feature selection techniques can be categorized as either wrappers or filters based on whether a learning algorithm is involved in the selection process, or be classified into feature subset selection and feature ranking depending on whether features are assessed collectively or individually [2]. Feature ranking scores the attributes based on their individual predictive power. A potential problem of feature ranking is that it neglects the possibility that a given attribute may have better predictability when combined with some other attributes, as compared to when used by itself. Feature subset selection that evaluates a subset of features as a group for suitability can overcome this problem. Wrappers evaluate each subset through a learning algorithm, while filters use a simpler statistical measure or some intrinsic characteristic to evaluate each subset rather than using a learning algorithm. Unfortunately, the building of the classifiers required for wrapper-based feature selection are frequently computationally infeasible. Thus, filter-based subset selection is a promising option as it evaluates subsets but is relatively faster than wrapper-based methods. In this study, we would like to examine one filter-based feature subset selection technique called correlation-based feature selection [3] in the context of software quality modeling.

In addition to an excess number of features, many real-world software datasets have the class imbalance problem,

(DOI reference number: 10.18293/SEKE2015-182)

wherein *nfp* modules significantly outnumber *fp* modules (the class of interest). When training data is imbalanced, traditional machine learning algorithms may have difficulty distinguishing between instances of the two classes. In this scenario, they tend to classify the *fp* modules as *nfp* modules to increase overall prediction accuracy. However, these models are rarely useful, because in software engineering practice, accurately detecting the few faulty modules is of upmost importance at the final stage of system testing, as it can avoid defective software in deployment and operation. Many solutions have been proposed to address the class imbalance problem. A frequently used method is data sampling [4], which attempts to achieve a certain balance (ratio) between the two classes by adding instances to (oversampling), or removing instances from (undersampling), the dataset. In this work, we employ a simple and effective sampling technique, random undersampling.

To cope with both high dimensionality and class imbalance, we proposed a data pre-processing technique in which feature selection is combined with data sampling. Some questions may arise when we combine the two techniques, such as which activity, feature selection or sampling, should be performed first? In addition, given the subset of selected features, should the training data be formed based on the sampled dataset or unsampled dataset? To answer all these questions, we investigate three different approaches: 1- data sampling performed prior to feature selection and the training data formed using selected features along with unsampled data; 2- data sampling performed prior to feature selection and the training data formed using selected features along with sampled data; and 3- data sampling performed after feature selection. In this study, we are interested in learning the impact of the feature subset selection technique on classification results when used along with a sampling method as well as the effects of three approaches on classification performance. To our knowledge, no study have been done for combining a filter-based feature subset selection method with data sampling and investigating the three approaches in the domain of software quality engineering.

The empirical study was carried out over two groups of datasets (each group having three datasets) from a real-world software system, all of which exhibit a high degree of class imbalance between the *fp* and *nfp* classes. Five different learners were used to build classification models. The experimental results demonstrate that data sampling performed prior to feature selection and the training data formed using selected features along with unsampled data (Approach 1) had significantly better performance than sampling performed after feature selection (Approach 3), or retaining the sampled data (Approach 2). As to the classification algorithms, Support Vector Machine presented the best (or close to the best) performance irrespective of training data or approach adopted, and therefore was recommented. Multilayer Perceptron and K Nearest Neighbors showed moderate performance, followed Naïve Bayes. Logistic Regression had fluctuate performance with respect to various approaches used.

The rest of the paper is organized as follows. Section II discusses related work. Section III provides methodology, including more detailed information about feature subset selection, data sampling, three combination approaches, learners, performance metric, and cross-validation applied in this work. A case study is described in Section IV. Finally, the conclusion and future work are summarized in Section V.

II. Related Work

Feature selection is an effective technique to solve the high dimensionality problem, and therefore has been significantly researched. Liu et al. [2] provided a comprehensive survey of feature selection and reviewed its developments with the growth of data mining. At present, feature selection has been widely applied in a range of fields, such as text categorization, remote sensing, intrusion detection, genomic analysis, and image retrieval [5]. Hall and Holmes [6] investigated six attribute selection techniques (information gain, ReliefF, principal components analysis, correlation-based feature selection (CFS), consistency-based subset evaluation (CNS), and wrapper subset evaluation) and applied them to 15 datasets. The comparison results show no single best approach for all situations. However, a wrapper-based approach is the best overall attribute selection schema in terms of accuracy if speed of execution is not a considered factor. Otherwise, CFS, CNS, and ReliefF are overall good performers. Feature selection also gets more attention in the software quality assurance domain [7]. Akalya et al. [8] proposed a hybrid feature selection model that combines wrapper and filter methods and applied it to NASA's public KC1 dataset obtained from the NASA IV&V Facility Metrics Data Program (MDP) data repository.

Besides an excess number of attributes, many real-world classification datasets suffer from the class imbalance problem. A considerable amount of research has been done to investigate this problem. Weiss [4] provided a survey of the class imbalance problem and techniques for reducing the negative impact imbalance has on classification performance. An important technique discussed for alleviating the problem of class imbalance is data sampling. The simplest form of sampling is random sampling. Besides that, several more intelligent algorithms for sampling data have been proposed, such as SMOTE [9] and Wilson's Editing [10].

While a great deal of work has been done for feature selection and data sampling separately, limited research has been done and reported on both together, especially in the context of software quality assurance. Among the few studies, Wahono et al. [11] proposed the combination of genetic algorithms with the bagging (bootstrap aggregation) technique for improving the performance of software defect prediction. Genetic algorithms were applied to deal with the feature selection, and bagging was employed to deal with the class imbalance problem. In one of our previous studies [12], we investigated combing feature ranking techniques with data sampling and also examined different combination scenarios. That previous study was focused on feature ranking, while the present research concentrates on feature subset selection.

III. Methodology

A. Feature Subset Selection

For any feature subset selection method, a key issue discussed is the search strategy which determines how the subsets are generated in the first place in order to avoid the $O(2^n)$ models built with exhaustive search. We use the Greedy Stepwise search mechanism in this paper. Greedy Stepwise starts with an empty working feature set and progressively add features, one at a time, until a stopping criterion is reached. Greedy Stepwise uses forward selection to build the full feature subset starting from the empty set. At each point in the process, the algorithm creates a new family of potential feature subsets by adding every feature (one at a time) to the current best-known set. The merit of all these sets are evaluated, and whichever performs best is the new known best set. This process is repeated until none of the new subsets improve performance. The final new "known-best" subset (that is, the last subset which improved performance over its predecessor) is then given as the procedure's output.

The main goal of feature selection is to select a subset of features that minimizes the prediction errors of classifiers. In this study, we employ correlation-based subset selection algorithm [3].

The correlation-based algorithm uses the Pearson correlation coefficient [3], which can be calculated using the following formula:

$$M_S = \frac{k\overline{r_{cf}}}{\sqrt{k + k(k-1)\overline{r_{ff}}}}$$

In this formula, M_S is the merit of the current subset of features, k is the number of features, $\overline{r_{cf}}$ is the mean of the correlations between each feature and the class variable, and $\overline{r_{ff}}$ is the mean of the pairwise correlations between every two features.

B. Data Sampling

A variety of data sampling techniques have been studied in the literature, including both majority undersampling and minority oversampling techniques [9], [13]. We employ random undersamping as the data sampling technique in this study. Random undersampling is a simple, yet effective, data sampling technique that achieves more balance in a given dataset by randomly removing instances from the majority (*nfp*) class. The post-sampling class ratio (between *fp* and *nfp* modules) was set to 50:50 throughout the experiments.

C. Three Combination Approaches

The primary goal of this study is to evaluate the data preprocessing technique in which the correlation-based feature subset selection technique is combined with random undersampling. Three different scenarios (also called approaches) would be produced depending on whether sampling is performed before or after feature selection and which dataset, sampled or unsampled data, is used to build a classifier. The three different approaches are described as follows:

- Approach 1: sampling then feature selection retaining the unsampled data instances
- Approach 2: sampling then feature selection retaining the sampled data instances
- Approach 3: feature selection then sampling

Fig. 1 shows the three approaches denoted as DS-FS-UnSam, DS-FS-Sam, and FS-DS, respectively.

D. Learners

The software defect prediction models in this study are built using five different classification algorithms, including Naïve Bayes (NB) [14], MultiLayer Perceptron (MLP) [14], K Nearest Neighbors (KNN) [14], Support Vector Machine (SVM) [15], and Logistic Regression (LR) [14]. Due to space limitations, we refer interested readers to these references to understand how these commonly-used learners function. The WEKA machine learning tool is used to instantiate the different classifiers. Generally, the default parameter settings for the different learners are used (for NB and LR), except for the below-mentioned changes. A preliminary investigation in the context of this study indicated that the modified parameter settings are appropriate.

In the case of MLP, the `hiddenLayers` parameter was changed to '3' to define a network with one hidden layer containing three nodes, and the `validationSetSize` parameter was changed to '10' to cause the classifier to leave 10% of the training data aside for use as a validation set to determine when to stop the iterative training process. For the KNN learner, the `distanceWeighting` parameter was set to 'Weight by 1/distance', the `kNN` parameter was set to '5', and the `crossValidate` parameter was turned on (set to 'true'). In the case of SVM, two changes were made: the `complexity constant c` was set to '5.0', and `build Logistic Models` was set to 'true'. A linear kernel was used by default.

E. Performance Metric

The Area Under the ROC (receiver operating characteristic) curve (i.e., AUC) is one of the most widely used single numeric measures that provides a general idea of the predictive potential of the classifier. The ROC curve graphs true positive rates versus the false positive rates. Traditional performance metrics for classifier evaluation consider only the default decision threshold of 0.5. ROC curves illustrate the performance across all decision thresholds. A classifier that provides a large area under the curve is preferable over a classifier with a smaller area under the curve. A perfect classifier provides an AUC that equals 1. AUC is of lower variance and is more reliable than other performance metrics such as precision, recall, and F-measure [16].

F. Cross-Validation

For all experiments, we employed 10 runs of 5-fold cross-validation (CV). That is, for each run the data was randomly divided into five folds, one of which was used as the test data while the other four folds were used as training data. All the

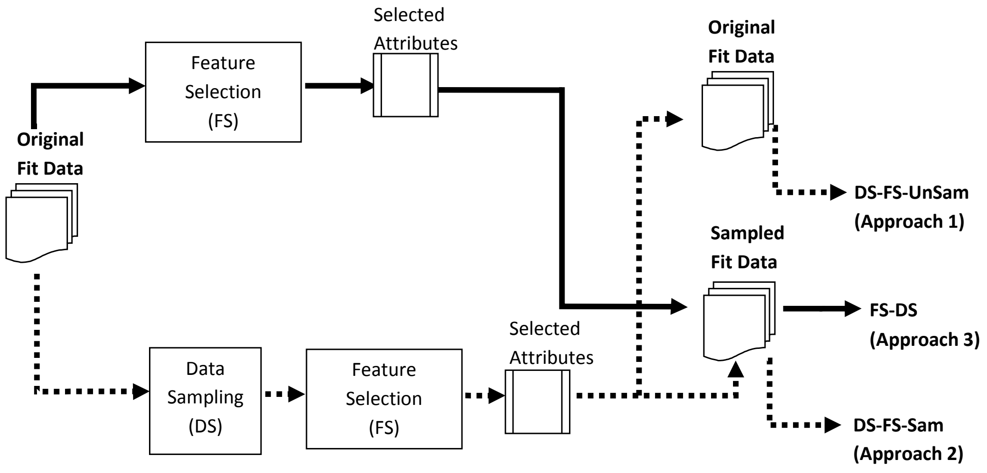


Fig. 1. Three approaches for combining feature selection with data sampling

TABLE I
DATA CHARACTERISTICS

Dataset	Rel.	thd	#Attr.	#Inst.	*fp*		*nfp*	
					#	%	#	%
Eclipse 1	2.0	10	209	377	23	6.1	354	93.9
	2.1	5	209	434	34	7.8	400	92.2
	3.0	10	209	661	41	6.2	620	93.8
Eclipse 2	2.0	5	209	377	52	13.8	325	86.2
	2.1	4	209	434	50	11.5	384	88.5
	3.0	5	209	661	98	14.8	563	85.2

preprocessing steps (feature selection and data sampling) were done on the training dataset. The processed training data was then used to build the classification model and the resulting model was applied to the test fold. This cross-validation was repeated five times, with each fold used exactly once as the test data. The five results from the five folds then was averaged to produce a single estimation. In order to lower the variance of the CV result, we repeated the CV with new random splits 10 times. The final estimation is the average results over the 10 runs of 5-fold CV.

IV. A CASE STUDY

A. Datasets

In our experiments, we use publicly available data, namely the Eclipse defect counts and complexity metrics dataset obtained from the PROMISE data repository (http://promisedata.org). In particular, we use the metrics and defects data at the software packages level. The original data for Eclipse packages consists of three releases denoted 2.0, 2.1, and 3.0 respectively. Each release as reported by Zimmerman et al. [17] contains the following information: the name of the package for which the metrics are collected (name), the number of defects reported six months prior to release (pre-release defects), the number of defects reported six months after release (post-release defects), a set of complexity metrics computed for classes or methods and aggregated in terms of average, maximum, and total (complexity metrics), and the abstract syntax tree of the package consisting of the node size, type, and frequency (structure of abstract syntax tree(s)). For our study we transform the original data by: (1) removing all non-numeric attributes, including the package names, and (2) converting the post-release defects attribute to a binary class attribute with fault-prone (*fp*) being the minority class and not-fault-prone (*nfp*), the majority class. Membership in each class is determined by a post-release defects threshold thd, which separates *fp* from *nfp* packages by classifying packages with thd or more post-release defects as *fp* and the remaining as *nfp*. In our study, we use $thd = \{10, 5\}$ for releases 2.0 and 3.0 while we use $thd = \{5, 4\}$ for release 2.1. This results in two groups. Each group contains three datasets, one for each release. The reason why a different set of thresholds is chosen for release 2.1 is that we would like to keep similar class distributions for the datasets in the same group. All datasets contain 209 attributes (208 independent attributes and 1 dependent attribute). Table I shows the characteristics of the datasets after transformation for each group. These datasets exhibit different distribution of class skew (i.e., the percentage of *fp* examples).

B. Results and Analysis

The results (in terms of AUC) of the correlated-based feature subset selection technique combined with random undersampling averaged over 10 runs of 5-fold CV for each dataset are reported in Table II, which contains the results for all five learners and three combination approaches. For a given learner, the best combination approach is highlighted in **bold** for each dataset. Among the 30 best performers, 23 are from Approach 1, 6 from Approach 3 and the remaining one from Approach 2.

Fig. 2 provides comparisons of three combination approaches along with various classification algorithms averaged over the respective groups of datasets. The charts intuitively demonstrate that

- Approach 1 performed better than the other two approaches for all the learners in Eclipse 1 (see Fig. 2(a)).

TABLE II
CLASSIFICATION PERFORMANCE

(a) Eclipse 1

Release	Approach	NB	MLP	KNN	SVM	LR
2.0	1	**0.8437**	**0.8513**	**0.8738**	**0.8772**	**0.8657**
	2	0.8234	0.8301	0.8555	0.8488	0.7626
	3	0.8205	0.8011	0.8606	0.8458	0.7193
2.1	1	0.8312	**0.8612**	0.8688	**0.9031**	**0.8730**
	2	0.8204	0.8327	0.8507	0.8734	0.7894
	3	**0.8319**	0.8371	**0.8867**	0.8861	0.7885
3.0	1	**0.8891**	**0.8844**	**0.8834**	**0.9146**	**0.9097**
	2	0.8843	0.8605	0.8747	0.9075	0.8445
	3	0.8783	0.8696	0.8628	0.9068	0.7721

(b) Eclipse 2

Release	Approach	NB	MLP	KNN	SVM	LR
2.0	1	0.8273	**0.8654**	0.8705	**0.9064**	**0.8880**
	2	0.8302	0.8624	**0.8736**	0.8807	0.8383
	3	**0.8397**	0.8580	0.8656	0.8934	0.7865
2.1	1	**0.8219**	**0.8509**	**0.8377**	**0.8909**	**0.8818**
	2	0.8150	0.8355	0.8364	0.8657	0.8395
	3	0.8119	0.8363	0.8544	0.8828	0.8548
3.0	1	0.8766	0.8963	0.8915	**0.9336**	**0.9348**
	2	0.8708	0.8951	0.8878	0.9180	0.9186
	3	**0.8777**	**0.9025**	**0.9006**	0.9222	0.9268

TABLE III
ANOVA FOR THE ECLIPSE DATASETS

(a) Eclipse 1

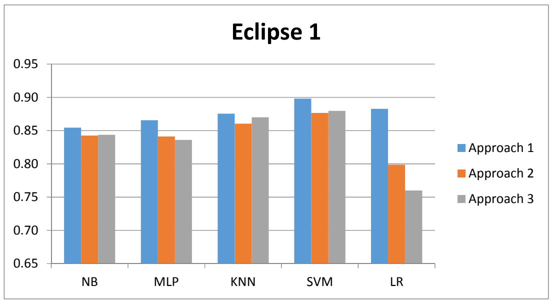
Source	Sum Sq.	d.f.	Mean Sq.	F	p-value
Approach	0.1217	2	0.0609	25.89	0.000
Error	1.0508	447	0.0024		
Total	1.1725	449			

(b) Eclipse 2

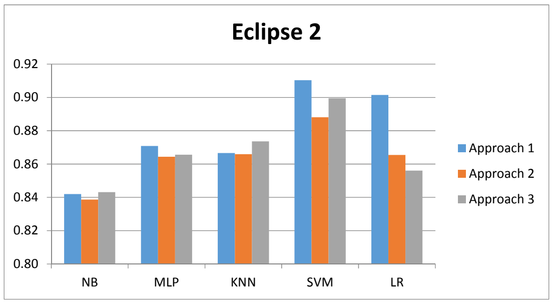
Source	Sum Sq.	d.f.	Mean Sq.	F	p-value
Approach	0.0156	2	0.0078	5.20	0.006
Error	0.6701	447	0.0015		
Total	0.6856	449			

- Approach 1 performed better than the other two approaches for the MLP, SVM, and LR learners in Eclipse 2, while for the NB and KNN learners, Approach 1 displayed similar or slightly worse performance than Approach 3 (see Fig. 2(b)).
- The advantage of Approach 1 is obvious when the SVM and LR learner were employed.
- Some learners, like LR, are significantly affected by the combination approach adopted, while others, like NB and KNN, are more robust with different approaches.

We further carried out a one-way analysis of variance (ANOVA) F-test on the classification performance to examine if the three combination approaches are statistically different or not. Note that all the statistical analysis was performed over each individual group of datasets, since each group displayed a distinct degree of class imbalance. In addition, as learner is not the focus of this paper, the factor taken into account only is the three combination approaches. The null hypothesis for the ANOVA test is that all the group population means are the same, while the alternate hypothesis is that at least one pair of means is different. Table III shows the ANOVA results. The p-value is less than the cutoff 0.05 for the factor, meaning that the alternate hypothesis is accepted, namely, at least two approach means are significantly different from each other.



(a) Eclipse 1



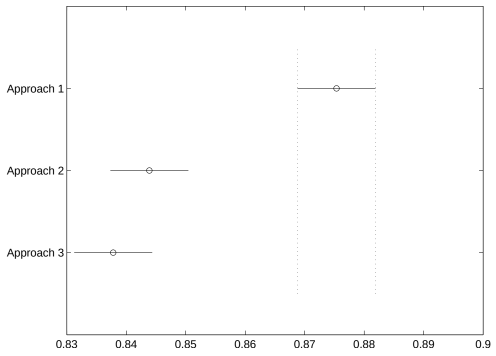
(b) Eclipse 2

Fig. 2. Comparisons of three approaches

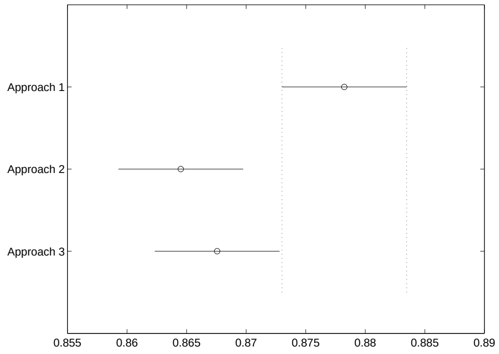
We further conducted a multiple comparison test on the factor with Tukey's honestly significant difference (HSD) criterion. For both the ANOVA and multiple comparison tests, the significance level was set to 0.05. Fig. 3 shows the multiple comparisons for both groups of datasets. The figures display graphs with each group mean represented by a symbol (○) and 95% confidence interval as a line around the symbol. Two means are significantly different if their intervals are disjoint, and are not significantly different if their intervals overlap. The assumptions for constructing ANOVA and Tukey's HSD models were validated. From these figures we can see the following points:

- Approach 1 had significantly better classification performance than Approaches 2 and 3 for both groups of datasets.
- Approach 2 and Approach 3 showed similar performance (no significant difference). Approach 2 performed slightly better than Approach 3 for Eclipse 1, while Approach 2 had slightly worse performance than Approach 3 for Eclipse 2.

Overall, when the correlation-based feature selection technique is used along with the random undersampling method, we strongly recommend the data pre-processing approach in which sampling is performed prior to feature selection and the training data is formed using selected features along with unsampled data. This approach is especially effective when SVM and LR are used as classifiers.



(a) Eclipse 1



(b) Eclipse 2

Fig. 3. Multiple comparison for three approaches

V. Conclusion

In this study, we proposed feature subset selection combined with data sampling to overcome the high dimensionality and class imbalance problems that often affect software quality classification. Three approaches were investigated: 1- sampling performed prior to feature selection, retaining the unsampled data instances; 2- sampling performed prior to feature selection, retaining the sampled data instances; and 3- sampling performed after feature selection. More specifically, we were interested in investigating the correlation-based feature selection method used along with random undersampling and studying the effect of three combination approaches.

In the experiments, we applied these techniques to six datasets from a real-world software system. We built classification models using five learners. The results demonstrate that among the three data pre-processing approaches, sampling performed prior to feature selection and retaining the unsampled data (Approach 1) had significantly better performance than sampling performed after feature selection (Approach 3) or sampling performed prior to feature selection but retaining the sampled data (Approach 2). Of the five learners, Support Vector Machine presented the best performance, while Multilayer Perceptron and K Nearest Neighbors demonstrated average performance. Logistic Regression performed variously with respect to different data pre-processing approaches. In contrast, Naïve Bayes showed relatively consistent performance for various approaches.

Future work will involve comparisons between feature ranking and feature subset selection as well as between wrapper subset selection and filter subset selection.

References

[1] A. K. Pandey and N. K. Goyal, "Predicting fault-prone software module using data mining technique and fuzzy logic," *Special Issue of International Journal of Computer and Communication Technology*, vol. 2, no. 2-4, pp. 56–63, 2010.

[2] H. Liu, H. Motoda, R. Setiono, and Z. Zhao, "Feature selection: An ever evolving frontier in data mining," in *Proceedings of the Fourth International Workshop on Feature Selection in Data Mining*, Hyderabad, India, 2010, pp. 4–13.

[3] M. A. Hall, "Correlation-based feature selection for machine learning," Ph.D. dissertation, The University of Waikato, Hamilton, New Zealand, 1999.

[4] G. M. Weiss, "Mining with rarity: A unifying framework," *SIGKDD Explorations*, vol. 6, no. 1, pp. 7–19, 2004.

[5] V. Kumar and S. Minz, "Feature selection: A literature review," *Smart Computing Review*, vol. 4, no. 3, pp. 211–229, June 2014.

[6] M. A. Hall and G. Holmes, "Benchmarking attribute selection techniques for discrete class data mining," *IEEE Transactions on Knowledge and Data Engineering*, vol. 15, no. 6, pp. 1437 – 1447, Nov/Dec 2003.

[7] K. Gao, T. M. Khoshgoftaar, and N. Seliya, "Predicting high-risk program modules by selecting the right software measurements," *Software Quality Journal*, vol. 20, no. 1, pp. 3–42, 2012.

[8] C. Akalya devi, K. E. Kannammal, and B. Surendiran, "A hybrid feature selection model for software fault prediction," *International Journal on Computational Sciences and Applications*, vol. 2, no. 2, pp. 25–35, Apr. 2012.

[9] N. V. Chawla, K. W. Bowyer, L. O. Hall, and P. W. Kegelmeyer, "SMOTE: Synthetic minority over-sampling technique," *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, 2002.

[10] R. Barandela, R. M. Valdovinos, J. S. Sanchez, and F. J. Ferri, "The imbalanced training sample problem: Under or over sampling?" *In Joint IAPR International Workshops on Structural, Syntactic, and Statistical Pattern Recognition (SSPR/SPR'04), Lecture Notes in Computer Science 3138*, no. 806-814, 2004.

[11] R. S. Wahono, N. Suryana, and S. Ahmad, "Metaheuristic optimization based feature selection for software defect prediction," *Journal of Software*, vol. 9, no. 5, pp. 1324–1333, May 2014.

[12] K. Gao and T. M. Khoshgoftaar, "Software defect prediction for high-dimensional and class-imbalanced data," in *Proceedings of the 23rd International Conference on Software Engineering & Knowledge Engineering (SEKE'2011), Eden Roc Renaissance, Miami Beach, USA, July 7-9, 2011*, 2011, pp. 89–94.

[13] C. Seiffert, T. M. Khoshgoftaar, J. V. Hulse, and A. Napolitano, "Rusboost: A hybrid approach to alleviating class imbalance," *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, vol. 40, no. 1, pp. 185–197, 2010.

[14] I. H. Witten, E. Frank, and M. Hall, *Data Mining: Practical Machine Learning Tools and Techniques*, 3rd ed. Morgan Kaufmann, 2011.

[15] J. Shawe-Taylor and N. Cristianini, *Support Vector Machines*, 2nd ed. Cambridge University Press, 2000.

[16] Y. Jiang, J. Lin, B. Cukic, and T. Menzies., "Variance analysis in software fault prediction models," in *Proceedings of the 20th IEEE International Symposium on Software Reliability Engineering*, Bangalore-Mysore, India, Nov. 16-19 2009, pp. 99–108.

[17] T. Zimmermann, R. Premraj, and A. Zeller, "Predicting defects for eclipse," in *Proceedings of the 29th International Conference on Software Engineering Workshops*. Washington, DC, USA: IEEE Computer Society, 2007, p. 76.