Attribute Selection and Imbalanced Data: Problems in Software Defect Prediction

Taghi M. Khoshgoftaar[*]
Kehan Gao[†]
Naeem Seliya[‡]

Abstract

The data mining and machine learning community is often faced with two key problems: working with imbalanced data and selecting the best features for machine learning. This paper presents a process involving a feature selection technique for selecting the important attributes and a data sampling technique for addressing class imbalance. The application domain of this study is software engineering, more specifically, software quality prediction using classification models. When using feature selection and data sampling together, different scenarios should be considered. The four possible scenarios are: (1) feature selection based on original data, and modeling (defect prediction) based on original data; (2) feature selection based on original data, and modeling based on sampled data; (3) feature selection based on sampled data, and modeling based on original data; and (4) feature selection based on sampled data, and modeling based on sampled data. The research objective is to compare the software defect prediction performances of models based on the four scenarios. The case study consists of nine software measurement data sets obtained from the PROMISE software project repository. Empirical results suggest that feature selection based on sampled data performs significantly better than feature selection based on original data, and that defect prediction models perform similarly regardless of whether the training data was formed using sampled or original data.

Keywords: *feature selection, data sampling, software measurements, defect prediction.*

[*]Taghi M. Khoshgoftaar is with Florida Atlantic University, Boca Raton, Florida 33431, taghi@cse.fau.edu.

[†]Kehan Gao is with Eastern Connecticut State University, Willimantic, Connecticut 06226, gaok@easternct.edu.

[‡]Naeem Seliya is with the University of Michigan - Dearborn, Dearborn, MI 48128, nseliya@umich.edu.

1. Introduction

The data mining and machine learning domain focuses on extracting hidden but useful information from data repositories. Considerable effort has been dedicated toward various aspects of data mining and machine learning, including data preprocessing, model training, model evaluation, and pattern verification and validation. The focus of this study is on data preprocessing activities, in particular, feature selection from a data set that suffers from class imbalance. Feature selection extracts the most important set of attributes for model training, while class imbalance occurs when the data set is skewed toward a specific class of instances. Class imbalance is a very common occurrence among a wide variety of application domains, including software quality estimation [11, 14], microarrary data analysis [12], network intrusion detection [7], and text mining problems [8].

Software quality engineering includes various techniques and processes for producing a high quality software product. One effective method is to employ data mining techniques and apply them to software metrics collected during the software development process to identify the potential fault-prone program modules. We examine the feature selection and class imbalance problems in the context of software quality estimation (also referred to as software defect prediction), where a classification model is used to predict program modules (instances) as fault-prone (*fp*) or not-fault-prone (*nfp*) [11, 14, 16]. Such a model is usually trained using software measurement and defect data from previous development experiences, and the model is then applied to predict the quality of target (under-development) program modules. As a result, practitioners can strategically allocate project resources and focus more on those potential problematic modules. In a given software measurement and defect data set, the proportion of *fp* instances is generally much lower than the proportion of *nfp* instances, which causes the class imbalance problem.

Practitioners and researchers often do not consider the quality of software measurement data when building defect predictors [11, 16], which can lead to undesirable consequences. The effectiveness and performance of software defect prediction models is affected by two important factors: (1) the set of software metrics (predictors or independent attributes) used to build the defect prediction models, and (2) the proportion of poor quality (the *fp* instances) modules in the software measurement data set. From a software engineering practice point of view, it is important to consider a smaller set of software metrics for defect prediction. Moreover, building software quality models without any data preprocessing will not produce effective defect predictors. The software quality model is adversely affected if redundant or ineffective software metrics are included in the training data. Class imbalance (between the *fp* and *nfp* instances) can lead to a model that is not practical, because most instances will be predicted as *nfp*.

This paper presents a process of using two data preprocessing steps, feature selection (for selecting the important attributes) and data sampling (for addressing class imbalance), together in the context of software defect prediction. We consider six commonly used feature ranking techniques [22] for feature selection purposes, and the random undersampling [21] technique for data sampling purposes.

The process of using feature selection and data sampling together can lead to four scenarios, each providing a different training data set for building the classification models. Feature selection can be based on either the sampled (after data sampling) or unsampled (original) data set, providing two different feature subsets. Given the subset of selected features, the training data can be based on either the sampled or unsampled data set. The four scenarios for forming the training data are: (1) the feature subset generated from the original data, and training data generated after feature subset applied to the original data; (2) the feature subset generated from the original data, and training data generated after feature subset applied to the sampled data; (3) the feature subset generated from the sampled data, and training data generated after feature subset applied to the original data; and (4) the feature subset generated from the sampled data, and training data generated after feature subset applied to the sampled data.

The research objective of this study is to empirically compare the performances of the defect prediction models based on each of the four scenarios of data preprocessing. Very limited work has been done on considering both feature selection and data sampling together, and to our knowledge, this is the first study to examine the above-described scenarios when using feature selection and data sampling together. The case study data is based on software measurement and defect data of nine software projects, obtained from the PROMISE software project repository [2]. The key conclusions from empirical results are: (1) feature selection based on the sampled data set significantly outperforms feature selection based on the original data set, and (2) performance of the defect prediction models is not affected by whether the training data set is produced from the sampled data or original data. This study provides the analyst with a practical solution to reducing the number of software metrics for defect prediction and also how to avoid the adverse effects of a skewed software measurement data set.

The remainder of the paper is organized as follows: Section 2 discusses key related work. Section 3 presents the methods and techniques used in this paper. Section 4 describes the software measurement data sets used in the case study. Section 5 presents the experiments including design, results and analysis. Finally, we summarize our conclusions and provide suggestions for future work in Section 6.

2. Related Work

Feature selection (or attribute selection) is an important data preprocessing activity and has been extensively studied in the data mining and machine learning community. A comprehensive survey of feature selection algorithms is presented in [18]. Typically, feature selection techniques are divided into two categories: *wrapper*-based approach and *filter*-based approach. The wrapper-based approach involves training a learner during the feature selection process, while the filter-based approach uses the intrinsic characteristics of the data, based on a given metric, for feature selection and does not depend on training a learner. The primary advantage of the filter-based approach over the wrapper-based approach is that it is computationally faster.

Hall and Holmes [10] investigated six attribute selection techniques that produce ranked lists of attributes and applied them to 15 data sets from the UCI repository. The comparison results showed that no one approach was the best for all situations. However, if computational complexity was not a factor, then a wrapper-based approach was the best overall attribute selection scheme in terms of accuracy. Saeys et al. [20] investigated the use of ensemble feature selection techniques, where multiple feature selection methods were combined to yield results. They showed that the ensemble approach presented great promise for high-dimensional domains with small sample sizes and provided more robust feature subsets than a single feature selection technique. In a recent study [22], we investigated various feature selection techniques including filter-based and wrapper-based methods in the software quality engineering domain. In that study, it was concluded that the performances of the classification models either improved or were not affected when over 85% of the features were eliminated from the original data sets.

The class imbalance problem is observed in various domains [7, 12, 24]. To overcome the difficulties associated with learning from imbalanced data, various techniques have been developed. Data sampling is the primary approach for handling class imbalance, and it involves balancing the relative class distributions of the given data set. For a two-group problem such as predicting program modules as either *fp* or *nfp*, there are two types of data sampling approaches: majority undersampling and minority oversampling [3, 5, 21]. In this study we consider an effective data sampling approach, random undersampling, for addressing the class imbalance problem in defect prediction modeling. Due to limited space, we could not consider other data sampling techniques [3, 5, 21] in this study.

Imbalanced data can also be dealt with directly when creating the classifier. This is easier with some classifiers than with others. For example, Naïve Bayes produces a probability that an instance belongs to a given class, and one can then adjust the threshold at which an instance is put into a certain class and solve the problem of attempting to classify everything as being in the majority class to improve accuracy. A related approach is cost-sensitive classification [6]. However, the costs of different types of misclassification errors are not easy to obtain or estimate during the modeling process. Future work will consider these approaches in the context of this study.

While considerable work has been done for feature selection and data sampling separately, limited research can be found on investigating them both together, particularly in the software engineering field. Chen et al. [4] have studied data row pruning (data sampling) and data column pruning (feature selection) in the context of software cost/effort estimation. However, the data sampling in their study does not focus on the class imbalance problem, and unlike this study, the classification models are meant for non-binary problems. Moreover, they focused only on the case in which data sampling is used prior to feature selection. Liu et al. [17] introduced the concept of active feature selection, and investigated a selective sampling approach to active feature selection in a filter model setting. However, the purpose of data sampling in their work was for reducing the data set size instead of addressing class imbalance. In this paper, we use both feature selection and data sampling together as a data preprocessing step to form the training data. Four different scenarios of forming the training data are investigated, which is unique to this study.

3. Methodology

3.1. Filter-Based Feature Ranking Techniques

The procedure of feature ranking is to score each feature (attribute) according to a particular method (metric), allowing the selection of the best set of features. In this study, we used six commonly used filter-based feature ranking techniques: chi-square (CS), information gain (IG), gain ratio (GR), two types of ReliefF (RF and RFW), and symmetrical uncertainty (SU). The chi-square, χ^2, test [19] is used to examine the distribution of the class as it relates to the values of the given feature. The null hypothesis is that there is no correlation, i.e. each value is as likely to have instances in any one class as any other class. Given the null hypothesis, the χ^2 statistic measures how far away the actual value is from the expected value:

$$\chi^2 = \sum_{i=1}^{r} \sum_{j=1}^{n_c} \frac{(O_{i,j} - E_{i,j})^2}{E_{i,j}}$$

where r is the number of different values of the feature, n_c is the number of classes (in this work, $n_c = 2$), $O_{i,j}$ is the observed number of instances with value i which are in class j, and $E_{i,j}$ is the expected number of instances with value i and class j. The larger this χ^2 statistic, the more likely it is that the distribution of values and classes are dependent, i.e. the feature is relevant to the class.

Information gain (IG) is the information provided about the target class attribute Y, given the value of another attribute X. IG measures the decrease of the weighted average impurity of the partitions compared to the impurity of the complete set of data. A drawback of IG is that it tends to prefer attributes with a larger number of possible values, i.e., if one attribute has a larger number of values, it will appear to gain more information than those with fewer values, even if it is actually no more informative. One strategy to solve this problem is to use the gain ratio (GR), which penalizes multiple-valued attributes. Symmetrical uncertainty (SU) is another way to overcome the problem of IG's bias toward attributes with more values, doing so by dividing by the sum of the entropies of X and Y.

Relief is an instance-based feature ranking technique introduced by Kira and Rendell [15]. ReliefF is an extension of the Relief algorithm that can handle noise and multiclass data sets, and is implemented in the WEKA tool [23]. When the `WeightByDistance` (weight nearest neighbors by their distance) parameter is set as default (false), the algorithm is referred to as RF; when the parameter is set to 'true', the algorithm is referred to as RFW.

3.2. Data Sampling Techniques

A number of data sampling techniques have been studied in the literature, including both majority undersampling and minority oversampling techniques [3, 8, 21]. We consider random undersamping (RUS) 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 data set by randomly removing instances from the majority class. In the context of our study, the *nfp* modules represent the majority class. The post-sampling class distribution is a parameter for any data sampling technique. In our study, the post-sampling distribution of the *nfp* and *fp* instances is 65% and 35% respectively. Other settings such as 50:50 were also considered, but those results are not presented due to similarity of conclusions.

3.3. Classifiers

The two learners that are selected for building the software quality prediction models are K-nearest-neighbors (KNN) and support vector machine (SVM) [23]. Selection of learners is based on their common use in the software engineering and data mining domains, and that they lack a built-in feature selection capability. We employ the WEKA tool [23] to implement these classifiers. Unless stated otherwise, we use default parameter settings as specified in WEKA.

The KNN classifier, also called instance-based learning, uses distance-based comparisons. The choice of distance metric is critical. In our study, KNN was built with changes to three parameters. 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'). Two changes were made to the default parameters of the SVM learner in WEKA: the `complexity constant c` was set to '5.0' and `build Logistic Models` was set to 'true'. By default, a linear kernel was used. In the case of both learners, preliminary investigations indicated the parameter changes yielded better results.

3.4. Performance Metric

The Area Under the ROC (Receiver Operating Characteristic) curve, abbreviated as AUC, is used for evaluating the defect prediction models in this study. The AUC metric is commonly used to evaluate software defect prediction models [11, 16]. If the *fp* modules are considered positive and the *nfp* modules are considered negative, then the ROC curve plots the true positive rates against the false positive rates. An ROC curve illustrates the classifier's performance across all decision thresholds, i.e., a value between 0 and 1 that theoretically separates the *fp* and *nfp* modules. The AUC values range from 0 to 1, where a perfect classifier provides an AUC value of 1 [23].

4. Case Study Data Description

The case study data is obtained from the publicly available PROMISE software project data repository [2]. We study the software measurement data sets for the Java-based Eclipse project [25]. The software metrics and defect data are aggregated at the software packages level; hence, a program module is a Java package in Eclipse. We consider three releases of the Eclipse system, where the releases are denoted as 2.0, 2.1, and 3.0 [25].

Table 1. Data Characteristics

Data set	Rel.	thd	#Attr.	#Inst.	#fp	%fp	#nfp	%nfp
1	2.0	10	209	377	23	6.1	354	93.9
2	2.0	5	209	377	52	13.8	325	86.2
3	2.0	3	209	377	101	26.8	276	73.2
4	2.1	5	209	434	34	7.8	400	92.2
5	2.1	4	209	434	50	11.5	384	88.5
6	2.1	2	209	434	125	28.8	309	71.2
7	3.0	10	209	661	41	6.2	620	93.8
8	3.0	5	209	661	98	14.8	563	85.2
9	3.0	3	209	661	157	23.8	504	76.2

Each system release contains the following information [25]: name of the package for which the metrics are collected (*name*); number of defects reported six months prior to release (*pre-release defects*); number of defects reported six months after release (*post-release defects*); a set of complexity metrics computed for classes or methods and aggregated by using average, maximum, and total at the package level (*complexity metrics*); and structure of abstract syntax tree(s) of the package consisting of the node size, type, and frequency (*structure of abstract syntax tree(s)*).

We modify 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. A program module's membership in a given class is determined by a post-release defects threshold, thd. A program module (package) with thd or more post-release defects is labeled as *fp*, while those with fewer than thd defects are labeled as *nfp*. For a given system release we consider three values for thd: for releases 2.0 and 3.0, thd = {10, 5, 3}, and for release 2.1, thd = {5, 4, 2}. This results in three different types of class distributions with respect to the *fp* and *nfp* modules for each release, as shown in Table 1. A different set of thresholds is chosen for release 2.1 because we wanted to maintain relatively similar types of class distributions as those of Releases 2.0 and 3.0.

Table 1 presents key details about the different data sets used in our study. All data sets contain 209 software attributes, which include 208 independent (predictor) attributes and one dependent attribute (*fp* or *nfp* class label). Each group of data sets exhibit different class distributions with respect to the *fp* and *nfp* modules. The most imbalanced data has 92%-94% of *nfp* instances, while the least imbalanced data has 71%-76% of *nfp* instances. The transformation of the data sets in terms of the respective class distributions provides a good representation of the class imbalance problem often seen in real-world data sets.

5. Experiments

5.1. Design

The primary goal of the experiments is to evaluate the effectiveness of feature selection techniques when combined with data sampling in four different scenarios. The four different scenarios (see Figure 1) include all the possible situations when feature selection and data sampling are used together to form the training data set. The scenarios are:

Scenario 1 (S1): Feature ranking technique is used alone based on the ***original*** data and the selected features are applied to the ***original*** data to form the training data set.

Scenario 2 (S2): Feature ranking technique is used alone based on the ***original*** data and the selected features are applied to a ***sampled*** data to form the training data set.

Scenario 3 (S3): Feature ranking technique is used based on a ***sampled*** data and the selected features are applied to the ***original*** data to form the training data set.

Scenario 4 (S4): Feature ranking technique is used based on a ***sampled*** data and the selected features are applied to the ***sampled*** data to form the training data set.

5.2. Results and Analysis

In our study, feature selection aims to select the best set of software metrics for defect prediction. The number of features to be selected is a parameter setting during feature selection. Related literature provides no specific guidance on the appropriate number of features to select. We found that $\lceil \log_2 n \rceil$ number of features (n is the total number of the independent attributes in the original data set) is appropriate when using WEKA to build Random Forests learners for binary classification with imbalanced data sets [13]. We use the same strategy in this study, after our preliminary investigation revealed it to be appropriate for various classification algorithms including KNN and SVM. $\lceil \log_2 n \rceil$ number of attributes that have the highest scores are selected during feature selection. For the nine Eclipse data sets in our case study, $\lceil \log_2 n \rceil = 8$, since $n = 208$.

The defect prediction models were built using a five-fold cross validation strategy. Each cross validation run was repeated 10 times to avoid bias due to a lucky/unlucky data split. Table 2 and Table 3 summarize the classification performances in terms of AUC across the four scenarios (S1 through S4) for KNN and SVM learners, respectively. Each value presented is the average over the ten runs of five-fold cross-validation. The last row shows the average performance of the classification models in each scenario over the six feature selection techniques and across nine data sets.

The above tables suggest that feature selection based on the sampled data (S3 and S4) performed better than feature selection based on the unsampled (original) data (S1 and S2). In addition, there is no significant performance distinction between the classification models based on the training data sets formed from the original data and sampled data (S1 vs. S2 and S3 vs. S4). If we rank the performances of the classification models of the different scenarios, Scenario 3 performed best followed by Scenario 4, then Scenario 1, and finally Scenario 2. In this paper we do not present the modeling results based on the complete set of software metrics, because in a recent study [9], we demonstrated that defect prediction models built on fewer software metrics (after feature selection) performed generally better than models based on the complete set of software metrics (no feature selection).

We also compared the defect prediction performance results for the different scenarios by performing a one-way ANalysis Of VAriance (ANOVA) F-test [1]. The ANOVA test examines the significance level of the differences in model performances. The underlying assumptions of ANOVA were tested and validated prior to statistical analysis. The factor of interest (Factor A) considered in our ANOVA experiment is the modeling scenario (S1, S2, S3 and S4). 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. The significance level for the ANOVA experiment is $\alpha = 0.05$. Table 4 and Table 5 respectively show the ANOVA results for KNN and SVM across the nine data sets. The p-value in each table is less than the value of α indicating that the alternate hypothesis is accepted, namely, at least two group means (scenarios) are significantly different from each other.

We wanted to find out which pairs of means (of the four scenarios) are significantly different, and which are not. We conducted a multiple pairwise comparison by using Tukey's Honestly Significant Difference (HSD) criterion [1]. The significance level for Tukey's HSD test is $\alpha = 0.05$. Figures 2 and 3 show the comparison results of the KNN and SVM learners, respectively. The figures display graphs with each group mean represented by a symbol ($\circ$) and the 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. We used Matlab to perform the ANOVA and multiple comparison tests. Figures 2 and 3 reveal that when data sampling was performed prior to feature selection (S3 and S4) it significantly improved classification performance compared to the cases where feature selection is used alone (S1 and S2). However, the training data produced from the sampled or unsampled (original) data did not significantly impact the classification performance (S1 vs. S2 and S3 vs. S4). The classification performance of four different scenarios and their distinctions are clearly depicted in the figures.

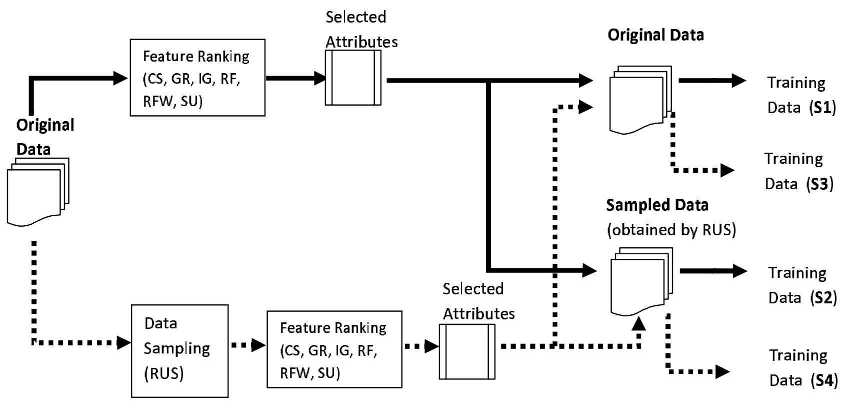

Figure 1. Scenarios of Using Feature Selection and Data Sampling Together.

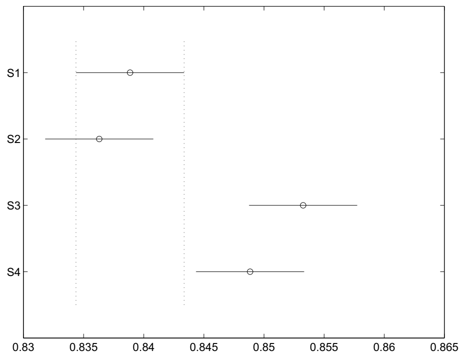

Figure 2. Multiple Comparisons for KNN

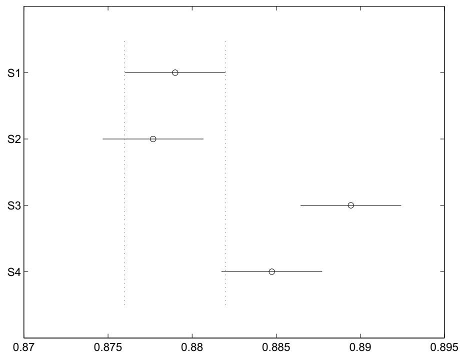

Figure 3. Multiple Comparisons for SVM

6. Conclusion

In a given classification problem, an important challenge is selecting the appropriate features when the underlying data is imbalanced or skewed. We present a process of using feature selection and data sampling together to form the training data for building a software defect prediction model. The study answers research questions such as whether to apply feature selection to the original or sampled data, and whether the training data should be formed based on the original or sampled data, given a set of selected features.

All four scenarios that cover the above two questions are investigated in this study, and they are: (1) feature selection based on original data, and training data based on original data; (2) feature selection based on original data, and training data based on sampled data; (3) feature selection based on sampled data, and training data based on original data; and (4) feature selection based on sampled data, and training data based on sampled data. Feature selection is based on using six different filter-based feature ranking techniques, while data sampling (for addressing class imbalance) is based on using the random undersampling technique. The feature ranking techniques are commonly used approaches [22], while random undersampling has been shown to be an effective data sampling technique [21].

The empirical case study is based on software measurement data obtained from the PROMISE repository. The results show that feature selection based on sampled data resulted in significantly better performance than feature selection based on original data. In addition, performance of the defect prediction models was not affected significantly regardless of whether the training data was formed using sampled data or original data. The conclusions of this study imply that selecting the right set of software metrics for defect prediction is very important. From a practical point of view, working with a smaller set (e.g., 8 in our study) of metrics for software quality modeling is more attractive than working with a large number (e.g., 208 in our study) of metrics.

Future work will include experiments using data sets from different software projects as well as other domains.

Table 2. Performance Metric, AUC, for KNN

Data	Filter	S1	S2	S3	S4
1	CS	0.8377	0.8217	0.8599	0.8382
	GR	0.8078	0.7750	0.8435	0.8227
	IG	0.8706	0.8575	0.8583	0.8416
	RF	0.8715	0.8518	0.8994	0.8722
	RFW	0.8085	0.8011	0.8357	0.8317
	SU	0.8349	0.8043	0.8708	0.8474
2	CS	0.8692	0.8736	0.8735	0.8774
	GR	0.8277	0.8310	0.8413	0.8468
	IG	0.8719	0.8809	0.8770	0.8806
	RF	0.8491	0.8406	0.8699	0.8622
	RFW	0.8413	0.8338	0.8666	0.8631
	SU	0.8674	0.8670	0.8792	0.8817
3	CS	0.8270	0.8245	0.8181	0.8158
	GR	0.7891	0.7900	0.7908	0.7864
	IG	0.8256	0.8244	0.8198	0.8187
	RF	0.7558	0.7541	0.7593	0.7579
	RFW	0.7634	0.7637	0.7568	0.7537
	SU	0.8233	0.8204	0.8175	0.8151
4	CS	0.9049	0.9040	0.9069	0.9083
	GR	0.8153	0.8159	0.8920	0.8943
	IG	0.9095	0.9180	0.9064	0.9091
	RF	0.7765	0.7488	0.8410	0.7885
	RFW	0.7420	0.7523	0.7717	0.7548
	SU	0.8942	0.9048	0.9073	0.9121
5	CS	0.8799	0.8802	0.8795	0.8832
	GR	0.8175	0.8223	0.8486	0.8579
	IG	0.8836	0.8855	0.8739	0.8799
	RF	0.7106	0.7222	0.8079	0.7833
	RFW	0.6602	0.6765	0.7359	0.7468
	SU	0.8773	0.8770	0.8732	0.8764
6	CS	0.8605	0.8644	0.8592	0.8634
	GR	0.8533	0.8573	0.8459	0.8450
	IG	0.8584	0.8618	0.8595	0.8636
	RF	0.7641	0.7681	0.7761	0.7680
	RFW	0.7625	0.7604	0.7717	0.7688
	SU	0.8520	0.8545	0.8551	0.8579
7	CS	0.9060	0.8977	0.9080	0.9019
	GR	0.8780	0.8623	0.8838	0.8747
	IG	0.9208	0.9112	0.9130	0.9080
	RF	0.8344	0.7917	0.8699	0.8467
	RFW	0.7751	0.7712	0.7682	0.7641
	SU	0.8774	0.8741	0.8993	0.8926
8	CS	0.9246	0.9197	0.9238	0.9206
	GR	0.9154	0.9159	0.9162	0.9168
	IG	0.9225	0.9179	0.9215	0.9192
	RF	0.7964	0.8103	0.8285	0.8273
	RFW	0.8037	0.8108	0.8451	0.8433
	SU	0.9154	0.9154	0.9203	0.9195
9	CS	0.8978	0.8987	0.8934	0.8935
	GR	0.8832	0.8847	0.8842	0.8821
	IG	0.8960	0.8960	0.8928	0.8930
	RF	0.7390	0.7447	0.7698	0.7697
	RFW	0.7556	0.7551	0.7997	0.8000
	SU	0.8933	0.8940	0.8889	0.8903
Average		**0.8389**	**0.8363**	**0.8533**	**0.8488**

Table 3. Performance Metric, AUC, for SVM

Data	Filter	S1	S2	S3	S4
1	CS	0.8488	0.8410	0.8662	0.8567
	GR	0.8410	0.8365	0.8611	0.8529
	IG	0.8625	0.8587	0.8665	0.8555
	RF	0.8465	0.8619	0.8753	0.8602
	RFW	0.8135	0.8189	0.8397	0.8386
	SU	0.8432	0.8438	0.8729	0.8663
2	CS	0.9143	0.9080	0.9092	0.9022
	GR	0.8710	0.8670	0.8857	0.8790
	IG	0.9186	0.9144	0.9132	0.9036
	RF	0.8941	0.8862	0.9078	0.8973
	RFW	0.8945	0.8872	0.9025	0.8873
	SU	0.9100	0.9010	0.9168	0.9100
3	CS	0.8621	0.8604	0.8636	0.8611
	GR	0.8124	0.8126	0.8218	0.8193
	IG	0.8645	0.8643	0.8666	0.8645
	RF	0.8377	0.8303	0.8351	0.8293
	RFW	0.8469	0.8432	0.8442	0.8372
	SU	0.8616	0.8603	0.8608	0.8584
4	CS	0.9008	0.9034	0.9174	0.9111
	GR	0.8530	0.8540	0.9024	0.9006
	IG	0.9107	0.9137	0.9199	0.9135
	RF	0.8318	0.8463	0.8485	0.8462
	RFW	0.8832	0.8651	0.8494	0.8321
	SU	0.9031	0.9044	0.9158	0.9108
5	CS	0.8979	0.9028	0.9028	0.9055
	GR	0.8650	0.8663	0.8929	0.8959
	IG	0.8998	0.9057	0.9034	0.9066
	RF	0.8204	0.8269	0.8436	0.8518
	RFW	0.7245	0.7592	0.8035	0.8214
	SU	0.8942	0.8987	0.9022	0.9044
6	CS	0.8876	0.8862	0.8885	0.8862
	GR	0.8855	0.8852	0.8851	0.8831
	IG	0.8878	0.8867	0.8881	0.8862
	RF	0.8620	0.8608	0.8728	0.8740
	RFW	0.8792	0.8766	0.8802	0.8793
	SU	0.8882	0.8864	0.8882	0.8862
7	CS	0.9244	0.9253	0.9292	0.9248
	GR	0.8875	0.8953	0.9011	0.9093
	IG	0.9338	0.9320	0.9325	0.9290
	RF	0.8825	0.8766	0.8925	0.8914
	RFW	0.8621	0.8643	0.8513	0.8456
	SU	0.9007	0.9051	0.9249	0.9208
8	CS	0.9413	0.9353	0.9409	0.9334
	GR	0.9373	0.9291	0.9367	0.9255
	IG	0.9407	0.9338	0.9408	0.9340
	RF	0.8999	0.8893	0.9156	0.9008
	RFW	0.9080	0.8955	0.9212	0.9033
	SU	0.9391	0.9305	0.9406	0.9318
9	CS	0.9060	0.9003	0.9062	0.8999
	GR	0.9027	0.8989	0.9036	0.8982
	IG	0.9060	0.9002	0.9063	0.9004
	RF	0.8284	0.8217	0.8717	0.8625
	RFW	0.8421	0.8368	0.8953	0.8902
	SU	0.9056	0.9016	0.9058	0.9009
Average		**0.8790**	**0.8777**	**0.8894**	**0.8847**

Table 4. One-way ANOVA Results for KNN

Source	Sum Sq.	d.f.	Mean Sq.	F	*p*-value
A	0.1049	3	0.0350	10.57	0
Error	7.1314	2156	0.0033		
Total	7.2364	2159			

Table 5. One-way ANOVA Results for SVM

Source	Sum Sq.	d.f.	Mean Sq.	F	*p*-value
A	0.0478	3	0.0159	10.85	0
Error	3.1640	2156	0.0015		
Total	3.2118	2159			

In addition, different data sampling techniques and different feature selection techniques will be considered in the context of this study.

References

[1] M. L. Berenson, M. Goldstein, and D. Levine. *Intermediate Statistical Methods and Applications: A Computer Package Approach*. Prentice-Hall, Englewood Cliffs, NJ, 2 edition, 1983.

[2] G. Boetticher, T. Menzies, and T. Ostrand. Promise repository of empirical software engineering data, 2007.

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

[4] Z. Chen, T. Menzies, D. Port, and B. Boehm. Finding the right data for software cost modeling. *IEEE Software*, (22):38–46, 2005.

[5] D. A. Cieslak, N. V. Chawla, and A. Striegel. Combating imbalance in network intrusion datasets. In *Proceedings of 2006 IEEE International Conference on Granular Computing*, pages 732– 737, Athens, Georgia, May 2006.

[6] C. Elkan. The foundations of cost-sensitive learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, page 239?246, 2001.

[7] V. Engen, J. Vincent, and K. Phalp. Enhancing network based intrusion detection for imbalanced data. *International Journal of Knowledge-Based and Intelligent Engineering Systems*, 12(5-6):357–367, 2008.

[8] A. Estabrooks, T. Jo, and N. Japkowicz. A multiple resampling method for learning from imbalanced data sets. *International Journal of Computational Intelligence*, 20(1):18–36, 2004.

[9] K. Gao, T. M. Khoshgoftaar, H. Wang, and N. Seliya. Choosing software metrics for defect prediction: An investigation on feature selection techniques. Technical Report FAU-CSETR-2009-11-2, Florida Atlantic University, Boca Raton, FL, November 2009.

[10] M. A. Hall and G. Holmes. Benchmarking attribute selection techniques for discrete class data mining. *IEEE Transactions on Knowledge and Data Engineering*, 15(6):1437 – 1447, Nov/Dec 2003.

[11] 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*, pages 99–108, Bangalore-Mysore, India, Nov. 16-19 2009.

[12] A. H. Kamal, X. Zhu, A. S. Pandya, S. Hsu, and M. Shoaib. The impact of gene selection on imbalanced microarray expression data. In *Proceedings of the 1st International Conference on Bioinformatics and Computational Biology; Lecture Notes in Bioinformatics; Vol. 5462*, pages 259–269, New Orleans, LA, 2009.

[13] T. M. Khoshgoftaar, M. Golawala, and J. V. Hulse. An empirical study of learning from imbalanced data using random forest. In *Proceedings of the 19th IEEE International Conference on Tools with Artificial Intelligence*, volume 2, pages 310–317, Washington, DC, USA, 2007. IEEE Computer Society.

[14] T. M. Khoshgoftaar, P. Rebours, and N. Seliya. Software quality analysis by combining multiple projects and learners. *Software Quality Journal*, 17(1):25–49, March 2009.

[15] K. Kira and L. A. Rendell. A practical approach to feature selection. In *Proceedings of 9th International Workshop on Machine Learning*, pages 249–256, 1992.

[16] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch. Benchmarking classification models for software defect prediction: A proposed framework and novel findings. *IEEE Transactions on Software Engineering*, 34(4):485–496, July-August 2008.

[17] H. Liu, H. Motoda, and L. Yu. A selective sampling approach to active feature selection. *Artificial Intelligence*, 159(1-2):49–74, November 2004.

[18] H. Liu and L. Yu. Toward integrating feature selection algorithms for classification and clustering. *IEEE Transactions on Knowledge and Data Engineering*, 17(4):491–502, 2005.

[19] R. L. Plackett. Karl pearson and the chi-squared test. *International Statistical Review*, 51(1):59?72, 1983.

[20] Y. Saeys, T. Abeel, and Y. Peer. Robust feature selection using ensemble feature selection techniques. In *Proceedings of the European conference on Machine Learning and Knowledge Discovery in Databases - Part II (2008)*, pages 313–325, 2008.

[21] C. Seiffert, T. Khoshgoftaar, and J. Van Hulse. Improving software-quality predictions with data sampling and boosting. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 39(6):1283–1294, Nov. 2009.

[22] H. Wang, T. M. Khoshgoftaar, K. Gao, and N. Seliya. Mining data from multiple software development projects. In *Proceedings of the 3rd IEEE International Workshop Mining Multiple Information Sources*, pages 551–557, Miami, FL, Dec. 6 2009.

[23] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 2 edition, 2005.

[24] X.-M. Zhao, X. Li, L. Chen, and K. Aihara. Protein classification with imbalanced data. *Proteins: Structure, Function, and Bioinformatics*, 70(4):1125 – 1132, 2007.

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