Combining Feature Selection and Ensemble Learning for Software Quality Estimation

Kehan Gao
Eastern Connecticut State University
83 Windham St., Willimantic, CT 06226
gaok@easternct.edu

Taghi Khoshgoftaar and **Randall Wald**
Florida Atlantic University
777 Glades Rd., Boca Raton, FL 33431
khoshgof@fau.edu; rwald1@fau.edu

Abstract

High dimensionality is a major problem that affects the quality of training datasets and therefore classification models. Feature selection is frequently used to deal with this problem. The goal of feature selection is to choose the most relevant and important attributes from the raw dataset. Another major challenge to building effective classification models from binary datasets is class imbalance, where the minority class has far fewer instances than the majority class. Data sampling (altering the dataset to change its balance level) and boosting (building multiple models, with each model tuned to work better on instances misclassified by previous models) are common techniques for resolving this problem. In particular, ensemble boosting, which integrates sampling with AdaBoost, has been shown to improve classification performance, especially for imbalanced training datasets. In this paper, we investigate approaches for combining feature selection with this ensemble learning (boosting) process. Six feature selection techniques and two forms of the ensemble learning method are examined. We focus on two different scenarios: feature selection performed prior to the ensemble learning process and feature selection performed inside the ensemble learning process. The experimental results demonstrate that performing feature selection inside of ensemble boosting generally performs better than using feature selection prior to ensemble boosting.

Introduction

The quality of training datasets can have a significant impact on the prediction accuracy of classification models in data mining and machine learning projects. Data collection is frequently a loosely controlled process, resulting in many data quality problems such as feature irrelevance and redundancy, data instance conflict and abnormality, and missing values. Data preprocessing is a critical initial step to improve the quality of training datasets.

In this study, we are particularly interested in investigating feature selection in the context of software quality prediction. Software quality prediction is a process of utilizing software metrics such as code-level measurements and defect data to build classification models that are able to estimate the quality of program modules (e.g., classify program modules as either fault-prone (*fp*) or not-fault-prone (*nfp*)) (Song *et al.* 2011). These kind of estimations can help practitioners effectively allocate limited project resources, forcing them to focus on program modules that are of poor quality or likely to have a high number of faults. However, it is often found in the modeling process that not all collected software metrics are useful or have the same contributions to classification results. The objective of feature selection is to choose the most relevant and important attributes from the raw dataset so that prediction performance will be improved, or at least maintained, while learning time is significantly reduced. Recent research (Van Hulse *et al.* 2012) has shown that filter-based feature ranking techniques are simple, fast, and effective methods for dealing with this problem. In this paper, we examine six filter-based feature ranking techniques to select subsets of features.

Another common classification challenge is class imbalance, which refers to datasets where one class has far fewer instances than the other class(es). Unfortunately, usually the minority class is the class of interest. In the context of software quality modeling, class imbalance manifests as *nfp* modules outnumbering *fp* modules. The main disadvantage of such an imbalanced training dataset is that a traditional classification algorithm would tend to classify minority class members (e.g., *fp* modules) as majority class (e.g., *nfp* modules) in order to improve overall prediction accuracy. This type of misclassification implies lost opportunities for correcting faulty modules before the software operation and deployment, resulting in serious consequences and high repair costs.

To alleviate the adverse impacts of imbalanced data on the prediction models, two techniques have been discussed: data sampling and boosting. Although boosting is not specifically developed to handle the class imbalance problem, it has been shown to be very effective in this regard (Seiffert *et al.* 2010). Data sampling is another effective method to cope with the class imbalance problem. Instances from the majority (or minority) class are selected and discarded (or added) until a certain balance (ratio) between the two classes is met. In this study, we use an ensemble boosting approach, in which data sampling is integrated into AdaBoost (Freund & Schapire 1996). As we study two sampling techniques, random undersampling (RUS) and synthetic minority over-

sampling (SMOTE), correspondingly we have two forms of the ensemble boosting technique: RUSBoost and SMOTEBoost.

To deal with both high dimensionality and class imbalance, we study approaches which combine feature selection with an ensemble learning (boosting) method. We investigate two different scenarios: feature selection performed right before the ensemble learning process (external feature selection) and feature selection performed inside the ensemble learning process (internal feature selection). As ensemble learning (boosting) is an iterative process (the number of repetitions is set to 10), internal feature selection performed after sampling inside the process naturally inherits the repetition. To avoid a biased assessment, for external feature selection we use an iterative process too. We first use a data sampling technique to alter the class ratio (balance). Then, we apply a feature ranking technique to the sampled data and rank the features according to their predictive power. We repeat these two steps 10 times, and then combine the rankings using mean (average) aggregation and choose the best set of features. The selected features are sent to the ensemble learning process thereafter.

In the experiments, we use six filter-based feature ranking techniques (chi-squared, information gain, gain ratio, two forms of ReliefF, and symmetrical uncertainty), five base learners (naïve Bayes, multilayer perceptron, k nearest neighbors, support vector machine, and logistic regression), and two sampling approaches (RUS and SMOTE). The experimental results demonstrate that feature selection performed inside the ensemble learning process provides better classification performance than when it is applied prior to the ensemble learning process. In addition, between the two ensemble learning approaches RUSBoost generally outperforms SMOTEBoost; among the six ranking techniques, two forms of ReliefF are the best; and of the five learners, support vector machine and logistic regression results in the best prediction.

Related Work

Feature selection is often used to resolve the high dimensionality problem found in many datasets. Significant research has been dedicated towards feature selection (Liu *et al.* 2010), and applied to a range of fields. Van Hulse et al. propose 11 new threshold-based feature selection techniques and apply them to 17 different bioinformatics datasets. The similarities of the feature selection techniques are analyzed using the Frobenius norm (Van Hulse *et al.* 2012). Yu et al. apply feature selection to gene expression microarray data and study the stability of feature selection via sample weighting (Yu, Han, & Berens 2012). In the context of text classification, Basu and Murthy propose a supervised feature selection approach that develops a similarity between a term and a class (Basu & Murthy 2012).

In addition to excess number of features, many datasets are plagued with the class imbalance problem. Two techniques that have been discussed for alleviating this problem are data sampling and boosting. The simplest form of sampling is random sampling. In addition, a few more intelligent algorithms for sampling data have been proposed (Chawla *et al.* 2002). Another technique for alleviating class imbalance is boosting. The most commonly used boosting algorithm is AdaBoost (Freund & Schapire 1996). Several variations have been proposed to improve AdaBoost's performance on imbalanced data. Two promising techniques are SMOTEBoost (Chawla *et al.* 2003) and RUSBoost (Seiffert *et al.* 2010). Each combines a sampling technique (SMOTE or RUS) with AdaBoost, resulting in a highly effective hybrid approach to learning from imbalanced data.

While a great deal of work has been done for feature selection and data sampling separately, research working on both together is starting to receive more attention. Yang et al. propose an ensemble-based wrapper approach for feature selection from data with highly imbalanced class distribution (Yang *et al.* 2013). They create multiple balanced datasets from the original imbalanced data via sampling, and then evaluate feature subsets using an ensemble of base classifiers each trained on a balanced dataset. Khoshgoftaar et al. use data sampling along with feature ranking techniques to deal with the high dimensionality and class imbalance problems in the context of software quality classification (Khoshgoftaar *et al.* 2013). The experimental results demonstrate that using feature ranking along with data sampling is more effective than using each technique individually for improving software defect prediction.

Methodology

Filter-Based Feature Ranking Techniques

The goal of feature ranking is to score each feature according to a particular method, allowing the selection of the best features. The six filter-based feature ranking techniques used in this work include: Chi-Squared (CS), Information Gain (IG), Gain Ratio (GR), two types of ReliefF (RF and RFW), and Symmetrical Uncertainty (SU).

The chi-squared (CS) test is used to examine whether the two variables are independent. CS is more likely to find significance to the extent that (1) the relationship is strong, (2) the sample size is large, and/or (3) the number of values of the two associated features is large. Information gain, gain ratio, and symmetrical uncertainty are measures based on the concept of entropy from information theory (Witten & Frank 2011). 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 with 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 counter 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 (Kira & Rendell 1992). ReliefF is an extension of the Relief algorithm that can handle noise and

multiclass datasets, and is implemented in the WEKA tool (Witten & Frank 2011). 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. The number of the features selected in the feature subsets is set to $\lceil \log_2 n \rceil$ (Khoshgoftaar *et al.* 2013), where n is the number of independent attributes in the original dataset.

Sampling Techniques

The two data sampling techniques used in this study are Random UnderSampling (RUS) and Synthetic Minority Oversampling TEchnique (SMOTE). RUS alleviates the problem with class imbalance in a dataset by randomly discarding instances from the majority class. SMOTE is an intelligent oversampling method proposed by Chawla et al. (Chawla *et al.* 2003). It adds new, artificial minority examples by extrapolating between preexisting minority instances rather than simply duplicating original examples. The newly created instances cause the minority regions of the feature-space to become fuller and more general.

The Iterative Feature Selection Technique

The iterative feature selection method is designed to deal with feature selection for imbalanced data. It consists of two basic steps: (1) using a technique to balance data, and (2) applying a filter-based feature ranking technique to the balanced data and ranking all the features according to their predictive powers (scores). In order to avoid biased results generated due to the sampling process, we repeat the two steps k times ($k = 10$ in this study) and aggregate k rankings using the average (mean) ranks. Finally, the best set of attributes is selected. More detailed information about this technique can be found in (Khoshgoftaar *et al.* 2013).

RUSBoost & SMOOTEBoost

RUSBoost combines random undersampling (RUS) and boosting for improving classification performance. Boosting is a meta-learning technique designed to improve the classification performance of weak learners by iteratively creating an ensemble of weak hypotheses which are combined to predict the class of unlabeled examples. This study uses AdaBoost (Freund & Schapire 1996), a well-known boosting algorithm shown to improve the classification performance of weak classifiers. Initially, all examples in the training dataset are assigned equal weights. During each iteration of AdaBoost, a weak hypothesis is formed by the base learner. The error associated with the hypothesis is calculated and the weight of each example is adjusted such that misclassified examples have their weights increased while correctly classified examples have their weights decreased. Therefore, subsequent iterations of boosting will generate hypotheses that are more likely to correctly classify the previously mislabeled examples. After all iterations are completed a weighted vote of all hypotheses are used to assign a class to unlabeled examples. In this study, the boosting algorithm is performed using 10 iterations. RUSBoost applies the same steps as the regular boosting, but prior to constructing the weak hypothesis during each round of boosting, random undersampling is applied to the training data to achieve a more balanced class distribution. The weak learners used in this work are NB, MLP, KNN, SVM, and LR, and these will be discussed in the next section. The procedure of RUSBoost is described in part of Figure 1. More details about the algorithm can be found in (Seiffert *et al.* 2010).

SMOTEBoost has the same mechanism as RUSBoost. Instead of using RUS, SMOTEBoost combines synthetic minority oversampling (SMOTE) with boosting in order to improve classification performance on imbalanced data. For the complete SMOTEBoost algorithm, one can refer to the RUSBoost algorithm (Seiffert *et al.* 2010), replacing RUS with SMOTE.

Learners

The software defect prediction models in this study are built using five different classification algorithms, including Naïve Bayes (NB) (Witten & Frank 2011), MultiLayer Perceptron (MLP) (Haykin 1998), K Nearest Neighbors (KNN) (Witten & Frank 2011), Support Vector Machine (SVM) (Shawe-Taylor & Cristianini 2000), and Logistic Regression (LR) (Witten & Frank 2011). Due to space limitations, we refer interested readers to these references to understand how these commonly-used learners function. The WEKA 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.

Performance Metric

One of the most popular methods for evaluating the performance of learners built using imbalanced data is the *receiver operating characteristic* (Fawcett 2006), or ROC, curve. ROC curves graph true positive rate on the y-axis versus the false positive rate on the x-axis. The ROC curve illustrates the performance of a classifier across the complete range of possible decision thresholds, and accordingly does not assume any particular misclassification costs or class prior probabilities. The area under the ROC curve (AUC) is used to provide a single numerical metric for comparing model performances. The AUC value ranges from 0 to 1. A model

Table 1: Data characteristics

Data	#Attri.	Total Inst.	*fp* Inst.		*nfp* Inst.	
		#	#	%	#	%
SP1	42	3649	229	6	3420	94
SP2	42	3981	189	5	3792	95
SP3	42	3541	47	1	3494	99
SP4	42	3978	92	2	3886	98

with more predictive power results in an AUC value closer to 1.

Datasets

Experiments conducted in this study used software metrics and defect data collected from a very large telecommunications software system (denoted as LLTS). The software measurement datasets of LLTS consist of 42 software metrics, including 24 product metrics, 14 process metrics, and four execution metrics. More details about these software metrics can be found in (Khoshgoftaar, Bullard, & Gao 2009). The dependent variable is the class of the program module. A module with one or more faults is considered *fp*, and *nfp* otherwise. The LLTS software system consists of four successive releases labeled SP1, SP2, SP3, and SP4. Table 1 shows the characteristics of the LLTS datasets.

Experiments

Design

The main objective of this study is to evaluate the two different scenarios of feature selection combined with the ensemble learning approach on the classification models in the context of software defect prediction. The procedure of the approach is shown in Figure 1. Two different scenarios are presented.

- **Scenario 1**: feature selection performed ***prior to*** the ensemble learning process (external feature selection)
- **Scenario 2**: feature selection performed ***inside*** the ensemble learning process (internal feature selection)

For the second scenario (internal feature selection), feature selection is applied inside of the iterative process of the ensemble boosting method. The number of repetitions is set to 10 in the experiment. For the first scenario (external feature selection), feature selection is performed outside the iterative process. In order to avoid biased evaluation of the two different scenarios, we apply the iterative process as discussed previously to the external feature selection. The number of repetitions is also set to 10. When used internally, feature selection is applied once per iteration of the boosting process (so, 10 times in total).

In the experiment, six feature selection methods (CS, IG, GR, RF, RFW, and SU) and two data sampling approaches (RUS and SMOTE) are applied. The post-sampling class ratio is set to 50:50 between *fp* and *nfp* modules throughout the experiment. In addition, the weak learners adopted in the boosting process are NB, MLP, KNN, SVM and LR.

For all experiments, we employ ten runs of five-fold cross-validation. That is, for each run the data is randomly divided into five folds, one of which is used as the test data while the other four folds are used as training data. All the preprocessing steps (feature selection and data sampling) are done on the training dataset. The processed training data is then used to build the classification model and the resulting model is applied to the test fold. This cross-validation is repeated five times (the folds), with each fold used exactly once as the test data. The five results from the five folds then are collected to produce a single estimation.

Table 2: Classification performance

Learner	Ranker	RUSBoost			SMOTEBoost		
		EFS	IFS	p	EFS	IFS	p
NB	CS	0.7617	0.7672	0.43	0.7855	0.7883	0.56
	GR	0.7498	0.7497	0.99	0.7321	0.7432	0.08
	IG	0.7602	0.7664	0.36	0.7853	0.7862	0.84
	RF	0.7951	0.8007	0.13	0.7962	**0.8068**	**0.02**
	RFW	0.7858	**0.8028**	**0.00**	0.7949	0.8020	0.12
	SU	0.7490	0.7606	0.06	0.7640	0.7723	0.13
MLP	CS	0.7919	**0.8090**	**0.00**	0.7918	0.7906	0.79
	GR	0.7868	**0.8028**	**0.00**	0.7377	**0.7736**	**0.00**
	IG	0.7907	**0.8069**	**0.00**	0.7911	0.7894	0.76
	RF	0.8148	0.8183	0.25	0.7922	0.7965	0.33
	RFW	0.8081	**0.8188**	**0.00**	0.7913	0.7983	0.07
	SU	0.7842	**0.8043**	**0.00**	0.7619	**0.7886**	**0.01**
KNN	CS	0.7722	**0.8088**	**0.00**	0.7420	0.7452	0.65
	GR	0.7804	**0.8088**	**0.00**	0.6231	**0.7447**	**0.00**
	IG	0.7724	**0.8099**	**0.00**	0.7394	0.7502	0.12
	RF	0.7948	**0.8095**	**0.00**	0.7361	**0.7715**	**0.00**
	RFW	0.7928	**0.8111**	**0.00**	0.7328	**0.7644**	**0.00**
	SU	0.7727	**0.8077**	**0.00**	0.7045	**0.7517**	**0.00**
SVM	CS	0.8054	**0.8250**	**0.00**	0.8177	**0.8248**	**0.02**
	GR	0.8040	**0.8256**	**0.00**	0.7697	**0.8042**	**0.00**
	IG	0.8059	**0.8257**	**0.00**	0.8176	**0.8244**	**0.04**
	RF	0.8290	0.8308	0.39	0.8233	0.8274	0.09
	RFW	0.8222	**0.8322**	**0.00**	0.8216	**0.8268**	**0.05**
	SU	0.8002	**0.8247**	**0.00**	0.7903	**0.8171**	**0.00**
LR	CS	0.8034	**0.8226**	**0.00**	0.8179	**0.8250**	**0.01**
	GR	0.7998	**0.8255**	**0.00**	0.7643	**0.8041**	**0.00**
	IG	0.8022	**0.8231**	**0.00**	0.8179	0.8238	0.07
	RF	0.8265	0.8300	0.18	0.8226	0.8266	0.08
	RFW	0.8188	**0.8282**	**0.00**	0.8208	**0.8260**	**0.03**
	SU	0.7970	**0.8229**	**0.00**	0.7883	**0.8168**	**0.00**

Results and Analysis

The results of two forms of ensemble learning method (RUSBoost and SMOTEBoost) are presented in Table 2, each containing the results for all five learners. Note that Although ten runs of five-fold cross validation were used on each dataset separately, the results present the average AUCs over the four datasets. The table shows the classification performance of two different scenarios: EFS (External Feature Selection) and IFS (Internal Feature Selection) across all six feature ranking techniques. For each combination of learner, ranker, and ensemble boosting method, we compared the classification performance of EFS and IFS using the Student's t-test. The t-test examines the null hypothesis that the population means related to two independent group samples are equal against the alternative hypothesis that the

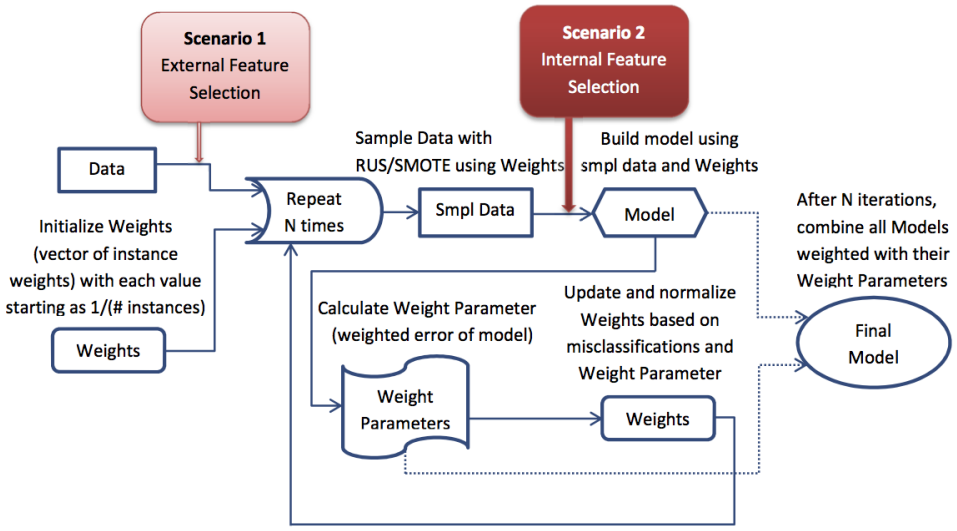


Figure 1: Two scenarios of feature selection combined with the ensemble learning approach

Table 3: ANOVA results

Source	Sum Sq.	d.f.	Mean Sq.	F	p
Learning process	0.6750	3	0.2250	287.35	0.000
Ranker	0.7215	5	0.1443	184.28	0.000
Learner	2.0693	4	0.5173	660.66	0.000
Error	3.7484	4787	0.0008		
Total	7.2142	4799			

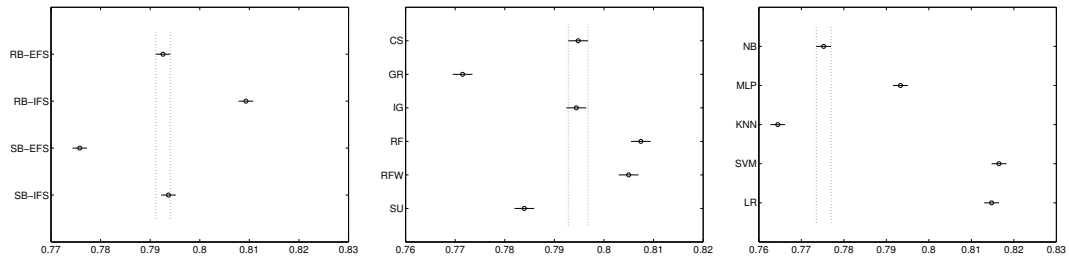
population means are different. All t-test results are presented in the list table, with the p-value provided for each pairwise comparison. The significance level is set to 0.05. When the p-value is less than 0.05, the two group means are significantly different from each other. The better performer is highlighted with **bold** in the table. The results demonstrate that feature selection performed inside the ensemble learning approach displays better or similar prediction quality than when it is performed outside the ensemble learning process. This phenomena is observed for both ensemble learning methods and across all five learners and six ranking techniques.

We further conducted a three-way analysis of variance (ANOVA) test on the classification performance for the datasets to examine if the performance difference is statistically significant. The three factors are A: the two types of ensemble learning approach (RUSBoost (RB) and SMOTEBoost (SB)), each along with two scenarios (EFS and IFS), B: the six feature ranking techniques, and C: the five learners. 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 3 shows the ANOVA results. All the p-values are less than the cutoff 0.05 for all factors, meaning that for each main factor the alternate hypothesis is accepted.

We continued carrying out a multiple comparison test on each main factor with Tukey's honestly significant difference (HSD) criterion. Figure 2 shows the multiple comparisons for all three factors. The figures display graphs with each group mean represented by a symbol ($\circ$) 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 multiple comparison results demonstrate that for both ensemble learning methods, feature selection performed inside the boosting process results in better classification performance than the situation when it is applied prior to the boosting process. In addition, the RUSBoost method presents better prediction behavior than its counterpart (SMOTEBoost) in both scenarios. As to the six rankers, two forms of ReliefF (RF and RFW) show the best performance, followed by CS and IG, then SU, and finally GR. For the five learners, SVM and LR exhibit significantly better prediction performance than the other three learners. For these three inferior learners, the order from best to worst in terms of their classification performance is MLP, NB, and KNN. In the experiment, all statistical analysis was performed using Matlab. The assumptions for constructing ANOVA and Tukey's HSD models were validated.

Conclusion

This paper presents feature selection working along with an ensemble learning approach, in which AdaBoost incorporates data sampling. We examine two sampling methods, random undersampling (RUS) and synthetic minority oversampling (SMOTE), which gives us two forms of the ensemble boosting technique: RUSBoost and SMOTEBoost. We investigate two different scenarios: feature selection performed right before the ensemble learning process and feature selection performed inside the ensemble learning process. In the experiments, the proposed method is applied to a group of datasets from a real-world software system by using six filter-based feature ranking techniques and five base



(a) Learning process (b) Ranker (c) Learner

Figure 2: Multiple comparison

learners. The results demonstrate that feature selection performed inside the ensemble learning approach results in better classification performance than when it is applied prior to the ensemble learning approach. As to two ensemble learning approaches, RUSBoost performs better than or similarly to SMOTEBoost. In addition, of the six ranking techniques two forms of ReliefF (RF and RFW) provide the best performance, and among the fiver learners, support vector machine and logistic regression perform much better than the multilayer perceptron, naïve Bayes, and k nearest neighbor algorithms. Future work will involve conducting additional empirical studies with software measurement and defect data from other software projects. More comparative study using other sampling approaches and rankers will also be considered in the future.

References

Basu, T., and Murthy, C. 2012. Effective text classification by a supervised feature selection approach. In *Proceedings of the 12th International Conference on Data Mining Workshops (ICDMW)*, 918–925.

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

Chawla, N. V.; Lazarevic, A.; Hall, L. O.; and Bowyer, K. 2003. SMOTEBoost: Improving prediction of the minority class in boosting. In *Proceedings of Principles of Knowledge Discovery in Databases*, 107–119.

Fawcett, T. 2006. An introduction to ROC analysis. *Pattern Recognition Letters* 27(8):861–874.

Freund, Y., and Schapire, R. E. 1996. Experiments with a new boosting algorithm. In *Proceedings of the 13th International Conference on Machine Learning*, 148–156.

Haykin, S. 1998. *Neural Networks: A Comprehensive Foundation*. Prentice-Hall, 2 edition.

Khoshgoftaar, T. M.; Gao, K.; Napolitano, A.; and Wald, R. 2013. A comparative study of iterative and non-iterative feature selection techniques for software defect prediction. *Information Systems Frontiers* 1–22.

Khoshgoftaar, T. M.; Bullard, L. A.; and Gao, K. 2009. Attribute selection using rough sets in software quality classification. *International Journal of Reliability, Quality and Safty Engineering* 16(1):73–89.

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

Liu, H.; Motoda, H.; Setiono, R.; and Zhao, Z. 2010. Feature selection: An ever evolving frontier in data mining. In *Proceedings of the 4th International Workshop on Feature Selection in Data Mining*, 4–13.

Seiffert, C.; Khoshgoftaar, T. M.; Van Hulse, J.; and Napolitano, A. 2010. Rusboost: A hybrid approach to alleviating class imbalance. *IEEE Trans. on Systems, Man and Cybernetics, Part A: Systems and Humans* 40(1):185–197.

Shawe-Taylor, J., and Cristianini, N. 2000. *Support Vector Machines*. Cambridge University Press, 2 edition.

Song, Q.; Jia, Z.; Shepperd, M.; Ying, S.; and Liu, J. 2011. A general software defect-proneness prediction framework. *IEEE Trans. on Software Engineering* 37(3):356–370.

Van Hulse, J.; Khoshgoftaar, T. M.; Napolitano, A.; and Wald, R. 2012. Threshold-based feature selection techniques for high-dimensional bioinformatics data. *Network Modeling Analysis in Health Informatics and Bioinformatics* 1(1-2):47–61.

Witten, I. H., and Frank, E. 2011. *Data Mining: Practical Machine Learning Tools and Techniques*. Burlington, MA: Morgan Kaufmann, 3 edition.

Yang, P.; Liu, W.; Zhou, B. B.; Chawla, S.; and Zomaya, A. Y. 2013. Ensemble-based wrapper methods for feature selection and class imbalance learning. In *Proceedings of the 17th Pacific-Asia Conference, PAKDD 2013, Gold Coast, Australia, Part I, Lecture Notes in Computer Science 7818*, 544–555. Springer-Verlag.

Yu, L.; Han, Y.; and Berens, M. E. 2012. Stable gene selection from microarray data via sample weighting. *IEEE/ACM Transactions On Computational Biology and Bioinformatics* 9(1):262–272.