2009 International Conference on Machine Learning and Applications

Feature Selection with Imbalanced Data for Software Defect Prediction

Taghi M. Khoshgoftaar
Florida Atlantic University
Boca Raton, Florida 33431
taghi@cse.fau.edu

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

Abstract

In this paper, we study the learning impact of data sampling followed by attribute selection on the classification models built with binary class imbalanced data within the scenario of software quality engineering. We use a wrapper-based attribute ranking technique to select a subset of attributes, and the random undersampling technique (RUS) on the majority class to alleviate the negative effects of imbalanced data on the prediction models. The datasets used in the empirical study were collected from numerous software projects. Five data preprocessing scenarios were explored in these experiments, including: (1) training on the original, unaltered fit dataset, (2) training on a sampled version of the fit dataset, (3) training on an unsampled version of the fit dataset using only the attributes chosen by feature selection based on the unsampled fit dataset, (4) training on an unsampled version of the fit dataset using only the attributes chosen by feature selection based on a sampled version of the fit dataset, and (5) training on a sampled version of the fit dataset using only the attributes chosen by feature selection based on the sampled version of the fit dataset. We compared the performances of the classification models constructed over these five different scenarios. The results demonstrate that the classification models constructed on the sampled fit data with or without feature selection (case 2 and case 5) significantly outperformed the classification models built with the other cases (unsampled fit data). Moreover, the two scenarios using sampled data (case 2 and case 5) showed very similar performances, but the subset of attributes (case 5) is only around 15% or 30% of the complete set of attributes (case 2).

1. Introduction

Software measurement data (such as code metrics, execution traces, historical code changes, and defect databases) that are collected during the software development process include valuable information about a software project's status, progress, quality, and evolution. Practitioners can utilize those software measurement data and various data mining techniques to extract knowledge that facilitates better project management and higher quality software production. In this paper, we concentrate on utilizing software metrics such as code-level measurements and defect data, to build software quality prediction models that estimate the quality of program modules, e.g., fault-prone (*fp*) and not fault-prone (*nfp*). With such a quality estimation, practitioners can prioritize the limited project resources and focus on inspecting program modules that are either of poor quality or likely to have a high number of defects.

A great deal of effort has been dedicated toward improving the predictive accuracy of software quality classification models in recent years [7, 11]. It has been shown in some studies that the performance of these models is affected by either the learner(s) used in the modeling process or the quality of the data that are involved in the model building process. We are more interested in the latter, namely, quality of data, for this study. We investigated attribute selection by the means of a *wrapper-based feature ranking* technique [6]. The idea of this method is using a classifier (learner) with a single independent attribute (as well as a class attribute) to build a classification model and select attributes according to their individual predictive capability. The predictive capability of the attribute can be measured in terms of various performance metrics. We used four different performance metrics in the study. They are overall accuracy, arithmetic mean, area under ROC, and area under PRC.

In software quality classification, class imbalance occurs when not fault-prone modules significantly outnumber fault-prone program modules in a given dataset. Generally, for a binary classification problem, two types of errors can occur: Type I and Type II. A Type I error or misclassification refers to a *nfp* module misclassified as *fp*, while a Type II error refers to a *fp* module misclassified as *nfp*. Class imbalance usually results in more prediction errors on Type II, especially when the different costs of misclassifications are not considered during the classification modeling process. In software engineering, a Type II error is usually far more

978-0-7695-3926-3/09 $26.00 © 2009 IEEE
DOI 10.1109/ICMLA.2009.18

expensive than a Type I error. The reason is that the cost of a Type I error may involve wasted effort inspecting high quality program modules, while a Type II error may indicate a missed opportunity of correcting a faulty module prior to operations. To alleviate the adverse impacts of imbalanced data on the prediction models, we used the random undersampling (RUS) technique for reducing the majority class. RUS randomly discards instances from the majority class (*nfp* class). With this strategy, the class distribution can be more balanced, however, important information may be lost when examples are discarded at random. The experiments of this study were carried out on eight datasets from real-world software systems. The distributions of the two classes (*fp* and *nfp*) are significantly different. In seven of the eight datasets, between 1% and 10% of the instances are in the *fp* class, while in the remaining dataset, 20% are. That is why we consider using the strategy of data sampling followed by attribute selection to deal with the problem. After sampling, the new dataset contains 65% *nfp* modules and 35% *fp* modules [9].

The main goal of this study is to investigate the learning impact of the data preprocessing approach that uses data sampling (RUS) followed by feature ranking on the classification models built with imbalanced data. For a given fit dataset, we can generate three different subsets when applying attribute selection on the original, unsampled fit dataset and the sampled fit dataset. The three cases include: (a) the training data that is created by selecting the attributes from the original, unsampled fit data, (b) the attribute subset selected based on the sampled fit data, but the final training data formed with instances from the original, unsampled fit data, and (c) the training data that is created by selecting the attributes from the sampled fit data and the selected sampled fit data used to train the model. In addition, we consider two other cases for comparison purposes, one is the original, unsampled fit dataset and the other is the full, sampled fit dataset. In total, five different scenarios of the training datasets are formed after a sequence of data preprocessing activities (three with subset of attributes plus two with complete set of attributes). We compare the performances of classification models constructed over the five scenarios of the training datasets. The SVM (Support Vector Machine) classifier (learner) is used during the attribute ranking and classification modeling process. The experimental results demonstrate that data sampling significantly improved the classification performance. In other words, the classification models built on the sampled fit data regardless of using the complete set of attributes or using the subset of attributes showed significantly better performance than those built with the unsampled training datasets. Moreover, between the two sampled fit data cases, although they presented very similar classification performance, the attribute subsets held only around 15% or 30% of the complete set of attributes. This would greatly benefit future software quality assurance efforts on metrics collection, model construction, and model validation of similar systems. To our knowledge, this is the first study to consider both data sampling and feature selection together in software quality classification.

The rest of the paper is organized as follows. Section 2 provides more detailed information about the attribute ranking technique, sampling strategy, and the performance metrics used in the study. The datasets used in the experiments are described in Section 3. Section 4 presents the experimental results. Finally, the conclusion and future work are summarized in Section 5.

2. Methodology

2.1. Feature Ranking Technique

Attribute selection (feature selection) is a process of selecting a subset of relevant attributes for building robust learning models. It includes various divisions. Some references divide attribute selection algorithms into *wrappers* and *filters* [10]. Wrappers refer to algorithms that use feedback from an induction (learning) algorithm to determine which attribute(s) to use in building a classification model, whereas, in filters, the attributes are selected using a method that is independent of an induction algorithm to determine which attributes are most relevant. In addition, attribute selection can be categorized into *feature ranking* and *feature subset selection* [6]. Feature ranking evaluates attributes individually and ranks attributes according to their individual predictive power. Feature subset selection approaches select subsets of attributes that collectively have good predictive capability.

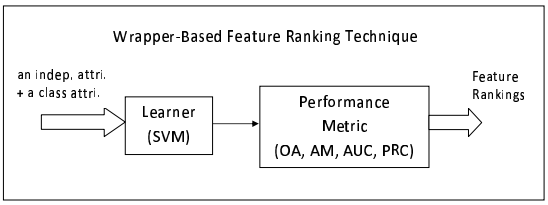


Figure 1. Wrapper-Based Feature Ranking

In this study, we used a wrapper-based feature ranking technique. Figure 1 presents the overall structure of the process. The technique consists of two parts: a learner (classifier) and a performance metric. We used the Support Vector Machine (SVM) learner implemented in the WEKA tool [5, 12] and four different performance metrics (discussed in the next section) for this feature ranking technique. The

WEKA default settings were employed, except for changing the complexity parameter, `c`, to '5.0' and setting the `buildLogisticModels` parameter to 'true' to produce proper probability estimates. Changes to the default parameter values in WEKA were done only when experimentation showed a general improvement in the classifier performance across all datasets based on preliminary analysis.

In the feature ranking technique, every attribute in a given fit (training) dataset was used individually in the model building process. For example, we used the first attribute of the fit data set as the single independent variable and the class attribute, *fp* or *nfp*, as the dependent variable. We used three-fold cross-validation to build the model and assessed the performance of the model based on four performance metrics. Then, we repeated for the second attribute, and so on. In total, we obtained n scores for each performance metric, where n is the number of independent attributes in a given dataset. We ranked the attributes based on the different performance metrics (scores). Then, we selected the top $\lceil \log_2 n \rceil$ attributes according to a given performance metric based ranking and used them as the set of the selected attributes. The reasons why we selected the top $\lceil \log_2 n \rceil$ features include (1) related literature lacks guidance on the number of features that should be selected when using a feature ranking technique; (2) our recent prior study [8] showed that it was appropriate to use $\lceil \log_2 n \rceil$ features when using WEKA to build random forests learners for binary classification in general and imbalanced datasets in particular. Although we used a different learner in this study, a preliminary study showed that $\lceil \log_2 n \rceil$ is still a good choice for various learners; and (3) a software engineering expert with more than 20 years experience recommended selecting $\lceil \log_2 n \rceil$ attributes for software systems such as those in our study. After the attribute selection, SVM was used again, but this time to build software quality classification models. The performances of the classification models were evaluated in terms of AUC, a commonly used performance metric in software engineering, data mining, and machine learning.

SVM was used in the case study because it is a well-known classifier in data mining and has been widely used in various classification problems. In addition, SVM itself does not possess the capability of attribute selection. Other learners besides SVM, such as naïve Bayes, multilayer perceptron, instance-based learning, and logistic regression have been investigated in the study too. However, due to space limitation, the results based on those learners are not presented in the paper. In addition, it is worthwhile to note that SVM in the study played two roles: a learner in the feature ranking process and a classifier in the program module classification process.

2.2. Performance Metrics

A binary classification problem, such as fault-prone (positive) and not fault-prone (negative), has four possible prediction outcomes: true positive (TP) (i.e., correctly classified positive instance), false positive (FP) (i.e., negative instance classified as positive), true negative (TN) (i.e., correctly classified negative instance), and false negative (FN) (i.e., positive instance classified as negative). The four values form the basis for several other performance measures that are well known and commonly used for classifier evaluation.

The **Overall Accuracy** (OA) provides a single value that ranges from 0 to 1. It can be calculated by the equation, $OA = (|TP| + |TN|)/N$, where N represents the total number of instances in a dataset. While the overall accuracy allows for easier comparisons of model performance, it is often not considered to be a reliable performance metric, especially in the presence of class imbalance [1].

The **Arithmetic Mean** (AM) is a single-value performance measure that ranges from 0 to 1, and a perfect classifier provides a value of 1. The arithmetic mean is $(TPR + TNR)/2$ [1].

The **Area Under the ROC** (receiver operating characteristic) curve (i.e., AUC) is a single-value measurement that originated from the field of signal detection. The value of the AUC ranges from 0 to 1. The ROC curve is used to characterize the trade-off between true positive rate (TPR) and false positive rate (FPR)[2] [4]. A classifier that provides a large area under the curve is preferable over a classifier with a smaller area under the curve.

The **Area Under the Precision-Recall Curve** (PRC) is a single-value measure, ranging from 0 to 1. The PRC diagram depicts the trade off between recall and precision[3] [3]. A perfect classifier results in an area under the PRC of 1.

2.3. Random Undersampling Technique

The Random UnderSampling (RUS) technique alleviates the problem with class imbalance in a dataset by randomly discarding instances from the majority class (*nfp* class for our study). With this strategy, the class distribution can be more balanced, however, important information may be lost when examples are eliminated at random. Some of our recent work [9] recommends that an appropriate ratio between the majority class (*nfp*) and the minority class (*fp*) is 65% to 35% when RUS is used, especially in the case of high-assurance and mission critical systems. Therefore, we used the same ratio for this study. We implemented our proposed

[1] $TPR = \frac{|TP|}{|TP|+|FN|}$ and $TNR = \frac{|TN|}{|FP|+|TN|}$
[2] $FPR = \frac{|FP|}{|FP|+|TN|}$
[3] $Recall = \frac{|TP|}{|TP|+|FN|}$ and $Precision = \frac{|TP|}{|TP|+|FP|}$

wrapper-based feature ranking procedure and random undersampling technique in the WEKA tool.

3. Data Set Description

We carried out the experiments on eight datasets, in which four datasets are from a very large legacy telecommunications software system (denoted as LLTS) and the other four are from NASA software projects: CM1, JM1, MW1, and PC1. The four NASA software project datasets were made available through the Metrics Data Program at NASA (http://mdp.ivv.nasa.gov/).

The LLTS software system was developed in a large organization by professional programmers using PROTEL, a proprietary high level procedural language (similar to C). The system consists of four successive releases of new versions of the system, and each release was comprised of several million lines of code. We refer to these four releases as TC1, TC2, TC3, and TC4. Each set of associated source code files was considered as a program module. The LLTS datasets consist of 42 software metrics, including 24 product metrics, 14 process metrics, and 4 execution metrics. All the 42 metrics are numerical metrics. The dependent variable is the class of the software module. The fault-proneness is based on a selected threshold, i.e., modules with one or more faults were considered as *fp*, *nfp* otherwise. One can refer to our recent work [7] for a more detailed discussion about the datasets.

The NASA datasets included software measurement data and associated error (fault) data collected at the function, subroutine, or method level. Hence, for the software systems, a function, subroutine, or method is considered as a software module or an instance in the data set. All the four projects were written in C. The CM1 project is a science instrument system used for mission measurements. The JM1 project is a real-time ground system that uses simulations to generate predictions for missions. The MW1 project is the software from a zero gravity experiment related to combustion. The PC1 project is flight software from an earth orbiting satellite that is no longer operational. The fault data collected for the software systems represent faults detected during software development. Each module in the respective datasets was characterized by 21 software measurements with numerical format, which included 13 basic software metrics and 8 derived Halstead metrics. The quality of the modules is described by their class labels, i.e., *nfp* and *fp*. A module was considered *fp* if it had one or more software faults, and *nfp* otherwise. The derived Halstead metrics were not used in our case studies.

Table 1 summarizes the numbers of the *fp* and *nfp* modules and their percentages in each data set. The original JM1 NASA data contained some inconsistent modules, i.e., those with identical software measurements but with different class labels. After eliminating all those modules and those with missing values, we had 8850 modules left for JM1.

Table 1. Dataset Summary

Dataset	# of nfp Modules	# of fp Modules	Total Modules
TC1	3420	229	3649
	93.72%	6.28%	
TC2	3792	189	3981
	95.25%	4.75%	
TC3	3494	47	3541
	98.67%	1.33%	
TC4	3886	92	3978
	97.69%	2.31%	
CM1	457	48	505
	90.50%	9.50%	
JM1	7163	1687	8850
	80.94%	19.06%	
MW1	372	31	403
	92.31%	7.69%	
PC1	1031	76	1107
	93.13%	6.87%	

4. Experiments

Since the class distributions of *fp* and *nfp* are significantly different, we consider using the RUS technique to sample the data. After data sampling, each new (sampled) dataset has 65% *nfp* modules and 35% *fp* modules. As mentioned previously, the experiments were performed to discover the impact of the data preprocessing strategy that uses data sampling followed by attribute selection, on the classification models constructed with imbalanced data. In this study, we used SVM as a feature ranking learner. Each software attribute and the class attribute (*fp* and *nfp*) from a given dataset were used to train a SVM classification model. For every round of feature ranking, we built 42 classification models (one for each attribute) for each LLTS dataset and 13 classification models for each NASA dataset. We evaluated the models with four different performance metrics, OA, AM, AUC, and PRC, and ranked the attributes accordingly. An attribute that achieves a higher performance measurement indicates that it is more relevant to the class attribute. We selected the top six attributes for the four LLTS datasets, and top four attributes for the four NASA datasets. SVM was used again but this time as a classifier to build a number of classification models with the selected attribute subsets.

As presented in Section 1, for a given fit dataset, three scenarios of the training datasets that contain only a subset of the attributes can be produced after a set of data preprocessing activities. Case 1: the attributes are ranked based on the original, unsampled fit data and the top $\lceil \log_2 n \rceil$ attributes are selected accordingly. The fit dataset formed in this scenario is denoted by `NS`; Case 2: the attributes are ranked based on the sampled fit data and the top $\lceil \log_2 n \rceil$

Table 2. Performance of SVM in terms of AUC

Data	Ranking in terms of OA			Full attributes	
	NS	RUS0	RUS1	NS	RUS
TC1	0.6441	0.6784	0.7981	0.7448	0.8160
TC2	0.6345	0.6957	0.8256	0.6863	0.8368
TC3	0.5927	0.6769	0.8149	0.5518	0.7711
TC4	0.6346	0.6381	0.8062	0.6329	0.8076
CM1	0.6481	0.6631	0.7705	0.6997	0.7988
JM1	0.6743	0.6404	0.7131	0.6816	0.7166
MW1	0.6651	0.6808	0.7593	0.7222	0.7459
PC1	0.6504	0.6716	0.7310	0.6853	0.7539
Data	Ranking in terms of AM			Full attributes	
	NS	RUS0	RUS1	NS	RUS
TC1	0.6412	0.6571	0.7986	0.7448	0.8160
TC2	0.6280	0.6959	0.8252	0.6863	0.8368
TC3	0.5927	0.6889	0.8145	0.5518	0.7711
TC4	0.6337	0.6401	0.8085	0.6329	0.8076
CM1	0.6609	0.6704	0.7744	0.6997	0.7988
JM1	0.6720	0.6225	0.7104	0.6816	0.7166
MW1	0.6628	0.6789	0.7570	0.7222	0.7459
PC1	0.6603	0.6579	0.7323	0.6853	0.7539
Data	Ranking in terms of AUC			Full attributes	
	NS	RUS0	RUS1	NS	RUS
TC1	0.6783	0.6824	0.7869	0.7448	0.8160
TC2	0.6994	0.6749	0.8183	0.6863	0.8368
TC3	0.6489	0.6682	0.8186	0.5518	0.7711
TC4	0.6696	0.6563	0.7969	0.6329	0.8076
CM1	0.6368	0.6848	0.7757	0.6997	0.7988
JM1	0.6631	0.6600	0.7169	0.6816	0.7166
MW1	0.6771	0.6708	0.7583	0.7222	0.7459
PC1	0.6883	0.6579	0.7374	0.6853	0.7539
Data	Ranking in terms of PRC			Full attributes	
	NS	RUS0	RUS1	NS	RUS
TC1	0.6361	0.6709	0.7965	0.7448	0.8160
TC2	0.6744	0.6814	0.8213	0.6863	0.8368
TC3	0.6736	0.6920	0.8163	0.5518	0.7711
TC4	0.6745	0.6654	0.8040	0.6329	0.8076
CM1	0.6686	0.6757	0.7753	0.6997	0.7988
JM1	0.6663	0.6509	0.7140	0.6816	0.7166
MW1	0.6690	0.6717	0.7550	0.7222	0.7459
PC1	0.6426	0.7042	0.7369	0.6853	0.7539

attributes are selected from the original, unsampled fit data. The fit dataset created in this scenario is denoted by `RUS0`; and Case 3: the attributes are ranked based on the sampled fit data and the top $\lceil \log_2 n \rceil$ attributes are selected also from the sampled fit data. The fit dataset produced in this scenario is denoted by `RUS1`. The performances of the classification models were evaluated in terms of AUC. We also applied the SVM classifier to the original, unsampled fit dataset and the sampled fit dataset (with the complete set of attributes) for comparison purposes. In summary, we have five cases.

In the experiments, ten runs of five-fold cross-validation were performed. For each of the five folds, one fold is used as the test data while the other four are used as the training data. First, RUS is applied to the training data (4 folds from the full dataset) if a sampled fit data is expected to be created. Then, the SVM is built on this sampled training data to get a ranking of the attributes. Once the attributes are ranked, the top $\lceil \log_2 n \rceil$ of attributes are selected (as well as the class attribute) from the sampled training data or from the original, unsampled training data to yield the final feature selected training data. This final training data is used to build the classification model and the resulting model is used on the test fold. All the results are reported in Table 2. The table lists the mean value of AUC for the classification models constructed over ten runs of five-fold cross-validation with the five cases of the training datasets. We divide the five training datasets into two groups based on whether a feature ranking technique is involved. `Full attributes` refers to no attribute ranking involved, while the `Ranking in terms of XXX` refers to that a wrapper-based ranking technique was involved. Four different performance metrics that were used during the ranking process are presented in the table. From the table, we can see the following facts:

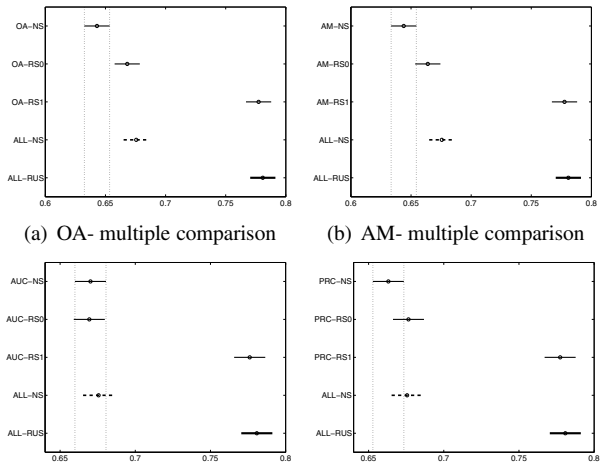
1. For the complete set of attributes (full attributes), the classification performance gets significantly improved when the sampled fit data were used to build the models. This is true across all eight datasets.

2. Among the three training data cases that contain only a subset of the full attributes (the first three data columns in the table), the classification models built with the `RUS1` scenario demonstrated significantly better performance than those built with the other two scenarios (`NS` and `RUS0`). In addition, between the two scenarios with inferior performances, `RUS0` showed a bit better performance than `NS`.

3. The classification models built using `RUS1` significantly outperformed those constructed on the original, unsampled fit dataset (fourth data column in the table) across all four performance metric-based feature rankings.

4. The classification models built on the sampled fit dataset regardless of using the complete set of attributes or using the subset of attributes showed superior performance than those built with the unsampled training datasets. Moreover, between the two sampled data cases, they displayed very similar classification performance, however, the subset of attributes is only a fraction (15% or 30%) of the complete set of attributes.

We also conducted a one-way ANalysis Of VAriance (ANOVA) F test [2] on AUC, to examine if the performance is significantly different (better/worse) among the five different training scenarios for each performance metric-based ranking. The ANOVA model can test the null hypothesis that all the group population means are the same against the alternate hypothesis that at least one pair of means is different. The ANOVA results indicate that the alternate hypothesis is accepted, since all the p-values are close to zero. We further carried out a multiple comparison test [2] using Tukey's honestly significant difference criterion



(a) OA- multiple comparison (b) AM- multiple comparison

(c) AUC- multiple comparison (d) PRC- multiple comparison

Figure 2. Multiple comparisons

to identify which of the means are significantly different from the others. The multiple comparison results, as shown in Figure 2, display graphs with each group mean represented by a symbol (○) and an interval around the symbol (95% confidence interval). Two means are significantly different ($p = 0.05$) if their intervals are disjoint, and are not significantly different if their intervals overlap. In the figures, `ALL-RUS` and `ALL-NS`, which represent the sampled training data and original, unsampled training data, are highlighted by a dot line and a thick line, respectively, and `XXX-NS`, `XXX-RUS0`, and `XXX-RUS1` have the same meanings as `NS`, `RUS0`, and `RUS1` defined previously, but are specified with a performance metric that is used in the feature ranking process. The results demonstrate the same conclusions as we obtained by directly observing the table, but more statistically substantiated.

5. Conclusion

This paper presents wrapper-based feature ranking techniques applied to imbalanced software quality datasets. Class imbalance often occurs in software engineering and data mining applications. Cost sensitive learning can help to compensate for the negative effect of the imbalanced data on the classification problems. However, the costs of different types of errors or misclassifications are difficult to estimate, especially during the modeling process. We used random undersampling (RUS) to solve the problem. An empirical study was performed through eight datasets collected from real-world software systems. The results show that for imbalanced datasets, attribute selection becomes more efficient when used after data sampling. In addition, the classification models built according to the strategy that selects attributes from the sampled fit data significantly outperformed the models constructed with the original, unsampled fit data. Moreover, the smaller subsets of attributes contained only 15% or 30% of the complete set of attributes. This aids the SQA (software quality assurance) team in managing software quality with fewer software metrics. Future work may involve conducting more experiments, investigation of different sampling techniques and filter-based feature ranking techniques, and comparisons between the wrapper and filter techniques.

References

[1] R. Arbel and L. Rokach. Classifier evaluation under limited resources. *Pattern Recognition Letters*, 27(14):1619–1631, 2006.

[2] 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.

[3] J. Davis and M. Goadrich. The relationship between Precision-Recall and ROC curves. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 233–240, Pittsburgh, Pennsylvania, 2006.

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

[5] G. Forman. Advances in kernel methods - support vector learning. *Advances in kernel methods - support vector learning*, pages 185–208, 1999.

[6] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, March 2003.

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

[8] 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.

[9] T. M. Khoshgoftaar, C. Seiffert, J. Van Hulse, A. Napolitano, and A. Folleco. Learning with limited minority class data. In *Proceedings of the IEEE International Conference on Machine Learning and Applications*, pages 348–353, Cincinnati, Ohio, USA, December 2007. IEEE Computer Society.

[10] 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.

[11] D. Rodriguez, R.Ruiz, J. Cuadrado-Gallego, and J. Aguilar-Ruiz. Detecting fault modules applying feature selection to classifiers. In *Proceedings of 8th IEEE International Conference on Information Reuse and Integration*, pages 667–672, Las Vegas, Nevada, August 13-15 2007.

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