Building Useful Models from Imbalanced Data with Sampling and Boosting

Chris Seiffert and **Taghi M. Khoshgoftaar** and **Jason Van Hulse** and **Amri Napolitano**
Computer Science and Engineering
Florida Atlantic University
777 Glades Rd., Boca Raton, FL 33431
chrisseiffert@gmail.com; taghi@cse.fau.edu; jvanhulse@gmail.com; anapoli1@fau.edu

Abstract

Building useful classification models can be a challenging endeavor, especially when training data is imbalanced. *Class imbalance* presents a problem when traditional classification algorithms are applied. These algorithms often attempt to build models with the goal of maximizing overall classification accuracy. While such a model may be very *accurate*, it is often not very *useful*. Consider the domain of software quality prediction where the goal is to identify program modules that are most likely to contain faults. Since these modules make up only a small fraction of the entire project, a highly accurate model may be generated by classifying all examples as not fault prone. Such a model would be useless. To alleviate the problems associated with class imbalance, several techniques have been proposed. We examine two such techniques: data sampling and boosting. Five data sampling techniques and one commonly used boosting algorithm are applied to five datasets from the software quality prediction domain. Our results suggest that while data sampling can be very effective at improving classification performance when training data is imbalance, boosting (which has received considerably less attention in research related to mining imbalanced data) usually results in even better performance.

Introduction

Many application domains, such as Software Quality Prediction, suffer from the problem of *class imbalance*. That is, datasets in these domains tend to be unevenly distributed with respect to class. When examples of one class are much more (or less) abundant than examples of the other class(es), this presents a problem for traditional classification algorithms. This is because most classification algorithms attempt to maximize classification accuracy without regard for the significance of the different classes. For example, in the Software Quality Prediction domain, models are typically built to distinguish between program modules that are likely to contain faults (fault prone, or fp) from those that are not fault prone (nfp). If only 2% of the modules in a software project are fault prone, then a model can achieve 98% accuracy by classifying all modules as being nfp. While this level of accuracy seems very good, such a model is useless to practitioners in this domain.

Many techniques have been proposed to alleviate the problems associated with class imbalance. This paper examines and compares the performance of two such techniques: data sampling and boosting. The first technique, data sampling, attempts to improve classification performance by artificially balancing the class distributions of training datasets. This can be done in one of two ways. Over sampling creates a more balanced dataset by increasing the number of examples in the the minority (underrepresented) class. Under sampling, on the other hand, reduces the number of examples belonging to the majority class. Both under and over sampling can be done randomly, or using more "intelligent" algorithms. In this work, we examine the performance of five different data sampling techniques, including both random and intelligent over and under sampling.

The second technique, boosting, is designed to improve the performance of weak classifiers. It does so by iteratively building an ensemble of classifiers, modifying the weight of training examples after each iteration. By assigning higher weights to misclassified examples, the model built in the following iteration is more likely to correctly classify those examples. While boosting was not designed specifically to deal with the class imbalance problem, our results show that it performs very well in this regard. In this work, we examine the performance of AdaBoost (Schapire 1999), a well known boosting algorithm that can improve the performance of any weak classifier.

While there is existing research that compares different data sampling techniques, and proposes modified boosting algorithms, these studies do not compare the performance of multiple data sampling to that of boosting for mining imbalanced data. Some studies, especially those that combine boosting and sampling into a single algorithm (Chawla *et al.* 2003), do compare the performance of the base data sampling technique, the base boosting technique and their combination, but to our knowledge no study performs a comprehensive comparison of the performance of boosting and the various data sampling techniques. This work does just that. We present a comprehensive empirical evaluation of data sampling and boosting using real world datasets from the domain of software quality prediction. In this study, for which 16,000 classification models were evaluated, we

demonstrate the performance of five data sampling techniques and boosting. Our results show that while data sampling does improve classification performance when training datasets are imbalanced, boosting often outperforms even the best data sampling technique.

Related Work

The difficulties associated with learning from imbalanced data are well documented in data mining and machine learning literature. Japkowicz (Japkowicz 2000) provides an in depth view of the imbalance problem, identifying different types of imbalance and which are the most detrimental to the learning process. (Japkowicz 2000) also examines different strategies for dealing with class imbalance, including random over and under sampling which are two of the techniques used in this work. Weiss (Weiss 2004) also investigates the problem of class imbalance, presenting a survey of many techniques for alleviating this problem, including boosting, another technique used in our research.

Data sampling has received significant attention among data mining researchers. Drummond and Holte (Drummond & Holte 2003), for example, compare over and under sampling using C4.5 decision trees. Their results show under sampling to be more effective than over sampling for improving the performance of models built using C4.5. Maloof's research (Maloof 2003), however, shows that under and over sampling result in roughly equivalent models when using C5.0 (C4.5's commercial successor) and Naive Bayes. In addition to random over and under sampling, several more "intelligent" data sampling techniques have been proposed to sample data in such a way that benefits the classifier. Barandela et al. (Barandela *et al.* 2004) and Han et al. (Han, Wang, & Mao 2005) examine the performance of some of these "intelligent" data sampling techniques such as SMOTE, borderline-SMOTE, and Wilson's Editing. Van Hulse et al. (Van Hulse, Khoshgoftaar, & Napolitano 2007) examine the performance of seven different sampling techniques including both random and "intelligent" techniques.

The other strategy this work examines, boosting, has received relatively less attention in the related research. Most of these studies deal with the introduction of new boosting techniques designed specifically for the class imbalance problem. DataBoost-IM, proposed by Gou and Viktor (Guo & Viktor 2004) combines boosting and data sampling to improve classification performance. SMOTEBoost (Chawla *et al.* 2003) is another technique that combines data sampling (SMOTE) and boosting to compensate for imbalanced data. RareBoost (Joshi, Kumar, & Agarwal 2001) modifies the boosting process by adjusting weights differently depending on the misclassified examples' class.

In this work, we present an empirical investigation comparing the performance of various data sampling techniques to that of boosting. Based on previous research (Van Hulse, Khoshgoftaar, & Napolitano 2007), we select the five data sampling techniques that most improve performance when learning from imbalanced data. We evaluate the performance of these five techniques on five datasets from the software quality prediction domain, and compare their performance to that of AdaBoost (Freund & Schapire 1996), one of the most popular boosting algorithms. To our knowledge, this is the first study to compare the performance of boosting and multiple data sampling techniques.

Table 1: Dataset characteristics

Dataset	#mods	#min	%min	#attr
C12	282	16	5.67	9
CM1	505	48	9.50	16
PC1	1107	76	6.87	16
SP1	3649	229	6.28	43
SP3	3525	47	1.33	43

Experimental Datasets

This work uses five datasets from the software quality prediction application domain. This domain is ideally suited for studying the class imbalance problem since almost all software quality datasets are imbalanced. It is well known in the software engineering field that the majority of a software project's faults lie in only a small percentage of the program modules. Therefore, software quality datasets are naturally imbalanced (Khoshgoftaar, Yuan, & Allen 2000). Table 1 provides details about the five software quality datasets used in this study, including the size of the dataset (#mods), the number of minority class (fp) examples (#min), the percentage of examples belonging to the minority class (%min), and the number of attributes in the dataset. The datasets, some of which are proprietary, used in this work represent software projects of different sizes with different levels of imbalance. They were obtained from three different industries including a large telecommunications company, the NASA Metrics Data Program (NASA/WVU IV&V Facility), and the Department of Defense.

Learners

This work uses two well-known learners: C4.5 and RIPPER. C4.5 (Quinlan 1993) is a learning algorithm that builds decision trees using an entropy-based splitting criterion stemming from information theory. It improves upon ID3 (Quinlan 1986) by adding support for tree pruning and dealing with missing values and numeric attributes. J48 is the WEKA (Witten & Frank 2005) implementation of C4.5. RIPPER (Repeated Incremental Pruning to Produce Error Reduction) (Cohen 1995) is a rule-based learner that modifies the IREP algorithm (Furnkranz & Widmer 1994) to improve accuracy without sacrificing efficiency. JRip is the WEKA implementation of RIPPER. The default parameter values (provided by WEKA) for both J48 and JRip are used in our experiments. These learners were selected since they have been shown to be greatly affected by the class imbalance problem (Van Hulse, Khoshgoftaar, & Napolitano 2007). Results may vary when other learners are used.

Sampling Techniques

This work uses five different data sampling techniques. Random over sampling (ROS) and random under sampling

(RUS) achieve more balanced datasets by randomly duplicating examples of the minority class (ROS) or randomly removing examples from the majority class (RUS). Synthetic Minority Oversampling Technique, or SMOTE (SM) (Chawla *et al.* 2002) creates new minority class examples by extrapolating between existing minority class examples. Borderline-SMOTE (BSM) (Han, Wang, & Mao 2005) modifies the SMOTE technique by only creating new examples based on minority class examples that lie near the decision border in feature space. Wilson's Editing (WE) is an under sampling technique that attempts to remove only majority class examples that are likely to contain noise as determined by the k-Nearest Neighbors algorithm (Aha 1997).

Each sampling technique was performed using a variety of parameters. ROS, RUS, SM, and BSM were performed with three different parameters: 35, 50 and 65. These parameters identify the percentage of examples in the post sampling dataset that will belong to the (pre-sampling) minority class. WE was performed twice, once using the Euclidean distance measure and once using the Weighted distance measure described in (Barandela *et al.* 2004).

Boosting

AdaBoost (Schapire 1999) is a well known boosting algorithm shown to improve the classification performance of weak classifiers. AdaBoost begins with all examples in the training dataset being assigned equal weights. AdaBoost then begins an iterative process where weak hypotheses are generated using the base learner (in this study, C4.5 or RIPPER). The error associated with a hypothesis is calculated and the weights of the training examples are adjusted, with misclassified examples receiving higher weights while the weights of correctly classified examples are reduced. Therefore in the next iteration the misclassified examples are more likely to be correctly classified by the new hypothesis. Once the stopping criteria is met (in this case, 10 iterations) all hypotheses participate in a weighted vote to assign classes to unlabeled examples.

Performance Metrics

Traditional performance measures such as overall classification accuracy, or its complement, misclassification rate, are inappropriate when dealing with the classification of rare events. When as few as 1% of examples belong to the minority class, a classifier can achieve an overall accuracy of 99% by simply labeling all examples as belonging to the majority class. In a domain such as software quality prediction, however, such a model is useless. Instead, we use two performance metrics that consider the ability of a classifer to differentiate between the two classes: The Kolmogorov-Smirnov statistic (KS) and the area under the ROC curve (AUC).

The Kolmogorov-Smirnov (KS) statistic measures the maximum difference between the empirical distribution function of the posterior probabilities of instances in each class. In other words, let $F_{c_i}(t) = P(p\ (x) \leq t \mid c_i)$, $0 \leq t \leq 1$. $F_{c_i}(t)$ is estimated by the proportion of class c_i instances $\leq t$:

$$F_{c_i}(t) = \frac{\#\text{ class } c_i \text{ instances with posterior probability} \leq t}{\#\text{ class } c_i \text{ instances}}.$$

Then the KS statistic is defined as:

$$KS = \max_{t \in [0,1]} \mid F_{c_1}(t) - F_{c_2}(t) \mid \quad (1)$$

The larger the distance between the two distribution functions, the better the learner is able to separate the two classes. The maximum possible value for KS is one (perfect separation), with a minimum of zero (no separation). The KS statistic is a commonly-used metric of classifier performance in the credit scoring application domain (Hand 2005).

Receiver Operating Characteristic curves (Provost & Fawcett 2001), or ROC curves, graph true positive rates on the y-axis versus the false positive rates on the x-axis. The resulting curve illustrates the trade-off between detection rate and false alarm rate. Traditional performance metrics consider only the default decision threshold of 0.5. ROC curves illustrate the performance across all decision thresholds. For a single numeric measure, the *area under the ROC curve* (AUC) is widely used, providing a general idea of the predictive potential of the classifier.

Experimental Design Summary

Using five software quality datasets, models are trained using two classification algorithms: C4.5 and RIPPER. Five different data sampling techniques, as well as boosting, are implemented to improve the performance of these learners. We employ 10-fold cross validation to build and test our models. That is, the datasets are broken into ten partitions, where nine of the ten partitions are used to train the model, and the remaining (hold out) partition is used to test the model. This is repeated ten times so that each partition is used as hold out data once. In addition, we perform 10 independent repetitions (runs) of each experiment to remove any biasing that may occur during the random selection process and to ensure the statistical significance of our results. The results reported in the following sections represent the average of these repetitions. In total, 16,000 models were evaluated during the course of our experiments.

Results: Data Sampling

This section presents the results of our experiments using data sampling. We compare the performance of C4.5 and RIPPER models built after applying five different data sampling techniques with the performance of models built without data sampling.

Tables 2 and 3 show the performance of C4.5 and RIPPER models, respectively, using the KS statistic to measure classification performance. In each of these tables, the best model for each dataset is indicated in **bold** print, while the worst performance is *italicized*. Asterisks (*'s) in these tables indicate results for which the statistical significance will be discussed later in this paper. Table 2 shows that data sampling almost always results in better performance than not performing sampling. For each dataset, the best performance is achieved using data sampling. In three of the

Table 2: Performance of C4.5 models measured using KS statistic

ST	C12	CM1	PC1	SP1	SP3
BSM	0.6974	0.3523	**0.5421**	0.3200	0.4064
ROS	*0.6015**	0.2158	0.3976	*0.1708*	0.1159
RUS	**0.7272***	0.3665	0.5351	**0.3832**	**0.4631**
SM	0.6944	**0.3768**	0.5385	0.3115	0.3402
WE	0.6764	0.2223	0.3556	0.2762	0.0290
None	0.6549	*0.1303*	*0.2806*	0.2714	*0.0180*

Table 3: Performance of RIPPER models measured using KS statistic

ST	C12	CM1	PC1	SP1	SP3
BSM	0.6754	0.3709	0.4979	0.3821	**0.4544**
ROS	0.6218	0.2355	0.4169	0.2220	0.0987
RUS	**0.7618**	**0.3919**	0.4907	**0.3988**	0.4497
SM	0.7050	0.3648	**0.5408**	0.3613	0.3564
WE	0.6722	0.1112	0.2494	0.1184	0.0129
None	*0.6097*	*0.0472*	*0.1808*	*0.0874*	*0.0121*

five datasets, RUS results in the best performance. For the two remaining datasets, the intelligent over sampling techniques results in the best performance. BSM results in the best performance using the PC1 dataset, while SM achieves the highest KS statistic using the CM1 dataset. None (no sampling) results in the worst performance in three of the five datasets. In the remaining two datasets (C12 and SP1), ROS actually performs worse than None. Of the five data sampling techniques, ROS is the only technique to result in worse performance than None for any dataset.

Table 3 shows the classification performance of models built using RIPPER. These results are even more convincing. Using RIPPER, every sampling technique outperforms None for every dataset. Unlike when C4.5 was used to build models, there is no case where models built without sampling outperform those built with sampling. Once again, RUS is shown to be a very strong sampling technique, achieving the highest KS statistic in three of the five datasets. SM achieved the best performance for the PC1 dataset, while BSM yielded the highest KS value for the SP3 dataset. In general, RUS is shown to be the best overall data sampling technique, while SM and BSM also perform very well. ROS and WE usually result in improved classification performance, but are almost always outperformed by the other three techniques.

Tables 4 and 5 show the performance of these data sampling techinques as measured using AUC. These results are similar to those obtained using the KS statistic. Once again, we see that data sampling almost always results in better classification performance than when sampling is not used. When models are built using C4.5 (Table 4), every data sampling technique outperforms None for every dataset. Using RIPPER to build models, there is only one case where None outperforms one of the data sampling techniques. As will be shown in later discussion, however, this performance difference (only 0.0003) is not statistically significant.

Table 4: Performance of C4.5 models measured using AUC

ST	C12	CM1	PC1	SP1	SP3
BSM	0.8409	0.6055	0.7479	0.6067	0.5262
ROS	0.7974	0.5929	0.6958	0.5833	0.5495
RUS	**0.8634**	**0.6682**	**0.7653**	**0.6784**	**0.6943**
SM	0.8380	0.6515	0.7322	0.6216	0.5091
WE	0.8176	0.5793	0.6899	0.5875	0.5000
None	*0.7728*	*0.5498*	*0.6529*	*0.5803*	*0.4974*

Table 5: Performance of RIPPER models measured using AUC

ST	C12	CM1	PC1	SP1	SP3
BSM	0.8299	0.6280	0.7377	0.6948	0.6253
ROS	0.8082	0.6018	0.7057	0.6077	0.5395
RUS	**0.8809**	**0.6923**	0.7471	**0.7084**	**0.7209**
SM	0.8461	0.6439	**0.7498**	0.6784	0.5616
WE	0.8358	0.5480	0.6233	0.5589	*0.5048**
None	*0.8026*	*0.5169*	*0.5898*	*0.5433*	0.5051

Tables 4 and 5 once again show RUS to be the superior data sampling technique. In fact, using AUC to measure performance, the data in these tables is even more convincing. Models built using C4.5 always result in the best performance when RUS is used, while RIPPER results in the best performance when combined with RUS in four of the five datasets. In the remaining dataset, PC1, SM outperforms RUS by a very small margin. The results presented in this section show that whether KS or AUC are used to measure performance, data sampling results in improved classification performance. In addition, RUS is shown to be the best of the five data sampling techniques.

Results: Boosting

In the previous section, we examined the performance of five different data sampling techniques. In this section we consider another technique for alleviating the class imbalance problem: boosting. The results presented in this section compare the performance of models built using C4.5 and RIPPER with Adaboost to the results of the best data sampling technique for each dataset/learner combination, as presented in the previous section.

Tables 6 and 7 compare the performance of the best data sampling technique from the previous section (usually RUS) with the performance achieved using Adaboost to improve classification performance. These tables use the KS statistic to measure classification performance. Tables 6 and 7 both show very similar results. For four of the five datasets, boosting out performs even the best data sampling technique (the best performance for each dataset is indicated by **bold** text). The only dataset where Adaboost is outperformed by data sampling is SP3. These results may be somewhat surprising, since these data sampling techniques are designed to improve classification performance when training data is imbalanced, while Adaboost was designed to improve perfor-

Table 6: Sampling vs. Boosting using C4.5, KS statistic

Tech	C12	CM1	PC1	SP1	SP3
BestST	0.7272	0.3768	0.5421	0.3832	**0.4631**
Boost	**0.8143***	**0.4510**	**0.6193**	**0.4146***	0.3746
None	*0.6549*	*0.1303*	*0.2806*	*0.2714*	*0.0180*

Table 7: Sampling vs. Boosting using RIPPER, KS statistic

Tech	C12	CM1	PC1	SP1	SP3
BestST	0.7618	0.3919	0.5408	0.3988	**0.4544***
Boost	**0.8127***	**0.4762**	**0.5975**	**0.4142***	0.4245
None	*0.6097*	*0.0472*	*0.1808*	*0.0874*	*0.0121*

Table 8: Sampling vs. Boosting using C4.5, AUC

Tech	C12	CM1	PC1	SP1	SP3
BestST	0.8634	0.6682	0.7653	0.6784	**0.6943***
Boost	**0.8995***	**0.7158**	**0.8457**	**0.7507**	0.6711
None	*0.7728*	*0.5498*	*0.6529*	*0.5803*	*0.4974*

Table 9: Sampling vs. Boosting using RIPPER, AUC

Tech	C12	CM1	PC1	SP1	SP3
BestST	0.8809	0.6923	0.7498	0.7084	**0.7209***
Boost	**0.8973***	**0.7284**	**0.8352**	**0.7432**	0.6928
None	*0.8026*	*0.5169*	*0.5898*	*0.5433*	*0.5051*

mance in general. Even though boosting, which receives less attention in class imbalance research than data sampling, is not designed to handle this problem, it results in better performance in four out of five of our real-world software quality datasets.

Tables 8 and 9 show similar results using AUC to measure performance. Once again, boosting outperforms even the best data sampling techniques for four of the five software quality datasets used in our experiments. As when KS was used to measure performance, the only dataset where data sampling results in better performance is SP3.

Significance of Results

In order to test the statistical significance of the results presented in this work, ANOVA analysis (Berenson, Levine, & Goldstein 1983) was performed. Although the entire analysis cannot be presented due to space considerations, we briefly discuss the results in this section. Tables 2 through 9 present the mean performance (across ten repetitions of ten-fold cross validation) for each dataset/learner/technique combinations. In addition, some entries in these tables include *'s, which identify values where discussion about statistical significance may be important.

In Tables 2 through 5, the best performing sampling technique is indicated by **bold** print, and the worst performance is indicated by *italicized* text. These two values may also be marked with *'s. If a bold value is marked with a *, then that value is not significantly different (at the 5% confidence level) than the value presented for None (no sampling). For example, while RUS results in the best KS value using C4.5 and the C12 dataset (Table 2), this value is not significantly better than the value presented for None. This is the only case in Tables 2 through 5 where the best performing sampling technique was not signifcantly better than None at the 5% confidence level.

In Tables 2 through 5, a * next to an italicized value means that that techniques performance was not significantly worse than None. For example, in Table 2, ROS is shown to perform worse than None for the C12 dataset. However, as indicated by the *, this difference is not significant at the 5% confidence level. In fact, in all of the experiments presented by Tables 2 through 5, only once is a sampling technique significantly outperformed by None (In Table 2, ROS performs significantly worse than None on the SP1 dataset).

The *'s in Tables 6 through 9 also provide information regarding the statistical significance of the results presented in those tables. In these tables, a * indicates that the value to which it is attached is not significantly better than the next highest value in that column. For example, in Table 6, Although boosting outperforms the best data sampling technique for the C12 dataset, the difference between the two values is not significant at the 5% confidence level.

Conclusion

The problem of learning from imbalanced training data is well documented in data mining and machine learning research. Traditional classification algorithms attempt to maximize overall classification accuracy, without regard for the significance of each class. Therefore, if one of the classes is underrepresented in the training data, classifiers will tend to misclassify those minority classes more often in an attempt to maximize overall accuracy. In a domain such as software quality classification, however, such a model has no value. In this domain, and many others, such as network security and medical diagnosis, the goal is to identify these minority class examples, not achieve high overall correct classification rates.

This study examines two different methods for alleviating the problems associated with class imbalance: data sampling and boosting. These techniques were applied to five different real world software quality datasets with different sizes and levels of imbalance. Our experiments show that data sampling, which has received much more attention in class imbalance related research, significantly improves the performance of software quality prediction models built using all five of our datasets. The best performance is usually achieved by random under sampling, but SMOTE and borderline-SMOTE also perform very well. While random over sampling and Wilson's Editing usually improve classification performance, these two techniques do not usually perform as well as the other three. ANOVA analysis was performed (but not presented due to space considerations) to test the statistical significance of our findings. In most cases the improvement achieved by sampling was shown to

be statistically significant.

Boosting, which was not designed to address the class imbalance issues, and which has received comparatively little attention in the research related to mining imbalanced data, was shown to perform extremely well in our experiments. In four out of the five datasets, AdaBoost resulted in better performance than even the best data sampling technique. ANOVA analysis shows that this difference is statistically significant for three of the four datasets, while in one dataset, where data sampling outperformed boosting, the difference was not significant. These results are fairly consistent regardless of the learner or performance metric used (exceptions are noted in the accompanying analysis).

Based on the results of our extensive experiments, we conclude that boosting is very effective at alleviating the problems associated with learning from imbalanced data. While data sampling techniques usually improve performance, this improvement is not as great as the improvement achieved by boosting. These promising results motivate future work which will include investigation of additional boosting algorithms, including those that are specifically designed to address the class imbalance problem. In addition, this study will be extended to additional datasets, in both the software quality prediction domain as well as other domains where imbalanced datasets are common.

References

Aha, D. W. 1997. *Lazy learning*. Norwell, MA, USA: Kluwer Academic Publishers.

Barandela, R.; Valdovinos, R. M.; Sanchez, J. S.; and Ferri, F. J. 2004. The imbalanced training sample problem: Under or over sampling? *In Joint IAPR International Workshops on Structural, Syntactic, and Statistical Pattern Recognition (SSPR/SPR'04), Lecture Notes in Computer Science 3138* (806-814).

Berenson, M. L.; Levine, D. M.; and Goldstein, M. 1983. *Intermediate Statistical Methods and Applications: A Computer Package Approach*. Prentice-Hall, Inc.

Chawla, N. V.; Hall, L. O.; Bowyer, K. W.; and Kegelmeyer, W. P. 2002. Smote: Synthetic minority oversampling 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 *In Proceedings of Principles of Knowledge Discovery in Databases*.

Cohen, W. W. 1995. Fast effective rule induction. In *In Proc. 12th International Conference on Machine Learning*, 115–123. Morgan Kaufmann.

Drummond, C., and Holte, R. C. 2003. C4.5, class imbalance, and cost sensitivity: why under-sampling beats oversampling. In *Workshop on Learning from Imbalanced Data Sets II, International Conference on Machine Learning*.

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

Furnkranz, J., and Widmer, G. 1994. Incremental reduced error pruning. In *International Conference on Machine Learning*, 70–77.

Guo, H., and Viktor, H. L. 2004. Learning from imbalanced data sets with boosting and data generation: the databoost-im approach. *ACM SIGKDD Explorations Newsletter* 6(1).

Han, H.; Wang, W. Y.; and Mao, B. H. 2005. Borderline-SMOTE: A new over-sampling method in imbalanced data sets learning. In *In International Conference on Intelligent Computing (ICIC'05). Lecture Notes in Computer Science 3644*, 878–887. Springer-Verlag.

Hand, D. J. 2005. Good practice in retail credit scorecard assessment. *Journal of the Operational Research Society* 56:1109–1117.

Japkowicz, N. 2000. Learning from imbalanced data sets: a comparison of various strategies. In *In Papers from the AAAI Workshop on Learning from Imbalanced Data Sets*, Tech. rep. WS–00–05. Menlo Park, CA: AAAI Press.

Joshi, M. V.; Kumar, V.; and Agarwal, R. C. 2001. Evaluating boosting algorithms to classify rare classes: Comparison and improvements. In *In Proceedings of IEEE International Conference on Data Mining*, 257–264.

Khoshgoftaar, T. M.; Yuan, X.; and Allen, E. B. 2000. Balancing misclassification rates in classification-tree models of software quality. *Empirical Software Engineering* 5(4):313 – 330.

Maloof, M. 2003. Learning when data sets are imbalanced and when costs are unequal and unknown. In *Proceedings of the ICML'03 Workshop on Learning from Imbalanced Data Sets*.

NASA/WVU IV&V Facility. Metrics data program. *http://mdp.ivv.nasa.gov*.

Provost, F., and Fawcett, T. 2001. Robust classification for imprecise environments. *Machine Learning* 42:203–231.

Quinlan, J. R. 1986. Induction of decision trees. *Machine Learning* 1(1):81–106.

Quinlan, J. R. 1993. *C4.5: Programs For Machine Learning*. San Mateo, California: Morgan Kaufmann.

Schapire, R. E. 1999. A brief introduction to boosting. In *International Joint Conference on Artificial Intelligence*, 1401–1406.

Van Hulse, J.; Khoshgoftaar, T. M.; and Napolitano, A. 2007. Experimental perspectives on learning from imbalanced data. In *In Proceedings of the 24th International Conference on Machine Learning*, 935–942.

Weiss, G. M. 2004. Mining with rarity: A unifying framework. *SIGKDD Explorations* 6(1):7–19.

Witten, I. H., and Frank, E. 2005. *Data Mining: Practical machine learning tools and techniques*. San Francisco, California: Morgan Kaufmann, 2nd edition.