Evaluating Stratification Alternatives to Improve Software Defect Prediction

Lourdes Pelayo and Scott Dick, *Member, IEEE*

***Abstract*—Numerous studies have applied machine learning to the software defect prediction problem, i.e. predicting which modules will experience a failure during operation based on software metrics. However, skewness in defect-prediction datasets can mean that the resulting classifiers often predict the faulty (minority) class less accurately. This problem is well known in machine learning, and is often referred to as "learning from imbalanced datasets." One common approach for mitigating skewness is to use stratification to homogenize class distributions; however, it is unclear what stratification techniques are most effective, both generally and specifically in software defect prediction. In this article, we investigate two major stratification alternatives (under-, and over-sampling) for software defect prediction using Analysis of Variance. Our analysis covers several modern software defect prediction datasets using a factorial design. We find that the main effect of under-sampling is significant at $\alpha = 0.05$, as is the interaction between under- and over-sampling. However, the main effect of over-sampling is not significant.**

***Index Terms*—Learning in imbalanced datasets, machine learning, non-parametric models, software fault-proneness, software reliability, stratification.**

Acronyms

SDP	Software Defect Prediction
ANOVA	Analysis of Variance
SMOTE	Synthetic Minority Oversampling Technique
WMC	Weighted Methods per Class
DIT	Depth of Inheritance Tree
NOC	Number of Children
RFC	Response for a Class
CBO	Coupling Between Object classes
LCOM	Lack of Cohesion of Methods
CPU	Central Processing Unit
k-NN	k-Nearest Neighbors
ROC	Receiver Operating Characteristic
AUC	Area Under the Curve
KDD	Knowledge Discovery in Databases
MSE	Mean Squared Error

Notation

i	Levels for the factor "undersampling"
j	Levels for the factor "oversampling"
k	Index for the datasets used in ANOVA analyses
l	Iteration of the tenfold cross-validation procedure
μ	Overall mean of out-of-sample prediction accuracy for the classifier
τ_{ik}	Effect of the i-th undersampling level on the k-th dataset
β_{jk}	Effect of the j-th oversampling level on the k-th dataset
$(\tau\beta)_{ijk}$	Interaction between the i-th undersampling and j-th oversampling levels on the k-th dataset
ε_{ijkl}	Random error for the i-th undersampling and j-th oversampling levels in the l-th iteration of the tenfold cross-validation procedure on the k-th dataset
y_{ijkl}	Out-of-sample prediction accuracy for the classifier on the i-th undersampling and j-th oversampling levels in the l-th iteration of the tenfold cross-validation procedure on the k-th dataset
K	Kappa Statistic
F_O	Observed agreement between two raters
F_C	Probability that two raters agree by chance
X, Y	Denotes two independent raters
X_m	The fraction of items rater X has assigned to category m
Y_m	The fraction of items rater Y has assigned to category m
CM_{ab}	The element of a contingency table (confusion matrix) at row a and column b
n	The total number of categories
N	The total number of samples

Manuscript received February 27, 2011; revised September 27, 2011; accepted October 31, 2011. Date of publication February 23, 2012; date of current version May 28, 2012. This work was supported in part by Consejo Nacional de Ciencia y Tecnologiá (CONACYT), México, and the Natural Sciences and Engineering Research Council of Canada under grant no. G121210906. Associate Editor: Dr. J.-C. Lu.

The authors are with the Department of Electrical & Computer Engineering, University of Alberta, Edmonton, AB, Canada (e-mail: pelayo@ece.ualberta.ca; dick@ece.ualberta.ca).

Digital Object Identifier 10.1109/TR.2012.2183912

T_α	The critical value for Tukey's Honestly Significant Difference test
$q_\alpha(e,f)$	The critical value of the studentized range distribution for significance α, number of means e and degrees of freedom f
N_O	Number of observations

I. Introduction

OUR MODERN technological society is absolutely dependent on software systems, and software failures can easily cause economic damage, personal injury, or even death. Software failures cost the U.S. economy alone more than $78 billion per year [1]. One approach to improving software reliability is to build models to determine how best to allocate quality improvement resources [2]. However, it is notoriously difficult to accurately estimate software reliability; the history of attempts stretches from the 1970s [3] through to the present day. A proxy for the actual reliability is the predicted number of defects (or simply their occurrence) in a module, as estimated using software metrics [4], [5]. This proxy has also proven stubbornly difficult to estimate; there is no accepted theory relating software metrics to reliability. This problem is referred to as software defect prediction (SDP), and its categorical form (defect occurrence) is the focus of this article.

Non-parametric techniques such as machine learning are often used for the SDP problem. However, this application is a difficult domain for these classifiers, because SDP datasets tend to be skewed towards small modules with a low chance of failure. In general, a dataset that is heavily skewed towards a majority class will often lead to classifiers that are inaccurate for, or never predict, the minority class; this problem is common in the machine learning community, known as "learning from imbalanced datasets." There are two main approaches to learning in imbalanced datasets: cost-sensitive classification, and stratification. Cost-sensitive classification has been applied in several investigations of SDP [6]–[9]; however, stratification has only recently been investigated in this domain [10]–[15]. Stratification works by reducing the number of samples of the majority class (under-sampling) or adding more samples to the minority class (over-sampling) to create a more balanced dataset. While some authors assert that only under-sampling is generally useful [16], others find over-sampling is a valuable technique [17], and still others find a combination of both is the best strategy [18]. This debate is ongoing in the machine learning community, which clearly has practical implications for the software reliability engineering community.

The goal of this article is to provide empirical evidence showing which alternative (under-sampling, over-sampling, or both) is *generally* (across multiple datasets) most effective in the SDP problem.

We employ the Analysis of Variance (ANOVA) to determine the relative contribution of under-sampling, over-sampling, and their interaction in SDP. We focus on uniform under-sampling, oversampling via the Synthetic Minority Oversampling Technique (SMOTE) algorithm [19], and a more recent variant of SMOTE [20]. Uniform under-sampling has been repeatedly identified as effective in many domains [16], [21], while SMOTE and its variants have been widely used and studied. Using a collection of modern SDP datasets, we test the performance of a single, standardized classification algorithm (the well-known C4.5 decision tree induction algorithm [71]) for different levels of under- and over-sampling. The ANOVA results show that, under-sampling, and the interaction between under- and over-sampling, are significant; but over-sampling is not. Tukey's Honestly Significant Difference test supports these results. From these findings, we can say that both under- and over-sampling have value in software defect prediction; the software quality analyst using machine learning is best served by employing both approaches.

The remainder of this paper is arranged as follows. We review related work in software defect prediction, cost-sensitive classification, and stratification for software defect prediction in Section II. In Section III, we review the current debate on stratification in the machine learning literature. We present our experimental methodology in Section IV, and our results in Section V. Threats to validity are discussed in Section VI, and finally we offer a summary and discussion of future work in Section VII.

II. Related Work

A. Software Defect Prediction

We formulate SDP as a binary classification problem in which software metrics for a given code module predict either a low risk of failure (class 0), or a high risk (class 1). To be specific, we focus only on software product metrics, e.g. [22], [23]. Software process metrics as well as cost estimation metrics (e.g. [24]) are beyond the scope of this article. Software product metrics tend to be strongly correlated with each other, as well as with the number of defects (multicollinearity). This problem is well-known in the literature [5], [25], and undermines regression models that assume s-independence between the predictor attributes. Principal Components Analysis (PCA) is commonly used to eliminate multicollinearity, as well as reduce the number of predictor attributes [4], [26]. For a recent comparison of parametric models, see [27]. The rise of object-oriented programming prompted the development of software product metrics focused on this new approach. The first and best-known of these are a suite of six metrics developed in [28], and known as the CK metrics. Lorenz and Kidd [29] proposed additional OO metrics dealing with inheritance and message passing in a class. Many other metrics have been proposed; we refer the reader to [30]–[33] for reviews and comparisons.

Despite the extensive research into SDP, there is still no parametric model that accurately forecasts defect rates or occurrence. Ultimately, this gap stems from the lack of a theory relating software metrics to defects. This gap reduces parametric approaches for SDP to a trial-and-error search through an infinite space of model forms (e.g. the models in [27]). Models can be fitted to a given project, but they are not transferable to others. For such reasons, nonparametric techniques, especially machine learning, have been employed for SDP. Some examples include neural networks [34]–[39], fuzzy classifiers [40], [41], genetic programming [11], [42], [43], clustering [12], [44]–[46], classification trees [8], [47], [48], and kernel-based methods [49]–[51].

B. Cost-Sensitive Classification for Software Defect Prediction

The imbalanced dataset problem is a bias (skewness) in the available data such that examples of one class significantly outnumber the other. This result is common in many real-world domains such as software defect prediction, credit card fraud detection [52], and breast cancer detection [53]. Imbalanced datasets undermine machine learning algorithms that do not explicitly consider differential error costs or class distributions. The basic design of most machine learning algorithms is to develop a problem representation (a decision tree, a neural network, etc.), and to measure its quality over all collected data. This representation will then be updated iteratively to optimize the performance measure (e.g. node weights in a neural network are changed) [54]. The difficulty is that the performance measure is naturally biased towards the majority class. Examples belonging to the minority class will thus tend to have less influence in updating the representation than majority class examples. Over the course of many iterations, the representation thus sacrifices accurate modeling of the minority class in favor of better modeling of the majority class [12].

Learning algorithms can compensate for imbalanced datasets by explicitly considering differential error costs. The cost of misclassification errors often differs when the errors are located in the minority class in contrast with the majority class [16]. When the minority class samples are the "interesting" ones (e.g. a faulty module), mistaking one such for an uninteresting case (not faulty) is more costly than the reverse. Cost-sensitive classifiers explicitly consider these differential costs, and will minimize the total *expected cost* of errors, rather than the base performance measure. Approaches to cost-sensitive classification include [55]–[59]. Domingos proposed a general method for cost-sensitive classification called MetaCost in [60]. This method is based on re-labeling training examples with their minimal-cost classes based on their class probabilities instead of their "optimal" class. The learning algorithm (which need not be modified) is then trained using this relabeled training set.

Cost-sensitive classification has been applied to SDP since 1999 [8]. Recently, Khoshgoftaar *et al.* [7] discuss the use of cost-sensitive boosting [57] for SDP. Ling *et al.* [6] proposed a system to predict the escalation risk of current defect reports for maximum return on investment (ROI), based on mining historic defect report data from an online repository. ROI was computed by estimating the cost of not correcting an escalated defect (false negative) to be seven times the cost of correcting a non-escalated defect (false positive). However, in a recent comparison on 15 benchmark datasets (including 7 SDP datasets), cost-sensitive classification was found to be inferior to random under-sampling when the cost ratios were high [61].

C. Stratification for Software Defect Prediction

Stratification is a sampling procedure (without replacement) used to alter the class distribution within a dataset, to balance it or make the classifier more sensitive towards one particular class of interest. The simplest approaches to stratification are (uniformly) under-sampling the majority class, or over-sampling the minority class by replicating examples. Some of these approaches include those reported in [10], [62]–[64]. Replication has recently been examined in [16], and found to be ineffective. However, other over-sampling techniques have substantially different properties; one of the best-known is SMOTE [19]. Instead of replicating minority class examples, SMOTE creates a new *synthetic* example via linear interpolation between a minority-class example and its nearest neighbor(s). Experimental evaluation showed that this approach was superior, as replication leads to very small decision regions around the replicated points, whereas SMOTE led to broader decision regions, and hence better out-of-sample accuracy. There are now several extensions of the basic SMOTE algorithm [17], [65], [66]. One of the most recent [20] uses "surrounding" neighbors (i.e. considering proximity and symmetry as well as distance) to generate synthetic examples. Alternative over-sampling techniques include those reported in [67]–[69].

Researchers have only recently investigated the use of stratification in SDP. Kaminsky and Boetticher [11] utilize genetic programming for software defect prediction. SMOTE was first applied to SDP in [12]. Pelayo and Dick [13] also used uniform under-sampling and SMOTE over-sampling for SDP. Seiffert *et al.* [14] compare five stratification techniques with a boosting algorithm (AdaBoost) on 15 SDP datasets.

III. Stratification in Machine Learning

Stratification is a preprocessing step in the well-known Knowledge Discovery in Databases (KDD) framework, proposed in [70]. In this framework, a dataset is first assembled that matches the analyst's goals. The data are then preprocessed; this is the stage where stratification would occur, along with other preprocessing steps (data cleaning, normalization, etc.). Optionally, there is then a feature-reduction step, and then the dataset is mined. The results are then consolidated with existing knowledge. Plainly, stratification is only valuable when used as a part of the KDD framework. Equally plainly, any improvement in stratification could be confounded with other elements of KDD. For this reason, stratification research often employs a simplified framework: stratification is the only preprocessing step, and a single classifier (using constant parameters) is used for all experiments in a study, which determine the out-of-sample error in a cross-validation design. To achieve meaningful results, the classification algorithm must be simple and powerful; the most common choice is the C4.5 decision tree [71]. In common with [16]–[19], we will use C4.5 and its default parameters in our experiments. With confounding factors eliminated, we can thus observe the effect of the stratification algorithm. However, this design is not universally used, e.g. [21] uses 11 different learning algorithms and 35 datasets to compare stratification techniques.

One major question in the machine learning community is the utility of over-sampling vs. under-sampling; it is unclear whether one or the other (or a combination of both) is most effective for learning from imbalanced datasets. Weiss and Provost [16] advocate pure under-sampling datasets, as their experiments showed it is effective, and over-sampling increases the time and memory requirements for classifiers; they also show replication is not helpful. In later work, [17] tests the SMOTE + Tomek and SMOTE+ENN techniques using thirteen UCI datasets [53]. They found that these over-sampling methods were superior to under-sampling. Estabrooks [18] compares uniform under-sampling with replication, or the use of both. A combination of both was found to be best. Chawla *et al.* [72] also propose a hybrid method combining random under-sampling and SMOTE. Seiffert *et al.* [73] compare a number of under- and over-sampling hybrids; uniform under-sampling and Wilson's editing were each combined with random over-sampling, SMOTE, and Borderline-SMOTE. Across 10 datasets (including 5 SDP datasets), random under-sampling, and either SMOTE or random over-sampling, were the top performers. In addition, uniform under-sampling was more beneficial in a hybrid situation than was Wilson's editing. A hybridization of random under-sampling and boosting (similar in philosophy to the SMOTEBoost technique) is presented in [74], [75]. Note that, while several of these studies tested for statistical significance using ANOVA and Tukey's test, none of them used a factorial experiment design between under- and over-sampling, making it impossible to quantify the significance of any interaction between them. Bagging and boosting techniques (which are different uses of resampling) are examined in [76].

IV. Experimental Design and Methodology

A. Methods

Our research question in this article asks whether under-sampling, over-sampling, or a combination of both would be most effective for software defect prediction. We can cast this question in terms of statistical experimental design as follows: a treatment will consist of modifying a defect-prediction training dataset using a certain level of under-sampling, and a certain level of over-sampling, followed by the training and testing of a known (constant) classifier on that dataset (the holdout sample is not resampled). Under-sampling and over-sampling thus become the factors in this design. Our research question then translates to inquiring if the mean responses of a classifier to the factors of under-sampling or over-sampling, or their interaction, are significantly different. This question is precisely what the ANOVA is intended to address [77]. We will employ ANOVA across a set of seven software defect-prediction datasets, employing the common tenfold cross-validation design in each cell; this approach will provide us with ten replicates per cell for determining the experimental error. We use the SMOTE implementation from [78], which employs K-D trees [79] instead of a brute-force neighbor search as in [19]. Other possibilities for a neighbor search include a box-assisted algorithm (also implemented in [78]), but the performance of a box-assisted search will be strongly influenced by the subspace selected for the search; K-D trees avoid this additional search of a parameter space, and were still an order of magnitude faster in timing experiments than a brute-force search. We will also test the more recent SMOTE-SN algorithm from [20]. Our classifier will be the C4.5 (release 8) decision-tree generator, employing its default parameters, as implemented in the WEKA package [80].

There is one significant complication in our experimental design: as with all pattern-recognition approaches, the outcomes of our experiments are expected to be dataset-dependent. Specifically, the relative importance of under-sampling and over-sampling in improving classifier performance is expected to vary from one dataset to another, and is not predictable *a priori*. This result is again a natural consequence of the well-known lack of a theory of pattern recognition [81]. Hence, we must accommodate this heterogeneity by employing blocking. Our experimental design is thus a blocked full-factorial experiment, where the blocks correspond to the datasets. We have seven different datasets, and will employ five levels for under-sampling and five levels for over-sampling. We select the over-sampling levels 0, 100, 200, 300, and 400%, which are the percent of synthetic minority samples added to the training dataset. We also select the under-sampling levels 0, 20, 40, 60, and 80%, which are the percent of majority class samples removed in the resampled dataset. These levels were determined from [13], where they were quite effective in improving SDP performance. The linear model for this design is

$$y_{ijkl} = \mu + \tau_{ik} + \beta_{jk} + (\tau\beta)_{ijk} + \varepsilon_{ijkl}$$
$$\begin{cases} i = 0, 20, 40, 60, 80 \\ j = 0, 100, 200, 300, 400 \\ k = CM1, KC1, KC1class-level, \\ KC2, JM1, PC1, Mozilla \\ l = 1, 2 \ldots 10 \end{cases} \tag{1}$$

We use the lack of fit test from [82], [83] to determine if this model is appropriate for our experiments. This test compares the residual error to the pure error from replicated design points, again using the F-test. The F-statistic is calculated by dividing the mean square error due to lack-of-fit by the mean square error due to replicate variation. If there is significant lack of fit, the model should not be used as a predictor of the response. The actual p-values for the lack of fit of our experiments are shown together with the analysis of variance results later in Section V.

We note that the reliability of the cross-validation design in general was questioned in [84], on the basis that two different models induced from the same dataset could not be homogeneously ordered across every fold. The authors generate 1000 samples from a theoretical (logarithmic) distribution fitted to a dataset relating function points to man-hours on software projects (the so-called "Finnish" dataset). A linear regression model and an Estimation-by-Analogy (EBA) model are then fitted using 1000-fold cross-validation (also known as the leave-one-out test). The authors determined that, across the 1000 test samples, neither the linear regression nor the EBA models were consistently superior, irrespective of the exact performance measure used. However, this rank-based analysis is questionable at best, as prediction errors on each fold were converted to ranks without first determining if the difference between the prediction errors for that fold was in fact s-significant.

For this experiment, our dependent variable is Cohen's kappa statistic [85], computed from the out-of-sample confusion matrix (contingency table). The kappa statistic is intended to represent the chance-corrected agreement between two raters. The kappa statistic is calculated by [86]

$$K = (F_O - F_C)/(1 - F_C) \tag{2}$$

F_O is computed as true positives plus true negatives divided by the total number of items. F_C, on the other hand, is given by [86]

$$F_C = \sum_{m=1}^{n} X_m \cdot Y_m \tag{3}$$

where $\mathrm{X_j}$ and $\mathrm{Y_j}$ are given by

$$X_b = \left(\sum_{a=1}^{n} CM_{ab}\right) \Big/ N \quad (4)$$

$$Y_b = \left(\sum_{a=1}^{n} CM_{ba}\right) \Big/ N \quad (5)$$

Eq. (3) takes the probabilities of the two raters agreeing on one category, and sums across all categories to obtain the total agreement by chance. In our experiments, we compute the kappa statistic between the outputs of a classifier and the correct classification for each input. This statistic essentially tells us the degree to which the classification is improved with respect to merely guessing the majority class.

In highly skewed datasets, where classifiers may simply predict the majority class for all instances, we believe that the kappa statistic will be more revealing than classification accuracy. The kappa statistic has been criticized as lacking a dataset-independent interpretation; values of kappa are influenced by the class distribution in a dataset [87], [88]. However, the kappa statistic is a valid tool for comparisons within a single dataset. As we are employing an experimental design with blocking between the individual datasets, kappa is thus an appropriate figure of merit. Other alternatives would include the geometric mean of the individual accuracies for each class (as in [12], [13]), the area under the Receiver Operating Characteristic (ROC) curve (often referred to as AUC), the area under the Precision-Recall curve, the Kolmogorov-Smirnov (K-S) statistic, or the F-Measure [89]. However, the AUC for a classifier is considerably more difficult to compute; one must generate several instances of a base classifier on the same dataset, varying the learning parameters to obtain different contingency tables. Each contingency table corresponds to one point on the ROC curve; the full curve itself is then approximated via interpolation between these points, and the AUC is then computed from these linear segments. As there is no know means to control the x-coordinate positions of the points identified on the ROC curve (they will be dataset- and classifier-dependent), the AUC is a somewhat approximate statistic [90]. ROC curves and AUC have been used to compare learners in previous stratification studies, but do not seem well-suited to our factorial experiment, as the classifier parameters are held constant (and thus the ROC curve cannot be generated). A similar critique applies to the area under the Precision-Recall curve. The geometric mean accuracy is again dataset-specific in that the optimal Bayes classifier (and hence the optimal classification accuracy) varies from one dataset to another [54]. As with the kappa statistic, the geometric mean accuracy is a valid tool for making comparisons within a single dataset. The F-measure is the harmonic mean of precision and recall, and the K-S statistic examines the maximum difference in the empirical distributions of the classes. Both are again dataset-specific, and valid tools for comparisons within a dataset. Furthermore, all three measures can be computed without varying the classifier, and would thus be candidates for our experimental design. However, unlike the kappa statistic, none of these three measures explicitly considers class imbalance. We have thus determined that the kappa statistic is the most congruent to our objectives for this experiment.

B. Datasets

Several SDP datasets created by the NASA Metrics Data Program are archived at the PROMISE repository [91]. We use the datasets denoted KC1, KC2, CM1, JM1, and PC1 in our experiments. These datasets can be described as quite large and skewed, but still having fifty or more examples of the minority class (meaning at least five minority samples in each test set), and low apparent noise (several PROMISE datasets have an inordinate number of "zero-size" modules).

- The project that KC1 was extracted from includes 43,000 lines of C++ code distributed into 2109 modules.
- KC2 covers over 43,000 lines of C++ code in 379 functions.
- CM1 covers 20,000 lines of C code in 498 functions.
- JM1 is a real-time system written in C, containing approximately 315,000 lines of code in 10,878 modules.
- PC1 covers 40,000 lines of C code in 941 functions.

These datasets contain twenty-one software product metrics covering the product's size, complexity [22], and vocabulary [23]. The complete list and definitions for all of these metrics are provided in [92]. The KC1 dataset is based on an object oriented program, and so a version of the KC1 dataset is available that includes ten object-oriented metrics (including the CK metrics) as well as the older procedural-programming metrics above. Method-level metrics were aggregated within a class using the arithmetic sum. The metrics in the KC1 class-level dataset and their definitions can be found in [92].

Finally, Mozilla-Delta is a new SDP dataset created and analysed in [44], [93], and comes from the 8,349 C++ classes that provide the back-end functionality for the Mozilla web browser. This dataset covers over 935,000 lines of C++ code; class-level metrics were extracted using the Krakatau Professional metrics tool [94], and associated with the number of deltas (atomic changes to a file resulting from patching a defect [95]) per class. This dataset was converted to a binary classification problem by labeling all classes with Delta > 0 as Class 1 (fault-prone), and the rest as Class 0 (fault-free). The result is a dataset of 18 numeric predictor attributes, and one categorical target attribute, with 8,349 records, and no missing values. It consists of the CK metrics (excluding CBO, which was uniformly 0) as well as several metrics for each class. Please refer to [93] for specifics.

V. Analysis of Variance Results

A. Preliminaries

Our stratification experiments combine uniform under-sampling with the original SMOTE algorithm and one of its most recent variations (SMOTE-SN). Our experiments are divided into four groups, as explained further below. To check the normality assumption of ANOVA, we use the Kolmogorov-Smirnov (K-S) normality test [96]. The p-values for the K-S test for each individual dataset and the four groupings, across both SMOTE and SMOTE-SN, are presented in Table I.

Normality is only rejected for two of the individual datasets under each algorithm at $\alpha = 0.05$, although it is rejected for all groupings. While normality is an assumption in ANOVA, violations of normality often do not lead to a material difference in the analysis outcome. Specifically, in a large sample, the type-1 error rate will be essentially unchanged even if normality is violated [97], [98]. In our experiments, we have 5 levels for each

TABLE I
KOLMOGOROV-SMIRNOV NORMALITY TEST p-VALUES

Datasets	SMOTE	SMOTE-SN
CM1	0.05	>0.15
KC1	>0.15	>0.15
KC2	0.07	0.09
KC1 class-level	<0.01	0.04
JM1	>0.15	0.04
PC1	0.08	0.15
Mozilla	>0.15	>0.15
CM1, KC1, KC2, JM1, PC1, KC1-Class, Mozilla	<0.01	< 0.01
CM1, KC1, KC2, JM1, PC1, KC1-Class	<0.01	< 0.01
CM1, KC1, KC2, JM1, PC1	<0.01	< 0.01
KC1-Class, Mozilla	<0.01	< 0.01

TABLE II
LEVENE'S TEST p-VALUES

Datasets	SMOTE	SMOTE-SN
CM1	0.221	0.507
KC1	0.743	0.527
KC2	0.905	0.732
KC1 class-level	0.834	0.896
JM1	0.096	0.143
PC1	0.447	0.531
Mozilla	0.103	0.156
CM1, KC1, KC2, JM1, PC1, KC1-Class, Mozilla	< 0.001	< 0.001
CM1, KC1, KC2, JM1, PC1, KC1-Class	< 0.001	< 0.001
CM1, KC1, KC2, JM1, PC1	< 0.001	< 0.001
KC1-Class, Mozilla	< 0.001	< 0.001

factor, yielding 25 cells; with 10 replicates per cell, we thus have a total of 250 samples for each individual dataset. Our largest grouping, consisting of all 7 datasets, thus has 1750 samples in total. Given this large size, we do not believe that the violations of normality we detect are material to the outcome of our experiments.

We check the homogeneity of variance assumptions for ANOVA, and the Tukey test using Levene's test as this test is not sensitive to the normality assumption [99]. The results of Levene's test applied to each dataset, and the different groups of datasets used in our experiments, are given in Table II.

All the individual datasets appear homoscedastic. Because the Tukey tests discussed below are conducted on individual datasets, they remain valid. However, when we group the datasets, homogeneity of variance no longer holds. To correct for heteroskedasticity, we follow Piepho's example [100], which employs the technique of [101] to adjust the degrees of freedom (df) by Box's method [102]. This procedure contains three steps. We first conduct a regular ANOVA analysis, and determine if any factors are significant. For those that are, we then apply Box's conservative test (in which we modify the degree of freedom for the numerator and denominator of the F-test to increase the critical value for the F-test; this is done by dividing out the degrees of freedom due to the factor treatments). Any factors that are still significant after Box's conservative test can be considered reliable. Finally, if any factors were significant after the basic ANOVA but were not significant per Box's test,

TABLE III
ANOVA RESULTS FOR CM1, KC1, KC2, JM1, PC1, KC1-CLASS, AND MOZILLA WITH SMOTE

Source	SS	df	Mean Square	F Value	P-value
Dataset	18.985	6	3.164	189.888	< 0.001
Under	0.428	4	0.428	25.688	< 0.001
Over	0.005	4	0.005	0.291	0.884
Under*Over	0.208	16	0.208	12.480	< 0.001
Lack of Fit	3.005	144	0.018	1.133	0.144
Pure Error	25.641	1575	0.016		

TABLE IV
p-VALUES AND ADJUSTED DEGREES OF FREEDOM FOR CM1, KC1, KC2, JM1, PC1, KC1-CLASS, AND MOZILLA WITH SMOTE

Factor	original df		adjusted df		p-value
	numerator	denominator	numerator	denominator	
Dataset	6	1719	1	286	< 0.001
Under	4	1719	1	430	< 0.001
Under*Over	16	1719	1	107	< 0.001

we then compute a *correction factor* for the degrees-of-freedom from the experimental data. In the following section, we will first present our ANOVA tables followed by the results of Box's conservative test. As will be seen, this test confirms all factors that were found to be significant in our basic ANOVA, and so it is not necessary to compute the correction factor. We then provide our Tukey test results.

B. Main Results

Our stratification experiments combine uniform under-sampling with the original SMOTE algorithm and one of its most recent variations (SMOTE-SN) [20]. We present the set of results using the original SMOTE followed by the results using SMOTE-SN. We have carried out our factorial experiments with blocking across all seven of our datasets (KC1, KC2, CM1, JM1, PC1, KC1-Class, and Mozilla). The results are analysed using the two-way ANOVA procedure based on (1). For both SMOTE and SMOTE-SN, under-sampling the datasets significantly affects the classification results. Surprisingly, the main effect of over-sampling by itself is not significant; however, the interaction between over- and under-sampling is significant at $\alpha = 0.05$. These results are given in Tables III Tables IV Tables V Tables VI. Table III presents our ANOVA analysis for all datasets using SMOTE. Box's conservative test is applied to the factors from Table III found to be significant at $\alpha = 0.05$; we present the reduced degrees-of-freedom, and the recomputed p-value for each factor in Table IV. Likewise, our results for all datasets using SMOTE-SN are presented in Table V, with the results of Box's test in Table VI.

Our ANOVA results clearly show a difference in means. To quantify those differences, we use Tukey's honestly significant difference (HSD) test [103] to determine whether under-sampling, over-sampling, or their interaction are s-significant in improving the classification of individual datasets. This test is known to be less vulnerable to a Type I error than the t-test would be in performing multiple comparisons. Tukey's HSD test considers all possible pairs of treatment means. For a

TABLE V
ANOVA RESULTS FOR CM1, KC1, KC2, JM1, PC1, KC1-CLASS, AND MOZILLA WITH SMOTE-SN

Source	SS	df	Mean Square	F Value	P-value
Dataset	21.008	6	3.501	228.667	< 0.001
Under	0.447	4	0.447	29.170	< 0.001
Over	0.007	4	0.007	0.467	0.494
Under*Over	0.183	16	0.183	11.952	< 0.001
Lack of fit	2.943	144	0.018	1.155	0.113
Pure Error	23.699	1575	0.015		

TABLE VI
p-VALUES AND ADJUSTED DEGREES OF FREEDOM FOR CM1, KC1, KC2, JM1, PC1, KC1-CLASS, AND MOZILLA WITH SMOTE-SN

Factor	original df		adjusted df		p-value
	numerator	denominator	numerator	denominator	
Dataset	6	1719	1	286	< 0.001
Under	4	1719	1	430	< 0.001
Under*Over	16	1719	1	107	< 0.001

TABLE VII
P-VALUES FOR TUKEY TESTS ON INDIVIDUAL DATASETS WITH SMOTE

Dataset	Undersampling	Oversampling	Interaction
CM1	0.516	0.086	0.042
KC1	0	0.031	0.014
KC2	0	0.048	0.001
KC1 class-level	0.336	0.228	0.152
JM1	0	0.504	0
PC1	0.054	0.760	0.934
Mozilla	0.234	0	0

pairwise comparison of all means, the null, and alternative hypotheses are respectively

$$\mathrm{H}_0 : \mu_\mathrm{i} = \mu_\mathrm{j}$$
$$\mathrm{H}_1 : \mu_\mathrm{i} \neq \mu_\mathrm{j}$$

for all i $\neq$ j. The null hypothesis is rejected if the absolute value of the sample difference exceeds $T_\alpha = q_\alpha(e,f)\sqrt{MSE/N_O}$.

For each dataset, we compare the response for the original dataset against the responses for pure under-sampling, the responses for pure over-sampling, and the responses for all resampling strategies. We again present our results for the original SMOTE, followed by the comparison using SMOTE-SN. The p-values of the Tukey tests are shown in Tables VII and VIII. We can see the individual datasets benefit differently from different resampling techniques, as expected.

C. Sensitivity Analysis

We undertake a sensitivity analysis by removing selected datasets. The first dataset to be removed is the Mozilla dataset; this dataset is unique in that it is significantly affected by both over-sampling and the under-over interaction, and yet was *not* significantly affected by under-sampling. Furthermore, this dataset was not sourced from the NASA MDP program, while the remaining datasets are commonly used in comparing different techniques within the SDP research community. Our ANOVA analysis with the Mozilla dataset removed is presented in the first row of Tables IX and X, and again shows that

TABLE VIII
P-VALUES FOR TUKEY TESTS ON INDIVIDUAL DATASETS WITH SMOTE-SN

Dataset	Undersampling	Oversampling	Interaction
CM1	0.619	0.007	0.525
KC1	0	0.671	0.025
KC2	0.001	0.028	0.001
KC1 class-level	0.104	0.464	0.472
JM1	0.153	0.315	0
PC1	0.291	0.956	0.357
Mozilla	0.163	0	0

TABLE IX
P-VALUES FOR ANOVA USING SMOTE

	Dataset	Under	Over	Under*Over	Lack of fit
CM1, KC1, KC2, JM1, PC1, KC1 Class level	0.0001	0.0001	0.2174	0.0015	0.1443
CM1, KC1, KC2, JM1, PC1	0.0001	0.0001	0.9930	0.0005	0.1058

TABLE X
P-VALUES FOR ANOVA USING SMOTE-SN

	Dataset	Under	Over	Under*Over	Lack of fit
CM1, KC1, KC2, JM1, PC1, KC1 Class level	0.0001	0.0001	0.7454	0.0034	0.2498
CM1, KC1, KC2, JM1, PC1	0.0001	0.0001	0.534	0.0114	0.1246

under-sampling is significant at $\alpha = 0.05$, the main effect of over-sampling is not significant, and the interaction between over- and under-sampling is significant. In both Tables IX and X, the first column names the datasets used in an experiment; the following four columns show the p-values for the four factors from (1): dataset, under-sampling, over-sampling, and the interaction of under- and over-sampling. The p-value for the lack of fit measure is displayed in the last column; this factor is insignificant for all groupings at $\alpha = 0.05$.

We next remove the KC1-Class dataset. While the KC1 dataset covers over 2,000 methods, these methods are grouped in approximately 170 classes, making the KC1-Class dataset actually quite small. This dataset is now also the only dataset containing object-oriented metrics. With the removal of this dataset, we are left purely with large SDP datasets containing function-level metrics. The results for this group are presented in the second row of Tables IX and X. Again, the results are the same: under-sampling is significant, the main effect of over-sampling is not, but the interaction between under- and over-sampling is. The results of these four datasets show that our main results were not caused by unique behaviors in the most “unusual” datasets. As before, Box’s conservative test shows that all factors found significant in the basic ANOVA remained significant; we omit these results due to space considerations.

D. Exploratory Analysis of Resampling Object-Oriented Datasets

As we have two datasets available that contain object-oriented metrics, we have undertaken an exploratory analysis of resampling in SDP datasets using object-oriented metrics. However, as KC1-Class contains a mixture of class-level and method-level metrics, and is in any case quite small, while the Mozilla dataset uses a different set of metrics, caution is urged in interpreting these results. An ANOVA analysis finds very different results

TABLE XI
ANOVA p-VALUES FOR KC1-CLASS AND MOZILLA

	Dataset	Under	Over	Under*Over	Lack of fit
SMOTE	0.0001	0.4590	0.2710	0.2546	0.3142
SMOTE-SN	0.0001	0.2990	0.9195	0.0136	0.2192

from the full analysis in part A. In Table XI, we find that none of the effects we are testing were significant using SMOTE. Under-sampling, over-sampling, and their interaction were all insignificant at $\alpha = 0.05$. However, with the SMOTE-SN algorithm, the interaction of under-sampling and over-sampling is significant. Again, Box's conservative test shows that all factors found significant in Table XI remain so.

E. Discussion

In general, our results indicate that uniform under-sampling, and the interaction between under-sampling and over-sampling, are valuable in improving the accuracy of SDP models. While over-sampling by itself is not significant at the 0.05 percent level, its utility in combination with random under-sampling cannot be overlooked. This finding holds for both the original SMOTE algorithm and the more recent SMOTE-SN variation. Based on these results, we do not advocate employing either over-sampling technique in isolation. Our results indicate that software quality analysts would benefit by applying both under- and over-sampling in creating SDP models; clearly, a structured method for exploring the resulting design space is needed. One possibility is to use the fivefold cross-validation strategy in [72]; another is to employ a classical response-surface methodology to find the optimal resampling strategy. An investigation of these and other possibilities is urged, but is beyond the scope of the current article.

VI. Threats to Validity

We examine the threats to internal and external validity in our experimental design. The threats to internal validity include adequacy of the linear model for ANOVA, the instrumentation threat of having metrics collected by different tools and personnel, and having a different set of metrics in two of the datasets. As to the first, we used the lack-of-fit test to protect against inadequacy of the model, as described in Section IV-A. Instrumentation threats arise from the fact that the datasets were collected at different locations and times by different personnel; they have been minimized as much as possible by using well-known benchmark datasets collected specifically for this purpose by NASA. Metrics in the Mozilla dataset, which is the sole exception to this rule, were collected using a commercial tool. Collection of defect data for this dataset was performed on a full copy of the Bugzilla defect-tracking database [93]. Finally, the different metrics employed in KC1-Class and Mozilla are a consequence of using object-oriented development; this instrumentation threat is unavoidable as object-oriented programs are not reasonably represented by method-level metrics alone. However, our sensitivity analysis in Section V-A indicates that the results were not unduly biased by these two datasets.

Threats to external validity include the choice of levels adopted in our study, and the preponderance of datasets from NASA's MDP program, which could introduce a source bias into the study. The levels we chose for our ANOVA experiments were based on our previous work [13]; in those experiments, we observed an improvement of at least 23% in the geometric mean accuracy of the C4.5 classifier on a subset of the datasets in the current study, using those levels. Source bias is an important concern, as six out of seven datasets were obtained from the NASA Metrics Data Program. These datasets, however, are now widely-used in comparing SDP techniques, and we have further alleviated the source bias by incorporating the Mozilla dataset.

VII. Conclusion and Future Work

We use stratification to deal with the imbalanced dataset problem in software defect prediction datasets. Using the ANOVA procedure, and a blocked factorial design, we respond to the debate about the usefulness of over-sampling vs. under-sampling within the machine learning community. Our analysis shows that under-sampling and the interaction with over-sampling significantly improves the quality of software defect predictions. However, the main effect of over-sampling was not significant.

In future work, we intend to investigate response surface methods as a structured approach to identify the best resampling strategy for a specific software defect prediction dataset in the context of test management. In addition, software defect prediction is one example of the more general problem of binary classification, and the characteristics of SDP datasets are not generally shared by other examples. We will therefore repeat the current study using a broad selection of binary classification problems. This work will contribute to the evidence on the efficacy of under- and over-sampling in the machine-learning literature. Finally, we will investigate the anomalous results obtained for class-level metrics using an expanded group of class-level datasets.

References

[1] M. Levinson, Let's Stop Wasting $78 Billion per Year CIO Magazine, October 15, 2001.

[2] L. L. Pullum, *Software Fault Tolerance Techniques and Implementation*. Boston, MA, USA: Artech House, 2001.

[3] Z. Jelinski and P. B. Moranda, "Software reliability research," in *Proc. Statistical Computer Performance Evaluation*, Providence, RI, USA, 1971, pp. 465–484.

[4] K. E. Emam, A Methodology for Validating Software Product Metrics National Research Council of Canada, , Technical Report NRC/ERB-1076, 2000.

[5] K. E. Emam, S. Benlarbi, N. Goel, and S. N. Rai, "The confounding effect of class size on the validity of object-oriented metrics," *IEEE Trans. Soft. Eng.*, vol. 27, pp. 630–650, 2001.

[6] C. X. Ling, S. Sheng, T. Bruckhaus, and N. H. Madhavji, "Predicting software escalations with maximum ROI," in *Proc. IEEE Int. C. Data Mining*, Houston, TX, USA, 2005, pp. 717–720.

[7] T. M. Khoshgoftaar, E. Geleyn, L. Nguyen, and L. Bullard, "Cost-sensitive boosting in software quality modeling," in *Proc. IEEE Int. Symp. High Assurance Systems Engineering*, Tokyo, Japan, 2002, pp. 51–60.

[8] T. M. Khoshgoftaar, E. B. Allen, W. D. Jones, and J. P. Hudepohl, "Data mining for predictors of software quality," *Int. J. Soft. Eng. Knowl. Eng.*, vol. 9, pp. 547–563, 1999.

[9] C. X. Ling, V. S. Sheng, T. Bruckhaus, and N. H. Madhavji, "Maximum profit mining and its application in software development," in *Proc. ACM SIGKDD Int. C. Knowledge Discovery and Data Mining*, Philadelphia, PA, USA, 2006, pp. 929–934.

[10] D. J. Drown, T. M. Khoshgoftaar, and N. Seliya, "Evolutionary sampling and software quality modeling of high-assurance systems," *IEEE Trans. Systems, Man and Cybernetics, Part A: Systems and Humans*, vol. 39, pp. 1097–1107, 2009.

[11] K. Kaminsky and G. D. Boetticher, "Better software defect prediction using equalized learning with machine learners," in *Proc. Knowledge Sharing and Collaborative Engineering*, St. Thomas, US Virgin Islands, 2004.
[12] S. Dick and A. Kandel, "Data mining with resampling in software metrics databases," in *Artificial Intelligence Methods in Software Testing*, M. Last, Ed. *et al.* : World Scientific, 2004, pp. 175–208.
[13] L. Pelayo and S. Dick, "Applying novel resampling strategies to software defect prediction," in *Proc. North American Fuzzy Information Processing Soc.*, San Diego, CA, 2007, pp. 69–72.
[14] C. Seiffert, T. M. Khoshgoftaar, and J. Van Hulse, "Improving software-quality predictions with data sampling and boosting," *IEEE Trans. Systems, Man and Cybernetics, Part A: Systems and Humans*, vol. 39, pp. 1283–1294, 2009.
[15] H. Haibo and E. A. Garcia, "Learning from Imbalanced Data," *IEEE Trans. Knowledge and Data Engineering*, vol. 21, pp. 1263–1284, 2009.
[16] G. Weiss and F. Provost, "Learning when training data are costly: The effect of class distribution on tree induction," *Journal of Artificial Intelligence Research*, vol. 19, pp. 315–354, 2003.
[17] G. Batista, R. Prati, and M. C. Monard, "A study of the behavior of several methods for balancing machine learning training data," *Sigkdd Explorations*, vol. 6, pp. 20–29, 2004.
[18] A. Estrabooks, T. Jo, and N. Japkowicz, "A multiple resampling method for learning from imbalanced datasets," *Computational Intelligence*, vol. 20, pp. 18–36, 2004.
[19] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Syntethic minority over-sampling technique," *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, Jun. 2002.
[20] V. Garcıacute;a, J. S. Sánchez, and R. A. Mollineda, "On the use of surrounding neighbors for synthetic over-sampling of the minority class," in *Proc. 8th C. Simulation, Modelling and Optimization Santander*, Cantabria, Spain, 2008, pp. 389–394.
[21] J. Van Hulse, T. M. Khoshgoftaar, and A. Napolitano, "Experimental perspectives on learning from imbalanced data," in *Proc. Int. C. Machine Learning*, Corvalis, OR, USA, 2007, pp. 935–942.
[22] T. J. McCabe, "A complexity measure," *IEEE Trans. Software Eng.*, vol. 2, pp. 308–320, 1976.
[23] M. H. Halstead, *Elements of Software Science*. New York: Elsevier, 1977.
[24] B. Boehm, B. Clark, E. Horowitz, and C. Westland, "Cost models for future software life cycle processes: COCOMO 2.0," *Annals of Software Engineering*, vol. 1, pp. 57–94, 1995.
[25] R. Harrison, S. Counsell, and R. Nithi, "Coupling metrics for object-oriented design," in *Int. Soft. Metrics Symp.*, Bethesda, MD, USA, 1998, pp. 150–157.
[26] J. C. Munson and T. M. Khoshgoftaar, "Software metrics for reliability assessment," in *Handbook of Software Reliability Engineering*, M. R. Lyu, Ed. New York, NY: McGraw-Hill, 1996, pp. 167–254.
[27] K. Gao and T. M. Khoshgoftaar, "A comprehensive empirical study of count models for software fault prediction," *IEEE Trans. Reliability*, vol. 56, pp. 223–236, 2007.
[28] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Trans. Software Engineering*, vol. 20, pp. 476–493, 1994.
[29] M. Lorenz and J. Kidd, *Object-Oriented Software Metrics*. Englewood Cliffs, NJ, USA: Prentice Hall, 1994.
[30] L. C. Briand, J. Daly, and J. Wust, "A unified framework for coupling measurement in object-oriented systems," *IEEE Trans. Soft. Eng.*, vol. 25, pp. 91–121, 1999.
[31] L. C. Briand, J. Daly, V. Porter, and J. Wust, "A comprehensive empirical validation of design measures for object-oriented systems," in *Proc. Int. Soft. Metrics. Symp.*, Bethesda, MD, USA, 1998, pp. 246–257.
[32] S. Counsell, S. Swift, and J. Crampton, "The interpretation and utility of three cohesion metrics for object-oriented design," *ACM Trans. Soft. Eng. Method*, vol. 15, pp. 123–149, 2006.
[33] T. M. Meyers and D. Binkley, "An empirical study of slice-based cohesion and coupling metrics," *ACM Trans. Software Engineering and Methodology*, vol. 17, pp. 2:1–2:27, 2008.
[34] A. R. Gray, "A simulation-based comparison of empirical modeling techniques for software metric models of development effort," in *Proc. Int. C. Neural Information Processing*, Perth, Australia, 1999, pp. 526–531.
[35] N. Karunanithi and Y. K. Malaiya, "Neural networks for software reliability engineering," in *Handbook of Software Reliability Engineering*, M. R. Lyu, Ed. New York, NY, USA: McGraw-Hill, 1996, pp. 699–728.
[36] T. M. Khoshgoftaar and R. M. Szabo, "Using neural networks to predict software faults during testing," *IEEE Trans. Reliability*, vol. 45, pp. 456–462, 1996.
[37] M. Shin and A. L. Goel, "Knowledge discovery and validation in software metrics databases," in *Proc. SPIE*, 1999, vol. 3695, pp. 226–233.
[38] E. Baisch, T. Bleile, and R. Belschner, "A neural fuzzy system to evaluate software development productivity," in *Proc. Int. C. Syst., Man, Cybern.*, Vancouver, BC, Canada, 1995, pp. 4603–4608.
[39] J. S. Mertoguno, R. Paul, N. G. Bourbakis, and C. V. Ramamoorthy, "A neuro-expert system for the prediction of software metrics," *Eng. Apps. Art. Int.*, vol. 9, pp. 153–161, 1996.
[40] C. Ebert, "Fuzzy classification for software criticality analysis," *Expert Sys. With Apps.*, vol. 11, pp. 323–342, 1996.
[41] C. Ebert and E. Baisch, "Knowledge-based techniques for software quality management," in *Computational Intelligence in Software Engineering*, W. Pedrycz and J. F. Peters, Eds. River Edge, NJ, USA: World Scientific, 1998, pp. 295–320.
[42] T. M. Khoshgoftaar, M. P. Evett, E. B. Allen, and P. D. Chien, "An application of genetic programming to software quality prediction," in *Computational Intelligence in Software Engineering*, W. Pedrycz and J.F. Peters, Eds. River Edge, NJ, USA: World Scientific, 1998, pp. 176–195.
[43] T. M. Khoshgoftaar and Y. Liu, "A multi-objective software quality classification model using genetic programming," *IEEE Trans. Reliability*, vol. 56, pp. 237–245, 2007.
[44] S. Dick and A. Sadia, "Fuzzy clustering of open-source software quality data: A case study of mozilla," in *Proc. Int. Joint. C. Neural Networks*, Vancouver, BC, 2006, pp. 4089–4096.
[45] X. Yuan, T. M. Khoshgoftaar, E. B. Allen, and K. Ganesan, "An application of fuzzy clustering to software quality prediction," in *Proc IEEE Symp. App. Spc. Soft. Eng. Tech.*, Richardson, TX, USA, 2000, pp. 85–90.
[46] N. Seliya and T. M. Khoshgoftaar, "Software quality analysis of unlabeled program modules with semisupervised clustering," *IEEE Trans. Systems, Man and Cybernetics, Part A: Systems and Humans*, vol. 37, pp. 201–211, 2007.
[47] T. M. Khoshgoftaar, E. B. Allen, W. D. Jones, and J. P. A. Hudepohl, "Classification-tree models of software-quality over multiple releases," *IEEE Trans. Reliability*, vol. 49, pp. 4–11, 2000.
[48] M. A. D. Almeida, H. Lounis, and W. L. Melo, "An investigation on the use of machine learned models for estimating software correctability," *Int. J. Soft. Eng. Knowl. Eng.*, vol. 9, pp. 185–215, 1997.
[49] Y. Bo and L. Xiang, "A study on software reliability prediction based on support vector machines," in *IEEE Int. C. Industrial Engineering and Engineering Management*, Singapore, 2007, pp. 1176–1180.
[50] K. O. Elish and M. O. Elish, "Predicting defect-prone software modules using support vector machines," *Journal of Systems and Software*, vol. 81, pp. 649–660, 2008.
[51] P.-F. Pai and W.-C. Hong, "Software reliability forecasting by support vector machines with simulated annealing algorithms," *Journal of Systems and Software*, vol. 79, pp. 747–755, 2006.
[52] T. Fawcett and F. Provost, "Adaptive fraud detection," *Data Mining and Knowledge Discovery*, vol. 1, Sep. 1997.
[53] D. J. Newman, S. Hettich, C. L. Blake, and C. J. Merz, UCI Repository of Machine Learning Databases [Online]. Available: http://www.ics.uci.edu/~mlearn/MLRepository.html
[54] T. M. Mitchell, *Machine Learning*. New York, NY: McGraw-Hill, 1997.
[55] C. Elkan, "The foundations of cost-sensitive learning," in *Proc. Int. Joint C. Artificial Intelligence*, Seattle, WA, USA, 2001, pp. 973–978.
[56] P. D. Turney, "Cost-sensitive classification: empirical evaluation of a hybrid genetic decision tree induction algorithm," *Journal of Artificial Intelligence Research*, vol. 2, pp. 369–409, 1995.
[57] K. M. Ting and Z. Zheng, "Boosting trees for cost-sensitive classifications," in *Proc. Eur. C. Machine Learning*, Chemnitz, Germany, 1998, pp. 191–195.
[58] P. Sen and L. Getoor, "Cost-sensitive learning with conditional Markov networks," *Data Mining and Knowledge Discovery*, vol. 17, pp. 136–163, 2008.
[59] G. M. Weiss and Y. Tian, "Maximizing classifier utility when there are data acquisition and modeling costs," *Data Mining and Knowledge Discovery*, vol. 17, pp. 253–282, 2008.
[60] P. Domingos, "MetaCost: A general method for making classifiers cost-sensitive," in *Proc Int. C. Knowledge Discovery and Data Mining*, San Diego, CA, USA, 1999, pp. 155–164.

[61] C. Seiffert, T. M. M. Khoshgoftaar, J. Van Hulse, and A. Napolitano, "A comparative study of data sampling and cost sensitive learning," in *IEEE Int. C. Data Mining Workshops*, Pisa, Italy, 2008, pp. 46–52.

[62] D. Wilson, "Asymptotic properties of nearest neighbor rules using edited data sets," *IEEE Trans. Systems, Man and Cybernetics*, vol. 2, pp. 408–421, 1972.

[63] M. Kubat and S. Matwin, "Addressing the curse of imbalanced training set: One-sided selection," in *Proc 14th Int. C. Machine Learning*, Nashville, TN, USA, 1997, pp. 179–186.

[64] H. Ahumada, G. L. Grinblat, L. C. Uzal, P. M. Granitto, and A. Ceccatto, "REPMAC: A new hybrid approach to highly imbalanced classification problems," in *Proc. Int. C. Hybrid Intelligent Systems*, Barcelona, Spain, 2008, pp. 386–391.

[65] N. V. Chawla, A. Lazarevic, L. O. Hall, and K. W. Bowyer, "Smoteboost: Improving prediction of the minority class in boosting," in *Proc. 7th Eur. C. Principles and Practice of Knowledge Discovery in Databases*, Dubrovnik, Croatia, 2003, pp. 107–119.

[66] H. Han, W.-Y. Wang, and B.-H. Mao, "Borderline-SMOTE: A new over-sampling method in imbalanced data sets learning," *Lecture Notes in Computer Science*, vol. 3644, pp. 878–887, 2005.

[67] A. Liu, C. Matinn, B. La Cour, and J. Ghosh, "Effects of oversampling versus cost-sensitive learning for bayesian and SVM classifiers," in *Data Mining*, R. Stahlbock, Ed. *et al.* New York, NY: Springer, 2010, pp. 159–192.

[68] T. Jo and N. Japkowicz, "Class imbalances versus small disjuncts," *SIGKDD Explorations*, vol. 6, pp. 40–49, 2004.

[69] S. Gazzah and N. E. B. Amara, "New oversampling approaches based on polynomial fitting for imbalanced data sets," in *Proc. IAPR Wkshp. Document Analysis Systems*, Nara, Japan, 2008, pp. 677–684.

[70] U. M. Fayyad, "Data mining and knowledge discovery: Making sense out of data," *IEEE Expert*, vol. 11, pp. 20–25, 1996.

[71] J. R. Quinlan, *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann Publishers, 1993.

[72] N. V. Chawla, D. A. Cieslak, L. O. Hall, and A. Joshi, "Automatically countering imbalance and its empirical relationship to cost," *Data Mining and Knowledge Discovery*, vol. 17, pp. 225–252, 2008.

[73] C. Seiffert, T. M. Khoshgoftaar, and J. Van Hulse, "Hybrid sampling for imbalanced data," *Integrated Computer-Aided Engineering*, vol. 16, pp. 193–210, 2009.

[74] C. Seiffert, T. M. Khoshgoftaar, J. Van Hulse, and A. Napolitano, "RUSBoost: Improving classification performance when training data is skewed," in *Proc. Int. C. Pattern Recognition*, Tampa, FL, USA, 2008, pp. 1–4.

[75] C. Seiffert, T. M. Khoshgoftaar, J. Van Hulse, and A. Napolitano, "RUSBoost: A hybrid approach to alleviating class imbalance," *IEEE Trans. Systems, Man and Cybernetics, Part A: Systems and Humans*, vol. 40, pp. 185–197, 2010.

[76] T. M. Khoshgoftaar, J. Van Hulse, and A. Napolitano, "Comparing boosting and bagging techniques with noisy and imbalanced data," *IEEE T. Systems, Man, Cybernetics, Part A: Systems and Humans*, vol. 41, pp. 552–568, 2011.

[77] D. Montgomery, *Design and Analysis of Experiments*, 5th ed. New York: John Wiley, 2001.

[78] J. Kam and S. Dick, "Comparing nearest-neighbour search strategies in the SMOTE algorithm," *Canadian J. Electrical and Computer Engineering*, vol. 31, pp. 203–210, 2006.

[79] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM*, vol. 18, pp. 509–517, 1975.

[80] M. Hall *et al.*, "The WEKA data mining software: An update," *SIGKDD Explorations*, vol. 11, pp. 10–18, 2009.

[81] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*. New York, NY: John Wiley & Sons, Inc., 2001.

[82] P. G. Mathews, *Design of Experiments with MINITAB*. Milwaukee, WI, USA: Quality Press, 2005.

[83] M. G. Marasinghe and W. J. Kennedy, *SAS for Data Analysis: Intermediate Statistical Methods*. New York, NY: Springer Science+Business Media LLC, 2008.

[84] I. Myrtveit, E. Stensrud, and M. Shepperd, "Reliability and validity in comparative studies of software prediction models," *IEEE Transactions on Software Engineering*, vol. 31, pp. 380–391, 2005.

[85] L. Rourke, T. Anderson, D. R. Garrison, and W. Archer, "Methodological issues in the content analysis of computer conference transcripts," *Int. J. Artificial Intelligence in Education*, vol. 12, pp. 8–22, 2001.

[86] B. D. Eugenio and M. Glass, "The kappa statistic: A second look," *Computational Linguistics*, vol. 30, pp. 95–101, 2004.

[87] W. D. Thompson and S. D. Walter, "A reappraisal of the kappa coefficient," *Journal of Clinical Epidemiology*, vol. 41, pp. 949–958, 1998.

[88] A. Feinstein and D. Cicchetti, "High agreement but low kappa: I. The problems of two paradoxes," *Journal of Clinical Epidemiology*, vol. 43, pp. 543–549, 1990.

[89] A. Folleco, T. M. Khoshgoftaar, and A. Napolitano, "Comparison of four performance metrics for evaluating sampling techniques for low quality class-imbalanced data," in *Proc. Int. C. Machine Learning and Applications*, San Diego, CA, USA, 2008, pp. 153–158.

[90] A. P. Bradley, "The use of the area under the ROC curve in the evaluation of machine learning algortihms," *Pattern Recognition*, vol. 30, pp. 1145–1159, 1997.

[91] G. Boetticher, T. Menzies, and T. Ostrand, PROMISE Repository of Empirical Software Engineering Data [Online]. Available: http://promisedata.org/repository 2008

[92] J. Long, Metrics Data Program 2008 [Online]. Available: http://mdp.ivv.nasa.gov/index.html

[93] L. Pelayo, A. Sadia, and S. Dick, "Predicting object-oriented software quality: A case study," *International Journal of Intelligent Control Systems*, vol. 14, pp. 180–192, 2009.

[94] Staff, Power Software - Products - Krakatau Professional [Online]. Available: http://www.powersoftware.com/kp/ 2008

[95] A. Mockus, T. Roy, and J. D. Herbsleb, "Two case studies of open source software development: Apache and Mozilla," *ACM Trans. Softw. Eng. Methodol*, vol. 11, pp. 309–346, Jul. 2002.

[96] E. H. Wolf and J. I. Naus, "Tables of critical values for a k-sample kolmogorov-smirnov test statistic," *Journal of the American Statistical Association*, vol. 68, pp. 994–997, 1973.

[97] H. Scheffe*, The Analysis of Variance*. New York, NY: John Wiley & Sons, Inc., 1959.

[98] H. R. Lindman, *Analysis of Variance in Complex Experimental Designs*. San Francisco, CA, USA: W.H. Freeman & Co., 1974.

[99] H. Levene, "Robust tests for equality of variance," in *Contributions to Probability and Statistics: Essays in Honor of Harold Hotelling*, I. Olkin, Ed. *et al.* Palo Alto, CA, USA: Stanford University Press, 1960, pp. 278–292.

[100] H. P. Piepho, "Detecting and handling heteroscedasticity in yield trial data," *Communications in Statistics-Simulation and Computation*, vol. 24, pp. 243–274, 1995.

[101] S. W. Greenhouse and S. Geisser, "On methods in the analysis of profile data," *Psychometrika*, vol. 24, pp. 95–112, 1959.

[102] G. E. P. Box, "Some theorems on quadratic forms in the study in analysis of variance problems II. Effect of inequality of variance and correlation between errors in the two-way classification," *Annals of Mathematical Statistics*, vol. 25, pp. 484–498, 1954.

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

Lourdes Pelayo received the B.S. degree in Digital Systems and Communications Engineering in 2000, and the MBA in 2003 from University of Juarez, Mexico. She is currently pursuing her Ph.D. in Computer Engineering at the University of Alberta, Canada. She was a member of the technical committee of FUZZ-IEEE 2008, and of the technical program committee of IFSA/EUSFLAT09.

Scott Dick (S'97–M'02) received his B.Sc. degree in 1997, his M.Sc. degree in 1999, and his Ph.D. in 2002, all from the University of South Florida. His Ph.D. dissertation received the USF Outstanding Dissertation Prize in 2003. From 2002 to 2008, he was an Assistant Professor of Electrical and Computer Engineering at the University of Alberta in Edmonton, AB. Since 2008, he has been an Associate Professor in the same department, and has published over 40 scientific articles in journals and conferences. Dr. Dick is a member of the ACM, IEEE, ASEE, and an associate member of Sigma Xi.