2008 20th IEEE International Conference on Tools with Artificial Intelligence

Resampling or Reweighting: A Comparison of Boosting Implementations

Chris Seiffert (chrisseiffert@gmail.com)
Taghi M. Khoshgoftaar (taghi@cse.fau.edu)
Jason Van Hulse (jvanhulse@gmail.com)
Amri Napolitano (anapoli1@fau.edu)
Florida Atlantic University, Boca Raton, Florida, USA

Abstract

Boosting has been shown to improve the performance of classifiers in many situations, including when data is imbalanced. There are, however, two possible implementations of boosting, and it is unclear which should be used. Boosting by reweighting is typically used, but can only be applied to base learners which are designed to handle example weights. On the other hand, boosting by resampling can be applied to any base learner. In this work, we empirically evaluate the differences between these two boosting implementations using imbalanced training data. Using 10 boosting algorithms, 4 learners and 15 datasets, we find that boosting by resampling performs as well as, or significantly better than, boosting by reweighting (which is often the default boosting implementation). We therefore conclude that in general, boosting by resampling is preferred over boosting by weighting.

1 Introduction

Boosting is a meta learning technique designed to improve classification performance. A weak learner's performance can be "boosted" by creating an ensemble of weak hypotheses and combining them to create a final, stronger, hypothesis. AdaBoost, a popular boosting algorithm proposed by Freund and Schapire [10], has been shown to improve the performance of any classifier with classification performance better than random guessing. AdaBoost iteratively builds an ensemble of classifiers, adjusting the weights of each example based on the performance of the iteration's classifier. Examples that were misclassified have their weights increased, while those that were correctly classified have their weights decreased. Therefore, in the next iteration the classifier is more likely to correctly classify examples that were misclassified during the current iteration. Once the stopping criteria is met, the classifiers built during each iteration participate in a weighted vote to classify unlabeled examples.

The weight changes that occur during boosting can be implemented in two ways: reweighting or resampling. When performing *boosting by reweighting*, the numerical weights for each example are passed directly to the base learner. The base learner uses the weighting information when forming its hypothesis. However, not all learning algorithms are designed to handle example weight information [10]. Therefore, the alternative approach, *boosting by resampling*, may be more appropriate. Rather than pass example weights to the learner, the training data can be resampled to reflect these weights. A new training dataset with the same size as the original is created by sampling (with replacement) from the original training dataset where the probability that each example is selected is proportional to its assigned weight. A model built on this resampled training data will therefore place more emphasis on examples that were assigned higher weights because they appear more frequently in the training data.

In this work, we examine the performance of boosting in the context of mining imbalanced data. Data is *imbalanced* if the examples of one class greatly outnumber the examples of the other class(es). Domains such as medical diagnosis, fraud detection, network security and software quality prediction often suffer from imbalanced data. When data is imbalanced, many traditional learners tend to favor classifying examples as belonging to the majority (overrepresented) class, frequently misclassifying examples of the minority (underrepresented) class. In most cases, however, it is the minority class that carries a higher cost of misclassification. Therefore, some modifications are required to force learners to focus more on examples of the minority, or positive, class. Boosting, while not specifically designed to address the issue of class imbalance, has performed very well in this regard [17]. Since examples of the minority class are likely to be misclassified, boosting will focus on correctly classifying these examples in later iterations. The result is a model that is better able to distinguish between minority and majority class examples than if the base learner was used alone.

We investigate the performance of AdaBoost along with nine variations of AdaBoost when learning from imbalanced data. Eight of these variations [9, 18, 19, 11] modify the way AdaBoost adjusts example weights, treating examples of the minority (positive) and majority (negative) classes differently. The final variation, SMOTEBoost [5], combines data sampling and boosting. It is not the goal of this work to compare the performance of each boosting algorithm. That is, we are not interested in determining which algorithm performs best in the context of class imbalance. Instead, we investigate which implementation (reweighting or resampling) is better for each boosting algorithm. While identifying the best performing boosting algorithm is not our goal, the reader may draw conclusions on this matter based on the results presented in this work.

This work presents a comprehensive empirical study of resampling and reweighting. A very preliminary work on this topic was performed by Botta [3], which compared the performance of AdaBoost by resampling and reweighting using WWIL (an extension of his Weak Inductive Learner, WIL). Our work includes a considerably more rigorous set of experiments using 4 commonly used learners, 10 boosting algorithms and 15 imbalanced datasets from various application domains. All results are tested for statistical significance, the results of which are presented and used when drawing conclusions. Our results show that, in general, boosting by resampling tends to perform as well or better than boosting by reweighting. This result is important, because boosting by reweighting tends to be the "default" technique.

2 Boosting Algorithms

With the exception of AdaBoost, each of these boosting algorithms was implemented using Java by our research group within the Weka [22] machine learning tool. AdaBoost is available in Weka using both the reweighting and resampling options. In this section, we present a brief description of the ten boosting algorithms used in this study: AdaBoost, AdaCost, CSB0, CSB1, CSB2, AdaC1, AdaC2, AdaC3, RareBoost and SMOTEBoost. For complete details on these algorithms, please refer to the referenced works.

AdaBoost [10] serves as the basis for all of the other boosting algorithms explored in this work. Proposed by Freund and Schapire, AdaBoost is the most well-known and frequently studied boosting algorithm. This study uses AdaBoost.M1, one of several AdaBoost variants. AdaBoost iteratively builds an ensemble of weak hypotheses. During each iteration, a weak hypothesis is formed. The error associated with the hypothesis is calculated and the weight of each example is adjusted such that misclassified examples have their weights increased while correctly classified examples have their weights decreased. In doing so, examples that were misclassified by the current iteration's hypothesis are more likely to be correctly classified by the next iteration's hypothesis. The hypotheses from each iteration participate in a weighted vote to classify unlabeled examples. In this study, we use ten iterations for all boosting algorithms. Preliminary experiments showed no significant improvement when additional iterations were performed.

AdaCost [9] provides a cost-sensitive alternative to AdaBoost. The key difference between AdaCost and AdaBoost is the formula for updating example weights. While AdaBoost treats examples of each class similarly, AdaCost differentiates between examples of the positive and negative classes. Further, it incorporates a user-specified cost ratio into its "misclassification cost adjustment function." In doing so, AdaCost can place emphasis on positive class examples based on input from the user. Information about the cost ratios used for this study are provided in Section 3.4.

We also examine six other variations of AdaBoost. CSB0, CSB1 and CSB2 [19] are three cost-sensitive variations of AdaBoost. Like AdaCost, these techniques use modified formulas for re-weighting examples in order to minimize the number of expensive errors, and therefore total cost. Although these algorithms are presented in the context of cost-sensitive learning, there is a close relationship between cost-sensitive learning and learning from imbalanced training data. Cost sensitive learners can be constructed by re-balancing data [8], and class imbalance can be overcome by cost-sensitive learning [20]. AdaC1, AdaC2 and AdaC3 [18] also modify AdaBoost's re-weighting formula. These three techniques are presented as cost sensitive boosting algorithms for imbalanced data. AdaCost, CSB0, CSB1, CSB2, AdaC1, AdaC2 and AdaC3 all require a user-specified cost ratio

A somewhat different boosting algorithm is RareBoost [11] (denoted RareBoost-1 in the referenced work), another variation of AdaBoost. RareBoost was designed specifically to address the problem of class imbalance. Unlike cost-sensitive boosting techniques, RareBoost does not take cost into account when reweighting examples. Instead, the updated weights are based on how well an iteration's generated hypothesis distinguishes between false positives and true positives as well as false negatives and true negatives. RareBoost does not require a user-specified cost ratio.

Finally, SMOTEBoost [5] combines data sampling with boosting. The data sampling aspect of SMOTEBoost uses the Synthetic Minority Oversampling Technique (SMOTE) [4]. SMOTE is an oversampling technique that creates new minority class examples by finding the k nearest neighbors [1] (we use the recommended value, $k = 5$) of each minority example and extrapolating between these examples. The number of examples created depends on a user specified level of oversampling. SMOTEBoost combines SMOTE with AdaBoost.M2 by performing SMOTE prior to constructing the weak hypothesis during each iteration to achieve the user-specified target class distribution.

Each of the above algorithms are implemented using both reweighting and resampling. *Boosting by reweighting* is the "default" approach to boosting. That is, if the base learner is able to use example weight information then boosting by reweighting is usually applied. However not all base learners are designed to incorporate example weights into their model building process. *Boosting by resampling* is a data-level approach to boosting that can be applied to any base learner. Rather than pass example weights to the learner, the training data is resampled (with replacement) according to the weight distribution to create a new training dataset. Examples with higher weights are more likely to be selected (often multiple times) while examples with lower weights are less likely to be selected. A model trained on such a resampled dataset is more likely to correctly classify examples that appear multiple times in the dataset over those that occur only once (if at all).

Boosting by reweighting is straightforward for nine of the ten boosting algorithms examined in this study. For each of the algorithms (except SMOTEBoost) examples are simply assigned weights based on the results of a given iteration's weak hypothesis. However, when using SMOTE, new examples are created and the question remains: What weights should these new examples receive? This topic is not addressed in the proposal of SMOTEBoost [5], and therefore we implement boosting by reweighting in a manner that is logical and approximates the boosting by resampling strategy as closely as possible. SMOTE creates new examples by selecting a point at some distance (determined by a random number between 0 and 1) along the line segment (in feature space) connecting a minority class example and one of its nearest neighbors. In our implementation, we assign weights to these examples which are the same distance along the line segment between the two examples as measured in a one dimensional feature space defined only by the instances' weights. That is, if instances x_i and x_j are selected to create a new instance, x_k, using the random value λ, then the new example's weight, $W(x_k)$, is defined as $\lambda \times W(x_i) + (1 - \lambda) \times W(x_j)$. The example weights are then scaled so that the each class accounts for the desired percentage of the total dataset cost. Our results show that this implementation performs very well when compared to SMOTEBoost by resampling.

Dataset	Size	# min	% min	# attr
SP3	3541	47	1.33	43
MAMMOGRAPHY	11183	260	2.32	7
SOLARFLAREF	1389	51	3.67	13
CAR3	1728	69	3.99	7
CCCS12	282	16	5.67	9
SP1	3649	229	6.28	43
PC1	1107	76	6.87	16
GLASS3	214	17	7.94	10
CM1	505	48	9.50	16
PENDIGITS5	10992	1055	9.60	17
SATIMAGE4	6435	626	9.73	37
ECOLI4	336	35	10.42	8
SEGMENT5	2310	330	14.29	20
CONTRA2	1473	333	22.61	10
VEHICLE1	846	212	25.06	19

Table 1. Dataset caracteristics

3 Experiments

3.1 Datasets

The results reported in this study are based on 15 datasets from various application domains. Details about these datasets are presented in Table 1. This table provides the total number of examples in the dataset (Size), as well as the number of minority examples (#min), the percentage of examples belonging to the minority class (%min), commonly referred to as the *level of imbalance*, and the number of attributes (#attr). The datasets are sorted by imbalance level. The datasets SP1, SP3 and CCCS12 are proprietary software project datasets obtained from Nortel Networks and the Department of Defense. PC1 and CM1 are publicly available software project datasets from the data metrics program at NASA [14]. The Mammography dataset is from the medical diagnosis domain and was generously provided by Dr. Nitesh Chawla [4]. The remaining datasets were obtained from the popular UCI repository [2] and represent various application domains. Since this work considers only the binary classification problem, it was necessary to transform some of the datasets to have a binary class.

3.2 Learners

This study uses four learning algorithms, all of which are implemented in Weka [22], an open source data mining suite. Two versions of the well-known C4.5 [16] decision tree classifier are used, denoted C4.5D and C4.5N. C4.5D uses the Weka default parameters, while C4.5N disables pruning and enables Laplace smoothing [21]. Naive Bayes (NB) uses Bayes' rule of conditional probability to classify examples. Repeated Incremental Pruning to Produce Error Reduction (RIPPER) is a rule-based learner [6]. Both NB and RIPPER models were built using the default Weka parameters.

3.3 Performance Metrics

When dealing with imbalanced data, it is inappropriate to use a performance measure such as overall correct classification rate. Such a performance metric can be misleading when evaluating a model that heavily favors classifying examples as belonging to the majority class. Instead, we prefer metrics which take into account the model's ability to identify examples of each class. We have selected two such measures for use in this work: area under the ROC curve and area under the PRC curve.

A receiver operating characteristic curve (ROC) [15] plots the true positive rate on the y-axis versus the false positive rate on the x-axis. The resulting curve represents the trade off between correctly identifying positive class examples and false alarms across the complete range of possible decision thresholds. A single numeric value, the area under the ROC curve (AROC), is used to measure the performance of a classifier. Another curve, the precision-recall operating characteristic curve (P-ROC) [13], is also used to measure classifier performance. This curve plots the true positive rate (recall) versus precision across the complete range of decision thresholds. As with ROC, the area under the P-ROC curve (APRC) is used to evaluate the performance of a classifier. An in depth analysis of the relationship between ROC and P-ROC curves is presented in [7].

3.4 Experimental Design

All experiments were performed using 10-fold cross validation. Each dataset is broken into ten partitions, nine of which are used to train a model, while the remaining partition is used to test the model. This process is repeated ten times so that each partition acts as test data once. In addition, 10 independent runs of 10-fold cross validation were performed to alleviate any biasing that may occur during the random partitioning process. Using 10 runs of 10-fold cross validation and 15 base datasets (described in Section 3.1), a total of 1500 training datasets were used in our experiments.

Each of the 4 learners (described in Section 3.2) were used in conjunction with each of the 10 boosting algorithms (described in Section 2). In addition, each learner was used to build models without boosting, providing a baseline for comparison. Two versions of each boosting algorithm were implemented (resampling and reweighting). Therefore, a total of $4 \times (10 \times 2 + 1) = 84$ models were built for each training dataset. The total number of models evaluated for this study is $84 \times 1500 = 126{,}000$.

For those boosting algorithms that require a cost ratio to be specified, the cost ratio was determined on a per-dataset basis. The cost ratio was selected to create an effective class distribution of 35:65, with 35% of the examples belonging to the minority class. Specifically, the cost ratio was selected such that the sum of the costs of all minority class examples was equal to 35% of the sum of the costs of all examples in the training data. This approach is based on Elkan's [8] theorem for adjusting the class distribution to achieve a desired cost ratio. SMOTEBoost was also performed to achieve a class balance of 35:65. This ratio of minority to majority class examples was chosen based on a previous study showing this to be a favorable class distribution [12]. Experiments were also performed using ratios of 50:50 as suggested by [21] and 65:35, producing results similar to those reported in this study.

Dataset	C45D	C45N	NB	RIPPER
SP3	*.503*	.738	**.809**	.508
Mammography	.827	.901	**.920**	*.785*
SolarFlareF	*.514*	.875	**.898**	.538
Car3	*.500*	**.991**	.975	.721
CCCS12	*.783*	.853	**.960**	.825
SP1	.590	.751	**.793**	*.543*
PC1	.672	**.840**	.698	*.583*
Glass3	.748	**.801**	.720	*.525*
CM1	.561	.685	**.772**	*.516*
Pendigits5	.971	**.991**	*.865*	.977
SatImage4	.751	.917	**.920**	*.748*
Ecoli4	*.773*	.870	**.935**	.777
Segment5	.941	**.981**	*.833*	.923
Contra2	.685	.688	**.721**	*.601*
Vehicle1	.756	**.818**	.698	*.663*
Average	.705	**.847**	.834	*.682*

Table 2. Performance (A-ROC) of learners without boosting

4 Empirical Results

4.1 Learner Performance

We begin by examining the relative "strengths" of the learners used in this study, with respect to learning from imbalanced data. The performance of all four learners on each dataset is provided as measured using the area under both the ROC and PRC curves (A-ROC and A-PRC, respectively).

Table 2 shows the performance of the four learners (measured using A-ROC) for each of the 15 datasets used in our experiments. Table 3 shows the same information using A-PRC to measure performance. A **bold** value indicates that the learner resulted in the highest A-ROC (or A-PRC) for the given dataset while an *italicized* value indicates the worst performing learner. The last row labeled "Average" provides the average performance across all datasets for each learner. This table shows that of the four learners, NB and C4.5N are the two "stronger" learners, while RIPPER and C4.5D are relatively weak. This distinction will be important to the subsequent analysis presented in this work. While boosting often results in significant improvement using a weaker base learner, it typically (as will be shown in

Dataset	C45D	C45N	NB	RIPPER
SP3	*.016*	**.082**	.078	.023
Mammography	.532	**.625**	*.509*	.509
SolarFlareF	*.049*	.282	**.316**	.096
Car3	*.040*	**.789**	.610	.343
CCCS12	.512	.536	**.682**	*.502*
SP1	.158	.207	**.209**	*.109*
PC1	.282	**.382**	.265	*.187*
Glass3	.387	**.443**	.341	*.127*
CM1	.147	.220	**.370**	*.119*
Pendigits5	.943	**.973**	*.675*	.945
SatImage4	.454	.612	**.635**	*.445*
Ecoli4	.561	.648	**.736**	*.514*
Segment5	.835	**.914**	*.411*	.823
Contra2	.404	.401	**.458**	*.316*
Vehicle1	.525	**.608**	.495	*.436*
Average	.390	**.515**	.453	*.366*

Table 3. Performance (A-PRC) of learners without boosting

	A-ROC				A-PRC			
	Sampling		Weighting		Sampling		Weighting	
Technique	μ	σ	μ	σ	μ	σ	μ	σ
AdaBoost	.849	.143	.851	.137	.571	.305	.571	.306
AdaC1	$\underline{.831}$	.148	.792	.167	$\underline{.568}$	.307	.523	.315
AdaC2	.848	.141	.845	.143	.570	.311	.567	.308
AdaC3	.865	.125	.862	.130	.563	.298	.559	.295
AdaCost	$\underline{.857}$	.134	.815	.152	$\underline{.573}$	.308	.516	.309
CSB0	$\underline{.849}$	.135	.751	.162	$\underline{.563}$	.303	.391	.286
CSB1	.845	.143	.845	.141	.569	.312	.567	.307
CSB2	.856	.135	.855	.139	.579	.307	.582	.309
RareBoost	.855	.138	.853	.144	.588	.310	.585	.313
SMOTEBoost	.878	.115	.881	.117	.596	.303	.594	.303
AdaBoost	.813	.139	.813	.134	.395	.240	.392	.237
AdaC1	$\underline{.773}$	.139	.743	.148	$\underline{.359}$	.240	.315	.238
AdaC2	.830	.125	.835	.125	.420	.225	$\underline{.438}$	.231
AdaC3	.839	.111	.840	.112	.384	.200	.384	.203
AdaCost	.712	.232	.706	.222	$\underline{.273}$	.165	.247	.162
CSB0	.794	.147	.787	.156	.316	.182	.307	.178
CSB1	$\underline{.647}$	.217	.623	.225	.294	.226	.286	.217
CSB2	$\underline{.747}$	.176	.725	.186	.335	.233	.320	.229
RareBoost	.842	.129	.840	.131	.495	.250	.492	.251
SMOTEBoost	.844	.130	.847	.121	.483	.254	.481	.252

Table 4. Comparing the results using sampling and weighting with C4.5N (top) and Naive Bayes (bottom).

the following sections) does not result in a major improvement when stronger learners are used (although it can).

The claim that NB and C4.5N are "strong", while C4.5D and RIPPER are relatively "weak", is supported by the data presented in Tables 2 and 3. Regardless of performance metric, C4.5N and NB obtain the highest average performances across all 15 datasets, with C4.5N outperforming NB. Both C4.5N and NB outperform C4.5D and RIPPER by a large margin, while C4.5D outperforms RIPPER by a relatively small margin.

4.2 Boosting: Resampling vs. Reweighting

We now investigate the impact of boosting on learning from imbalanced data. The objective of this work, however, is not to identify which boosting algorithms are superior, but instead to compare each algorithm's performance using resampling and reweighting. In other words, when implementing each of these boosting algorithms, how should example weights be passed to the base learner?

4.2.1 Results: Strong Learners

Table 4 (top) shows the performance of C4.5N using each of the boosting algorithms as measured by A-ROC (left) and A-PRC (right). Both the mean μ and standard deviation σ are provided based on the A-ROC and A-PRC values obtained using all 15 datasets. A pairwise t-test was performed for each boosting technique - if resampling resulted in a significantly higher A-ROC or A-PRC (using a 95% confidence level), the value is underlined. For example, the mean A-ROC obtained by resampling AdaC1 is .831, which is significantly better than the mean A-ROC obtained by reweighting AdaC1. As shown in Table 4, three (out of ten) boosting algorithms, AdaC1, AdaCost and CSB0, perform significantly better when resampling is used, regardless of performance metric.

When using A-PRC to measure the performance of C4.5N (right side of Table 4), the same three boosting algorithms (AdaC1, AdaCost and CSB0) perform significantly better when implemented with resampling. CSB0 (with reweighting) was the only boosting algorithm to result in a performance decrease when compared to C4.5N built without boosting. C4.5N (without boosting) achieved an A-PRC of .515 (Table 3), while achieving an A-PRC of only .391 (a decrease of 24%) when CSB0 was used with reweighting. With resampling, however, the performance of CSB0 is greatly improved, resulting in an A-PRC of .563. Even a "strong" learner can be improved by boosting, especially when the boosting algorithm is implemented using resampling.

Using A-ROC to evaluate the models built using NB (bottom of Table 4), three boosting algorithms (AdaC1, CSB0 and CSB1) showed a significant improvement when resampling was used. As was the case with C4.5N, boosting by resampling performs as well as or significantly better than boosting by reweighting with all ten boosting algorithms. This is an interesting conclusion since boosting by reweighting is the "default" method of boosting (if the base learner can incorporate example weights in its decision making process, then reweighting is typically used).

The results are slightly different when A-PRC is used to evaluate performance. Two boosting algorithms (AdaC1 and AdaCost) perform significantly better when implemented with resampling. However, there is one boosting

	A-ROC				A-PRC			
	Sampling		Weighting		Sampling		Weighting	
Technique	μ	σ	μ	σ	μ	σ	μ	σ
AdaBoost	.849	.144	.852	.142	.574	.308	.578	.308
AdaC1	.815	.160	.772	.177	.554	.307	.503	.325
AdaC2	.843	.144	.842	.146	.564	.312	.570	.315
AdaC3	.803	.179	.769	.194	.510	.328	.476	.344
AdaCost	.861	.134	.830	.146	.578	.304	.530	.305
CSB0	.853	.130	.762	.157	.559	.297	.385	.273
CSB1	.845	.141	.839	.145	.563	.308	.552	.304
CSB2	.852	.139	.855	.139	.577	.310	.577	.309
RareBoost	.858	.139	.847	.153	.587	.309	.585	.316
SMOTEBoost	.862	.132	.876	.121	.590	.304	.589	.299
AdaBoost	.848	.141	.852	.140	.561	.299	.564	.305
AdaC1	.813	.157	.777	.177	.545	.304	.512	.322
AdaC2	.844	.146	.850	.137	.571	.312	.558	.310
AdaC3	.810	.184	.777	.203	.543	.339	.489	.351
AdaCost	.866	.131	.865	.141	.583	.307	.529	.293
CSB0	.859	.136	.814	.142	.549	.294	.391	.273
CSB1	.840	.144	.835	.157	.542	.307	.548	.320
CSB2	.851	.138	.855	.137	.570	.312	.568	.312
RareBoost	.853	.141	.840	.162	.579	.312	.565	.315
SMOTEBoost	.851	.137	.865	.127	.566	.299	.583	.302

Table 5. Comparing the results using sampling and weighting with C4.5D (top) and RIPPER (bottom).

algorithm, AdaC2, that performs significantly better when the reweighting version is used. This is the only combination of learner and boosting algorithm that violates the general trend we have seen thus far. Besides SMOTEBoost, AdaC2 (measured relative to A-PRC) is the only scenario in all of our experiments where reweighting performs significantly better than resampling.

4.2.2 Results: Weak Learners

Somewhat more interesting is the performance of these boosting algorithm implementations when applied to weak learners. It is, after all, weak learners that benefit most from boosting. In this section we compare the performance of boosting by resampling to boosting by reweighting using the two relatively "weak" learners, C4.5D and RIPPER.

Table 5 shows the performance of the various boosting algorithms using C4.5D (top) and RIPPER (bottom). For these two learning algorithms, resampling is clearly the preferred method for implementing boosting. For all boosting algorithms (except SMOTEBoost), resampling results in performance that is as good as or better than reweighting. SMOTEBoost with reweighting performs significantly better relative to the A-ROC than SMOTEBoost with resampling for both C4.5D and RIPPER. In 17 of the 40 scenarios in Table 5, however, resampling significantly outperforms reweighting, while conversely, reweighting significantly outperforms resampling only twice.

	A-ROC				A-PRC			
	Sampling		Weighting		Sampling		Weighting	
Technique	μ	σ	μ	σ	μ	σ	μ	σ
AdaBoost	.839	.142	.842	.139	.525	.299	.526	.301
AdaC1	.808	.153	.771	.169	.507	.303	.463	.314
AdaC2	.841	.139	.843	.138	.531	.299	.533	.298
AdaC3	.829	.155	.812	.169	.500	.304	.477	.310
AdaCost	.824	.176	.804	.179	.502	.307	.455	.299
CSB0	.839	.139	.779	.157	.497	.293	.369	.259
CSB1	.794	.185	.785	.194	.492	.312	.488	.313
CSB2	.827	.155	.822	.162	.515	.310	.512	.312
RareBoost	.852	.137	.845	.148	.562	.299	.557	.302
SMOTEBoost	.859	.130	.867	.122	.559	.294	.562	.294

Table 6. Comparing the results of using sampling and weighting for each Boosting Technique

	A-ROC				A-PRC			
	Sampling		Weighting		Sampling		Weighting	
Learner	μ	σ	μ	σ	μ	σ	μ	σ
C45D	.844	.146	.824	.158	.566	.309	.535	.316
C45N	.853	.136	.835	.148	.574	.306	.546	.310
NB	.784	.171	.776	.176	.375	.234	.366	.236
RIPPER	.843	.147	.833	.157	.561	.309	.531	.316

Table 7. Summary comparison of sampling and weighting by learner

4.2.3 Summarized Results

This section presents a summary of the results comparing resampling and reweighting for each of the ten boosting algorithms. The left side of Table 6 shows the performance of the ten boosting algorithms using A-ROC to measure learner performance based on the average performance across all four learners. For seven of the ten boosting algorithms, there is a significant difference between resampling and reweighting. In every case except SMOTEBoost, resampling performs as well as or better than reweighting. For SMOTEBoost, reweighting performs significantly better than resampling. The improvement in performance of CSB0 is particularly interesting, as it is the worst of the CSB algorithms using reweighting, but it is the best of the CSB algorithms using resampling.

The results are similar when A-PRC is used to measure performance, as shown in on the right side of Table 6. Here, we see that four (AdaC1, AdaC3, AdaCost and CSB0) of the ten boosting algorithms are significantly impacted by the selection of resampling or reweighting. Once again, resampling always results in performance that is at least as good as, and often significantly better than, reweighting.

Table 7 compares resampling and reweighting using both the A-ROC and A-PRC measures for each learner individually. In other words, the A-ROC and A-PRC values were averaged for all 10 boosting techniques over all 15 datasets for each learner. Resampling is significantly better than reweighting for each learner based on both performance measures.

5 Conclusion

This work presented a thorough empirical comparison of *boosting by reweighting* and *boosting by resampling* using a variety of boosting algorithms, 4 learners and 15 datasets with various levels of imbalance. Since this is an empirical study of the performance of these boosting techniques on imbalanced data, all boosting algorithms evaluated in this work are variations of AdaBoost designed to be used in a cost-sensitive or imbalanced data environment. Our results show that, in general, boosting by resampling (which can be applied to any base learner) performs as well as, or better than, boosting by reweighting. This conclusion is important because boosting by reweighting is typically used when the base learner is able to incorporate examples weights into the learning process. The primary exception to this general result is SMOTEBoost, which sometimes performed better when using boosting by reweighting. Since there were no guidelines set forth regarding how to implement SMOTEBoost with reweighting, we provide our own implementation, finding that it often outperforms SMOTEBoost by resampling.

Future work will include an investigation of these two boosting implementations using alternate boosting algorithms (which are not designed for imbalanced data), including but not limited to LogitBoost and InfoBoost. Experiments will also be performed using additional learners and datasets that present challenges other than imbalance, such as noise.

References

[1] D. W. Aha. *Lazy learning*. Kluwer Academic Publishers, Norwell, MA, USA, 1997.

[2] C. Blake and C. Merz. UCI repository of machine learning databases. *http://www.ics.uci.edu/ mlearn/ MLRepository.html*, 1998. Department of Information and Computer Sciences, University of California, Irvine.

[3] M. Botta. Resampling vs reweighting in boosting a relational weak learner. In *Proceedings of the 7th Congress of the Italian Association for Artificial Intelligence on Advances in Artificial Intelligence*, pages 70–80, London, UK, 2001. Springer-Verlag.

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

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

[6] W. W. Cohen. Fast effective rule induction. In *Proceedings 12th International Conference on Machine Learning*, pages 115–123. Morgan Kaufmann, 1995.

[7] J. Davis and M. Goadrich. The relationship between precision-recall and roc curves. In *ICML '06: Proceedings of the 23rd international conference on machine learning*, pages 233–240, Pittsburgh, Pennsylvania, 2006. ACM.

[8] C. Elkan. The foundations of cost-sensitive learning. In *Proceedings of the Seventeenth International Joint Conference on Articial Intelligence*, pages 973–978, 2001.

[9] W. Fan, S. J. Stolfo, J. Zhang, and P. K. Chan. AdaCost: misclassification cost-sensitive boosting. In *Proc. 16th International Conf. on Machine Learning*, pages 97–105. Morgan Kaufmann, San Francisco, CA, 1999.

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

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

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

[13] T. Landgrebe, P. Paclik, and R. Duin. Precision-recall operating characteristic (P-ROC) curves in imprecise environments. *18th International Conference on Pattern Recognition*, 4:123–127, 2006.

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

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

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

[17] C. Seiffert, T. M. Khoshgoftaar, J. Van Hulse, and A. Napolitano. Building useful models from imbalanced data with sampling and boosting. In *Proc. 21st Annual FLAIRS Conference*, Coconut Grove, FL, USA, May 2008, In Press.

[18] Y. Sun, M. S. Kamel, A. K. C. Wong, and Y. Wang. Cost-sensitive boosting for classification of imbalanced data. *Pattern Recognition*, 40(12):3358–3378, 2007.

[19] K. M. Ting. A comparative study of cost-sensitive boosting algorithms. In *Proc. 17th International Conf. on Machine Learning*, pages 983–990. Morgan Kaufmann, San Francisco, CA, 2000.

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

[21] G. M. Weiss and F. Provost. Learning when training data are costly: the effect of class distribution on tree induction. *Journal of Artificial Intelligence Research*, (19):315–354, 2003.

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