Exploratory Undersampling for Class-Imbalance Learning

Xu-Ying Liu, Jianxin Wu, and Zhi-Hua Zhou, *Senior Member, IEEE*

***Abstract*—Under-sampling is a popular method in dealing with class-imbalance problems, which uses only a subset of the majority class and thus is very efficient. The main deficiency is that many majority class examples are ignored. We propose two algorithms to overcome this deficiency. `EasyEnsemble` samples several subsets from the majority class, trains a learner using each of them, and combines the outputs of those learners. `BalanceCascade` trains the learners sequentially, where in each step the majority class examples which are correctly classified by the current trained learners are removed from further consideration. Experimental results show that both methods have higher AUC, F-measure and G-mean values than many existing class-imbalance learning methods. Moreover, they have approximately the same training time as that of under-sampling when the same number of weak classifiers are used, which is significantly faster than other methods.**

***Index Terms*—Data mining, machine learning, class-imbalance learning, under-sampling, ensemble learning.**

I. Introduction

IN many real-world problems, the data sets are typically imbalanced, i.e., some classes have much more instances than others. The level of imbalance (ratio of size of the majority class to minority class) can be as huge as 10^6 [41]. It is noteworthy that class-imbalance is emerging as an important issue in designing classifiers [11], [23], [37].

Imbalance has a serious impact on the performance of classifiers. Learning algorithms that do not consider class-imbalance tend to be overwhelmed by the majority class and ignore the minority class [10]. For example, in a problem with imbalance level 99, a learning algorithm that minimizes error rate could decide to classify all examples as the majority class in order to achieve a low error rate of 1%. However, all minority class examples will be wrongly classified in this case. In problems where the imbalance level is huge, class-imbalance must be carefully handled to build a good classifier.

Class-imbalance is also closely related to cost-sensitive learning, another important issue in machine learning. Misclassifying a minority class instance is usually more serious than misclassifying a majority class one. For example, approving a fraudulent credit card application is more costly than declining a credible one. Breiman et al. [7] pointed out that training set size, class priors, cost of errors in different classes, and placement of decision boundaries are all closely connected. In fact, many existing methods for dealing with class-imbalance rely on connections among these four components. Sampling methods handle class-imbalance by varying the minority and majority class sizes in the training set. Cost-sensitive learning deals with class-imbalance by incurring different costs for the two classes and is considered as an important class of methods to handle class-imbalance [37]. More details about class-imbalance learning methods are presented in Section II.

In this paper we examine only binary classification problems by ensembling classifiers built from multiple under-sampled training sets. Under-sampling is an efficient method for class-imbalance learning. This method uses a subset of the majority class to train the classifier. Since many majority class examples are ignored, the training set becomes more balanced and the training process becomes faster. However, the main drawback of under-sampling is that potentially useful information contained in these ignored examples is neglected. The intuition of our proposed methods is then to wisely explore these ignored data, while keeping the fast training speed of under-sampling.

We propose two ways to use these data. One straightforward way is to sample several subsets independently from $\mathcal{N}$ (the majority class), use these subsets to train classifiers separately, and combine the trained classifiers. Another method is to use trained classifiers to guide the sampling process for subsequent classifiers. After we have trained n classifiers, examples correctly classified by them will be removed from $\mathcal{N}$. Experiments on 16 UCI data sets [3] show that both methods have higher

Manuscript received October 26, 2007; revised June 24, 2008, and accepted October 5, 2008.

X.-Y. Liu and Z.-H. Zhou are with National Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210093, China email: {liuxy, zhouzh}@lamda.nju.edu.cn

J. Wu is with School of Interactive Computing, College of Computing, Georgia Institute of Technology, Atlanta, GA 30332 email: wujx@cc.gatech.edu

AUC, F-measure and G-mean values than many existing class-imbalance learning methods.

The rest of this paper is organized as follows. Section II reviews related methods. Section III presents `EasyEnsemble` and `BalanceCascade`. Section IV reports the experiments. Finally, Section V concludes this paper.

II. Related Work

As mentioned in the previous section, many existing class-imbalance learning methods manipulate the following four components: training set size, class prior, cost matrix, and placement of decision boundary. Here we pay special attention to two classes of methods that are most widely used: sampling and cost-sensitive learning. For other methods, we refer the readers to [37] for a more complete and detailed review.

Sampling is a class of methods that alters the size of training sets. Under-sampling and over-sampling change the training sets by sampling a smaller majority training set and repeating instances in the minority training set, respectively [15]. The level of imbalance is reduced in both methods, with the hope that a more balanced training set can give better results. Both sampling methods are easy to implement and have been shown to be helpful in imbalanced problems [37], [47]. Under-sampling requires shorter training time, at the cost of ignoring potentially useful data. Over-sampling increases the training set size, and thus requires longer training time. Furthermore, it tends to lead to overfitting since it repeats minority class examples [9], [15]. Besides the basic under-sampling and over-sampling methods, there are also methods that sample in more complex ways. SMOTE [9] added new synthetic minority class examples by randomly interpolating pairs of closest neighbors in the minority class. The one-sided selection procedures [25] tried to find a representative subset of majority class examples by only removing 'borderline' and 'noisy' majority examples. Some other methods combine different sampling strategies to achieve further improvement [1]. Also, researchers have studied the effect of varying the level of imbalance and how to find the best ratio when a C4.5 tree classifier was used [38].

Cost-sensitive learning [14], [16] is another important class of class-imbalance learning methods. Although many learning algorithms have been adapted to accommodate class-imbalance and cost-sensitive problems, variants of AdaBoost appear to be the most popular ones. Many cost-sensitive boosting algorithms have been proposed [31]. A common strategy of these variants was to intentionally increase the weights of examples with higher misclassification cost in the boosting process. In [30] the initial weights of high cost examples were increased. It was reported that, however, the weight differences between examples in different classes disappear quickly when the boosting process proceeds [33]. Thus, many algorithms raised high cost examples' weights in every iteration of the boosting process, for example, AsymBoost [33], AdaCost [17], CSB [31], DataBoost [21], AdaUBoost [24], just to name a few. Another way to adapt a boosting algorithm to cost-sensitive problems is to change the weights of the weak classifiers in forming the final ensemble classifier, such as BMPM [22] and LAC [41]. Unlike the heuristic methods mentioned above, Asymmetric Boosting [28] directly minimized a cost-sensitive loss function in the statistical interpretation of boosting.

SMOTEBoost [12] is designed for class-imbalance learning, which is very similar to AsymBoost. Both methods alter the distribution for the minority class and majority class in separate ways. The only difference is how these distributions are altered. AsymBoost directly updates instance weights for the majority class and minority class differently in each iteration, while SMOTEBoost alters distribution by first updating instance weights for majority class and minority class equally and then using SMOTE to get new minority class instances.

Chan and Stolfo [8] introduced an approach to explore majority class examples. They split the majority class into several non-overlapping subsets, with each subset having approximately the same number of examples as the minority class. One classifier was trained from each of these subsets and the minority class. The final classifier ensembled these classifiers using stacking [40]. However, when a data set is highly imbalanced, this approach requires a much longer training time than under-sampling. Also, since the minority class examples are used by every classifier, stacking these classifiers will have a high probability of suffering from overfitting when the number of minority class examples is limited.

III. EasyEnsemble & BalanceCascade

As was shown by [15], under-sampling is an efficient strategy to deal with class-imbalance. However, the drawback of under-sampling is that it throws away many potentially useful data. In this section, we propose two strategies to explore the majority class examples ignored by under-sampling: `EasyEnsemble` and `BalanceCascade`.

A. `EasyEnsemble`

Given the minority training set $\mathcal{P}$ and the majority training set $\mathcal{N}$, the under-sampling method randomly samples a subset $\mathcal{N}'$ from $\mathcal{N}$, where $|\mathcal{N}'| < |\mathcal{N}|$. Usually we choose $|\mathcal{N}'| = |\mathcal{P}|$, and therefore have $|\mathcal{N}'| \ll |\mathcal{N}|$ for highly imbalanced problems.

`EasyEnsemble` is probably the most straightforward way to further exploit the majority class examples ignored by under-sampling, i.e. examples in $\mathcal{N} \bigcap \overline{\mathcal{N}'}$. In this method, we independently sample several subsets $\mathcal{N}_1, \mathcal{N}_2, \ldots, \mathcal{N}_T$ from $\mathcal{N}$. For each subset $\mathcal{N}_i$ ($1 \leq i \leq T$), a classifier H_i is trained using $\mathcal{N}_i$ and all of $\mathcal{P}$. All generated classifiers are combined for the final decision. AdaBoost [29] is used to train the classifier H_i. The pseudo-code for `EasyEnsemble` is shown in Algorithm 1.

Algorithm 1 The `EasyEnsemble` algorithm.

1: $\{$Input: A set of minority class examples $\mathcal{P}$, a set of majority class examples $\mathcal{N}$, $|\mathcal{P}| < |\mathcal{N}|$, the number of subsets T to sample from $\mathcal{N}$, and s_i, the number of iterations to train an AdaBoost ensemble $H_i\}$
2: $i \Leftarrow 0$
3: **repeat**
4: $\quad i \Leftarrow i + 1$
5: $\quad$ Randomly sample a subset $\mathcal{N}_i$ from $\mathcal{N}$, $|\mathcal{N}_i| = |\mathcal{P}|$.
6: $\quad$ Learn H_i using $\mathcal{P}$ and $\mathcal{N}_i$. H_i is an AdaBoost ensemble with s_i weak classifiers $h_{i,j}$ and corresponding weights $\alpha_{i,j}$. The ensemble's threshold is θ_i, i.e.
$\quad H_i(x) = \mathrm{sgn}\left(\sum_{j=1}^{s_i} \alpha_{i,j} h_{i,j}(x) - \theta_i\right).$
7: **until** $i = T$
8: Output: An ensemble:
$\quad H(x) = \mathrm{sgn}\left(\sum_{i=1}^{T}\sum_{j=1}^{s_i} \alpha_{i,j} h_{i,j}(x) - \sum_{i=1}^{T} \theta_i\right).$

The idea behind `EasyEnsemble` is simple. Similar to the balanced Random Forests [13], `EasyEnsemble` generates T balanced sub-problems. The output of the ith sub-problem is AdaBoost classifier H_i, an ensemble with s_i weak classifiers $\{h_{i,j}\}$. An alternative view of $h_{i,j}$ is to treat it as a feature that is extracted by the ensemble learning method and can only take binary values [41]. H_i, in this viewpoint, is simply a linear classifier built on these features. Features extracted from different subsets $\mathcal{N}_i$ thus contain information of different aspects of the original majority training set $\mathcal{N}$. Finally, instead of counting votes from $\{H_i\}_{i=1\ldots T}$, we collect all the features $h_{i,j}$ ($i = 1, 2, \ldots, T$, $j = 1, 2, \ldots, s_i$), and form an ensemble classifier from them.

The output of `EasyEnsemble` is a single ensemble, but it looks like an 'ensemble of ensembles'. It is known that, Boosting mainly reduces bias while Bagging mainly reduces variance. Several works [19], [35], [36], [42] combine different ensemble strategies to achieve stronger generalization. MultiBoosting [35], [36] combines boosting with bagging/wagging [2] by using boosted ensembles as base learners. Stochastic Gradient Boosting [19] and Cocktail Ensemble [42] also combine different ensemble strategies. It is evident that `EasyEnsemble` has benefited from the combination of boosting and a bagging-like strategy with balanced class distribution.

Both `EasyEnsemble` and Balanced Random Forests try to use balanced bootstrap samples, however, the former uses the samples to generate boosted ensembles while the latter uses the samples to train decision trees randomly. Costing [43] also uses multiple samples of the original training set. Costing was initially proposed as a cost-sensitive learning method, while `EasyEnsemble` is proposed to deal with class-imbalance directly. Besides, the working style of `EasyEnsemble` is quite different from Costing. For example, the Costing method samples the examples with probability in proportion to their costs (Rejection Sampling). Since this is a probability-based sampling method, no positive example will definitely appear in all the samples (in fact, the probability of a positive example appearing in all the samples is small). While in `EasyEnsemble`, all the positive examples will definitely appear in all the samples. When the size of minority class is very small, it is important to utilize every minority class example.

B. `BalanceCascade`

`EasyEnsemble` is an unsupervised strategy to explore $\mathcal{N}$ since it uses independent random sampling with replacement. Our second algorithm, `BalanceCascade`, explores $\mathcal{N}$ in a supervised manner. The idea is as follows. After H_1 is trained, if an example $x_1 \in \mathcal{N}$ is correctly classified to be in the majority class by H_1, it is reasonable to conjecture that x_1 is somewhat redundant in $\mathcal{N}$, given that we already have H_1. Thus, we can remove some correctly classified majority class examples from $\mathcal{N}$. As in `EasyEnsemble`, we use AdaBoost in this method. The pseudo-code of `BalanceCascade` is described in Algorithm 2.

This method is called `BalanceCascade` since it is somewhat similar to the cascade classifier in [34]. The majority training set $\mathcal{N}$ is shrunk after every H_i is trained, and every node H_i is dealing with a balanced sub-problem ($|\mathcal{N}_i| = |\mathcal{P}|$). However, the final classifier is different. A cascade classifier is the conjunction of all $\{H_i\}_{i=1\ldots T}$, i.e. $H(x)$ predicts positive if and only if all $H_i(x)$ ($i = 1, 2, \ldots, T$) predict positive. Viola and Jones

Algorithm 2 The `BalanceCascade` algorithm.

1: {Input: A set of minority class examples $\mathcal{P}$, a set of majority class examples $\mathcal{N}$, $|\mathcal{P}| < |\mathcal{N}|$, the number of subsets T to sample from $\mathcal{N}$, and s_i, the number of iterations to train an AdaBoost ensemble H_i}
2: $i \Leftarrow 0$, $f \Leftarrow \sqrt[T-1]{\frac{|\mathcal{P}|}{|\mathcal{N}|}}$, f is the false positive rate (the error rate of misclassifying a majority class example to the minority class) that H_i should achieve.
3: **repeat**
4: $\quad i \Leftarrow i + 1$
5: $\quad$ Randomly sample a subset $\mathcal{N}_i$ from $\mathcal{N}$, $|\mathcal{N}_i| = |\mathcal{P}|$.
6: $\quad$ Learn H_i using $\mathcal{P}$ and $\mathcal{N}_i$. H_i is an AdaBoost ensemble with s_i weak classifiers $h_{i,j}$ and corresponding weights $\alpha_{i,j}$. The ensemble's threshold is θ_i i.e. $H_i(x) = \mathrm{sgn}\left(\sum_{j=1}^{s_i} \alpha_{i,j} h_{i,j}(x) - \theta_i\right)$.
7: $\quad$ Adjust θ_i such that H_i's false positive rate is f.
8: $\quad$ Remove from $\mathcal{N}$ all examples that are correctly classified by H_i.
9: **until** $i = T$
10: Output: A single ensemble: $H(x) = \mathrm{sgn}\left(\sum_{i=1}^{T} \sum_{j=1}^{s_i} \alpha_{i,j} h_{i,j}(x) - \sum_{i=1}^{T} \theta_i\right)$.

[34] used the cascade classifier mainly to achieve fast testing speed. While in `BalanceCascade`, sequential dependency between classifiers is mainly exploited for reducing the redundant information in the majority class. This sampling strategy leads to a restricted sample space for the following under-sampling process, to explore as much useful information as possible.

`BalanceCascade` is similar to `EasyEnsemble` in their structures. The main difference between them is the line III-B and III-B of Algorithm 2. Line III-B removes the true majority class examples from $\mathcal{N}$, and line III-B specifies how many majority class examples can be removed. At the beginning of the T-th iteration, $\mathcal{N}$ has been shrunk $T-1$ times, and therefore its current size is $|\mathcal{N}| \cdot f^{T-1} = |\mathcal{P}|$. Thus, after H_T is trained and $\mathcal{N}$ is shrunk again, the size of $\mathcal{N}$ is smaller than $|\mathcal{P}|$. We can stop the training process at this time.

There are other ways to combine weak classifiers in `EasyEnsemble` and `BalanceCascade`. A popular one is stacking [40]. It takes the outputs of other classifiers as input to train a generalizer. However, Ting [32] stated that, the use of class probabilities is crucial for the successful application of stacked generalization in classification tasks. Furthermore, since minority class examples are used to train each weak classifier, stacking these classifiers is likely to suffer from overfitting when the number of minority class examples is limited. To verify this, stacking is compared with the ensemble strategy used in the proposed methods in section IV-E.

Chan and Stolfo's method [8] (abbreviated as `Chan`) is closely related to `EasyEnsemble` and `BalanceCascade`. It splits the majority class into several non-overlapping subsets, with each subset having similar size to the the minority class. Classifiers trained from each majority class subset and the minority class are combined by stacking. The differences between `Chan` and the proposed methods are obvious: (1) `Chan` uses all majority class examples, while `EasyEnsemble` and `BalanceCascade` use only part of them. When a data set is highly imbalanced, `Chan` requires a much longer training time than the proposed methods. However, the experimental results reveal that it is not necessary to use all majority class examples to achieve good performances. (2) `Chan` uses stacking to combine classifiers trained from each subset. As stated above, since the minority class is used repeatedly, stacking is likely to suffer from overfitting when the number of minority class examples is limited.

Both `EasyEnsemble` and `BalanceCascade` are very efficient. Their training time is roughly the same as that of under-sampling when the same number of weak classifiers are used. Detailed analysis of training time and empirical running time are presented in section IV-C.

IV. Experiments

A. Evaluation Criteria

It is now well-known that error rate is not an appropriate evaluation criterion when there are class-imbalance or unequal costs. In this paper, we use F-measure, G-mean, and AUC (Area Under the ROC Curve) [4] as performance evaluation measures. F-measure and G-mean are functions of the confusion matrix as shown in Table I. F-measure and G-mean are then defined as follows. Here, we take minority class as positive class.

$$\begin{aligned}
False\ Positive\ Rate\ (fpr) &= \tfrac{FP}{FP+TN} \\
True\ Positive\ Rate\ (Acc_+) &= \tfrac{TP}{TP+FN} \\
True\ Negative\ Rate\ (Acc_-) &= \tfrac{TN}{TN+FP} \\
G{-}mean &= \sqrt{Acc_+ \times Acc_-} \\
Precision &= \tfrac{TP}{TP+FP} \\
Recall &= \tfrac{TP}{TP+FN} = Acc_+ \\
F{-}measure &= \tfrac{2 \times Precision \times Recall}{Precision + Recall}
\end{aligned} \quad (1)$$

AUC has proved to be a reliable performance measure for imbalanced and cost-sensitive problems [18]. Given a binary classification problem, an ROC curve depicts the performance of a method using the (*fpr*,*tpr*) pairs, as illustrated in Figure 1. *fpr* is the false positive rate of the classifier, and *tpr* is the true positive rate (Acc_+). AUC is the area below the curve (shaded region in Fig. 1). It integrates performance of the classification method over

TABLE I
CONFUSION MATRIX.

	Predicted Positive Class	Predicted Negative Class
Actual Positive Class	TP (True Positives)	FN (False Negatives)
Actual Negative Class	FP (False Positives)	TN (True Negatives)

all possible values of *fpr* and is proved to be a reliable performance measure for imbalanced and cost-sensitive problems [18].

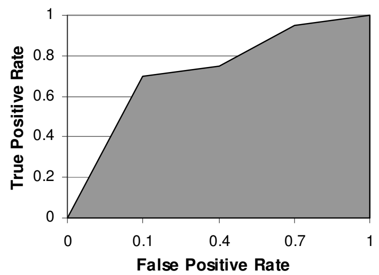

Fig. 1. Example of an ROC curve.

In our experiments, for ensemble classifiers in the form $H(x) = \mathrm{sgn}(\sum_{i=1}^{T} \alpha_i h_i(x) - \theta)$, we alter the value of θ from $-\infty$ to ∞. In this way we get a full range of (*fpr*,*tpr*) pairs and build an ROC curve from these data. We then use the Algorithm 3 in [18] to calculate the AUC score. Details of AUC can be found in [18].

B. Experimental Settings

We tested the proposed methods on 16 UCI data sets [3]. Information about these data sets is summarized in Table II.

TABLE II
BASIC INFORMATION OF DATA SETS. ***Size*** IS NUMBER OF EXAMPLES. ***Target*** IS USED AS MINORITY CLASS, AND ALL OTHERS ARE USED AS MAJORITY CLASS. IN ***Attribute***, ***B***: BINARY, ***N***: NOMINAL, ***C***: CONTINUOUS. ***#min/#maj*** IS THE SIZE OF MINORITY AND MAJORITY CLASS, AND ***Ratio*** IS THE SIZE OF MAJORITY CLASS DIVIDED BY THAT OF MINORITY CLASS.

Dataset	Size	Attribute	Target	#min/#maj	Ratio
abalone	4177	1N,7C	Ring=7	391/3786	9.7
balance	625	4C	Balance	49/576	11.8
car	1728	6N	acc	384/1344	3.5
cmc	1473	3B,4N,2C	class 2	333/1140	3.4
haberman	306	1N,2C	class 2	81/225	2.8
housing	506	1B,12C	[20, 23]	106/400	3.8
ionosphere	351	33C	bad	126/225	1.8
letter	20000	16C	A	789/19211	24.3
mf-morph	2000	6C	class 10	200/1800	9.0
mf-zernike	2000	47C	class 10	200/1800	9.0
phoneme	5404	5C	class 1	1586/3818	2.4
pima	768	8C	class 1	268/500	1.9
satimage	6435	36C	class 4	626/5809	9.3
vehicle	846	18C	opel	212/634	3.0
wdbc	569	30C	malignant	212/357	1.7
wpbc	198	33C	recur	47/151	3.2

For every data set, we perform a 10-fold stratified cross validation. Within each fold, the classification method is repeated 10 times considering that the sampling of subsets introduces randomness. The AUC, F-measure and G-mean of this cross validation process are averaged from these 10 runs. The whole cross validation process is repeated for 5 times, and the final values from this method are the averages of these 5 cross validation runs.

We compared the performance of 15 methods, including:

- CART. Classification and regression trees [7]. It uses the entire data set ($\mathcal{P}$ and $\mathcal{N}$) to train a single classifier.
- Bagging (abbreviated as `Bagg`): Bagging [5] uses the entire data set ($\mathcal{P}$ and $\mathcal{N}$). CART is used to train weak classifiers. The number of iterations is 40.
- AdaBoost (abbreviated as `Ada`). AdaBoost uses the entire data set ($\mathcal{P}$ and $\mathcal{N}$). CART is used to train weak classifiers. The number of iterations is 40.
- AsymBoost (abbreviated as `Asym`). AsymBoost is a typical cost-sensitive variant of AdaBoost[1]. Let $r = |\mathcal{N}|/|\mathcal{P}|$ be the imbalance level. At each iteration, the weight of every positive example is multiplied by $\sqrt[T]{r}$, where T is the number of iterations [33]. AsymBoost uses the entire data set ($\mathcal{P}$ and $\mathcal{N}$). CART is used to train weak classifiers. The number of iterations is 40.
- SMOTEBoost (abbreviated as `SMB`). SMOTE adds synthetic minority class examples [9]. For data sets having nominal attributes, we use SMOTE-NC. Details for implementing SMOTE and SMOTE-NC can be found in [9]. SMOTEBoost uses SMOTE to get new minority class examples in each iteration. CART is used to train weak classifiers. The number of iterations is 40. The k nearest neighbor parameter of SMOTE is 5. The amount of new data generated using SMOTE in each iteration is $|\mathcal{P}|$.
- Under-sampling + AdaBoost (abbreviated as `Under`). A subset $\mathcal{N}'$ is sampled (without replacement) from $\mathcal{N}$, $|\mathcal{N}'| = |\mathcal{P}|$. Then, AdaBoost is used to train a classifier using $\mathcal{P}$ and $\mathcal{N}'$, since the problem is balanced after under-sampling.

[1]It is also equivalent to the CSB2 algorithm in [31].

CART is used to train weak classifiers. The number of iterations is 40.

- Over-sampling + AdaBoost (abbreviated as `Over`). A new minority training set is sampled (with replacement) from the original minority class, $|\mathcal{P}'| = |\mathcal{N}|$. Then, AdaBoost is used to train a classifier using $\mathcal{P}'$ and $\mathcal{N}$. CART is used to train weak classifiers. The number of iterations is 40.
- SMOTE + AdaBoost (abbreviated as `SMOTE`). In our experiments, we first generate $\mathcal{P}'$ using SMOTE, a set of synthetic minority class examples with $|\mathcal{P}'| = |\mathcal{P}|$. We sample a new majority training set $\mathcal{N}'$ with $|\mathcal{N}'| = 2|\mathcal{P}|$ when $|\mathcal{N}| > 2|\mathcal{P}|$, and let $\mathcal{N}' = \mathcal{N}$ otherwise. Then we use AdaBoost to train a classifier with $\mathcal{P}$, $\mathcal{P}'$, and $\mathcal{N}'$. CART is used to train weak classifiers. The number of iterations is 40. The settings of SMOTE are the same as SMOTEBoost ($k = 5$).
- Chan & Stolfo's method + AdaBoost. (abbreviated as `Chan`). It splits $\mathcal{N}$ into $\lfloor|\mathcal{N}|/|\mathcal{P}|\rfloor$ non-overlapping subsets. An AdaBoost classifier was trained from each of these subsets and $\mathcal{P}$. Fisher Discriminant Analysis [20] is used as the stacking method. CART is used to train weak classifiers. AdaBoost classifiers are trained for $\lceil 40|P|/|N|\rceil$ iterations when $\lfloor|N|/|P|\rfloor < 40$, otherwise, only one iteration is allowed.
- `BalanceCascade` (abbreviated as `Cascade`). CART is used to train weak classifiers. Number of subsets $T = 4$, number of rounds in each AdaBoost ensemble $s_i = 10$.
- `EasyEnsemble` (abbreviated as `Easy`). CART is used to train weak classifiers. Number of subsets $T = 4$, number of rounds in each AdaBoost ensemble $s_i = 10$.
- Random Forests (abbreviated as `RF`). Random Forests [6] uses bootstrap samples of training data to generate random trees and then form an ensemble. Here, we use RandomForest in WEKA [39], in which a random tree is a variant of REPTree, using random feature selection in the tree induction process, and not pruned. `RF` uses the entire data set ($\mathcal{P}$ and $\mathcal{N}$). The number of iterations is 40.
- Under-sampling + Random Forests (abbreviated as `Under-RF`). A subset $\mathcal{N}'$ is sampled (without replacement) from $\mathcal{N}$, $|\mathcal{N}'| = |\mathcal{P}|$. Then, Random Forests is used to train a classifier using $\mathcal{P}$ and $\mathcal{N}'$, The number of iterations is 40.
- Over-sampling + Random Forests (abbreviated as `Over-RF`). A new minority training set is sampled (with replacement) from the original minority class, $|\mathcal{P}'| = |\mathcal{N}|$. Then, Random Forests is used to train a classifier using $\mathcal{P}'$ and $\mathcal{N}$. The number of iterations is 40.
- Balanced Random Forests (abbreviated as `BRF`). Balanced Random Forests is different from Random Forests in that it uses balanced bootstrap samples of training data. It is different from under-sampling + Random Forests, because the latter preprocesses the training data and then learns a Random Forests classifier. Here, we use RandomTree in WEKA to train weak classifiers, which is the same weak classifier learning method used by RandomForest in WEKA. The number of iterations is 40.

The settings of CART are the same. In CART, pruning is used, and impure nodes must have at least 10 examples to be split. CART and `Ada` are baseline methods. All other classifiers have 40 weak classifiers. In `Chan`, the amount of classifiers is also 40 since the imbalance levels of data sets in Table II are all lower than 40.

C. Analysis of Training Time

Random Forests series (`RF`, `Under-RF`, `Over-RF`, and `BRF`) use random decision trees, which train much faster than CART. Moreover, they are implemented in Java code, while the other methods are in Matlab code. Therefore, it is not fair to compare the running time of them directly. Here, we only analyze the training time of CART based methods.

Since all methods use the same weak learner and have the same amount of weak classifiers, the training time of these methods mainly depends on the number of training examples.

From the descriptions in section IV-B, `Under` uses the smallest number ($2|\mathcal{P}|$) of examples and is the fastest among all methods. The proposed methods (`Cascade` and `Easy`) and `Chan` use the same number of weak classifiers as `Under`, and use the same number of examples as `Under` to train every weak classifier[2]. These methods require additional time to sample or split subsets of $\mathcal{N}$. However, this time is negligible. Thus, the proposed methods and `Chan` have approximately the same training time as `Under`. Note that, the imbalance level of data sets used in the experiment happens to be lower than 40, so the number of weak classifiers in `Chan` can be the same with `Cascade` and `Easy`. However, when the data set is highly imbalanced (say the imbalance level is 1000), `Chan` will require extremely more training time than the proposed methods. Furthermore, `Easy` has a potential computational advantage since each under-sampling process can be executed in parallel.

[2]Although different subsets of $\mathcal{N}$ are used in the training process, the number of active training examples is always $2|\mathcal{P}|$ at all times.

TABLE III
RUNNING TIMES (IN SECONDS). THE ROW `AVG.` SHOWS THE AVERAGE RUNNING TIME OF EACH METHOD.

	CART	Bagg	Ada	Asym	SMB	Under	Over	SMOTE	Chan	Easy	Casc
abalone	0.21	8.39	18.06	18.04	35.51	3.47	39.11	7.78	4.83	3.72	6.22
balance	0.03	0.96	2.16	2.20	2.72	0.53	3.41	0.99	0.97	0.65	0.85
car	0.09	2.50	6.42	5.71	9.70	3.18	9.10	5.14	4.20	3.10	4.79
cmc	0.25	6.64	8.64	9.01	11.54	4.42	11.44	7.85	6.06	4.51	7.33
haberman	0.03	0.71	1.35	1.15	1.32	0.84	1.26	1.11	1.26	0.80	0.86
ionosphere	0.09	2.11	2.30	2.19	2.77	1.75	2.30	2.87	2.46	1.70	2.83
letter	0.41	19.70	153.47	138.11	1120.99	3.92	549.73	9.72	5.19	3.87	5.62
phoneme	0.34	9.72	23.20	22.87	150.09	12.03	38.78	30.18	16.67	11.64	20.12
pima	0.07	2.51	3.42	3.58	4.91	2.51	3.97	4.22	4.24	2.37	2.38
sat	0.78	27.74	54.83	53.29	102.84	9.62	116.83	21.24	11.66	10.08	13.96
wdbc	0.06	2.05	2.53	2.42	3.44	1.70	2.62	3.03	2.47	1.93	2.63
wpbc	0.06	1.97	2.04	1.85	2.26	1.00	2.29	2.01	1.17	1.27	1.50
vehicle	0.13	4.54	5.82	5.67	7.63	2.90	6.58	5.90	4.28	3.17	3.69
housing	0.08	2.17	2.66	2.92	3.84	1.32	3.35	2.42	1.89	1.15	0.77
mf-morph	0.06	2.06	5.69	5.78	11.62	1.08	11.22	2.34	1.60	1.17	2.57
mf-zernike	0.50	17.47	24.74	23.28	35.65	5.01	37.47	10.26	5.39	4.91	11.81
avg.	0.20	6.95	19.83	18.63	94.18	3.45	52.47	7.31	4.65	3.50	5.50

Both `Ada` and `Asym` use $|\mathcal{P}| + |\mathcal{N}|$ examples. Since $|\mathcal{N}| > |\mathcal{P}|$, these methods are slower than `Under`. When the imbalance level is high, these methods have much longer training time than that of `Under` and the proposed methods.

In our experiments, `SMOTE` uses either $4|\mathcal{P}|$ or $2|\mathcal{P}| + |\mathcal{N}|$ examples. `SMB` uses $2|\mathcal{P}| + |\mathcal{N}|$ examples. And both of them require to compute the distance between minority class examples. Thus they are much slower than `Under` and the proposed methods.

`Over` uses $2|\mathcal{N}|$ examples, which has the largest training set. `SMB` and `Over` are the most expensive ones. For data sets with a large number of examples, e.g. *letter*, the time to train a over-sampled or SMOTEBoost classifier is too long to be practical.

CART uses $|\mathcal{P}| + |\mathcal{N}|$ examples. CART trains only one classifier, so it indicates the time baseline.

Running times of these methods are recorded in Table III, on a computer with a 3.0GHz Intel Xeon CPU. It shows that `Chan`, `Easy` and `Cascade` are as efficient as `Under`. The most expensive ones are `SMB` and `Over`, followed by `Ada` and `Asym`, and then by `SMOTE`.

D. Results and Analyses

The average AUC of the compared methods are summarized in Table IV and Table V. On *car, ionosphere, letter, phoneme, sat* and *wdbc*, `Ada` achieves very high AUC values, which are all greater than 0.95. Applying class-imbalance learning methods on these data sets is not necessarily beneficial. On the other 10 data sets, `Ada`'s AUC values are not high and these data sets seem suffer from class-imbalance problem. Therefore, we divide the 16 data sets into two groups. The first group contains 6 'easy' tasks, on which the AUC values of `Ada` are greater than 0.95. The second group contains 10 'hard' tasks, on which the AUC values of `Ada` are lower than 0.95. The AUC results are shown separately in Table IV and V[3]. The results of t-test (significance level 0.05) of AUC are also shown separately in the upper and lower triangles in Table VI. The average F-measure of the compared methods are summarized in Table VII and VIII, and the t-test result is shown in Table IX. The average G-mean of the compared methods are summarized in Table X and XI, and the t-test result is shown in Table XII.

The results show that on 'easy' tasks, all class-imbalance learning methods have lower AUC and F-measure than `Ada`, except that `Asym` has similar AUC and F-measure to it. While on 'hard' tasks, class-imbalance learning methods generally have higher AUC and F-measure than `Ada`, including `SMOTE`, `Chan`, `Cascade` and `Easy`. We argue that for tasks on which ordinary methods can achieve high AUC (e.g. ≥ 0.95), class-imbalance learning is generally not helpful with AUC and F-measure. However, `Easy` and `Cascade` can be used to reduce the training time, while their average AUC are close to that of `Ada` and `Asym`.

We are more interested in the results on 'hard' tasks, where class-imbalance learning really helps. Compared with the results on 'easy' tasks, they reveal more properties of class-imbalance learning and the proposed methods.

`Under` is not performing well with AUC and F-measure. Its AUC and F-measure are lower than `Ada` and `Asym` on all 'easy' tasks, and lower than many other class-imbalance learning methods on 'hard' tasks. Our conjecture is that this is due to the information

[3]Note that the performance of `Over` and `SMB` on the data sets in the former group has not been obtained due to its large training time costs. CART gives discrete outputs, so its AUC is not available.

TABLE IV

AUC OF THE COMPARED METHODS (PART 1). THIS TABLE SHOWS RESULTS FOR DATA SETS ON WHICH ADABOOST'S AUC IS HIGHER THAN 0.95. FOR EACH METHOD AND EACH DATA SET, THE AVERAGE AUC IS FOLLOWED BY A STANDARD DEVIATION. THE COLUMN AVG. SHOWS THE AVERAGE AUC OF EACH METHOD.

AUC	*car*	*ionosphere*	*letter*	*phoneme*	*sat*	*wdbc*	***avg.***
Bagg	$.995 \pm .000$	$.962 \pm .004$	$.997 \pm .001$	$.955 \pm .001$	$.946 \pm .001$	$.987 \pm .001$	$.974 \pm .020$
Ada	$.998 \pm .000$	$.978 \pm .003$	$1.000 \pm .000$	$.965 \pm .000$	$.953 \pm .001$	$.994 \pm .001$	$.981 \pm .018$
Asym	$.998 \pm .000$	$.979 \pm .002$	$1.000 \pm .000$	$.965 \pm .001$	$.953 \pm .001$	$.994 \pm .000$	$.982 \pm .018$
Under	$.989 \pm .001$	$.973 \pm .002$	$1.000 \pm .000$	$.953 \pm .001$	$.941 \pm .001$	$.993 \pm .001$	$.975 \pm .021$
SMOTE	$.995 \pm .000$	$.978 \pm .002$	$1.000 \pm .000$	$.964 \pm .000$	$.946 \pm .001$	$.994 \pm .001$	$.979 \pm .019$
Chan	$.996 \pm .000$	$.979 \pm .002$	$1.000 \pm .000$	$.960 \pm .000$	$.955 \pm .000$	$.993 \pm .000$	$.980 \pm .018$
Cascade	$.996 \pm .000$	$.976 \pm .002$	$1.000 \pm .000$	$.962 \pm .000$	$.949 \pm .001$	$.994 \pm .000$	$.979 \pm .019$
Easy	$.994 \pm .000$	$.974 \pm .002$	$1.000 \pm .000$	$.958 \pm .000$	$.947 \pm .000$	$.993 \pm .000$	$.978 \pm .020$
RF	$.784 \pm .003$	$.981 \pm .004$	$1.000 \pm .000$	$.965 \pm .001$	$.961 \pm .002$	$.991 \pm .000$	$.947 \pm .074$
BRF	$.749 \pm .004$	$.969 \pm .003$	$.999 \pm .000$	$.960 \pm .001$	$.952 \pm .001$	$.990 \pm .001$	$.937 \pm .085$
Under-RF	$.786 \pm .001$	$.976 \pm .002$	$1.000 \pm .000$	$.952 \pm .001$	$.953 \pm .000$	$.991 \pm .001$	$.943 \pm .072$
Over-RF	$.785 \pm .002$	$.981 \pm .001$	$1.000 \pm .000$	$.964 \pm .001$	$.962 \pm .001$	$.991 \pm .001$	$.947 \pm .074$

TABLE V

AUC OF THE COMPARED METHODS (PART 2). THIS TABLE SHOWS RESULTS FOR DATA SETS ON WHICH ADABOOST'S AUC IS LOWER THAN 0.95.

AUC	*abalone*	*balance*	*cmc*	*haberman*	*housing*	
Bagg	$.824 \pm .002$	$.439 \pm .018$	$.705 \pm .004$	$.669 \pm .014$	$.825 \pm .011$	
Ada	$.811 \pm .001$	$.616 \pm .009$	$.675 \pm .008$	$.641 \pm .015$	$.815 \pm .010$	
Asym	$.812 \pm .003$	$.619 \pm .012$	$.675 \pm .010$	$.639 \pm .015$	$.815 \pm .010$	
SMB	$.818 \pm .002$	$.599 \pm .010$	$.687 \pm .011$	$.646 \pm .006$	$.824 \pm .014$	
Under	$.830 \pm .002$	$.617 \pm .011$	$.671 \pm .007$	$.646 \pm .010$	$.805 \pm .007$	
Over	$.817 \pm .002$	$.540 \pm .010$	$.675 \pm .008$	$.637 \pm .017$	$.821 \pm .010$	
SMOTE	$.831 \pm .001$	$.617 \pm .015$	$.680 \pm .008$	$.647 \pm .017$	$.816 \pm .008$	
Chan	$.850 \pm .001$	$.652 \pm .011$	$.696 \pm .006$	$.638 \pm .008$	$.811 \pm .010$	
Cascade	$.828 \pm .002$	$.637 \pm .011$	$.686 \pm .007$	$.653 \pm .012$	$.808 \pm .008$	
Easy	$.847 \pm .002$	$.633 \pm .008$	$.704 \pm .008$	$.668 \pm .011$	$.825 \pm .008$	
RF	$.827 \pm .004$	$.435 \pm .029$	$.669 \pm .007$	$.645 \pm .021$	$.828 \pm .015$	
BRF	$.853 \pm .001$	$.558 \pm .013$	$.683 \pm .003$	$.677 \pm .013$	$.798 \pm .018$	
Under-RF	$.842 \pm .002$	$.593 \pm .014$	$.676 \pm .002$	$.643 \pm .009$	$.820 \pm .010$	
Over-RF	$.823 \pm .001$	$.458 \pm .014$	$.660 \pm .005$	$.641 \pm .014$	$.826 \pm .014$	
AUC	*mf-morph*	*mf-zernike*	*pima*	*vehicle*	*wpbc*	***avg.***
Bagg	$.887 \pm .004$	$.855 \pm .002$	$.821 \pm .003$	$.859 \pm .003$	$.688 \pm .009$	$.757 \pm .129$
Ada	$.888 \pm .002$	$.795 \pm .003$	$.788 \pm .006$	$.854 \pm .003$	$.716 \pm .009$	$.760 \pm .088$
Asym	$.888 \pm .001$	$.801 \pm .005$	$.788 \pm .005$	$.853 \pm .002$	$.721 \pm .012$	$.761 \pm .088$
SMB	$.897 \pm .002$	$.788 \pm .007$	$.790 \pm .003$	$.864 \pm .003$	$.720 \pm .013$	$.763 \pm .092$
Under	$.916 \pm .001$	$.881 \pm .003$	$.789 \pm .002$	$.846 \pm .003$	$.694 \pm .010$	$.769 \pm .100$
Over	$.889 \pm .002$	$.779 \pm .007$	$.791 \pm .004$	$.855 \pm .003$	$.711 \pm .010$	$.751 \pm .103$
SMOTE	$.912 \pm .001$	$.862 \pm .004$	$.792 \pm .003$	$.858 \pm .004$	$.709 \pm .004$	$.772 \pm .097$
Chan	$.912 \pm .002$	$.903 \pm .002$	$.786 \pm .007$	$.856 \pm .002$	$.706 \pm .009$	$.781 \pm .097$
Cascade	$.905 \pm .001$	$.891 \pm .001$	$.799 \pm .005$	$.856 \pm .002$	$.712 \pm .011$	$.778 \pm .093$
Easy	$.918 \pm .002$	$.904 \pm .001$	$.809 \pm .004$	$.859 \pm .004$	$.707 \pm .009$	$.787 \pm .096$
RF	$.880 \pm .007$	$.840 \pm .008$	$.821 \pm .004$	$.869 \pm .008$	$.677 \pm .030$	$.749 \pm .133$
BRF	$.901 \pm .002$	$.866 \pm .009$	$.809 \pm .003$	$.850 \pm .002$	$.646 \pm .014$	$.764 \pm .109$
Under-RF	$.919 \pm .003$	$.889 \pm .002$	$.818 \pm .004$	$.855 \pm .002$	$.661 \pm .008$	$.772 \pm .110$
Over-RF	$.881 \pm .004$	$.854 \pm .003$	$.819 \pm .004$	$.866 \pm .003$	$.670 \pm .010$	$.750 \pm .130$

TABLE VI

SUMMARY OF t-TEST OF AUC WITH SIGNIFICANCE LEVEL AT 0.05. THE UPPER TRIANGLE SHOWS THE RESULT OF 6 'EASY' TASKS AND THE LOWER TRIANGLE SHOWS THE RESULT OF 10 'HARD' TASKS. EACH TABULAR SHOWS THE AMOUNT OF WIN-TIE-LOSE OF A METHOD IN A ROW COMPARING WITH THE METHOD IN A COLUMN.

	Bagg	Ada	Asym	SMB	Under	Over	SMOTE	Chan	Cascade	Easy	RF	BRF	Under-RF	Over-RF
Bagg	–	0-0-6	0-0-6	NA	3-0-3	NA	0-2-4	0-0-6	0-0-6	1-1-4	1-0-5	1-0-5	2-0-4	1-0-5
Ada	2-1-7	–	0-5-1	NA	6-0-0	NA	4-2-0	4-0-2	5-1-0	5-1-0	3-2-1	5-1-0	5-1-0	3-1-2
Asym	2-1-7	1-9-0	–	NA	6-0-0	NA	4-2-0	4-1-1	5-1-0	5-1-0	3-2-1	6-0-0	6-0-0	3-1-2
SMB	4-1-5	5-3-2	5-3-2	–	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
Under	5-0-5	3-4-3	3-3-4	4-2-4	–	NA	0-0-6	1-1-4	0-0-6	0-2-4	2-0-4	4-0-2	3-1-2	2-0-4
Over	2-1-7	3-5-2	4-4-2	0-5-5	4-2-4	–	NA	NA	NA	NA	NA	NA	NA	NA
SMOTE	5-1-4	6-4-0	6-4-0	3-5-2	4-4-2	5-5-0	–	3-1-2	2-1-3	4-1-1	2-2-2	5-0-1	5-0-1	2-1-3
Chan	5-1-4	5-4-1	6-2-2	5-0-5	6-2-2	5-3-2	4-5-1	–	3-0-3	4-1-1	2-1-3	5-1-0	5-0-1	2-1-3
Cascade	5-1-4	7-2-1	8-1-1	5-3-2	7-2-1	7-2-1	4-3-3	2-3-5	–	6-0-0	2-0-4	5-0-1	4-1-1	2-0-4
Easy	5-4-1	9-1-0	8-2-0	7-2-1	9-1-0	8-2-0	8-2-0	6-2-2	8-2-0	–	2-0-4	4-0-2	4-0-2	2-0-4
RF	1-5-4	5-2-3	5-1-4	3-3-4	3-3-4	5-1-4	3-3-4	3-2-5	3-2-5	2-2-6	–	5-1-0	4-2-0	1-3-2
BRF	5-0-5	6-0-4	6-0-4	5-1-4	5-1-4	7-0-3	3-2-5	3-0-7	3-2-5	2-1-7	6-0-4	–	1-2-3	0-1-5
Under-RF	4-0-6	5-3-2	5-3-2	4-3-3	6-1-3	5-4-1	4-4-2	4-1-5	4-1-5	1-1-8	5-3-2	6-1-3	–	0-2-4
Over-RF	2-3-5	5-1-4	5-1-4	3-3-4	3-1-6	5-1-4	3-1-6	3-1-6	3-0-7	2-1-7	1-7-2	4-0-6	2-3-5	–

TABLE VII
F-MEASURE OF THE COMPARED METHODS ON ‘EASY’ TASKS (PART 1).

F Measure	*car*	*ionosphere*	*letter*	*phoneme*	*sat*	*wdbc*	***avg.***
CART	$.857 \pm .011$	$.831 \pm .024$	$.945 \pm .005$	$.773 \pm .007$	$.546 \pm .014$	$.895 \pm .004$	$.808 \pm .128$
Bagg	$.933 \pm .004$	$.883 \pm .005$	$.962 \pm .003$	$.834 \pm .002$	$.641 \pm .007$	$.938 \pm .004$	$.865 \pm .109$
Ada	$.967 \pm .002$	$.907 \pm .004$	$.988 \pm .002$	$.850 \pm .002$	$.664 \pm .006$	$.956 \pm .003$	$.889 \pm .110$
Asym	$.966 \pm .002$	$.910 \pm .004$	$.987 \pm .001$	$.852 \pm .002$	$.668 \pm .004$	$.956 \pm .003$	$.890 \pm .109$
Under	$.884 \pm .001$	$.900 \pm .004$	$.903 \pm .002$	$.819 \pm .001$	$.546 \pm .002$	$.952 \pm .002$	$.834 \pm .134$
SMOTE	$.930 \pm .004$	$.907 \pm .003$	$.954 \pm .002$	$.847 \pm .002$	$.610 \pm .003$	$.957 \pm .003$	$.867 \pm .121$
Chan	$.916 \pm .003$	$.910 \pm .006$	$.905 \pm .001$	$.837 \pm .002$	$.607 \pm .001$	$.954 \pm .002$	$.855 \pm .116$
Cascade	$.917 \pm .002$	$.905 \pm .003$	$.976 \pm .002$	$.839 \pm .002$	$.619 \pm .002$	$.957 \pm .002$	$.869 \pm .120$
Easy	$.880 \pm .002$	$.901 \pm .005$	$.910 \pm .002$	$.821 \pm .002$	$.554 \pm .001$	$.951 \pm .004$	$.836 \pm .132$
RF	$.307 \pm .013$	$.906 \pm .005$	$.979 \pm .003$	$.850 \pm .004$	$.666 \pm .008$	$.954 \pm .002$	$.777 \pm .234$
BRF	$.521 \pm .001$	$.887 \pm .004$	$.889 \pm .016$	$.821 \pm .003$	$.553 \pm .001$	$.945 \pm .005$	$.769 \pm .168$
Under-RF	$.513 \pm .001$	$.895 \pm .005$	$.895 \pm .003$	$.813 \pm .001$	$.557 \pm .001$	$.948 \pm .002$	$.770 \pm .171$
Over-RF	$.518 \pm .001$	$.904 \pm .004$	$.986 \pm .001$	$.851 \pm .002$	$.689 \pm .004$	$.955 \pm .003$	$.817 \pm .164$

TABLE VIII
F-MEASURE OF THE COMPARED METHODS ON ‘HARD’ TASKS (PART 2).

F Measure	*abalone*	*balance*	*cmc*	*haberman*	*housing*	
CART	$.232 \pm .018$	$.000 \pm .000$	$.356 \pm .009$	$.335 \pm .046$	$.420 \pm .031$	
Bagg	$.170 \pm .010$	$.000 \pm .000$	$.362 \pm .011$	$.334 \pm .030$	$.419 \pm .029$	
Ada	$.210 \pm .008$	$.000 \pm .000$	$.388 \pm .009$	$.348 \pm .022$	$.475 \pm .022$	
Asym	$.222 \pm .006$	$.000 \pm .001$	$.400 \pm .011$	$.360 \pm .020$	$.485 \pm .015$	
SMB	$.286 \pm .008$	$.001 \pm .001$	$.393 \pm .013$	$.377 \pm .024$	$.530 \pm .016$	
Under	$.367 \pm .001$	$.175 \pm .009$	$.429 \pm .007$	$.442 \pm .017$	$.529 \pm .006$	
Over	$.195 \pm .005$	$.000 \pm .000$	$.383 \pm .011$	$.338 \pm .024$	$.470 \pm .016$	
SMOTE	$.379 \pm .005$	$.149 \pm .011$	$.421 \pm .007$	$.405 \pm .016$	$.532 \pm .017$	
Chan	$.400 \pm .002$	$.156 \pm .005$	$.437 \pm .007$	$.380 \pm .018$	$.523 \pm .010$	
Cascade	$.384 \pm .002$	$.194 \pm .011$	$.436 \pm .009$	$.438 \pm .014$	$.529 \pm .008$	
Easy	$.382 \pm .003$	$.184 \pm .007$	$.454 \pm .008$	$.466 \pm .013$	$.543 \pm .007$	
RF	$.189 \pm .015$	$.000 \pm .000$	$.347 \pm .017$	$.321 \pm .027$	$.445 \pm .035$	
BRF	$.382 \pm .002$	$.167 \pm .006$	$.441 \pm .004$	$.468 \pm .015$	$.515 \pm .018$	
Under-RF	$.375 \pm .002$	$.168 \pm .007$	$.435 \pm .003$	$.445 \pm .011$	$.537 \pm .006$	
Over-RF	$.253 \pm .004$	$.000 \pm .000$	$.408 \pm .008$	$.348 \pm .015$	$.490 \pm .025$	
F Measure	*mf-morph*	*mf-zernike*	*pima*	*vehicle*	*wpbc*	**avg.**
CART	$.251 \pm .022$	$.216 \pm .015$	$.584 \pm .029$	$.523 \pm .019$	$.373 \pm .023$	$.329 \pm .158$
Bagg	$.263 \pm .016$	$.183 \pm .014$	$.644 \pm .007$	$.526 \pm .011$	$.410 \pm .019$	$.331 \pm .177$
Ada	$.321 \pm .014$	$.188 \pm .017$	$.611 \pm .007$	$.545 \pm .010$	$.432 \pm .014$	$.352 \pm .173$
Asym	$.344 \pm .015$	$.191 \pm .010$	$.613 \pm .011$	$.561 \pm .008$	$.444 \pm .015$	$.362 \pm .175$
SMB	$.351 \pm .013$	$.295 \pm .018$	$.641 \pm .006$	$.606 \pm .012$	$.452 \pm .011$	$.393 \pm .175$
Under	$.579 \pm .004$	$.538 \pm .004$	$.644 \pm .002$	$.623 \pm .005$	$.449 \pm .008$	$.477 \pm .132$
Over	$.319 \pm .012$	$.166 \pm .011$	$.609 \pm .009$	$.539 \pm .017$	$.427 \pm .010$	$.345 \pm .175$
SMOTE	$.560 \pm .005$	$.538 \pm .007$	$.627 \pm .004$	$.615 \pm .006$	$.459 \pm .009$	$.468 \pm .134$
Chan	$.635 \pm .001$	$.577 \pm .002$	$.618 \pm .006$	$.608 \pm .003$	$.448 \pm .018$	$.478 \pm .140$
Cascade	$.596 \pm .006$	$.549 \pm .004$	$.649 \pm .007$	$.623 \pm .012$	$.454 \pm .007$	$.485 \pm .128$
Easy	$.624 \pm .002$	$.564 \pm .002$	$.660 \pm .005$	$.638 \pm .007$	$.452 \pm .014$	$.497 \pm .136$
RF	$.261 \pm .023$	$.144 \pm .034$	$.641 \pm .013$	$.544 \pm .024$	$.393 \pm .027$	$.328 \pm .181$
BRF	$.627 \pm .003$	$.500 \pm .013$	$.663 \pm .005$	$.633 \pm .007$	$.401 \pm .006$	$.480 \pm .140$
Under-RF	$.602 \pm .004$	$.530 \pm .004$	$.668 \pm .006$	$.633 \pm .007$	$.419 \pm .008$	$.481 \pm .140$
Over-RF	$.349 \pm .014$	$.292 \pm .012$	$.656 \pm .005$	$.564 \pm .015$	$.397 \pm .019$	$.376 \pm .171$

TABLE IX
SUMMARY OF t-TEST OF F-MEASURE WITH SIGNIFICANCE LEVEL AT 0.05. THE UPPER TRIANGLE SHOWS THE RESULT OF 6 ‘EASY’ TASKS AND THE LOWER TRIANGLE SHOWS THE RESULT OF 10 ‘HARD’ TASKS. EACH TABULAR SHOWS THE AMOUNT OF WIN-TIE-LOSE OF A METHOD IN A ROW COMPARING WITH THE METHOD IN A COLUMN.

	Cart	Bagg	Ada	Asym	SMB	Under	Over	SMOTE	Chan	Cascade	Easy	RF	BRF	Under-RF	Over-RF
CART	–	0-0-6	0-0-6	0-0-6	NA	1-1-4	NA	0-0-6	1-0-5	0-0-6	1-1-4	1-0-5	2-1-3	2-1-3	1-0-5
Bagg	2-6-2	–	0-0-6	0-0-6	NA	4-0-2	NA	3-0-3	3-0-3	2-0-4	4-0-2	1-0-5	4-1-1	4-0-2	1-0-5
Ada	6-2-2	7-2-1	–	0-4-2	NA	6-0-0	NA	4-2-0	4-2-0	5-1-0	6-0-0	2-4-0	6-0-0	6-0-0	3-2-1
Asym	5-4-1	7-2-1	5-5-0	–	NA	6-0-0	NA	4-2-0	5-1-0	5-1-0	6-0-0	2-4-0	6-0-0	6-0-0	3-2-1
SMB	9-1-0	8-2-0	8-2-0	7-3-0	–	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
Under	10-0-0	9-1-0	10-0-0	9-1-0	7-3-0	–	NA	0-0-6	0-1-5	0-0-6	1-2-3	1-2-3	3-2-1	5-0-1	1-0-5
Over	4-4-2	5-3-2	0-6-4	0-2-8	0-1-9	0-0-10	–	NA	NA	NA	NA	NA	NA	NA	NA
SMOTE	10-0-0	9-0-1	10-0-0	9-1-0	7-2-1	1-3-6	10-0-0	–	5-1-0	2-2-2	6-0-0	2-2-2	6-0-0	6-0-0	1-2-3
Chan	10-0-0	9-0-1	9-1-0	9-1-0	5-4-1	4-1-5	10-0-0	4-2-4	–	1-2-3	4-1-1	1-2-3	6-0-0	6-0-0	2-1-3
Cascade	10-0-0	9-1-0	10-0-0	9-1-0	8-2-0	5-5-0	10-0-0	8-2-0	4-3-3	–	6-0-0	1-2-3	6-0-0	6-0-0	2-1-3
Easy	10-0-0	10-0-0	10-0-0	9-1-0	9-1-0	9-1-0	10-0-0	8-2-0	6-1-3	7-2-1	–	1-2-3	3-3-0	4-1-1	1-0-5
RF	1-7-2	3-6-1	1-2-7	1-2-7	0-2-8	0-1-9	1-5-4	1-0-9	1-0-9	0-1-9	0-0-10	–	5-0-1	5-0-1	0-3-3
BRF	10-0-0	9-1-0	9-0-1	9-0-1	8-0-2	6-1-3	9-0-1	6-1-3	4-2-4	4-1-5	0-5-5	9-1-0	–	2-2-2	1-0-5
Under-RF	10-0-0	9-1-0	9-0-1	9-0-1	8-1-1	6-2-2	9-1-0	6-2-2	5-1-4	5-1-4	1-0-9	10-0-0	3-3-4	–	0-0-6
Over-RF	7-3-0	7-3-0	7-2-1	5-3-2	2-3-5	1-0-9	7-2-1	1-0-9	1-0-9	1-0-9	0-0-10	8-2-0	0-1-9	0-1-9	–

TABLE X

G-MEAN OF THE COMPARED METHODS ON 'EASY' TASKS (PART 1). THE ROW AVG. SHOWS THE AVERAGE G-MEAN OF EACH METHOD.

G-mean	*car*	*ionosphere*	*letter*	*phoneme*	*sat*	*wdbc*	***avg.***
CART	$.910 \pm .011$	$.867 \pm .021$	$.968 \pm .003$	$.836 \pm .006$	$.716 \pm .009$	$.918 \pm .004$	$.869 \pm .080$
Bagg	$.964 \pm .002$	$.906 \pm .003$	$.972 \pm .002$	$.880 \pm .001$	$.729 \pm .005$	$.950 \pm .003$	$.900 \pm .083$
Ada	$.980 \pm .001$	$.920 \pm .003$	$.989 \pm .002$	$.890 \pm .001$	$.754 \pm .005$	$.963 \pm .003$	$.916 \pm .080$
Asym	$.981 \pm .001$	$.922 \pm .003$	$.988 \pm .001$	$.892 \pm .002$	$.761 \pm .003$	$.963 \pm .002$	$.918 \pm .078$
Under	$.956 \pm .001$	$.918 \pm .003$	$.994 \pm .000$	$.889 \pm .001$	$.871 \pm .002$	$.963 \pm .001$	$.932 \pm .043$
SMOTE	$.969 \pm .002$	$.922 \pm .002$	$.995 \pm .001$	$.899 \pm .001$	$.862 \pm .003$	$.964 \pm .003$	$.935 \pm .046$
Chan	$.970 \pm .001$	$.923 \pm .005$	$.992 \pm .001$	$.897 \pm .001$	$.881 \pm .001$	$.962 \pm .002$	$.937 \pm .040$
Cascade	$.969 \pm .001$	$.921 \pm .002$	$.996 \pm .001$	$.897 \pm .001$	$.867 \pm .002$	$.967 \pm .002$	$.936 \pm .045$
Easy	$.958 \pm .001$	$.919 \pm .003$	$.995 \pm .000$	$.892 \pm .001$	$.876 \pm .001$	$.962 \pm .003$	$.934 \pm .042$
RF	$.452 \pm .013$	$.918 \pm .005$	$.980 \pm .002$	$.892 \pm .003$	$.744 \pm .006$	$.962 \pm .003$	$.825 \pm .183$
BRF	$.693 \pm .001$	$.911 \pm .004$	$.989 \pm .002$	$.893 \pm .002$	$.881 \pm .001$	$.957 \pm .004$	$.887 \pm .095$
Under-RF	$.687 \pm .001$	$.916 \pm .003$	$.993 \pm .001$	$.887 \pm .001$	$.883 \pm .000$	$.960 \pm .002$	$.888 \pm .098$
Over-RF	$.690 \pm .001$	$.918 \pm .002$	$.987 \pm .001$	$.897 \pm .001$	$.782 \pm .003$	$.963 \pm .003$	$.873 \pm .104$

TABLE XI

G-MEAN OF THE COMPARED METHODS ON 'HARD' TASKS (PART 2).

G-mean	*abalone*	*balance*	*cmc*	*haberman*	*housing*	
CART	$.453 \pm .021$	$.000 \pm .000$	$.525 \pm .008$	$.483 \pm .045$	$.586 \pm .026$	
Bagg	$.337 \pm .011$	$.000 \pm .000$	$.509 \pm .010$	$.476 \pm .036$	$.553 \pm .032$	
Ada	$.396 \pm .008$	$.001 \pm .002$	$.561 \pm .007$	$.502 \pm .025$	$.615 \pm .017$	
Asym	$.412 \pm .007$	$.002 \pm .004$	$.577 \pm .010$	$.515 \pm .023$	$.627 \pm .011$	
SMB	$.511 \pm .010$	$.002 \pm .004$	$.560 \pm .011$	$.536 \pm .022$	$.686 \pm .013$	
Under	$.765 \pm .003$	$.560 \pm .020$	$.623 \pm .007$	$.592 \pm .018$	$.725 \pm .005$	
Over	$.372 \pm .005$	$.000 \pm .000$	$.555 \pm .009$	$.491 \pm .028$	$.607 \pm .010$	
SMOTE	$.742 \pm .006$	$.465 \pm .027$	$.605 \pm .006$	$.562 \pm .016$	$.710 \pm .014$	
Chan	$.778 \pm .001$	$.465 \pm .011$	$.622 \pm .006$	$.536 \pm .020$	$.698 \pm .007$	
Cascade	$.755 \pm .001$	$.595 \pm .021$	$.628 \pm .008$	$.591 \pm .013$	$.721 \pm .007$	
Easy	$.785 \pm .004$	$.577 \pm .015$	$.646 \pm .007$	$.615 \pm .012$	$.739 \pm .006$	
RF	$.363 \pm .016$	$.000 \pm .000$	$.516 \pm .015$	$.476 \pm .028$	$.580 \pm .031$	
BRF	$.790 \pm .003$	$.548 \pm .012$	$.634 \pm .004$	$.618 \pm .014$	$.718 \pm .018$	
Under-RF	$.778 \pm .002$	$.548 \pm .015$	$.627 \pm .003$	$.593 \pm .011$	$.735 \pm .005$	
Over-RF	$.457 \pm .004$	$.000 \pm .000$	$.587 \pm .006$	$.504 \pm .016$	$.638 \pm .019$	
G-mean	*mf-morph*	*mf-zernike*	*pima*	*vehicle*	*wpbc*	**avg.**
CART	$.473 \pm .022$	$.428 \pm .020$	$.673 \pm .024$	$.658 \pm .013$	$.513 \pm .032$	$.479 \pm .178$
Bagg	$.483 \pm .016$	$.378 \pm .021$	$.720 \pm .006$	$.642 \pm .008$	$.510 \pm .032$	$.461 \pm .187$
Ada	$.560 \pm .012$	$.386 \pm .020$	$.694 \pm .006$	$.664 \pm .008$	$.537 \pm .025$	$.492 \pm .189$
Asym	$.594 \pm .014$	$.392 \pm .013$	$.696 \pm .009$	$.679 \pm .007$	$.549 \pm .028$	$.504 \pm .193$
SMB	$.605 \pm .013$	$.524 \pm .019$	$.719 \pm .006$	$.728 \pm .009$	$.584 \pm .021$	$.545 \pm .196$
Under	$.873 \pm .003$	$.848 \pm .004$	$.719 \pm .001$	$.768 \pm .004$	$.617 \pm .008$	$.709 \pm .102$
Over	$.559 \pm .012$	$.358 \pm .015$	$.692 \pm .007$	$.657 \pm .013$	$.527 \pm .013$	$.482 \pm .191$
SMOTE	$.841 \pm .006$	$.813 \pm .007$	$.708 \pm .003$	$.743 \pm .005$	$.610 \pm .009$	$.680 \pm .111$
Chan	$.920 \pm .001$	$.854 \pm .002$	$.700 \pm .005$	$.738 \pm .004$	$.585 \pm .021$	$.690 \pm .134$
Cascade	$.874 \pm .006$	$.820 \pm .003$	$.725 \pm .005$	$.760 \pm .011$	$.623 \pm .007$	$.709 \pm .092$
Easy	$.914 \pm .001$	$.869 \pm .003$	$.734 \pm .004$	$.781 \pm .005$	$.623 \pm .014$	$.728 \pm .107$
RF	$.479 \pm .022$	$.326 \pm .049$	$.717 \pm .010$	$.659 \pm .018$	$.477 \pm .019$	$.459 \pm .190$
BRF	$.918 \pm .002$	$.831 \pm .007$	$.735 \pm .004$	$.780 \pm .007$	$.567 \pm .007$	$.714 \pm .114$
Under-RF	$.888 \pm .005$	$.844 \pm .002$	$.740 \pm .005$	$.779 \pm .006$	$.588 \pm .011$	$.712 \pm .111$
Over-RF	$.597 \pm .013$	$.519 \pm .016$	$.731 \pm .004$	$.689 \pm .013$	$.494 \pm .022$	$.522 \pm .193$

TABLE XII

SUMMARY OF t-TEST OF G-MEAN WITH SIGNIFICANCE LEVEL AT 0.05. THE UPPER TRIANGLE SHOWS THE RESULT OF 6 'EASY' TASKS AND THE LOWER TRIANGLE SHOWS THE RESULT OF 10 'HARD' TASKS. EACH TABULAR SHOWS THE AMOUNT OF WIN-TIE-LOSE OF A METHOD IN A ROW COMPARING WITH THE METHOD IN A COLUMN.

	CART	Bagg	Ada	Asym	SMB	Under	Over	SMOTE	Chan	Cascade	Easy	RF	BRF	Under-RF	Over-RF
CART	–	0-0-6	0-0-6	0-0-6	NA	0-0-6	NA	0-0-6	0-0-6	0-0-6	0-0-6	1-0-5	1-0-5	1-0-5	1-0-5
Bagg	1-7-2	–	0-0-6	0-0-6	NA	1-0-5	NA	0-0-6	0-0-6	0-0-6	1-0-5	1-0-5	1-1-4	1-0-5	1-0-5
Ada	5-3-2	7-2-1	–	0-3-3	NA	2-2-2	NA	1-2-3	1-2-3	1-1-4	1-2-3	3-3-0	3-1-2	4-0-2	3-1-2
Asym	6-2-2	7-2-1	6-4-0	–	NA	3-1-2	NA	1-1-4	1-2-3	1-1-4	2-2-2	3-3-0	3-1-2	4-0-2	3-1-2
SMB	9-1-0	8-2-0	8-2-0	8-1-1	–	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
Under	10-0-0	9-1-0	10-0-0	10-0-0	9-1-0	–	NA	1-2-3	1-1-4	1-0-5	0-3-3	3-2-1	4-0-2	5-0-1	3-2-1
Over	3-5-2	5-3-2	0-5-5	0-2-8	0-2-8	0-0-10	–	NA	NA	NA	NA	NA	NA	NA	NA
SMOTE	10-0-0	9-0-1	10-0-0	10-0-0	9-0-1	0-1-9	10-0-0	–	3-2-1	1-3-2	2-3-1	5-1-0	5-0-1	5-0-1	5-1-0
Chan	10-0-0	9-0-1	10-0-0	10-0-0	7-2-1	3-1-6	10-0-0	4-1-5	–	1-3-2	4-1-1	4-2-0	5-1-0	4-1-1	4-2-0
Cascade	10-0-0	9-1-0	10-0-0	10-0-0	10-0-0	2-5-3	10-0-0	10-0-0	7-0-3	–	4-1-1	5-1-0	5-0-1	5-0-1	5-1-0
Easy	10-0-0	10-0-0	10-0-0	10-0-0	10-0-0	9-1-0	10-0-0	10-0-0	9-0-1	8-2-0	–	3-3-0	3-1-2	4-1-1	3-2-1
RF	1-6-3	2-6-2	1-2-7	1-1-8	0-2-8	0-1-9	1-5-4	1-0-9	1-0-9	0-0-10	0-0-10	–	2-1-3	1-2-3	0-2-4
BRF	10-0-0	10-0-0	10-0-0	9-1-0	9-1-0	6-2-2	10-0-0	9-0-1	7-1-2	6-2-2	2-3-5	10-0-0	–	2-1-3	2-1-3
Under-RF	10-0-0	10-0-0	10-0-0	10-0-0	9-1-0	5-3-2	10-0-0	9-0-1	6-2-2	6-2-2	1-1-8	10-0-0	3-3-4	–	2-2-2
Over-RF	6-4-0	8-2-0	7-2-1	6-3-1	2-2-6	1-0-9	7-2-1	1-0-9	1-0-9	1-0-9	0-0-10	8-2-0	0-1-9	0-0-10	–

contained in the majority class which is ignored by `Under`. Both our proposed methods can improve upon `Under`, no matter on 'easy' tasks or 'hard' tasks. This result supports our argument that `Easy` and `Cascade` can effectively explore the majority class examples.

`Chan` uses all the majority class examples, and it generally has higher AUC and F-measure than `Under`. But the results show that on 'hard' tasks, its AUC, F-measure and G-mean are comparable to or slightly lower than `Cascade`, and they are lower than `Easy` on most of the data sets. This implies that using all majority class examples is not necessary. In particular, when the data set is highly imbalanced, `Chan` will consume a lot of time.

Both `Easy` and `Cascade` attain higher average AUC, F-measure and G-mean than almost all the other methods on 'hard' tasks, except that `Cascade` is comparable to `Chan` with AUC and F-measure, and slightly worse than `BRF` and `Under-RF` with G-mean. But `Chan` has much lower G-mean, and, `BRF` & `Under-RF` have much lower AUC and F-measure than many other class-imbalance learning methods. While both `Easy` and `Cascade` are very robust with different performance measures.

`Easy` and `Cascade` can not only improve the AUC scores, but also reduce the training time. They require approximately the same training time as `Under`, and are faster than other methods. Considering both classification performance and training time, they are better than all other compared methods.

The results on 'hard' tasks show that `Cascade` is inferior to `Easy`. The way `Cascade` explores the majority class examples might be responsible for this observation. In `Cascade`, the majority training set of H_{i+1} is produced by H_i. Such a supervised, cascading way of sampling might suffer from overfitting. In other words, the correctly predicted majority class examples that have been filtered out may be useful [27]. In particular, some examples that are deemed redundant and discarded in earlier rounds may be helpful in some later rounds, after some other examples have been discarded. Note that there are also situations in which `Cascade` is preferred. From the results on "easy" tasks we can see that `Cascade` has higher AUC, F-measure and G-mean than `Easy` on almost all data sets. This suggests that `Cascade` can focus on more useful data. In addition, note that `Cascade` is more favorable than `Easy` on data set *balance* and *wpbc*. Both of these data sets have a very small minority class. In fact, if the number of examples in a class is very small, there is a significant chance that the examples will scatter around broadly. It is difficult to get a representative subset by using under-sampling alone. Focusing on more informative examples may be particularly helpful in this case. Also, `Cascade` is more suitable for highly imbalanced problems. For example, in the face detection problem described in [41], there are 5000 positive examples and 2284 million negative ones. The independent random sampling strategy of `Easy` requires T, the number of subsets, to be very large in order to catch all the information in $\mathcal{N}$. Furthermore, the number of subsets is hard to decide since no prior information is available. Thus, `Easy` is computationally infeasible for this problem. But for `Cascade`, it is much easier to set the iteration number since it is reasonable to set fp rate around 0.5. So, $T = 20$ is sufficient for the face detection problem, since $\log_2(2.284 \times 10^9/5000) \approx 19$ (assuming a false positive rate of 0.5).

E. Analysis of the Ensemble Strategy

As stated above, since minority class examples are used to train each weak classifier in the proposed method, stacking these classifiers may cause overfitting when the number of minority class examples is limited. To verify this, the 16 data sets in Table II were used to compare stacking with the ensemble strategy used in `Easy` and `Cascade`.

The AUC values are summarized in Table XIII. Similar to the experiments in the previous subsection, the 16 data sets are divided into groups based on the performance of AdaBoost. When `Cascade` is used on 'easy' tasks, stacking is inferior to the original ensemble strategy on 3 out of 6 data sets, while it is superior on only one data set. However, the difference between the two strategies is small. The same observation holds for `Easy`. On 'hard' tasks, the performance of `Cascade` dominates that of stacking on all data sets. As for `Easy`, there is only one data set on which stacking is better. Generally speaking, there are significant differences between the performance of stacking and the current ensemble strategy used in our proposed methods.

Therefore, stacking is not very suitable for the case when minority class examples are used in each weak classifier. In such a case, stacking may cause overfitting. This is probably a major reason for `Chan` to be inferior to `Easy`.

F. Additional Remarks

We have the following remarks regarding the results in AUC, F-measure and G-mean on both 'easy' and 'hard' tasks:

- The proposed methods `EasyEnsemble` and `BalanceCascade` are more robust than many other class-imbalance learning methods. When

TABLE XIII

COMPARISON OF STACKING WITH ENSEMBLE STRATEGY IN BALANCECASCADE AND EASYENSEMBLE. THIS TABLE SHOWS AUC'S OF THE COMPARED METHODS. THE FIRST GROUP DATA SETS IS 'EASY' TASKS, AND THE SECOND GROUP IS 'HARD' TASKS. THE ROW ***avg1.*** SHOWS THE AVERAGE AUC OF EACH METHOD ON 'EASY' TASKS. THE ROW ***avg2.*** SHOWS THE AVERAGE AUC ON 'HARD' TASKS. THE ROW ***avg.*** SHOWS THE OVERALL AVERAGE AUC. TABULAR IN BOLD DENOTES THE SUPERIOR ENSEMBLE STRATEGY BETWEEN THE ORIGINAL ONE AND STACKING.

Data Set	BalanceCascade		EasyEnsemble	
	original	stacking	original	stacking
car	0.996 ± 0.000	$\mathbf{0.997 \pm 0.000}$	0.994 ± 0.000	$\mathbf{0.995 \pm 0.000}$
letter	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
ionosphere	0.976 ± 0.002	0.976 ± 0.002	0.974 ± 0.002	0.974 ± 0.002
phoneme	$\mathbf{0.962 \pm 0.000}$	0.960 ± 0.000	$\mathbf{0.958 \pm 0.000}$	0.957 ± 0.000
sat	$\mathbf{0.949 \pm 0.001}$	0.944 ± 0.001	$\mathbf{0.947 \pm 0.000}$	0.946 ± 0.001
wdbc	$\mathbf{0.994 \pm 0.000}$	0.992 ± 0.001	$\mathbf{0.993 \pm 0.000}$	0.992 ± 0.001
avg1.	$\mathbf{0.979 \pm 0.018}$	0.978 ± 0.019	$\mathbf{0.978 \pm 0.018}$	0.977 ± 0.019
abalone	$\mathbf{0.828 \pm 0.002}$	0.802 ± 0.002	$\mathbf{0.847 \pm 0.002}$	0.844 ± 0.002
balance	$\mathbf{0.637 \pm 0.011}$	0.631 ± 0.008	0.633 ± 0.008	$\mathbf{0.640 \pm 0.012}$
cmc	$\mathbf{0.686 \pm 0.007}$	0.679 ± 0.006	$\mathbf{0.704 \pm 0.008}$	0.698 ± 0.009
haberman	$\mathbf{0.653 \pm 0.012}$	0.637 ± 0.013	$\mathbf{0.668 \pm 0.011}$	0.647 ± 0.011
housing	$\mathbf{0.809 \pm 0.008}$	0.800 ± 0.009	$\mathbf{0.827 \pm 0.005}$	0.811 ± 0.013
mf-morph	$\mathbf{0.904 \pm 0.002}$	0.903 ± 0.002	$\mathbf{0.917 \pm 0.001}$	0.916 ± 0.002
mf-zernike	$\mathbf{0.890 \pm 0.002}$	0.864 ± 0.003	$\mathbf{0.904 \pm 0.002}$	0.901 ± 0.001
pima	$\mathbf{0.799 \pm 0.005}$	0.792 ± 0.005	$\mathbf{0.809 \pm 0.004}$	0.802 ± 0.004
vehicle	$\mathbf{0.856 \pm 0.002}$	0.848 ± 0.002	$\mathbf{0.860 \pm 0.001}$	0.857 ± 0.004
wpbc	$\mathbf{0.712 \pm 0.011}$	0.707 ± 0.009	$\mathbf{0.707 \pm 0.009}$	0.705 ± 0.012
avg2.	$\mathbf{0.778 \pm 0.089}$	0.766 ± 0.087	$\mathbf{0.788 \pm 0.092}$	0.782 ± 0.092
avg.	$\mathbf{0.853 \pm 0.119}$	0.846 ± 0.122	$\mathbf{0.859 \pm 0.116}$	0.855 ± 0.119

class-imbalance is not harmful, they don't cause serious degeneration of performance. When class-imbalance is indeed harmful, they are better than almost all other methods we have compared with.

- Class-imbalance is not harmful for some tasks and applying class-imbalance learning methods in such cases may lead to performance degeneration. A consequence of this observation is that, class-imbalance learning methods should only be applied to tasks which suffer from class imbalance. For this purpose, we need to develop some methods to judge whether a task suffers from class imbalance or not, before applying class-imbalance learning methods to it.
- We observed that, on tasks which do not suffer from class-imbalance, AdaBoost and Bagging can improve the performance of decision trees significantly; while on tasks which suffer from class-imbalance, they could not help and sometimes even deteriorate the performance. This might give us some clues on judging whether a task suffers from class imbalance or not, which will be studied in the future.

V. CONCLUSION

This paper extends our preliminary work [26] which proposed two algorithms `EasyEnsemble` and `BalanceCascade` for class-imbalance learning. Both algorithms are designed to utilize the majority class examples ignored by under-sampling, while at the same time keeping its fast training speed. Both algorithms sample multiple subsets of the majority class, train an ensemble from each of these subsets, and combine all weak classifiers in these ensembles into a final output. Both algorithms make better use of the majority class than under-sampling, since multiple subsets contain more information than a single one. The main difference is that `EasyEnsemble` samples independent subsets, while `BalanceCascade` uses trained classifiers to guide the sampling process for subsequent classifiers. Both algorithms have approximately the same training time as that of under-sampling when the same number of weak classifiers are used.

Empirical results suggest that for problems on which ordinary methods achieve high AUC (e.g. ≥ 0.95), class-imbalance learning is not helpful. However, the proposed methods can be used to reduce training time. For problems where class-imbalance learning methods really help, both `EasyEnsemble` and `BalanceCascade` have higher AUC, F-measure and G-mean than almost all other compared methods and the former is superior than the latter. However, since `BalanceCascade` removes correctly classified majority class examples in each iteration, it will be more efficient on highly imbalanced data sets. In addition, the comparison of `Chan` and our proposed methods reveals that, it is not necessary to use all examples in the majority class.

In the current version of the proposed methods, we use the $\alpha_{i,j}$ returned by the weak learner directly. Further improvements are possible by learning $\alpha_{i,j}$, as shown in [22], [41]. Note that both `EasyEnsemble` and

`BalanceCascade` are ensemble methods. So, while they provide strong generalization ability, they also inherit the weaknesses of ensemble methods. An apparent weakness is the lack of comprehensibility. Even when the base classifiers are comprehensible symbolic learners, ensembles are still black-boxes. There are some research on this problem [44]–[46] and it is possible to use those research outputs to enhance the comprehensibility of `EasyEnsemble` and `BalanceCascade`.

ACKNOWLEDGMENT

The authors want to thank the anonymous reviewers and the associate editor for helpful comments and suggestions. This research was partially supported by the National Science Foundation of China (60635030, 60721002), the Jiangsu Science Foundation (BK2008018) and the National High Technology Research and Development Program of China (2007AA01Z169).

REFERENCES

[1] G. Batista, R. C. Prati, and M. C. Monard, “A study of the behavior of several methods for balancing machine learning training data,” *ACM SIGKDD Explorations*, vol. 6, no. 1, pp. 20–29, 2004.

[2] E. Bauer and R. Kohavi, “An empirical comparison of voting classification algorithms: Bagging, boosting, and variants,” *Machine Learning*, vol. 36, no. 1-2, pp. 105–139, 1999.

[3] C. Blake, E. Keogh, and C. J. Merz, “UCI repository of machine learning databases,” [http://www.ics.uci.edu/∼mlearn/MLRepository.html], Department of Information and Computer Science, University of California, Irvine, CA.

[4] A. P. Bradley, “The use of the area under the ROC curve in the evaluation of machine learning algorithms,” *Pattern Recognition*, vol. 30, no. 6, pp. 1145–1159, 1997.

[5] L. Breiman, “Bagging predictors,” *Machine Learning*, vol. 24, pp. 123–140, 1996.

[6] ——, “Random forest,” *Machine Learning*, vol. 45, pp. 5–32, 2001.

[7] L. Breiman, J. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*. CRC Press, 1984.

[8] P. K. Chan and S. J. Stolfo, “Toward scalable learning with non-uniform class and cost distributions: A case study in credit card fraud detection,” in *Proceedings of the 4th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, New York, NY, 1998, pp. 164–168.

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

[10] N. V. Chawla, N. Japkowicz, and A. Kolcz, “Editorial: Special issue on learning from imbalanced data sets,” *ACM SIGKDD Explorations*, vol. 6, no. 1, pp. 1–6, 2004.

[11] N. V. Chawla, N. Japkowicz, and A. Kotcz, Eds., *ICML’2003 Workshop on Learning from Imbalanced Data Sets*, 2003.

[12] N. V. Chawla, A. Lazarevic, L. O. Hall, and K. W. Bowyer, “SMOTEBoost: Improving prediction of the minority class in boosting,” in *Proceedings of the 7th European Conference on Principles and Practice of Knowledge Discovery in Databases*, Cavtat-Dubrovnik, Croatia, 2003, pp. 107–119.

[13] C. Chen, A. Liaw, and L. Breiman, “Using random forest to learn imbalanced data,” Deptarment of Statistics, UC Berkeley, Tech. Rep. 666, 2004.

[14] P. Domingos, “MetaCost: A general method for making classifiers cost-sensitive,” in *Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Diego, CA, 1999, pp. 155–164.

[15] C. Drummond and R. C. Holte, “C4.5, class imbalance, and cost sensitivity: Why under-sampling beats over-sampling,” in *Working Notes of the ICML’03 Workshop on Learning from Imbalanced Data Sets*, Washington, DC, 2003.

[16] C. Elkan, “The foundations of cost-senstive learning,” in *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, Seattle, WA, 2001, pp. 973–978.

[17] W. Fan, S. J. Stolfo, J. Zhang, and P. K. Chan, “AdaCost: Misclassification cost-sensitive boosting,” in *Proceedings of the 16th International Confernece on Machine Learning*, Bled, Slovenia, 1999, pp. 97–105.

[18] T. Fawcett, “ROC graphs: Notes and practical considerations for researchers,” HP Labs, Tech. Rep. HPL-2003-4, 2003.

[19] J. H. Friedman, “Stochastic gradient boosting,” *Computational Statistics and Data Analysis*, vol. 38, no. 4, pp. 367–378, 2002.

[20] K. Fukunaga, *Introduction to Statistical Pattern Recognition*. Academic Press, 1990.

[21] H. Guo and H. L. Viktor, “Learning from imbalanced data sets with boosting and data generation: The DataBoost-IM approach,” *ACM SIGKDD Explorations*, vol. 6, no. 1, pp. 30–39, 2004.

[22] K. Huang, H. Yang, I. King, and M. R. Lyu, “Learning classifiers from imbalanced data based on biased minimax probability machine,” in *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Washington, DC, 2004, pp. 558–563.

[23] N. Japkowicz, Ed., *AAAI’2000 Workshop on Learning from Imbalanced Data Sets*, 2000.

[24] G. J. Karakoulas and J. Shawe-Taylor, “Optimizing classifiers for imbalanced training sets,” in *Advances in Neural Information Processing Systems 11*. Cambridge, MA: MIT Press, 1999, pp. 253–259.

[25] M. Kubat and S. Matwin, “Addressing the curse of imbalanced training sets: One-sided selection,” in *Proceedings of the 14th International Conference on Machine Learning*, Nashville, TN, 1997, pp. 179–186.

[26] X.-Y. Liu, J. Wu, and Z.-H. Zhou, “Exploratory under-sampling for class-imbalance learning,” in *Proceedings of the 6th IEEE International Conference on Data Mining*, Hong Kong, 2006, pp. 965–969.

[27] F.-Z. Marcos, “On the usefulness of almost-redundant information for pattern recognition,” in *Summer School on Neural Networks*, 2004, pp. 357–364.

[28] H. Masnadi-Shirazi and N. Vasconcelos, “Asymmetric boosting,” in *Proceedings of the 24th International Confernece on Machine Learning*, Corvallis, OR, 2007.

[29] R. E. Schapire, “A brief introduction to Boosting,” in *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, Stockholm, Sweden, 1999, pp. 1401–1406.

[30] R. E. Schapire, Y. Singer, and A. Singhal, “Boosting and rocchio applied to text filtering,” in *Proceedings of the 4th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1998, pp. 215–223.

[31] K. M. Ting, “An empirical study of MetaCost using boosting algorithms,” in *Proceedings of the 11th European Conference on Machine Learning*, Barcelona, Spain, 2000, pp. 413–425.
[32] K. M. Ting and I. H. Witten, “Issues in stacked generalization,” *Journal of Artificial Intelligence Research*, vol. 10, pp. 271–289, 1999.
[33] P. Viola and M. Jones, “Fast and robust classification using asymmetric AdaBoost and a detector cascade,” in *Advances in Neural Information Processing Systems 14*, T. G. Dietterich, S. Becker, and Z. Ghahramani, Eds. Cambridge, MA: MIT Press, 2002, pp. 1311–1318.
[34] ——, “Robust real-time face detection,” *International Journal of Computer Vision*, vol. 57, no. 2, pp. 137–154, 2004.
[35] G. I. Webb, “MultiBoosting: A technique for combining boosting and wagging,” *Machine Learning*, vol. 40, pp. 159–196, 2000.
[36] G. I. Webb and Z. Zheng, “Multistrategy ensemble learning: Reducing error by combining ensemble learning techniques,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 8, pp. 980–991, 2004.
[37] G. M. Weiss, “Mining with rarity: A unifying framework,” *ACM SIGKDD Explorations*, vol. 6, no. 1, pp. 7–19, 2004.
[38] G. M. Weiss and F. Provost, “Learning when training data are costly: The effect of class distributions on tree induction,” *Journal of Artificial Intelligence Research*, vol. 19, pp. 315–354, 2003.
[39] I. H. Witten and E. Frank, *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.
[40] D. H. Wolpert, “Stacked generalization,” *Neural Networks*, vol. 5, no. 2, pp. 241–260, 1992.
[41] J. Wu, S. C. Brubaker, M. D. Mullin, and J. M. Rehg, “Fast asymmetric learning for cascade face detection,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, no. 3, pp. 369–382, 2008.
[42] Y. Yu, Z.-H. Zhou, and K. M. Ting., “Cocktail ensemble for regression,” in *Proceedings of the 7th IEEE International Conference on Data Mining*, Omeha, NE, 2007, pp. 721–726.
[43] B. Zadrozny, J. Langford, and N. Abe, “Cost-sensitive learning by cost-proportionate example weighting,” in *Proceedings of the 3rd IEEE International Conference on Data Mining*, Melbourne, FL, 2003, pp. 435–442.
[44] Z.-H. Zhou and Y. Jiang, “Medical diagnosis with C4.5 rule preceded by artificial neural network ensemble,” *IEEE Transactions on Information Technology in Biomedicine*, vol. 7, no. 1, pp. 37–42, 2003.
[45] ——, “NeC4.5: Neural ensemble based C4.5,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 6, pp. 770–773, 2004.
[46] Z.-H. Zhou, Y. Jiang, and S.-F. Chen, “Extracting symbolic rules from trained neural network ensembles,” *AI Communications*, vol. 16, no. 1, pp. 3–15, 2003.
[47] Z.-H. Zhou and X.-Y. Liu, “Training cost-sensitive neural networks with methods addressing the class imbalance problem,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, no. 1, pp. 63–77, 2006.

Xu-Ying Liu received her BSc and MSc degree in computer science from Nanjing University of Aeronautics and Astronautics, China in 2003 and Nanjing University, China in 2006, respectively. Currently she is a PhD candidate in Nanjing University and is a member of the LAMDA Group. Her research interests are in machine learning and data mining, especially in cost-sensitive and class imbalance learning.

Jianxin Wu received the BS degree and MSc degree in computer science, both from Nanjing University, China. He is currently a PhD candidate in Georgia Institute of Technology under the supervision of Dr. James M. Rehg. His research interests are computer vision, machine learning, and robotics.

Zhi-Hua Zhou (S’00-M’01-SM’06) received the BSc, MSc and PhD degrees in computer science from Nanjing University, China, in 1996, 1998 and 2000, respectively, all with the highest honors.

He joined the Department of Computer Science & Technology at Nanjing University as an assistant professor in 2001, and is currently Cheung Kong Professor and Director of the LAMDA group. His research interests are in artificial intelligence, machine learning, data mining, pattern recognition, information retrieval, evolutionary computation, and neural computation. In these areas he has published over 60 papers in leading international journals or conference proceedings.

Dr. Zhou has won various awards/honors including the National Science & Technology Award for Young Scholars of China (2006), the Award of National Science Fund for Distinguished Young Scholars of China (2003), the National Excellent Doctoral Dissertation Award of China (2003), the Microsoft Young Professorship Award (2006), etc. He is an Associate Editor of *IEEE Transactions on Knowledge and Data Engineering*, Associate Editor-in-Chief of *Chinese Science Bulletin*, and on the editorial boards of *Artificial Intelligence in Medicine*, *Intelligent Data Analysis*, *Knowledge and Information Systems*, *Science in China*, etc. He is/was a PAKDD Steering Committee member, Program Committee Chair/Co-Chair of PAKDD’07 and PRICAI’08, Vice Chair/Area Chair of ICDM’06, ICDM’08, etc., Program Committee member of various international conferences including AAAI, ICML, ECML, SIGKDD, ICML, ACM Multimedia, etc., and General Chair/Co-Chair or Program Committee Chair/Co-Chair of a dozen of native conferences. He is a senior member of China Computer Federation (CCF), the Vice Chair of the CCF Artificial Intelligence & Pattern Recognition Society, an Executive Committee member of Chinese Association of Artificial Intelligence (CAAI), the Chair of the CAAI Machine Learning Society, and the Chair of the IEEE Computer Society Nanjing Chapter. He is a member of AAAI and ACM, and a senior member of IEEE and IEEE Computer Society.