# Exploratory Under-Sampling for Class-Imbalance Learning

Xu-Ying Liu [1] Jianxin Wu [2] Zhi-Hua Zhou [1]
[1] National Laboratory for Novel Software Technology
Nanjing University, Nanjing 210093, China
[2] College of Computing and GVU Center
Georgia Institute of Technology, Atlanta, GA 30339, USA
{liuxy, zhouzh}@lamda.nju.edu.cn wujx@cc.gatech.edu

## Abstract

*Under-sampling is a class-imbalance learning method which uses only a subset of major class examples and thus is very efficient. The main deficiency is that many major class examples are ignored. We propose two algorithms to overcome the deficiency.* `EasyEnsemble` *samples several subsets from the major class, trains a learner using each of them, and combines the outputs of those learners.* `BalanceCascade` *is similar to* `EasyEnsemble` *except that it removes correctly classified major class examples of trained learners from further consideration. Experiments show that both of the proposed algorithms have better AUC scores than many existing class-imbalance learning methods. Moreover, they have approximately the same training time as that of under-sampling, which trains significantly faster than other methods.*

## 1 Introduction

Data in real-world tasks are usually imbalanced, i.e. some classes have much more instances than others. The level of imbalance (ratio of size of major class to that of minor class) can be as huge as $10^6$ [16]. Learning algorithms that do not consider class-imbalance tend to be overwhelmed by the major class and ignore the minor one [6].

Sampling is a class of methods that alters the sizes of training sets. Under-sampling and over-sampling change the training sets by sampling a smaller major training set and repeating minor class instances, respectively [8]. 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 learning problems [14, 17]. Besides the basic under-sampling and over-sampling methods, there are also methods that sample in more complex ways. SMOTE [5] is an example, which adds new synthetic minor class examples by randomly interpolating pairs of closest neighbors in the minor class.

Among various class-imbalance learning methods, under-sampling has been popularly used [14, 17]. Given the minor example set $\mathcal{P}$ and the major example set $\mathcal{N}$, under-sampling randomly samples a subset $\mathcal{N}'$ from $\mathcal{N}$, where $|\mathcal{N}'| < |\mathcal{N}|$. Usually we choose $|\mathcal{N}'| = |\mathcal{P}|$, and have $|\mathcal{N}'| \ll |\mathcal{N}|$ for highly imbalanced problems.

Since under-sampling uses only a subset of the major class examples to train the classifier, the training process is very efficient. However, potentially useful information contained in these ignored examples, i.e. examples in $\mathcal{N} \bigcap \overline{\mathcal{N}'}$, are neglected, which is the main deficiency of under-sampling. In this paper, we propose two algorithms, `EasyEnsemble` and `BalanceCascade`, to overcome the deficiency but keeping the efficiency of under-sampling. The intuition is to build ensemble of classifiers generated from multiple under-sampled training sets such that these ignored data can be wisely explored. Experiments show that both algorithms have achieved better performance than many existing class-imbalance learning methods.

## 2 EasyEnsemble

`EasyEnsemble` is probably the most straightforward way to further exploit the 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 $\mathcal{N}_i$ $(1 \le i \le T)$, a classifier $H_i$ is trained using $\mathcal{N}_i$ and $\mathcal{P}$. All generated classifiers are then combined for the final decision. Many learning algorithms can be employed to generate the individual classifiers. Here AdaBoost [11] is used. The pseudo-code of `EasyEnsemble` is shown in Algorithm 1.

The idea behind `EasyEnsemble` is quite simple. Similar to the balanced Random Forest [7], `EasyEnsemble` generates $T$ balanced sub-problems. The output of the $i$th sub-problem is $H_i$, an ensemble with weak learners $\{h_{i,j}\}$.



Proceedings of the Sixth International Conference on Data Mining (ICDM'06)
0-7695-2701-9/06 $20.00 © 2006 IEEE

**Algorithm 1** The `EasyEnsemble` algorithm.

1: {Input: A set of minor class examples $\mathcal{P}$, a set of major class examples $\mathcal{N}$, $|\mathcal{P}| < |\mathcal{N}|$, and $T$, the number of subsets to be sampled from $\mathcal{N}$.}
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 weak classifiers $h_{i,j}$ and corresponding weights $\alpha_{i,j}$, i.e.
$$H_i(\boldsymbol{x}) = \operatorname{sgn}\left(\sum_{j=1}^{s_i} \alpha_{i,j} h_{i,j}(\boldsymbol{x}) - \theta_i\right).$$
7: **until** $i = T$
8: Output: An ensemble:
$$H(\boldsymbol{x}) = \operatorname{sgn}\left(\sum_{i=1}^{T}\sum_{j=1}^{s_i} \alpha_{i,j} h_{i,j}(\boldsymbol{x}) - \sum_{i=1}^{T} \theta_i\right).$$

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. $H_i$, 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 major training set $\mathcal{N}$. Finally, instead of counting votes from the $H_i$'s, 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.

## 3 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 $\boldsymbol{x}^* \in \mathcal{N}$ is classified correctly as major class by $H_1$, it is reasonable to conjecture that $\boldsymbol{x}^*$ is somewhat redundant in $\mathcal{N}$, given that we already have $H_1$. Thus, we can remove part of the correctly classified major 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 similar to the cascade classifier in [13]. $\mathcal{N}$ is shrank after every node $H_i$ is trained, and every $H_i$ deals with a balanced sub-problem ($|\mathcal{N}_i| = |\mathcal{P}|$). However, the final classifier is different. A cascade classifier is the conjunction of all $H_i$'s, i.e. $H(x)$ predicts positive if and only if every $H_i(x)$ $(i = 1, 2, \ldots, T)$ predicts positive. Viola and Jones [13] 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 major class. This sampling strategy leads to a restricted sample space for the following under-sampling process, hoping to explore as much useful information as possible.

**Algorithm 2** The `BalanceCascade` algorithm.

1: {Input: A set of minor class examples $\mathcal{P}$, a set of major class examples $\mathcal{N}$, $|\mathcal{P}| < |\mathcal{N}|$, and $T$, the number of subsets to be sampled from $\mathcal{N}$.}
2: $i \leftarrow 0$
3: **repeat**
4: $\quad i \leftarrow i + 1$, $f \leftarrow \sqrt[T-1]{\frac{|\mathcal{P}|}{|\mathcal{N}|}}$
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 weak learners $h_{i,j}$ and corresponding weights $\alpha_{i,j}$, i.e.
$$H_i(\boldsymbol{x}) = \operatorname{sgn}\left(\sum_{j=1}^{s_i} \alpha_{i,j} h_{i,j}(\boldsymbol{x}) - \theta_i\right).$$
7: $\quad$ Adjust $\theta_i$ such that $H_i$'s false positive rate (the error rate of misclassifying a major class example to the minor class) 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:
$$H(\boldsymbol{x}) = \operatorname{sgn}\left(\sum_{i=1}^{T}\sum_{j=1}^{s_i} \alpha_{i,j} h_{i,j}(\boldsymbol{x}) - \sum_{i=1}^{T} \theta_i\right).$$

`BalanceCascade` is similar to `EasyEnsemble` in structure. The main difference between them is the lines 7 and 8 of Algorithm 2. Line 8 removes the true major class examples from $\mathcal{N}$, and line 7 specifies how many major class examples can be removed. At the beginning of the $T$-th iteration, $\mathcal{N}$ has been shrank $T - 1$ 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 shrank again, the size of $\mathcal{N}$ is smaller than $|\mathcal{P}|$. We can stop the training process at this time.

## 4 Experiments

### 4.1 Settings

Sixteen UCI data sets [1] are used. Information about these data sets are summarized in Table 1, where *Size* is the number of examples, *Target* is used as minor class while the union of all other classes is used as major class, *#min/#maj* is the size of minor/major class, and *Ratio* is the size of major class divided by that of minor class.

Since the data sets are imbalanced, AUC [2] is used as the performance measure. For every data set, 5 times 10-fold stratified cross validation are executed. In order to reduce the influence of the randomness introduced by the sampling process on evaluation, within each fold the learning process is repeated for 10 times. Finally, the averaged AUC score is recorded.



Proceedings of the Sixth International Conference on Data Mining (ICDM'06)
0-7695-2701-9/06 $20.00 © 2006 IEEE

**Table 1. Experimental Data Sets**

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

Our proposed algorithms (abbr. `Easy` and `Cascade`) are compared with seven existing class-imbalance learning methods, including `Bagging` (abbr. `Bagg`) [3], `AdaBoost` (abbr. `Ada`), `AsymBoost` (abbr. `Asym`) [12], `Under-sampling` (abbr. `Under`), `Over-sampling` (abbr. `Over`), `SMOTE`, and Chan & Stolfo's method (abbr. `Chan`) [4]. In `Under`, a subset $\mathcal{N}'$ is sampled from $\mathcal{N}$, $|\mathcal{N}'| = |\mathcal{P}|$. In `Over`, a new minor training set is sampled (with replacement) from the original minor class, $|\mathcal{P}'| = |\mathcal{N}|$. As for `SMOTE`, we first generate $\mathcal{P}'$, a set of synthetic minor class examples with $|\mathcal{P}'| = |\mathcal{P}|$; then, we sample a new major training set $\mathcal{N}'$ with $|\mathcal{N}'| = 2|\mathcal{P}|$ if $|\mathcal{N}| > 2|\mathcal{P}|$, and $\mathcal{N}' = \mathcal{N}$ otherwise. On data sets having nominal attributes we use `SMOTE-NC` [5]. `Chan` splits $\mathcal{N}$ into $|\mathcal{N}|/|\mathcal{P}|$ non-overlapping subsets, and the final ensemble is obtained by stacking [15] these classifiers. In our implementation, we use Fisher Discriminant Analysis [9] as the stacker. Note that the base classifiers of `Under`, `Over`, `SMOTE` and `Chan` are all AdaBoost ensembles, just as those used in `Easy` and `Cascade`.

In `Ada`, `Asym`, `Under`, `Over` and `SMOTE`, 40 weak learners are used for each ensemble. For `Easy` and `Cascade`, 4 subsets are sampled (i.e. $T$ is set to 4 in both algorithms), on each an ensemble containing 10 weak learners are trained. Thus, the final ensemble generated by `Easy` and `Cascade` also contain $10 \times 4 = 40$ weak learners. In `Chan`, there are $|N|/|P|$ subsets, where ensemble classifiers are trained for $40|P|/|N|$ iterations when $|N|/|P| < 40$, and one iteration otherwise. Thus, since the imbalance level of all the data sets in Table 1 is lower than 40, all these methods use the same number of weak learners. Thus, the comparison among these methods is fair. In all experiments, we use the classification tree in Matlab's statistics toolbox as the weak learner.

Since all methods use the same type and same number of weak learners, the training time of these methods mainly depends on the number of training examples. `Under` uses the smallest number ($2|\mathcal{P}|$) of examples and therefore is the fastest. `Easy`, `Cascade` and `Chan` use as same number of weak learners as `Under`, and use as same number of examples as `Under` to train every weak learner [1]. These methods require additional time cost to sample or split subsets of $\mathcal{N}$, but this time cost is obviously negligible. Thus, `Easy`, `Cascade` and `Chan` have approximately the same training time cost as that of `Under`. Note that when the imbalance level is higher than 40, `Chan` will use more weak learners and therefore its time cost will be bigger than that of `Easy` and `Cascade`. Both `Ada` and `Asym` use $|\mathcal{P}| + |\mathcal{N}|$ examples. Since $|\mathcal{N}| \gg |\mathcal{P}|$, these methods are much slower than `Under`. `SMOTE` uses either $4|\mathcal{P}|$ or $2|\mathcal{P}| + |\mathcal{N}|$ examples, and therefore it is slower than `Under`, `Easy` and `Cascade`. Among all the compared methods, `Over` is the most time-consuming since it uses $2|\mathcal{N}|$ examples. On data sets with large number of examples, e.g. *letter*, the training time of `Over` is too long to be practical.

## 4.2 Results

We divide the sixteen data sets into two groups according to the performance of `Ada`. The AUC scores of `Ada` are higher than 0.95 on six data sets, and Table 2 summarizes the performance on these "easy" data sets. Results on the remaining ten "hard" data sets are tabulated in Table 3. Note that the performance of `Over` on the data sets in the former group has not been obtained due to its large training time costs. In the tables the best performance on each data set is bolded, while the worst performance is underlined. The *avg.* row shows the average AUC score of each method. Besides, in Table 3 the entries marked with '*' denotes the first runner-up, and the *avg2.* row shows the average AUC score when the *cmc* data set is excluded.

Table 2 contains examples of problems on which class-imbalance learning methods do not help. The observations from Table 2 are summarized as follows:

- Although the imbalance level ranges from 1.7 to 24.3 on these 6 data sets, performance of the compared methods are quite similar. Except `Bagg` and `Under`, the AUC scores of the other 6 methods are almost indistinguishable. Thus, class-imbalance does not necessarily imply degenerated performance.
- `Bagg` is with the worst performance. This is probably because that the sampling strategy in `Bagg` makes class-imbalance more serious sometimes, and the profit from varying distribution is overwhelmed by the negative effect it brings.

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



Proceedings of the Sixth International Conference on Data Mining (ICDM'06)
0-7695-2701-9/06 $20.00 © 2006 IEEE

**Table 2. AUC Scores on "easy" data sets**

| Data Set | Bagg | Ada | Asym | Under |
|---|---|---|---|---|
| *car* | 0.995 | **0.998** | **0.998** | 0.989 |
| *letter* | 0.997 | **1.000** | **1.000** | **1.000** |
| *ionosphere* | 0.962 | 0.978 | **0.979** | 0.973 |
| *phoneme* | 0.955 | **0.965** | **0.965** | 0.953 |
| *sat* | 0.946 | 0.953 | 0.953 | 0.941 |
| *wdbc* | 0.987 | **0.994** | **0.994** | 0.993 |
| *avg.* | 0.974 | 0.981 | **0.982** | 0.975 |
| Data Set | SMOTE | Chan | Cascase | Easy |
| *car* | 0.989 | 0.996 | 0.996 | 0.994 |
| *letter* | **1.000** | **1.000** | **1.000** | **1.000** |
| *ionosphere* | 0.978 | 0.977 | 0.976 | 0.974 |
| *phoneme* | 0.964 | 0.960 | 0.962 | 0.958 |
| *sat* | 0.946 | **0.955** | 0.949 | 0.947 |
| *wdbc* | **0.994** | 0.993 | **0.994** | 0.993 |
| *avg.* | 0.978 | 0.980 | 0.979 | 0.978 |

- Under has lower AUC scores than other methods on 3 of the 6 data sets. Our conjecture is that this is due to the information contained in the major class has been ignored. It is worth noting that both our proposed methods can improve upon Under, and have similar AUC scores as Ada. This result supports our claim that Easy and Cascade can effectively explore the major training set.
- Chan is superior to Under, but there is no significant difference between Chan and Easy & Cascade. This implies that using all major class examples is not necessary at all. Especially when the data set is highly imbalanced, Chan will consume much more time.

Based on the above observations, we argue that for tasks on which ordinary methods can achieve a high AUC score (e.g. $\geq 0.95$), class-imbalance learning is generally not helpful. However, our proposed methods can be used to save the training time.

We are more interested in Table 3 which reports on data sets where class-imbalance learning really helps. The observations from Table 3 are summarized as follows:

- Although Bagg is helpful on some data sets, it is not reliable at all. It has the worst performance on 3 data sets. In particular, it performs extremely poor on data sets which have very small minor class, such as *balance* and *wpbc*.
- Asym, although proven to be effective for a fixed cost ratio [12], is not better than Ada if we consider the AUC score which is the overall performance in the entire range of false positive rate.
- Over, although reported in some paper, e.g. [10], to be effective in handling class-imbalance problems, helps little in improving the AUC score in our experiments. Conversely, it leads to the decrease of performance on the "hard" data sets. This observation is consistent with the results in [8].

**Table 3. AUC Scores on "hard" data sets**

| Data Set | Bagg | Ada | Asym | Under | Over |
|---|---|---|---|---|---|
| *abalone* | 0.825 | 0.811 | 0.812 | 0.830 | 0.816 |
| *balance* | 0.440 | 0.616 | 0.619 | 0.617 | 0.542 |
| *cmc* | **0.705** | 0.675 | 0.675 | 0.671 | 0.674 |
| *haberman* | 0.666* | 0.641 | 0.639 | 0.646 | 0.639 |
| *housing* | 0.825* | 0.815 | 0.817 | 0.806 | 0.819 |
| *mf-morph* | 0.887 | 0.888 | 0.888 | 0.916* | 0.890 |
| *mf-zernike* | 0.854 | 0.795 | 0.800 | 0.879 | 0.779 |
| *pima* | **0.820** | 0.788 | 0.788 | 0.789 | 0.790 |
| *vehicle* | 0.859* | 0.855 | 0.855 | 0.847 | 0.856 |
| *wpbc* | 0.682 | 0.716 | **0.721** | 0.694 | 0.720* |
| *avg.* | 0.756 | 0.760 | 0.761 | 0.769 | 0.752 |
| *avg2.* | 0.762 | 0.770 | 0.771 | 0.780 | 0.761 |
| Data Set | SMOTE | Chan | Cascase | Easy | |
| *abalone* | 0.832 | **0.850** | 0.828 | 0.847* | |
| *balance* | 0.617 | **0.648** | 0.637* | 0.633 | |
| *cmc* | 0.549 | 0.699 | 0.686 | 0.704* | |
| *haberman* | 0.647 | 0.643 | 0.653 | **0.668** | |
| *housing* | 0.814 | 0.812 | 0.809 | **0.827** | |
| *mf-morph* | 0.913 | 0.912 | 0.904 | **0.917** | |
| *mf-zernike* | 0.862 | 0.902* | 0.890 | **0.904** | |
| *pima* | 0.792 | 0.786 | 0.799 | 0.809* | |
| *vehicle* | 0.857 | 0.859* | 0.856 | **0.860** | |
| *wpbc* | 0.709 | 0.709 | 0.712 | 0.707 | |
| *avg.* | 0.759 | 0.782* | 0.778 | **0.788** | |
| *avg2.* | 0.782 | 0.791* | 0.788 | **0.797** | |

- SMOTE has better results than Ada on most data sets. This observation corroborates the results in [5]. However, SMOTE was designed for data sets with only continuous attributes. Since all attributes of *cmc* are nominal, SMOTE-NC is used, which attains a significantly lower score than other methods. By excluding *cmc*, the *avg2.* row shows that SMOTE's AUC score is higher than that of Ada but similar to that of Under.
- Chan achieves the best performance on two data sets. It is comparable with Cascade on average but much worse than Easy. Moreover, Chan causes negative effect on *pima*. This reveals again that, using all the major class examples is generally not necessary. By using a subset of major class examples, Easy and Cascade perform comparable to or even much better than Chan.
- Expect Over, all class-imbalance learning methods achieve higher average AUC scores than Ada on these "hard" data sets. Although Under has a low average score among the compared methods, it can be used as a very efficient baseline in highly imbalanced problems.

Observations about the proposed methods, Easy and Cascade, deserve a separate thread of discussions:



Proceedings of the Sixth International Conference on Data Mining (ICDM'06)
0-7695-2701-9/06 $20.00 © 2006 IEEE

- Both `Easy` and `Cascade` attain higher average AUC scores than almost all the other methods (except that `Chan` is comparable to `Cascade`). Moreover, both have higher scores than `Ada`, `Asym`, `Over` and `Under` on almost all data sets. `Easy` achieves the highest scores on 5 out of 10 data sets, and 3 first runner-ups. In addition, the proposed methods never cause negative effect, while the other methods will.
- `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 both superior to all other compared methods.
- Table 3 shows that `Cascade` is inferior to `Easy`. The way `Cascade` explores the major class examples might be responsible for this observation. In `Cascade`, the major training set of $H_{i+1}$ is produced by $H_i$. Such a supervised, cascade way of sampling could suffer from overfitting, or, the correctly-predicted major class examples that have been filtered out may be useful.

## 5 Conclusion

In this paper, we propose `EasyEnsemble` and `BalanceCascade` for class-imbalance learning. They are designed to utilize the major class examples ignored by under-sampling, while at the same time to keep the fast training speed of under-sampling. Both methods sample multiple subsets of the major class, train an ensemble from each of these subsets, and combine all weak learners in these ensembles into a final ensemble. Both methods make better use of the major class than under-sampling, since multiple subsets contain more information than a single one. The main difference is that `EasyEnsemble` samples independent subsets, while in `BalanceCascade` correctly predicted major class examples are removed from further considerations. Both methods have approximately the same training time as that of under-sampling.

Empirical results suggest that, for problems on which ordinary methods achieve high AUC scores (e.g. $\geq 0.95$), class-imbalance learning is not helpful. However, the proposed methods can be used to save the training time. For problems where class-imbalance learning methods really help, both `EasyEnsemble` and `BalanceCascade` have higher AUC scores than almost all other compared methods and the former is superior than the latter. However, since `BalanceCascade` removes correctly classified major class examples in each iteration, it would 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 major class examples.

**Acknowledgement**: Z.-H. Zhou was supported by NSFC (60325207) and JiangsuSF (BK2004001).

## References

[1] 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, 1998.

[2] A. P. Bradley. The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30(7):1145–1159, 1997.

[3] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[4] P. Chan and S. Stolfo. Toward scalable learning with non-uniform class and cost distributions: A case study in credit card fraud detection. In *Proceeding of KDD'98*, pages 164–168, New York, NY.

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

[6] N. V. Chawla, N. Japkowicz, and A. Kotcz. Editorial to the special issue on learning from imbalanced data sets. *ACM SIGKDD Explorations*, 6(1):1–6, 2004.

[7] C. Chen, A. Liaw, and L. Breiman. Using random forest to learn imbalanced data. Technical Report 666, Department of Statistics, UC Berkeley, 2004.

[8] 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.

[9] K. Fukunaga. *Introduction to Statistical Pattern Recognition*. Academic Press, Boston, MA, 2nd edition, 1990.

[10] N. Japkowicz and S. Stephen. The class imbalance problem: A systematic study. *Intelligent Data Analysis*, 6(5):429–449, 2002.

[11] R. E. Schapire. A brief introduction to Boosting. In *Proceedings of IJCAI'99*, pages 1401–1406, Stockholm, Sweden.

[12] P. Viola and M. Jones. Fast and robust classification using asymmetric AdaBoost and a detector cascade. In *Advances in Neural Information Processing Systems 14*, pages 1311–1318. MIT Press, Cambridge, MA, 2002.

[13] P. Viola and M. Jones. Robust real-time object detection. *International Journal of Computer Vision*, 57(2):137–154, 2004.

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

[15] D. H. Wolpert. Stacked generalization. *Neural Networks*, 5(2):241–260, 1992.

[16] J. Wu, M. D. Mullin, and J. M. Rehg. Linear asymmetric classifier for cascade detectors. In *Proceedings of ICML'05*, pages 993–1000, Bonn, Germany.

[17] Z.-H. Zhou and X.-Y. Liu. Training cost-sensitive neural networks with methods addressing the class imbalance problem. *IEEE Trans Knowledge and Data Engineering*, 18(1):63–77, 2006.



Proceedings of the Sixth International Conference on Data Mining (ICDM'06)
0-7695-2701-9/06 $20.00 © 2006 IEEE