Online Bagging and Boosting for Imbalanced Data Streams

Boyu Wang and Joelle Pineau

Abstract—While both cost-sensitive learning and online learning have been studied separately, these two issues have seldom been addressed simultaneously. Yet, there are many applications where both aspects are important. This paper investigates a class of algorithmic approaches suitable for online cost-sensitive learning, designed for such problems. The basic idea is to leverage existing methods for online ensemble algorithms, and combine these with batch mode methods for cost-sensitive bagging/boosting algorithms. Within this framework, we describe several theoretically sound online cost-sensitive bagging and online cost-sensitive boosting algorithms, and show that the convergence of the proposed algorithms is guaranteed under certain conditions. We then present extensive experimental results on benchmark datasets to compare the performance of the various proposed approaches.

Index Terms—Bagging, boosting, ensemble learning, cost-sensitive learning, online learning, class imbalance problem

1 Introduction

In many real world applications, such as medical diagnosis [38], network intrusion detection [10], and spam filtering [16], the distribution between the classes of examples is not well balanced. Consider a scenario where we want to automatically detect whether an incoming individual is an intruder or not, we will typically have many more negative examples (i.e., non-intruders) than positive examples (i.e., intruders). Conventional machine learning algorithms usually have difficulty learning from imbalanced class problems since their objective is to minimize the overall error rate, thus implicitly treating all misclassification costs equally. As a result, these algorithms may produce trivial results, typically classifying all test examples as negative (i.e., the majority class). Additionally, it is often the case that the positive (minority) class is the one of greater interest: the expected cost of missing an intruder may be much higher than the cost of mis-identifying a non-intruder. An effective way to handle the class imbalance problem is to define the learning objective using a cost-sensitive criterion, increasing the cost of misclassifying rare examples compared to the cost of classifying common ones [15], [21].

For some applications, it is necessary to handle class imbalance in the context of learning from data streams. Consider for example the task of automatically detecting epileptic seizures from EEG records [19]: seizures are relatively rare compared to non-seizure brain activities, and the EEG data is collected incrementally over time, yet one would like to deploy the seizure detection system as soon as some modest amount of training data is available for that patient (e.g., a handful of seizures), with the possibility of gradually improving the system as more data becomes available. Another example is credit card fraud detection, where the positive instances are rare, data arrives in a stream, and furthermore the size of the dataset prevents holding it all in memory at the same time.

Many techniques have been proposed to deal with the class imbalance problem in the setting of batch learning. Of these, ensemble learning approaches have been shown to be particularly effective [18]. In this paper, we will therefore restrict our attention to bagging and boosting approaches to cost-sensitive learning. Note that, as of yet, none of these approaches have been extended to the online learning problem.[1]

The main goal of this paper is to shed light on the joint problem of learning from imbalanced datasets in a data stream setting. Although these two issues have been well studied independently in the past decades, the research on simultaneously tackling both problems is limited. The main contribution of this paper consists of an online cost-sensitive ensemble learning framework, which generalizes a batch of widely used bagging and boosting based cost-sensitive learning algorithms to their online versions. In particular, we present online versions of UnderOverBagging [48], SMOTEBagging [48], AdaC2 [43], CSB2 [44], RUSBoost [41], and SMOTEBoost [7]. In addition, we provide theoretical analysis showing the consistency between the new online variants and their original batch mode counterparts. This analysis confirms that under certain conditions, as the number of examples approaches infinity, the models generated by online cost-sensitive ensemble learning algorithms converge to that of batch cost-sensitive ensembles.

The performance of the proposed online cost-sensitive bagging and boosting algorithms is evaluated on several benchmark datasets. This empirical analysis primarily focuses on the consistency between the new online variants and the original batch mode versions. We observe that the bagging-based algorithms achieve better performance than boosting-based algorithms in terms of both consistency and value of the area under the ROC curve (AUC). In addition, we also empirically demonstrate that our algorithms

• *The authors are with the School of Computer Science, McGill University, McConnell Engineering Bldg., 3480 University Street, Montreal, QC H3A 0E9, Canada. E-mail: boyu.wang@mail.mcgill.ca, jpineau@cs.mcgill.ca.*

Manuscript received 10 Nov. 2015; revised 9 Sept. 2016; accepted 11 Sept. 2016. Date of publication 14 Sept. 2016; date of current version 2 Nov. 2016.
Recommended for acceptance by J. Gama.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TKDE.2016.2609424

1. In this paper, online learning is referred to the learning paradigm responding immediately to a new instance and then discarding it. In some cases, the positive examples (minority class) are stored to create additional synthetic samples.

outperform two recent state-of-the-art online cost-sensitive learning algorithms in terms of misclassification cost.

The paper is organized as follows. Section 2 reviews related work of class imbalance learning and online learning. Section 3 presents the technical background for our work. The full online cost-sensitive ensemble learning framework and its many instantiations is described in Section 4, followed by the theoretical analysis in Section 5. The experimental results are reported in Section 6. Section 7 completes the paper with conclusions and avenues for future work.

2 Related Work

Existing approaches to the class imbalance problem can be roughly categorized into algorithm-level approaches and data-level approaches. The former directly modify traditional algorithms to achieve cost sensitivity by taking different misclassification costs into consideration when designing the algorithms, such that the misclassification cost of positive examples is higher than that of negative ones. The classifier is then trained to minimize the cost (over all examples), rather than the classification error. There are several ways to incorporate the modified classification costs. For example, a cost-sensitive SVM can be derived by kernel modification [49], biased penalty [32], or loss function modification [34]. For decision tree, cost sensitivity can be introduced by probabilistic estimate calibration [50] or using different pruning methods [13]. Cost-sensitivity can also be imposed on the model by a post-processing step, such as associating costs to prediction errors and then tuning the model, or by moving the decision threshold to minimize the expected cost [3]. In the case of data-level approaches, a pre-processing step is added (prior to learning) to artificially rebalance the class distribution by *undersampling* the negative class, or *oversampling* the positive class, or even creating synthetic positive examples [6]. After resampling, any conventional (cost-insensitive) algorithm can then be applied on the rebalanced dataset. One advantage of this strategy is that it is applicable to any existing classification algorithm. Such over-/under-sampling techniques can also be incorporated within ensemble learning algorithms [39], and have received much attention recently due to their convincing performance on imbalanced datasets [18], [26].

Regarding the problem of learning from data streams, some basic classification algorithms can easily be extended to the online setting, including *k*-NN, naive Bayes classifier, binary linear discriminant analysis (LDA), and quadratic discriminant analysis (QDA). In addition, the incremental/online versions of several more sophisticated algorithms have been proposed in the literature, including but not limited to decision trees [45], random forests [12], [40], multi-class LDA [27], [33], logistic regression [29], support vector machines [5], [30], and other kernel methods [24], [28]. Online versions of ensemble learning techniques, bagging and boosting, were also derived in [1], [8], [36].

More recently, online learning algorithms for imbalanced data streams have been proposed and studied empirically and theoretically in [47], [51]. The work in [51] aims at maximizing the AUC value, and the algorithms proposed in [47] directly deal with the cost-sensitive measures (e.g., misclassification cost). However, both of these methods only focus on a simple linear model (i.e., perceptron). Our proposed framework improves upon this in two significant ways: (i) accommodating any online learning algorithm, and (ii) producing more flexible (i.e., non-linear) class boundaries.

3 Technical Background

In this section, we briefly review the standard bagging and boosting algorithms, as well as their online and cost-sensitive versions, which motivate the proposed online cost-sensitive ensemble framework.

3.1 Standard Bagging and Boosting

Ensemble learning algorithms work by combining the outputs of multiple base learners. The basic insight is to improve the generalization ability of individual classifiers by training multiple base classifiers on different datasets (sampled from the original dataset) and combining the results. Averaging the outputs of several classifiers helps reduce the variance component and/or the bias in the classification error [2]. Varying the approach used to generate the datasets and to average the base classifiers produces distinct ensemble methods. Two representative techniques among them are bagging [4] and boosting (AdaBoost) [17].

Given a data set $S = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ of size N, where $x_n \in X$, $y_n \in Y = \{0, 1\}$, bagging constructs M classifiers $\{h_m\}, m = \{1, \ldots, M\}$ with bootstrap replicas $\{S_m\}$ of S, where S_m is obtained by drawing examples from the original data set S with replacement, usually having the same number of examples as S. The diversity among the classifiers is introduced by independently constructing different subsets of the original dataset. After constructing the ensemble of solutions, the prediction of the class of a new example is given by majority voting. The pseudo-code for bagging is shown in Algorithm 1, where $I(\omega)$ is the indicator function of event ω.

Algorithm 1. Bagging [4]

```
Input: S: data set, M: number of base learners
1: for $m = 1, \ldots, M$ do
2:     $S_m = Sample\_with\_replacement(S, N)$
3:     Train a base learner $h_m \rightarrow Y$ using $S_m$
4: end for
Output: $H(x) = \arg\max_{y \in Y} \sum_{m=1}^{M} I(h_m(x) = y)$
```

AdaBoost (Algorithm 2) is another widely used ensemble learning algorithm [17]. Unlike Bagging, which treats all examples equally at each iteration, AdaBoost focuses more on difficult examples. In particular, AdaBoost sequentially constructs a series of base learners in such a way that examples that are misclassified by the current base learner h_m are given more weight in the training set for the following learner h_{m+1}, whereas the correctly classified examples are given less weight. More specifically, the weights of all examples are equally initialized at the first iteration (i.e., $D_1(n) = \frac{1}{N}$ for all $n \in \{1, \ldots, N\}$). At the mth iteration, the base learner h_m is trained on examples weighted by the distribution D_m, and then the weight of x_n at the next iteration is updated according to

$$D_{m+1}(n) = \frac{D_m(n) exp(-\alpha_m(2I(h_m(x_n) = y_n) - 1))}{Z_m}, \quad (1)$$

where $\alpha_m = \frac{1}{2} \log \frac{\sum_{n=1}^{N} I(h_m(x_n)=y_n)}{\sum_{n=1}^{N} I(h_m(x_n) \neq y_n)}$, and Z_m is the normalization factor to ensure that the weights $D_{m+1}(n)$ sum to one. However, Eq. 1 cannot be used to implement online boosting since the normalization factor Z_m is unavailable during the online learning process. To sidestep this problem, the update rule can be reformulated as

$$D_{m+1}(n) = D_m(n) \times \begin{cases} \frac{1}{2(1-\epsilon_m)}, & h_m(x_n) = y_n \\ \frac{1}{2\epsilon_m}, & h_m(x_n) \neq y_n \end{cases} \tag{2}$$

where $\epsilon_m = \sum_{n=1}^{N} D_m(n) I(h_m(x_n) \neq y_n)$, and one can check that Eq. 2 is equivalent to Eq. 1. That is, the summation $\sum_{n=1}^{N} D_{m+1}(n) = 1$ after each update without normalization, which is crucial for the online version of boosting [36], and will be equally important in our proposed online cost-sensitive boosting algorithms. In standard AdaBoost, after each update, either the reweighted examples can be directly used to train the next base learner, or they are first resampled according to the weights and then the unweighted samples are used to train the base learner. In this paper, all boosting techniques are implemented by resampling. The reasons are three-fold: first, sampling based boosting algorithms are consistent with bagging techniques; second, they are also consistent with their online counterparts introduced later, which simulate sampling with replacement by using Poisson distribution; finally, for some learning algorithms reweighting the training set is not feasible.

Algorithm 2. AdaBoost [17]

```
Input: S: data set, M: number of base learners
1: Initialize $D_1(n) = \frac{1}{N}$ for all $n \in \{1, \dots, N\}$
2: for $m = 1, \dots, M$ do
3:   Train a base learner $h_m \to Y$ using $S$ with distribution $D_m$
4:   $\epsilon_m = \sum_{n=1}^{N} D_m(n) I(h_m(x_n) \neq y_n)$
5:   for $n = 1, \dots, N$ do
6:     $D_{m+1}(n) = D_m(n) \times \begin{cases} \frac{1}{2(1-\epsilon_m)}, & h_m(x_n) = y_n \\ \frac{1}{2\epsilon_m}, & h_m(x_n) \neq y_n \end{cases}$
7:   end for
8: end for
Output: $H(x) = \arg\max_{y \in Y} \sum_{m=1}^{M} \log(\frac{1-\epsilon_m}{\epsilon_m}) I(h_m(x) = y)$
```

Algorithm 3. Online Bagging [36]

```
Input: S: data set, M: number of base learners
1: for $n = 1, \dots, N$ do
2:   for $m = 1, \dots, M$ do
3:     Let $k \sim Poisson(1)$
4:     Do $k$ times
         Train the base learner $h_m \to Y$ using $(x_n, y_n)$
5:   end for
6: end for
Output: $H(x) = \arg\max_{y \in Y} \sum_{m=1}^{M} I(h_m(x) = y)$
```

3.2 Online Bagging and Boosting

The framework for online ensemble learning proposed by Oza et al. [36] is inspired by the observation that the binomial distribution $Binomial(p, N)$ can be approximated by a Poisson distribution $Poisson(\lambda)$ with $\lambda = Np$ as $N \to \infty$. Let the probability of success p in the binomial distribution be equivalent to $D(n)$ in bagging and boosting algorithms. For example, since $D(n) = \frac{1}{N}$ for all examples in the bagging algorithm, the uniform sampling with replacement of the bagging algorithm can be approximated by $Poisson(1)$ in its online version. For online boosting, λ can be computed by tracking the total weights of correctly classified and misclassified examples for each base learner (denoted as λ_m^{SC} and λ_m^{SW} respectively[2]). The online bagging and boosting algorithms are described in Algorithms 3 and 4 respectively. The asymptotic properties of online bagging and boosting have been studied in [36], [37], showing consistency between the online variant and their batch counterpart.

Algorithm 4. Online AdaBoost [36]

```
Input: S: data set, M: number of base learners
1: Initialize $\lambda_m^{SC} = 0, \lambda_m^{SW} = 0$ for all $m \in \{1, \dots, M\}$
2: for $n = 1, \dots, N$ do
3:   Set $\lambda = 1$
4:   for $m = 1, \dots, M$ do
5:     Let $k \sim Poisson(\lambda)$
6:     Do $k$ times
7:       Train the base learner $h_m \to Y$ using $(x_n, y_n)$
8:     if $h_m(x_n) = y_n$ then
9:       $\lambda_m^{SC} \leftarrow \lambda_m^{SC} + \lambda, \epsilon_m \leftarrow \frac{\lambda_m^{SW}}{\lambda_m^{SC}+\lambda_m^{SW}}, \lambda \leftarrow \frac{\lambda}{2(1-\epsilon_m)}$
10:    else
11:      $\lambda_m^{SW} \leftarrow \lambda_m^{SW} + \lambda, \epsilon_m \leftarrow \frac{\lambda_m^{SW}}{\lambda_m^{SC}+\lambda_m^{SW}}, \lambda \leftarrow \frac{\lambda}{2\epsilon_m}$
12:    end if
13:  end for
14: end for
Output: $H(x) = \arg\max_{y \in Y} \sum_{m=1}^{M} \log(\frac{1-\epsilon_m}{\epsilon_m}) I(h_m(x) = y)$
```

3.3 Cost-Sensitive Bagging and Boosting

Cost-sensitive ensemble learning methods typically manipulate misclassification costs via biased resampling/reweighting of the data before each iteration of bagging/boosting. With this, any cost-insensitive classifier can become cost-sensitive, and be incorporated into an ensemble learning framework to take advantage of the useful properties of ensembles with respect to bias/variance reduction. The main difference between various approaches to cost-sensitive bagging and boosting lies in the choice of the resampling mechanism. We briefly review six popular cost-sensitive ensemble learning algorithms: UnderOverBagging [48], SMOTEBagging [48], AdaC2 [43], CSB2 [44], RUSBoost [41], and SMOTEBoost [7]. The derivation of online extensions for these algorithms is the main contribution of this paper and is described formally in the next section.

Given an imbalanced dataset containing N^+ examples in the minority class S^+ and N^- examples in the majority class S^-, one straightforward approach to implement bagging-based learning algorithms is to undersample the majority class or oversample the minority class, which respectively yields approaches known as UnderBagging and OverBagging. UnderOverBagging [48] is a uniform approach combining both UnderBagging and OverBagging. In addition, the resampling rate a varies over the bagging iterations, which further boosts the diversity among the base learners. Algorithm 5 presents UnderOverBagging. We note that the sampling method is gradually switched from undersampling the majority class to oversampling the minority class. The number of training examples for the first base learner is lower than the last one. Since both UnderBagging and OverBagging can be regarded as special cases of UnderOverBagging, only UnderOverBagging is investigated in this paper.

In SMOTEBagging [48] (Algorithm 6), the negative class is sampled with replacement at rate 100 percent (i.e., N^-

2. The superscripts SC and SW stand for the *samples* that are *correctly* classified and *wrongly* classified, respectively.

negative examples are generated), while CN^+ positive examples are generated for each base learner, for some $C > 1$. Of these examples, m/M are generated by resampling, and the rest via the synthetic minority oversampling technique (SMOTE) [6] described in Algorithm 7. The main idea of SMOTE is to generate synthetic examples by interpolating the positive examples. As a result, all of the base learners are trained on a more balanced and diverse dataset. The diversity is further boosted by varying a so that the ratio of bootstrap replicates and synthetic examples generated by SMOTE varies over the bagging iterations.

Algorithm 5. Batch UnderOverBagging [48]

```
Input: $S$: data set, $M$: number of base learners, $C > 1$: sampling rate
1: for $m = 1, \dots, M$ do
2:     $a = \frac{m}{M}$
3:     $S^-_{train} = Sample\_with\_replacement(S^-, aN^-)$
4:     $S^+_{train} = Sample\_with\_replacement(S^+, aCN^+)$
5:     $S_m = S^+_{train} + S^-_{train}$
6:     Train a base learner $h_m \rightarrow Y$ using $S_m$
7: end for
Output: $H(x) = \arg\max_{y \in Y} \sum_{m=1}^{M} I(h_m(x) = y)$
```

Algorithm 6. Batch SMOTEBagging [48]

```
Input: $S$: data set, $M$: number of base learners, $C > 1$: sampling rate, $k$: number of nearest neighbors
1: for $m = 1, \dots, M$ do
2:     $a = \frac{m}{M}$
3:     $S^-_{train} = Sample\_with\_replacement(S^-, N^-)$
4:     $S^+_{train} = Sample\_with\_replacement(S^+, aCN^+)$
5:     $S^+_S = SMOTE(S^+, C(1-a), k)$
6:     $S_m = S^+_{train} + S^-_{train} + S^+_S$
7:     Train a base learner $h_m \rightarrow Y$ using $S_m$
8: end for
Output: $H(x) = \arg\max_{y \in Y} \sum_{m=1}^{M} I(h_m(x) = y)$
```

In [48], the sampling rate C for UnderOverBagging and SMOTEBagging was set as $C = \frac{N^-}{N^+}$ to obtain a balanced class distribution. However, it has been reported that the optimal class ratio need not to be 1. To eliminate this effect, we vary the resampling rate to generate an ROC to compare different online and batch cost-sensitive algorithms.

AdaC2[3] [43] is a boosting algorithm that takes the different misclassification costs into consideration when calculating the classifier weights, and updates the sample weights accordingly. In particular, AdaC2 increases weight more on the misclassified positive examples than the misclassified negative examples. Similarly, it decreases weight less on correctly classified positive examples compared to correctly classified negative examples. The pseudo-code for AdaC2 is shown in Algorithm 8, where C_P is the cost of misclassifying a positive example as a negative one, and C_N is the cost of the contrary case. In cost-sensitive learning problems, as $C_P > C_N$, it can be observed that by embedding different costs for each class into the update formula, AdaC2 puts more weight on positive examples than negative ones.

3. The meaning of AdaC2 and CSB2 is not specified in [43] and [44], but we guess the acronyms stand for *AdaBoost for cost-sensitive learning* and *cost-sensitive boosting* respectively.

Algorithm 7. SMOTE [6]

```
Input: $S$: data set of size $N$, $T$: oversampling rate, $k$: number of nearest neighbors
1: for $n = 1, \dots, N$ do
2:     for $i = 1, \dots, T$ do
3:         Randomly choose one of the $k$ nearest neighbors of $x_n$, $x'_n$
4:         Calculate the difference between $x_n$ and $x'_n$
5:         Generate a random number $\gamma$ between 0 and 1
6:         Create a synthetic instance:
           $x^i_{S,n} = x_n + \gamma(x'_n - x_n)$
7:     end for
8: end for
Output: $TN$ synthetic instances $\{x^1_{S,1}, \dots, x^T_{S,N}\}$
```

Algorithm 8. Batch AdaC2 [43]

```
Input: $S$: data set, $M$: number of base learners, $C_P$: cost of false negative, $C_N$: cost of false positive
1: Initialize $D_1(n) = \frac{1}{N}$ for all $n \in \{1, \dots, N\}$
2: for $m = 1, \dots, M$ do
3:     Train a base learner $h_m \rightarrow Y$ using $S$ with distribution $D_m$
4:     Calculate $\alpha_m = \frac{1}{2}\log\left(\frac{\sum_{n=1}^{N} C_n D_m(n) I(h_m(x_n)=y_n)}{\sum_{n=1}^{N} C_n D_m(n) I(h_m(x_n)\neq y_n)}\right)$,
       where $C_n = C_P$ if $y = 1$, $C_n = C_N$ otherwise
5:     for $n = 1, \dots, N$ do
6:         $D_{m+1}(n) = \frac{C_n D_m(n)\exp(-\alpha_m(2I(h_m(x_n)=y_n)-1))}{Z_m}$
           where $Z_m$ is a normalization factor
7:     end for
8: end for
Output: $H(x) = \arg\max_{y \in Y} \sum_{m=1}^{M} \alpha_m I(h_m(x) = y)$
```

Algorithm 9. Batch CSB2 [44]

```
Input: $S$: data set, $M$: number of base learners, $C_P$: cost of false negative, $C_N$: cost of false positive
1: Initialize $D_1(n) = \frac{1}{N}$ for all $n \in \{1, \dots, N\}$
2: for $m = 1, \dots, M$ do
3:     Train a base learner $h_m \rightarrow Y$ using $S$ with distribution $D_m$
4:     Calculate $\alpha_m = \frac{1}{2}\log\left(\frac{\sum_{n,h_m(x_n)\neq y_n} D_m(n)}{\sum_{n,h_m(x_n)=y_n} D_m(n)}\right)$,
5:     for $n = 1, \dots, N$ do
6:         $D_{m+1}(n) = \frac{C_\delta D_m(n)\exp(-\alpha_m(2I(h_m(x_n)=y_n)-1))}{Z_m}$
           where $Z_m$ is a normalization factor,
           $C_\delta = 1$ if $h_m(x_n) = y_n$, $C_\delta = C_n$ otherwise
7:     end for
8: end for
Output: $H(x) = \arg\max_{y \in Y} \sum_{m=1}^{M} \alpha_m I(h_m(x) = y)$
```

The CSB2 algorithm [44] (Algorithm 9) combines techniques from AdaBoost and AdaC2. For correctly classified examples, CSB2 updates weights as in AdaBoost, whereas for misclassified examples, it updates weights as in AdaC2. In addition, the voting weight of each base learner in CSB2 is the same as in AdaBoost. AdaC2 and CSB2 incorporate different misclassification costs by directly modifying the weight update equations. The key feature of this category of algorithms is that the cost sensitivity is introduced by treating the examples from different classes differently. Therefore, such approaches are categorized as cost-sensitive boosting [18].

Algorithm 10. Batch RUSBoost [41]

```
Input: $S$: data set, $M$: number of base learners, $C > 1$: sampling rate
1: Initialize $D_1(n) = \frac{1}{N}$ for all $n \in \{1, \dots, N\}$
2: for $m = 1, \dots, M$ do
3:     Generate a new training set $S'$ by undersampling the majority class
4:     Train a base learner $h_m \to Y$ using $S'$
5:     $\epsilon_m = \sum_{n=1}^{N} D_m(n) I(h_m(x_n) \neq y_n)$
6:     for $n = 1, \dots, N$ do
7:         $D_{m+1}(n) = D_m(n) \times \begin{cases} \frac{1}{2(1-\epsilon_m)}, & h_m(x_n) = y_n \\ \frac{1}{2\epsilon_m}, & h_m(x_n) \neq y_n \end{cases}$
8:     end for
9: end for
Output: $H(x) = \arg\max_{y \in Y} \sum_{m=1}^{M} \log(\frac{1-\epsilon_m}{\epsilon_m}) I(h_m(x) = y)$
```

RUSBoost1 (fix the class ratio)
Generate a new training set S' by undersampling the majority class:
Randomly remove majority class examples until CN^+ of them left, and then renormalize the weight distribution of the remaining training examples with respect to their total sum of weights.

RUSBoost2 (fix the example distribution)
Generate a new training set S' by undersampling the majority class:
Generate CN^+ and N^+ majority and minority examples respectively by sampling D_m.

RUSBoost3 (fix the sampling rate)
Generate a new training set S' by undersampling the majority class:
Generate N examples according to distribution D_m, and then undersample majority class until $\frac{1}{C}$ of them left.

Cost sensitivity in boosting can also be achieved by sampling techniques as a pre-processing step before each iteration of the standard AdaBoost algorithm, as in UnderOverBagging and SMOTEBagging, to bias boosting algorithms towards the positive class, resulting in RUSBoost [41] and SMOTEBoost [7].[4] These two algorithms differ in the way they undersample the negative class, or oversample the positive class using SMOTE. In particular, both algorithms rebalance the class distribution before feeding the training data to base learners, yet the error estimate of base learners is still measured on the original dataset. The RUSBoost and SMOTEBoost algorithms are shown in Algorithms 10 and 11 respectively.[5] For each algorithm, there are at least three different implementations. In RUSBoost1 (RUS1), the class ratio is fixed, which means that the ratio of the weighted positive to negative examples is fixed, and then the modified training set is generated according to the weights. The proportion of examples of the two classes for each base learner is not necessarily fixed. In RUSBoost2 (RUS2) it is the ratio of unweighted examples of the classes that is fixed, which means that the proportion of examples of each class that passes through each base learner is fixed. RUSBoost3 (RUS3) is easier to understand: the sampling rate of the negative class is $\frac{1}{C}$ times that of positive class. The three implementations of SMOTEBoost (SBO1–SBO3) are analogous to the ones of RUSBoost. The only difference is that instead of undersampling the majority class, SMOTEBoost variants oversample the minority class by generating synthetic positive examples using SMOTE.

4. AdC2, CSB2, and RUSBoost, SMOTEBoost were respectively categorized into different families in [18] Here, we do not distinguish them since all of these methods can be formulated within the framework of combining resampling techniques and conventional AdaBoost, and we simply refer to all of them as cost-sensitive boosting algorithms.
5. RUSBoost and SMOTEBoost are originally based on AdaBoost.M2, which is a variant of AdaBoost for multi-class classification problems. As for binary classification problems, AdaBoost.M2 with a base learner outputting hard labels is equivalent to AdaBoost. Therefore, in order to develop online RUSBoost and SMOTEBoost, both of them are based on AdaBoost in this paper.

Algorithm 11. Batch SMOTEBoost [7]

```
Input: $S$: data set, $M$: number of base learners, $C > 1$: sampling rate
1: Initialize $D_1(n) = \frac{1}{N}$ for all $n \in \{1, \dots, N\}$
2: for $m = 1, \dots, M$ do
3:     Generate a new training set $S'$ by creating synthetic examples from minority class using SMOTE
4:     Train a base learner $h_m \to Y$ using $S'$
5:     $\epsilon_m = \sum_{n=1}^{N} D_m(n) I(h_m(x_n) \neq y_n)$
6:     for $n = 1, \dots, N$ do
7:         $D_{m+1}(n) = D_m(n) \times \begin{cases} \frac{1}{2(1-\epsilon_m)}, & h_m(x_n) = y_n \\ \frac{1}{2\epsilon_m}, & h_m(x_n) \neq y_n \end{cases}$
8:     end for
9: end for
Output: $H(x) = \arg\max_{y \in Y} \sum_{m=1}^{M} \log(\frac{1-\epsilon_m}{\epsilon_m}) I(h_m(x) = y)$
```

SMOTEBoost1 (fix the class ratio)
Generate a new training set S' by creating synthetic examples from minority class using SMOTE:
Randomly generate $\frac{N^-}{C} - N^+$ synthetic examples from minority class, and then renormalize the weight distribution with respect to their total sum of weights.

SMOTEBoost2 (fix the example distribution)
Generate a new training set S' by creating synthetic examples from minority class using SMOTE
Generate N^- and N^+ majority and minority examples respectively by sampling D_m, and then create $\frac{N^-}{C} - N^+$ synthetic minority examples

SMOTEBoost3 (fix the sampling rate)
Generate a new training set S' by creating synthetic examples from minority class using SMOTE:
Generate N examples according to distribution D_m, create $(C-1)N'^+$ synthetic minority examples, where N'^+ is the number of minority examples on the mth iteration.

4 Methods

We now derive a new collection of algorithms for online cost-sensitive ensemble learning based on the combination of online ensemble methods and batch cost-sensitive ensembles described above. The main challenge for generalizing cost-sensitive ensemble algorithms to the online setting resides in finding a way to embed costs into online ensembles for boosting algorithms without seeing all of the data. Thus a key point for our online algorithms is to reformulate the batch cost-sensitive boosting algorithms in a way that there is no normalization step at each iteration, and then to incrementally estimate the quantities embedded with the cost setting in the online learning scenario. Whereas cost sensitivity in the batch setting is achieved by different resampling mechanisms, in the online ensembles it is achieved by manipulating the parameters of the Poisson distribution for different classes. In this section, we define online extensions of all of the popular methods described in the previous section. Note that the proposed framework can also be similarly applied to other batch cost-sensitive ensembles, such as RBBagging [22], CSRoulette [42], RareBoost [25], or DataBoost-IM [20], and so on.

4.1 Online Cost-Sensitive Bagging

We begin with the derivation of online UnderOverBagging. Noting that the Poisson distribution parameter $\lambda = 1$ corresponds to sampling with probability $\frac{1}{N}$, we can simulate sampling a given example with probability $\frac{C}{N}$ by instead presenting this example to the base model $k \sim Poisson(C)$ times and performing online bagging. This yields the simple algorithm shown in Algorithm 12.

Algorithm 12. Online UnderOverBagging

```
Input: $S$: data set, $M$: number of base learners, $C > 1$: sampling rate
1: for $n = 1, \ldots, N$ do
2:   for $m = 1, \ldots, M$ do
3:     $a = \frac{m}{M}$
4:     If $y_n = 1$, $\lambda = aC$, else $\lambda = a$
5:     Let $k \sim Poisson(\lambda)$
6:     Do $k$ times
         Train the base learner $h_m \rightarrow Y$ using $(x_n, y_n)$
7:   end for
8: end for
Output: $H(x) = \arg\max_{y \in Y} \sum_{m=1}^{M} I(h_m(x) = y)$
```

For the online version of SMOTEBagging, we propose a similar generalization, shown in Algorithm 13, together with the online version of the original SMOTE in Algorithm 14. It is worth noting that online SMOTE may not be a good approximation of its batch counterpart, since at the early stages of the learning process, the positive examples are extremely rare, and therefore the generated synthetic examples by online SMOTE can be very different from the ones generated in the batch setting. Nonetheless, we investigated it for completeness, and the empirical results presented below show that it works surprisingly well despite this intrinsic limitation.

4.2 Online Cost-Sensitive Boosting

As stated at the beginning of this section, the key step to implement online boosting is to formulate the weight update rule such that the normalization step can be avoided, as we cannot track the normalization factor online.

To derive such weight update rule for the AdaC2 algorithm, we reformulate step 6 of Algorithm 8 as

$$\begin{aligned}
&D_{m+1}(n)\\
&\quad \propto C_n D_m(n)\exp(-\alpha_m(2I(h_m(x_n) = y_n) - 1))\\
&= C_n D_m(n)\exp(-2\alpha_m I(h_m(x_n) = y_n))\exp(\alpha_m)\\
&\quad \propto C_n D_m(n)\exp(-2\alpha_m I(h_m(x_n) = y_n))\\
&= D_m \times \begin{cases} \frac{C_n \sum_{n=1}^{N} C_n D_m(n) I(h_m(x_n) \neq y_n)}{\sum_{n=1}^{N} C_n D_m(n) I(h_m(x_n) = y_n)}, & h_m(x_n) = y_n \\ C_n, & h_m(x_n) \neq y_n \end{cases}\\
&\propto D_m \times \begin{cases} \frac{C_n}{2\sum_{n=1}^{N} C_n D_m(n) I(h_m(x_n) = y_n)}, & h_m(x_n) = y_n \\ \frac{C_n}{2\sum_{n=1}^{N} C_n D_m(n) I(h_m(x_n) \neq y_n)}, & h_m(x_n) \neq y_n \end{cases}.
\end{aligned}$$

Algorithm 13. Online SMOTEBagging

```
Input: $S$: data set, $M$: number of base learners, $C > 1$: sampling rate
1: $X^+ = \{\}$
2: for $n = 1, \ldots, N$ do
3:   for $m = 1, \ldots, M$ do
4:     $a = \frac{m}{M}$
5:     if $y_n = 1$ then
6:       $X^+ = \{X^+; x_n\}$,
7:       $\lambda = aC, \lambda_{SMOTE} = (1 - a)C$
8:       Let $k \sim Poisson(\lambda)$
9:       Do $k$ times
           Train the base learner $h_m \rightarrow Y$ using $(x_n, y_n)$
10:      Let $k_{SMOTE} \sim Poisson(\lambda_{SMOTE})$
11:      Do $k_{SMOTE}$ times
           $x_S = Online\_SMOTE(X^+)$
           Train the base learner $h_m \rightarrow Y$ using $(x_S, y_n)$
12:    else
13:      Let $k \sim Poisson(\lambda)$
14:      Do $k$ times
           Train the base learner $h_m \rightarrow Y$ using $(x_n, y_n)$
15:    end if
16:  end for
17: end for
Output: $H(x) = \arg\max_{y \in Y} \sum_{m=1}^{M} I(h_m(x) = y)$
```

Algorithm 14. Online SMOTE

```
Input: $X$: collection of positive samples
1: $x = X(end, :)$
2: Randomly choose one of the $k$ nearest neighbors of $x$, $x'$
3: Calculate the difference between $x$ and $x'$
4: Generate a random number $\gamma$ between 0 and 1
5: Create a synthetic instance:
   $x_S = x + \gamma(x' - x)$
Output: a synthetic instance $x_{SMOTE}$
```

Let $werr_m = \sum_{n=1}^{N} C_n D_m(n) I(h_m(x_n) \neq y_n)$ be the weighted error and $wacc = \sum_{n=1}^{N} C_n D_m(n) I(h_m(x_n) = y_n)$ be the weighted accuracy. Then, we can reformulate the weight update step (step 6) of AdaC2 as

$$D_{m+1}(n) = D_m \times \begin{cases} h_m(x_n) = y_n \begin{cases} y_n = 1, \frac{C_P}{2wacc_m} \\ y_n = 0, \frac{C_N}{2wacc_m} \end{cases} \\ h_m(x_n) \neq y_n \begin{cases} y_n = 1, \frac{C_P}{2werr_m} \\ y_n = 0, \frac{C_N}{2werr_m} \end{cases} \end{cases}.$$

One can check that it is equivalent to step 6 of Algorithm 8 but without the normalization factor, and the sum of the weights after the update is still one. Therefore, to implement online AdaC2, we only need to track $werr$ and $wacc$ to update the Poisson parameter λ for each base learner. In particular, as AdaC2 treats differently true positive, true negative, false positive and false negative examples, we need four parameters $(\lambda_m^{TP}, \lambda_m^{TN}, \lambda_m^{FP}, \lambda_m^{FN})$ to track the sum of weighted Poisson distribution parameter λ of each category of the examples for the mth base model respectively. Then $werr$ and $wacc$ can be calculated by $\frac{\lambda_m^{FP} + \lambda_m^{FN}}{\lambda_m^{SUM}}$ and $\frac{\lambda_m^{TP} + \lambda_m^{TN}}{\lambda_m^{SUM}}$ respectively, where λ_m^{SUM} is the sum of total Poisson distribution parameter for the mth base learner. The pseudo-code of online AdaC2 is shown in Algorithm 15.

Algorithm 15. Online AdaC2

```
Input: $S$: data set, $M$: number of base learners, $C_P$: cost of false negative, $C_N$: cost of false positive
 1: Initialize $\lambda_m^{TP} = 0, \lambda_m^{TN} = 0, \lambda_m^{FP} = 0, \lambda_m^{FN} = 0,\ \lambda_m^{SUM} = 0$ for all $m \in \{1, \ldots, M\}$
 2: for $n = 1, \ldots, N$ do
 3:   Set $\lambda = 1$
 4:   for $m = 1, \ldots, M$ do
 5:     $\lambda_m^{SUM} = \lambda_m^{SUM} + \lambda$
 6:     Let $k \sim Poisson(\lambda)$
 7:     Do $k$ times
 8:       Train the base learner $h_m \to Y$ using $(x_n, y_n)$
 9:     if $h_m(x_n) = 1 \&\& y_n = 1$ then
10:       $\lambda_m^{TP} \leftarrow \lambda_m^{TP} + C_P\lambda$,
          $wacc_m \leftarrow \frac{\lambda_m^{TP}+\lambda_m^{TN}}{\lambda_m^{SUM}}, werr_m \leftarrow \frac{\lambda_m^{FP}+\lambda_m^{FN}}{\lambda_m^{SUM}}$,
          $\lambda \leftarrow \frac{C_P\lambda}{2wacc_m}$
11:     else if $h_m(x_n) = 0 \&\& y_n = 0$ then
12:       $\lambda_m^{TN} \leftarrow \lambda_m^{TN} + C_N\lambda$,
          $wacc_m \leftarrow \frac{\lambda_m^{TP}+\lambda_m^{TN}}{\lambda_m^{SUM}}, werr_m \leftarrow \frac{\lambda_m^{FP}+\lambda_m^{FN}}{\lambda_m^{SUM}}$,
          $\lambda \leftarrow \frac{C_N\lambda}{2wacc_m}$
13:     else if $h_m(x_n) = 0 \&\& y_n = 1$ then
14:       $\lambda_m^{FN} \leftarrow \lambda_m^{FN} + C_P\lambda$,
          $wacc_m \leftarrow \frac{\lambda_m^{TP}+\lambda_m^{TN}}{\lambda_m^{SUM}}, werr_m \leftarrow \frac{\lambda_m^{FP}+\lambda_m^{FN}}{\lambda_m^{SUM}}$,
          $\lambda \leftarrow \frac{C_P\lambda}{2werr_m}$
15:     else if $h_m(x_n) = 1 \&\& y_n = 0$ then
16:       $\lambda_m^{FP} \leftarrow \lambda_m^{FP} + C_N\lambda$,
          $wacc_m \leftarrow \frac{\lambda_m^{TP}+\lambda_m^{TN}}{\lambda_m^{SUM}}, werr_m \leftarrow \frac{\lambda_m^{FP}+\lambda_m^{FN}}{\lambda_m^{SUM}}$,
          $\lambda \leftarrow \frac{C_N\lambda}{2werr_m}$
17:     end if
18:   end for
19: end for
Output: $H(x) = \arg\max_{y\in Y} \sum_{m=1}^{M} \log\left(\frac{wacc_m}{werr_m}\right) I(h_m(x) = y)$
```

By similar calculations, an update formula for CSB2 without normalization can be formulated as

$$D_{m+1}(n) = D_m \times \begin{cases} h_m(x_n) = y_n & \frac{\epsilon_m}{(1-\epsilon_m)(\epsilon_m + werr_m)} \\ h_m(x_n) \neq y_n \begin{cases} y_n = 1, & \frac{C_P}{\epsilon_m + werr_m} \\ y_n = 0, & \frac{C_N}{\epsilon_m + werr_m} \end{cases} \end{cases}.$$

Then, the online CSB2 can be implemented by tracking the weighted error and unweighted error, which is achieved by tracking $\lambda_m^{FP}, \lambda_m^{FN}$, λ_m^{SW} and λ_m^{SUM} respectively. The pseudo-code of online CSB2 is shown in Algorithm 16.

Moving on to RUSBoost (Algorithm 10), we note that it is the same as the standard boosting algorithm except that there is a pre-processing step at the beginning of every iteration to rebalance the example distribution. As a result, each example should be reweighted according to different undersampling strategies. In RUSBoost1, as all negative examples are first undersampled at a rate of $\frac{CN^+}{N^-}$, the expected total weights of the remaining examples are $D_m^+ + \frac{CN^+}{N^-} D_m^-$, where $D_m^+ = \sum_{n=1}^{N^+} D_m^+(n)$ is the sum of the weights of positive examples, and $D_m^- = \sum_{n=1}^{N^-} D_m^-(n)$ is the sum of negative examples. Since the size of training examples is $(C+1)N^+$ after undersampling, we set the new weight of each example to:

$$D'_m(n) = D_m(n) \times \begin{cases} \frac{1}{D_m^+ + \frac{CN^+}{N^-} D_m^-} \frac{(C+1)N^+}{N^+ + N^-}, & y_n = 1 \\ \frac{CN^+}{N^-} \frac{1}{D_m^+ + \frac{CN^+}{N^-} D_m^-} \frac{(C+1)N^+}{N^+ + N^-}, & y_n = 0 \end{cases}.$$

To approximate these quantities in online RUSBoost1, we need to record the total weights of positive examples and negative examples, as well as their counts. This can be done by introducing four parameters: λ_m^{POS}, λ_m^{NEG}, n^+, and n^-. Here, N^+ and N^- are approximated by n^+, and n^-. Since $D_m(n)$ in batch boosting (Algorithm 2) corresponds to the parameter λ in online boosting (Algorithm 4), D_m^+ and D_m^- can be tracked by $\frac{\lambda_m^{POS}}{\lambda_m^{POS} + \lambda_m^{NEG}}$ and $\frac{\lambda_m^{NEG}}{\lambda_m^{POS} + \lambda_m^{NEG}}$ respectively. The derivation of online RUSBoost2 follows a similar approach. In RUSBoost2, the number of examples of each class is fixed regardless of the total weight of each class. Therefore, the positive (negative) examples will be undersampled if their total weight is greater than $\frac{N^+}{N^+ + N^-}$ ($\frac{CN^+}{N^+ + N^-}$), and will be oversampled otherwise. Therefore, each example is reweighted by

$$D'_m(n) = D_m(n) \times \begin{cases} \frac{N^+}{N^+ + N^-} / D_m^+, & y_n = 1 \\ \frac{CN^+}{N^+ + N^-} / D_m^-, & y_n = 0. \end{cases}$$

The implementation of online RUSBoost3 is straightforward, positive examples are generated as in standard online boosting. For a negative example, let $\lambda^{RUS} = \frac{\lambda}{C}$, and use this example $k \sim Poisson(\lambda^{RUS})$ times to update a base learner. The pseudo-code of these three online RUSBoost algorithms is included in Algorithm 17.

Algorithm 16. Online CSB2

```
Input: $S$: data set, $M$: number of base learners, $C_P$: cost of false negative, $C_N$: cost of false positive
 1: Initialize $\lambda_m^{FP} = 0, \lambda_m^{FN} = 0, \lambda_m^{SW} = 0, \lambda_m^{SUM} = 0$ for all $m \in \{1, \ldots, M\}$
 2: for $n = 1, \ldots, N$ do
 3:   Set $\lambda = 1$
 4:   for $m = 1, \ldots, M$ do
 5:     $\lambda_m^{SUM} = \lambda_m^{SUM} + \lambda$
 6:     Let $k \sim Poisson(\lambda)$
 7:     Do $k$ times
 8:       Train the base learner $h_m \to Y$ using $(x_n, y_n)$
 9:     if $h_m(x_n) = y_n$ then
10:       $\epsilon_m \leftarrow \frac{\lambda_m^{SW}}{\lambda_m^{SUM}}, werr_m \leftarrow \frac{\lambda_m^{FP}+\lambda_m^{FN}}{\lambda_m^{SUM}}$,
          $\lambda \leftarrow \frac{\epsilon_m}{(1-\epsilon_m)(\epsilon_m+werr_m)}$
11:     else
12:       if $h_m(x_n) = 0 \&\& y_n = 1$ then
13:         $\lambda_m^{FN} \leftarrow \lambda_m^{FN} + C_P\lambda, \lambda_m^{SW} \leftarrow \lambda_m^{SW} + \lambda$,
            $\epsilon_m \leftarrow \frac{\lambda_m^{SW}}{\lambda_m^{SUM}}, werr_m \leftarrow \frac{\lambda_m^{FP}+\lambda_m^{FN}}{\lambda_m^{SUM}}$,
            $\lambda \leftarrow \frac{C_P\lambda}{\epsilon_m+werr_m}$
14:       else
15:         $\lambda_m^{FP} \leftarrow \lambda_m^{FP} + C_P\lambda, \lambda_m^{SW} \leftarrow \lambda_m^{SW} + \lambda$,
            $\epsilon_m \leftarrow \frac{\lambda_m^{SW}}{\lambda_m^{SUM}}, werr_m \leftarrow \frac{\lambda_m^{FP}+\lambda_m^{FN}}{\lambda_m^{SUM}}$,
            $\lambda \leftarrow \frac{C_N\lambda}{\epsilon_m+werr_m}$
16:       end if
17:     end if
18:   end for
19: end for
Output: $H(x) = \arg\max_{y\in Y} \sum_{m=1}^{M} \log\left(\frac{1-\epsilon_m}{\epsilon_m}\right) I(h_m(x) = y)$
```

Algorithm 17. Online RUSBoost

```
Input: $S$: data set, $M$: number of base learners, $C > 1$: sampling rate
 1: Initialize $\lambda_m^{SC} = 0, \lambda_m^{SW} = 0, \lambda_m^{POS} = 0, \lambda_m^{NEG} = 0$, for all $m \in \{1, \dots, M\}, n^+ = 0, n^- = 0$
 2: for $n = 1, \dots, N$ do
 3:   Set $\lambda = 1$
 4:   for $m = 1, \dots, M$ do
 5:     if $y_n = 1$ then
 6:       $\lambda_m^{POS} \leftarrow \lambda_m^{POS} + \lambda, n^+ \leftarrow n^+ + 1,$
 7:     else
 8:       $\lambda_m^{NEG} \leftarrow \lambda_m^{NEG} + \lambda, n^- \leftarrow n^- + 1,$
 9:     end if
10:     Compute $\lambda^{RUS}$ (step 21 – 23)
11:     Let $k \sim Poisson(\lambda^{RUS})$
12:     Do $k$ times
13:       Train the base learner $h_m \rightarrow Y$ using $(x_n, y_n)$
14:     if $h_m(x_n) = y_n$ then
15:       $\lambda_m^{SC} \leftarrow \lambda_m^{SC} + \lambda, \epsilon_m \leftarrow \frac{\lambda_m^{SW}}{\lambda_m^{SC} + \lambda_m^{SW}}, \lambda \leftarrow \frac{\lambda}{2(1-\epsilon_m)}$
16:     else
17:       $\lambda_m^{SW} \leftarrow \lambda_m^{SW} + \lambda, \epsilon_m \leftarrow \frac{\lambda_m^{SW}}{\lambda_m^{SC} + \lambda_m^{SW}}, \lambda \leftarrow \frac{\lambda}{2\epsilon_m}$
18:     end if
19:   end for
20: end for
Output: $H(x) = \arg\max_{y \in Y} \sum_{m=1}^{M} \log(\frac{1-\epsilon_m}{\epsilon_m}) I(h_m(x) = y)$
```

online RUSBoost1

21: Compute λ^{RUS}: $\begin{cases} \lambda^{RUS} \leftarrow \lambda \frac{\lambda_m^{POS} + \lambda_m^{NEG}}{\lambda_m^{POS} + \lambda_m^{NEG} \frac{Cn^+}{n^-}} \frac{(C+1)n^+}{n^+ + n^-}, & y_n = 1 \\ \lambda^{RUS} \leftarrow \lambda \frac{\lambda_m^{POS} + \lambda_m^{NEG}}{\lambda_m^{NEG} + \lambda_m^{POS} \frac{n^-}{Cn^+}} \frac{(C+1)n^+}{n^+ + n^-}, & y_n = 0 \end{cases}$

online RUSBoost2

22: Compute λ^{RUS}: $\begin{cases} \lambda^{RUS} \leftarrow \lambda \frac{n^+}{n^+ + n^-} / \frac{\lambda^{POS}}{\lambda^{POS} + \lambda^{NEG}}, & y_n = 1 \\ \lambda^{RUS} \leftarrow \lambda \frac{Cn^+}{n^+ + n^-} / \frac{\lambda^{NEG}}{\lambda^{POS} + \lambda^{NEG}}, & y_n = 0 \end{cases}$

online RUSBoost3

23: Compute λ^{RUS}: $\begin{cases} \lambda^{RUS} \leftarrow \lambda, & y_n = 1 \\ \lambda^{RUS} \leftarrow \frac{\lambda}{C}, & y_n = 0 \end{cases}$

In online SMOTEBoost (Algorithm 18) we similarly keep track of the number of positive and negative examples, n^+, and n^-, a parameter λ' controlling the re-use frequency of each new sample during training, and an additional parameter λ^{SMOTE} controlling the number of generated synthetic positive examples. Three variants are proposed, as online generalizations of the three cases in Algorithm 11. In online SMOTEBoost1, the parameter λ' is analogous to λ in standard online boosting, and synthetic examples are generated by uniformly sampling from positive examples and applying an online version of SMOTE. In online SMOTEBoost2, the derivation of λ' is the same as λ^{RUS} in online RUSBoost2. Since SMOTE is applied after sampling the original dataset, the probability of a positive example being chosen to generate synthetic examples is proportional to its weight. Therefore, λ^{SMOTE} is given by $\lambda'(\frac{N^-}{CN^+} - 1)$. Finally, online SMOTEBoost3 is analogous to standard online boosting, except for the additional parameter $\lambda^{SMOTE} = \lambda(C - 1)$.

4.3 A Unified Perspective

The proposed online cost-sensitive ensemble algorithms and their batch mode counterparts can be characterized from the perspective of data sampling, and the approach to dealing with the class imbalanced problem, as summarized in Table 1. More specifically, in all batch learning algorithms, data are sampled according to their weights (distribution), while in online learning algorithms, each instance is sampled from a Poisson distribution with parameter λ. The key point is that as the size of dataset approaches infinity, sampling from a uniform is equivalent to sampling from a Poisson distribution with $\lambda = 1$. Therefore, updating the weights in batch algorithms can be approximated by properly updating the Poisson parameter in online algorithms.

Algorithm 18. Online SMOTEBoost

```
Input: $S$: data set, $M$: number of base learners, $C > 1$: sampling rate
 1: Initialize $\lambda_m^{SC} = 0, \lambda_m^{SW} = 0$, for all $m \in \{1, \dots, M\}$, $n^+ = 0$, $n^- = 0, X^+ = \{\}$
 2: for $n = 1, \dots, N$ do
 3:   Set $\lambda = 1$
 4:   for $m = 1, \dots, M$ do
 5:     if $y_n = 1$ then
 6:       $n^+ \leftarrow n^+ + 1, \lambda_m^{POS} \leftarrow \lambda_m^{POS} + \lambda,$
          $X^+ = \{X^+; x_n\}$
 7:     else
 8:       $n^- \leftarrow n^- + 1, \lambda_m^{NEG} \leftarrow \lambda_m^{NEG} + \lambda,$
 9:     end if
10:     Compute $\lambda'$ (step 26, 28, 30)
11:     $k \sim Poisson(\lambda')$
12:     Do $k$ times
13:       Train the base learner $h_m \rightarrow Y$ using $(x_n, y_n)$
14:     Compute $\lambda^{SMOTE}$ (step 27, 29, 31)
15:     $k^{SMOTE} \sim Poisson(\lambda^{SMOTE})$
16:     Do $k^{SMOTE}$ times
17:       $x_S = online\_SMOTE(X^+)$
18:       Train the base learner $h_m \rightarrow Y$ using $(x_S, y_n)$
19:     if $h_m(x_n) = y_n$ then
20:       $\lambda_m^{SC} \leftarrow \lambda_m^{SC} + \lambda, \epsilon_m \leftarrow \frac{\lambda_m^{SW}}{\lambda_m^{SC} + \lambda_m^{SW}}, \lambda \leftarrow \frac{\lambda}{2(1-\epsilon_m)}$
21:     else
22:       $\lambda_m^{SW} \leftarrow \lambda_m^{SW} + \lambda, \epsilon_m \leftarrow \frac{\lambda_m^{SW}}{\lambda_m^{SC} + \lambda_m^{SW}}, \lambda \leftarrow \frac{\lambda}{2\epsilon_m}$
23:     end if
24:   end for
25: end for
Output: $H(x) = \arg\max_{y \in Y} \sum_{m=1}^{M} \log(\frac{1-\epsilon_m}{\epsilon_m}) I(h_m(x) = y)$
```

online SMOTEBoost1

26: Compute λ': $\lambda' \leftarrow \lambda$

27: Compute λ^{SMOTE}: $\begin{cases} \lambda^{SMOTE} \leftarrow \frac{n^-}{Cn^+} - 1, & y_n = 1 \\ \lambda^{SMOTE} \leftarrow 0, & y_n = 0 \end{cases}$

online SMOTEBoost2

28: Compute λ': $\begin{cases} \lambda' \leftarrow \lambda \frac{n^+}{n^+ + n^-} / \frac{\lambda^{POS}}{\lambda^{POS} + \lambda^{NEG}}, & y_n = 1 \\ \lambda' \leftarrow \lambda \frac{n^-}{n^+ + n^-} / \frac{\lambda^{NEG}}{\lambda^{POS} + \lambda^{NEG}}, & y_n = 0 \end{cases}$

29: Compute λ^{SMOTE}: $\begin{cases} \lambda^{SMOTE} \leftarrow \lambda'(\frac{n^-}{Cn^+} - 1), & y_n = 1 \\ \lambda^{SMOTE} \leftarrow 0, & y_n = 0 \end{cases}$

online SMOTEBoost3

30: Compute λ': $\lambda' \leftarrow \lambda$

31: Compute λ^{SMOTE}: $\begin{cases} \lambda^{SMOTE} \leftarrow (C-1)\lambda, & y_n = 1 \\ \lambda^{SMOTE} \leftarrow 0, & y_n = 0 \end{cases}$

TABLE 1
A Unified Perspective of Cost-Sensitive Ensemble

	Quantity used for data sampling	
	Weight $D(n)$	Poisson parameter λ
Batch Ensemble	✓	✘
Online Ensemble	✘	✓

	Approaches to dealing with class imbalance problem			
	Different Costs	Undersampling	Oversampling	SMOTE
UnderOverBagging	✘	✓	✓	✘
SMOTEBagging	✘	✘	✓	✓
AdaC2	✓	✘	✘	✘
CSB2	✓	✘	✘	✘
RUSBoost	✘	✓	✘	✘
SMOTEBoost	✘	✘	✓	✓

To deal with the class imbalance problem, AdaC2 and CSB2 impose different costs of misclassification on positive and negative classes, and then minimize overall cost instead of classification error. In such a way, examples from different classes are given different weight update rules at each boosting iteration. SMOTEBagging and SMOTEBoost add an additional oversampling step over the positive class based on SMOTE at each bagging/boosting iterations, while RUSBoost undersamples the negative class before feeding the examples to a base learner. Finally, UnderOverBagging smoothly switches from undersampling over negative class to oversampling positive class (without SMOTE) over the iterations to rebalance the class distribution.

4.4 Computational Complexity

At each time step, sampling from a Poisson distribution and updating the parameters (e.g., λ) takes $\mathcal{O}(1)$. If we assume that the complexity of updating a base learner and making a prediction are $\mathcal{O}(\xi(d))$ and $\mathcal{O}(\varsigma(d))$ respectively, where d is the feature dimension, then, the overall complexity of non-SMOTE based online ensemble algorithms at each time step with M base learners is $\mathcal{O}(M(\varsigma(d)+\xi(d)))$.[6] In other words, compared with a single online base learner algorithm, the complexity of ensemble approaches only increases linearly with the number of base learners. Compared with online cost-insensitive ensemble algorithms, there is almost no extra cost to pay, since the cost of extra parameter update steps is negligible. For SMOTE-based online ensemble algorithms, the online SMOTE algorithm takes $\mathcal{O}(kdn^+)$ to generate k synthetic examples, where n^+ is the number of positive examples stored in memory by the time step n. In addition, it also requires $\mathcal{O}(dn^+)$ to store these positive examples. Since the complexity grows linearly with n^+, and in the cost-sensitive learning setting, the positive examples are usually rare, the overall complexity is still tractable. Even though n^+ might be large in some applications, we can always control the complexity by only maintaining $\tilde{n}$ ($\tilde{n} \ll n^+$) positive examples in the online learning scenario (e.g., dropping old ones and keeping only the $\tilde{n}$ most recent positive examples).

5 Theoretical Analysis

The asymptotic properties of online bagging and boosting have been studied in [37], showing consistency between the online approaches and the original bagging and boosting. We now extend the analysis for the cost-sensitive case, to show that online UnderOverBagging, AdaC2, CSB2 and RUSBoost converge to the same solution as their batch counterparts under similar conditions as in [36], [37]. We note that online SMOTEBagging and online SMOTEBoost cannot be shown to converge to their batch mode counterparts since there is no guarantee that online SMOTE will converge to batch SMOTE.

Since our theoretical analysis follows quite readily from the work by [37], we give here only the statements of the main theorems. The proofs are available in the supplementary materials, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TKDE.2016.2609424.

Theorem 1. *As $N^+ \to \infty$, $N^- \to \infty$ and $M \to \infty$, if the base learning algorithms are* proportional,[7] *and converge in probability to some classifier, online UnderOverBagging converges to its batch mode counterpart.*

Theorem 2. *As $N^+ \to \infty$ and $N^- \to \infty$, if the base learners are naive Bayes classifiers, online AdaC2, CSB2 and RUSBoost algorithms converge to their batch mode counterparts.*

6 Experiments

In this section, the proposed online learning algorithms are compared with their batch counterparts using benchmark datasets. As the data distributions are highly imbalanced, the performance of each algorithm is measured by its AUC value (computed using five-fold cross-validation), a widely used metric for imbalanced data distribution, rather than classification accuracy. Since the performances of different batch ensemble algorithms have been thoroughly compared [18], [26], [41], [43], this paper mainly focuses on the performances of online algorithms, and the comparison between the two different types of learning scenarios. We aim to answer the following questions:

1. Which online cost-sensitive ensemble method achieves the best performance?
2. For each online cost-sensitive ensemble algorithm, how close is the performance of the online variant to its batch counterpart?
3. How does the choice of base learners affect the performance and convergence of the online ensemble algorithms?
4. Though the asymptotic properties of some online ensemble algorithms are guaranteed, what can we observe empirically about the convergence speed for the batch versus online algorithms?

6.1 Data Sets and Experimental Setup

Eighteen datasets from the UCI repository[8] with different class ratios were selected to compare the performances of batch and online ensemble learning algorithms. Multiclass datasets were converted to binary class datasets by selecting one class to be the positive and the rest of the classes to be the negative. The detailed description of the datasets can found be in KEEL and UCI dataset websites. Table 2 summarizes the characteristics of the datasets, including the number of examples (#ex), number of features (#fea), the

6. Mostly, updating an algorithm k times can be accomplished by updating it once with a weight k.

7. For the detailed definition of proportional and other related definitions, we refer the reader to [37].

8. Downloaded from KEEL website: http://sci2s.ugr.es/keel/datasets.php

TABLE 2
Summary of Datasets Used in Experiments

dataset	#ex	#fea	%pos	%neg	%neg/%pos
Sonar	208	60	46.63	53.37	1.14
Glass1	214	9	35.51	64.49	1.82
Pima	768	8	34.84	66.16	1.90
Iris0	150	4	33.33	66.67	2.00
Glass0	214	9	32.71	67.29	2.06
Ecoli1	336	7	22.92	77.08	3.36
Ecoli2	336	7	15.48	84.52	5.46
Segment0	2,308	19	14.26	85.74	6.01
Glass6	214	9	13.55	86.45	6.38
Yeast3	1,484	8	10.98	89.02	8.11
Ecoli3	336	7	10.88	89.12	8.19
Satimage	6,435	36	9.73	90.27	9.28
Led7digit	443	7	8.35	91.65	10.97
Ecoli4	336	7	6.74	93.26	13.84
Glass4	214	9	6.07	93.93	15.47
Glass5	214	9	4.20	95.80	22.81
Yeast5	1,484	8	2.96	97.04	32.78
Yeast6	1,484	8	2.49	97.51	39.15

TABLE 3
Parameter Setting

Algorithms	Parameters
ALL	Number of base learners $M = 10$
SMOTE	Number of neighbors $k = 5$
	Distance = Euclidian distance
AC2, CSB2	$C_P = 1$
	$C_N = 0.1$ to 1
UOB, SB,RUS3,SBO3	$C = 1$ to original class ratio
RUS1,RUS2,SBO1,SBO2	$C =$ original class ratio to 1

percentage of positive and negative examples (%pos; %neg), and the class ratio (%neg/%pos).

As the purpose of the experiments is to make fair comparisons between the proposed online algorithms and their batch mode counterparts, the effects of base learners should be reduced to a minimum. In other words, given the same training set, the difference between batch and online algorithms should be due to the inconsistency between ensembles themselves, rather than the base learners. In the theoretical analysis, this requirement is formulated as the *proportional* property, which requires that given the same replicas or proportion of training examples, the base learners returned by batch and online learning algorithms should be the same. From a practical perspective, it is desirable to have base learners that are easy to implement and for which the computation cost of an update step is low. Based on these considerations, three classifiers are considered in the experiments: linear discriminant analysis (LDA), quadratic discriminant analysis (QDA), and naive Bayes (NB).

There are a few hyper-parameters to specify. We use ten base learners per ensemble throughout our experiments, as suggested in [18]. Other parameters specific to each approach are summarized in Table 3, where UOB, SB, AC2 are the abbreviations of UnderOverBagging, SMOTEBagging, AdaC2; RUS1—RUS3 and SBO1—SBO3 are the three variants of RUSBoost and SMOTEBoost respectively.

6.2 Results

First, we show in Fig. 1 the overall performance (average AUC with standard deviation) for each algorithm and each

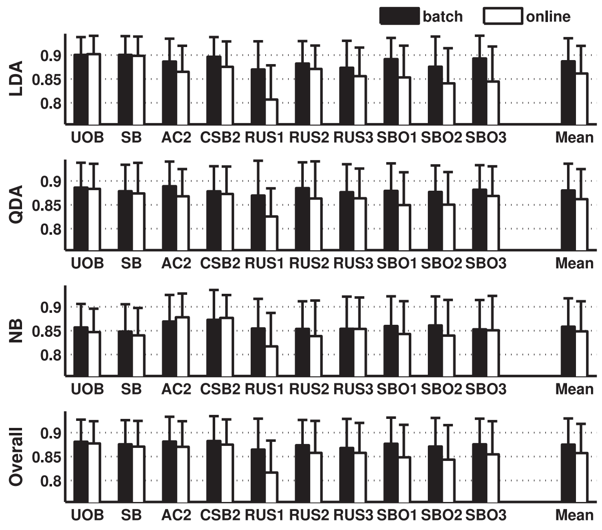


Fig. 1. Overall performances in terms of AUC with standard deviation.

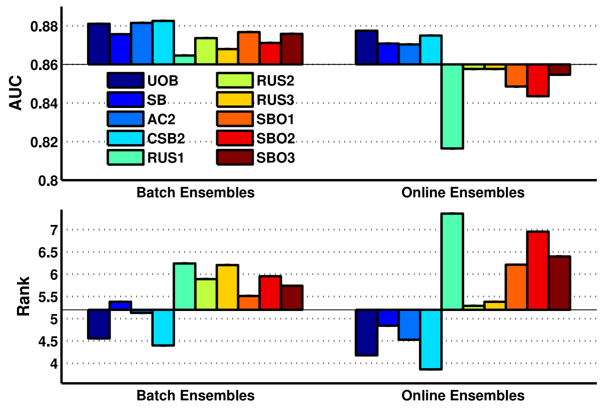


Fig. 2. Overall AUC and rank of batch and online learning algorithms.

base learner. The detailed results are presented in the supplementary materials, available online. Roughly speaking, bagging algorithms achieve better performance than boosting algorithms in terms of consistency. Bagging algorithms, AdaC2, and CSB2 perform better than the RUSBoost and SMOTEBoost algorithms in terms of AUC in both batch and online modes.

6.2.1 *Online Learning Performance*

The overall AUC and rank of batch and online learning algorithms are grouped and shown in Fig. 2. From the left-hand side of the figure, it can be observed that batch bagging algorithms, AdaC2, and CSB2 perform better than the RUSBoost and SMOTEBoost algorithms, though perhaps not significantly in terms of AUC values. However, the difference between them is enhanced in the online learning scenario, as shown in the right-hand side of Fig. 2. Therefore, we attribute the better performances of oUOB,[9] oSB, oAC2, and oCSB2 to two reasons: 1. the performances of their batch mode counterparts; and 2. the consistency the online learning algorithms, as further investigated in the next section.

9. Hereafter, we use *b*- and *o*- as the abbreviations of *batch* and *online*. (e.g., oUOB is the abbreviation of online UnderOverBagging algorithm.)

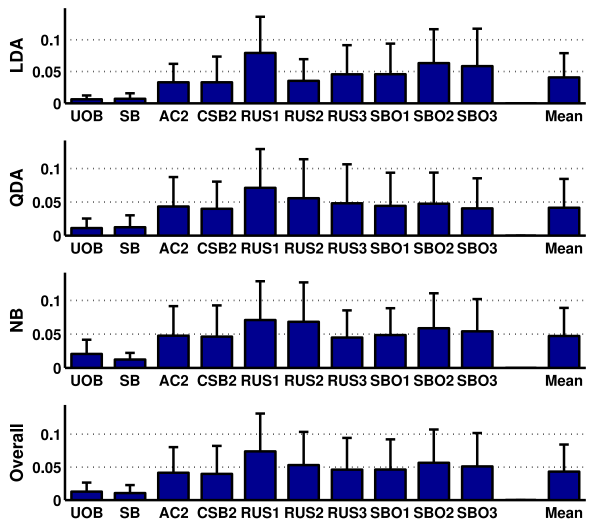


Fig. 3. Bar-plot of the absolute difference of AUC between batch and online ensemble algorithms.

6.2.2 *Consistency Between Batch and Online Algorithms*

Fig. 3 shows the bar-plot of absolute difference (with standard deviation) of AUC between batch and online ensemble algorithms for each algorithm and each base learner. We observe that bagging algorithms have much better consistency than boosting algorithms across all base learners. oAC2 and oCSB2 also achieve better consistency than RUSBoost and SMOTE-Boost algorithms, though not so significantly as oUOB and oSB. We also observe that oRUS1 demonstrates the worst consistency among all algorithms, which explains the poor online learning performances shown in Fig. 2.

Fig. 4 presents the full ROCs of different algorithms with different base learners for a particular dataset (satimage). Here we observe that the ROCs of oUOB and oSB are much closer to their batch counterparts than to any other algorithm. The ROCs of batch and online bagging algorithms are almost overlapped. oAC2 and oCSB2 achieve comparable consistency, as their ROCs of batch and online mode are very close to each other. Overall, while differences between the performances of batch ensembles are not so significant, the performances of the online algorithms are largely determined by the consistency with their batch counterparts.

6.2.3 *The Effect of Base Learners*

From Figs. 1 and 3, we observe that LDA and QDA lead to slightly better performance in terms of average AUC than NB, yet NB has slightly better consistency. Nonetheless, we see that oUOB and oSB perform consistently well for all base learners, while oRUS1 is consistently worse across all base learners. Therefore, there is no strong evidence that the choice of base learner significantly affects the consistency of the online ensemble algorithms.

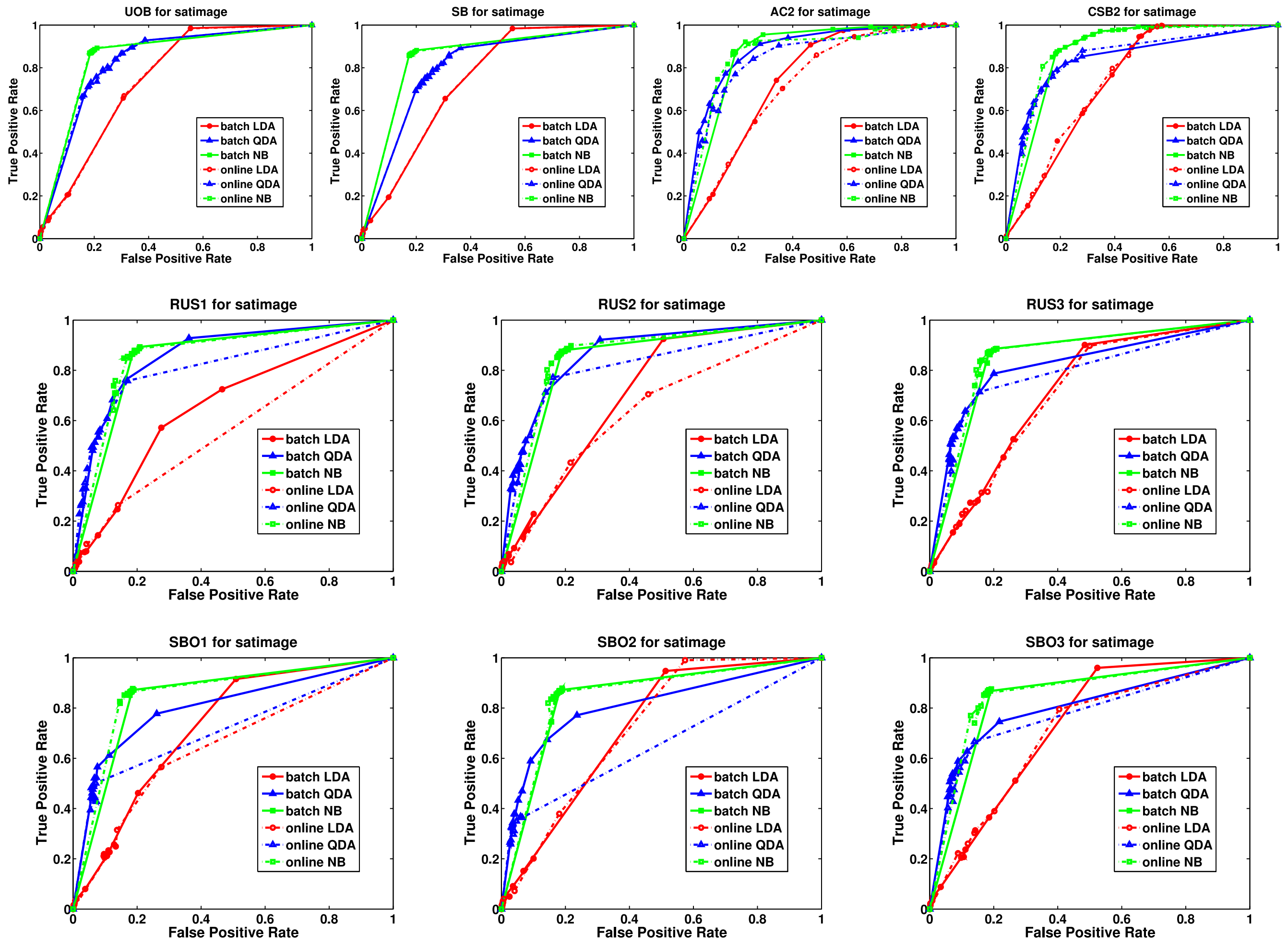


Fig. 4. ROCs of different algorithms with different base learners for the satimage data set.

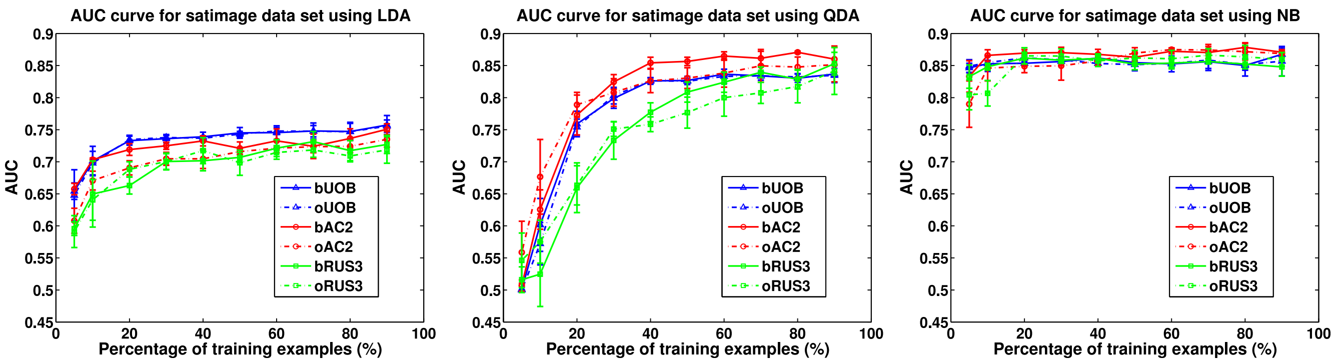


Fig. 5. Learning curves with different base learners for the satimage data set.

6.2.4 Convergence Speed of the Algorithms

The last question is the convergence speed of the proposed algorithms. Because of the better performances of oUOB, oSB, oAC2, and oCSB2 in terms of both consistency and AUC, as well as the similarity between oUOB and oSB, oAC2 and oCSB2, we only include oUOB and oAC2 in these results. We also include oRUS3 for comparison purposes. We vary the number of examples of the satimage data set from 5 to 90 percent used as the training set, and the rest are used as the testing set. The experiment for each algorithm is repeated five times, and the average AUCs (with standard deviation) are shown in Fig. 5. We observe that oUOB converges to bUOB even with a very small amount of training examples. Across all methods, performance improves gradually with addition of more examples, and throughout the range considered, the performance of each online ensemble learner is close to that of its batch counterpart.

6.3 Comparison with State-of-the-Art Algorithms

In this section, we select two representative algorithms, oUOB and oAC2, and compare them with two recently proposed cost-sensitive online learning algorithms (CSOGD-I and CSOGD-II) [47]. For comparison purposes, we follow the same experimental settings and use misclassification cost as a performance metric as in [47], defined as

$$cost = \alpha_p \times N_p + \alpha_n \times N_n,$$

where $\alpha_p + \alpha_n = 1$ and $0 \le \alpha_p, \alpha_n \le 1$ are the misclassification cost parameters for positive and negative classes respectively, and N_p, N_n are the numbers of false negatives and false positives respectively. It is worth noting that CSOGD-I and CSOGD-II are restricted to using perceptron learning, thus this is what we use also as the base learner (oUOB-PER, oAC2-PER). In addition, we also consider NB as the base learner (oUOB-NB, oAC2-NB). To demonstrate the effectiveness of the cost-sensitive algorithms, the performances of a perceptron classifier, as well as standard online bagging and boosting with perceptron as the base learner (oBag-PER and oBoost-PER) are presented as baselines.

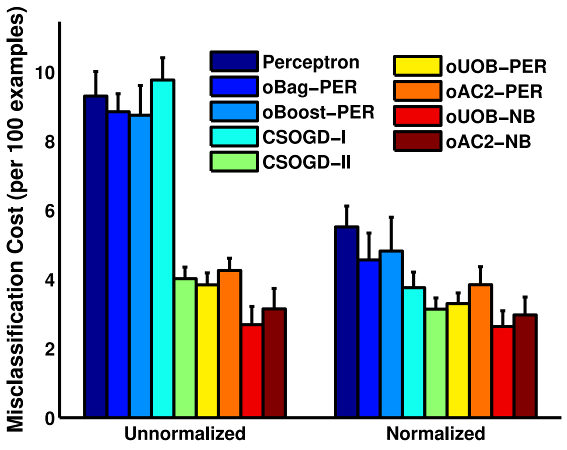


Fig. 6. Average misclassification costs of different algorithms on UCI datasets.

Fig. 6 shows the average costs of different algorithms on 18 UCI datasets, where we have considered two scenarios: with and without normalization as a pre-processing step of learning process. We observe that in both scenarios, the costs of cost-sensitive ensembles are lower than that of cost-insensitive ones. In addition, cost-sensitive ensembles with NB as the base learner outperforms the other algorithms, and CSOGD-II performs slightly better than oAC2-PER. On the other hand, CSOGD-I completely fails without normalization as a pre-processing step. However, in the context of online learning, it is usually unrealistic to perform normalization before the entire learning process, since we usually do not have prior knowledge of the data (e.g., mean, variance).

We also conduct experiments on the larger scale datasets as in [47], and the results are summarized in Table 4. Similarly, the cost-sensitive ensemble learning algorithms and CSOGD-II are insensitive to the normalization of datasets, while CSOGD-I fails on unnormalized datasets. Furthermore, in both scenarios, all the cost-sensitive ensemble learning algorithms outperform CSOGD-I; oUOB-PER and oAC2-NB outperform both CSOGD algorithms. Overall, the cost-sensitive ensemble approaches are insensitive to the scaling of the data and achieve better performance than linear CSOGD algorithms.

7 Conclusions

The paper generalizes a number of batch cost-sensitive ensemble learning algorithms to the online setting, including new online extensions of UnderOverBagging, SMOTEBagging, AdaC2, CSB2, RUSBoost, and SMOTEBoost. We discuss the properties necessary for theoretical convergence of the online variants to the original batch version. We provide extensive experimental results analyzing the performance of the proposed algorithms.

The performances of the online ensembles depend not only on that of their batch counterparts, but also on their consistency. As the batch learning algorithms perform relatively similarly to each other, the online learning performances are greatly affected by consistency. Some previous studies [11],

TABLE 4
Performance Comparison with CSOGD Algorithms on Larger Scale Datasets

Dataset	*cost* (per 100 samples) on different datasets (without normalization)								
	Perceptron	oBag-PER	oBoost-PER	CSOGD-I	CSOGD-II	oUOB-PER	oAC2-PER	oUOB-NB	oAC2-NB
covtype	23.498 ± 0.031	22.783 ± 0.031	23.323 ± 0.028	23.503 ± 0.034	4.793 ± 0.005	2.580 ± 0.002	**2.562** ± 0.001	4.139 ± 0.313	2.714 ± 0.227
spambase	12.770 ± 0.560	16.087 ± 0.341	11.577 ± 0.301	12.823 ± 0.532	3.730 ± 0.228	3.045 ± 0.017	3.044 ± 0.011	2.974 ± 0.140	**2.897** ± 0.174
german	19.324 ± 0.587	18.645 ± 0.623	16.995 ± 0.617	19.318 ± 0.508	6.046 ± 0.233	3.733 ± 0.140	**3.619** ± 0.076	7.978 ± 2.112	5.030 ± 1.356
svmguide3	15.879 ± 0.531	11.925 ± 0.189	11.943 ± 0.246	19.081 ± 0.581	5.116 ± 0.260	**3.876** ± 0.061	3.912 ± 0.064	12.184 ± 0.713	4.925 ± 1.304
a9a	10.454 ± 0.075	8.551 ± 0.058	9.539 ± 0.051	5.108 ± 0.038	3.417 ± 0.040	**2.447** ± 0.023	3.797 ± 0.008	5.088 ± 0.713	3.946 ± 0.189
w8a	1.337 ± 0.011	1.254 ± 0.023	2.968 ± 0.324	1.077 ± 0.016	1.053 ± 0.015	**0.958** ± 0.012	2.863 ± 0.208	1.037 ± 0.011	1.782 ± 0.160
Average	13.877	13.208	12.724	13.485	4.026	**2.773**	3.299	5.566	3.549
Dataset	***cost* (per 100 samples) on different datasets (with normalization)**								
	Perceptron	oBag-PER	oBoost-PER	CSOGD-I	CSOGD-II	oUOB-PER	oAC2-PER	oUOB-NB	oAC2-NB
covtype	16.275 ± 0.026	13.234 ± 0.021	14.165 ± 0.018	6.118 ± 0.014	2.539 ± 0.005	**2.532** ± 0.005	2.535 ± 0.003	4.274 ± 0.171	2.886 ± 0.390
spambase	5.122 ± 0.127	4.752 ± 0.096	4.365 ± 0.134	3.560 ± 0.105	1.874 ± 0.079	**1.841** ± 0.093	1.959 ± 0.104	3.122 ± 0.179	2.645 ± 0.204
german	11.418 ± 0.631	9.723 ± 0.592	10.792 ± 0.785	7.731 ± 0.351	8.475 ± 0.464	7.272 ± 0.317	8.816 ± 0.508	7.231 ± 1.208	**4.210** ± 0.478
svmguide3	10.021 ± 0.272	9.672 ± 0.365	9.491 ± 0.524	9.294 ± 0.340	7.524 ± 0.487	7.323 ± 0.376	8.123 ± 0.410	8.864 ± 0.619	**4.184** ± 0.186
a9a	8.167 ± 0.052	7.782 ± 0.054	8.294 ± 0.098	3.544 ± 0.034	2.836 ± 0.035	**2.442** ± 0.019	2.914 ± 0.046	4.843 ± 0.349	3.862 ± 0.059
w8a	1.346 ± 0.019	1.176 ± 0.023	1.672 ± 0.078	1.053 ± 0.015	1.008 ± 0.013	1.030 ± 0.016	1.582 ± 0.062	**0.921** ± 0.020	1.222 ± 0.262
Average	8.724	7.7232	8.130	5.217	4.043	3.740	4.322	4.876	**3.168**

[14], [18], [26] have shown that simpler techniques, such as bagging algorithms and undersampling techniques, often perform better than more complex ones. In the context of online learning, we emphasize that in terms of consistency, it is how we construct the ensembles (bagging or boosting), rather than how we sample the data (undersampling or oversampling/SMOTE) that matters. Therefore, it is not surprising that online bagging based algorithms are superior to most boosting algorithms since they have much better consistency. In addition, online UnderOverBagging also outperforms recent state-of-art online cost-sensitive learning algorithms [47] in terms of misclassification cost. AdaC2 and CSB2 also achieve comparable performance, even though bagging algorithms show more consistency. This is mainly because of the good performances of their batch counterparts. The RUSBoost and SMOTEBoost algorithms have relatively worse performances mainly because of their inferior consistency due to the less reliable approximation of the weight update formula, and the higher chance of violating the requirements of boosting.

Even though online SMOTE performs well with SMOTEBagging, it might still be problematic due to the intrinsic limitation discussed in Section 4.1. One option to sidestep this issue would be to generate synthetic positive examples by Gaussian noise instead of by SMOTE [31]. On the other hand, in some scenarios, we may gather a number of positive examples before the learning process, and therefore the problem of insufficient positive examples can be mitigated. For example, for the task of seizure detection, before training a system for a new patient using an online learning algorithm, we may already have a number of annotated seizures from a library of previous patients.

An interesting direction for future work is to extend our framework to handle non-stationary distributions [35]. Such behavior often arises in online data, due to subtle (or in some case sudden) changes in the underlying data generation process. One of the challenges that arises is to incorporate a mechanism for forgetting old data. Another similar scenario is to process incoming data in mini-batches (chunks) [9], [23], assuming that data distribution is stationary within each chunk, but may evolve over chunks. One common approach used in this context is to maintain all hypotheses over all chunks and make the prediction on the current chunk based on weighted voting of the hypotheses [46]. In addition, in the case of non-stationary data, it is also less obvious what to use as a reference performance: comparison to the batch is not as pertinent. The notion of *regret* has been used in the online learning literature to characterize performance in this case [52]. Extending the learning objective of our suite of algorithms to this criterion is an interesting challenge.

ACKNOWLEDGMENTS

The authors gratefully acknowledge financial support from the Natural Sciences and Engineering Research Council of Canada (NSERC), Discovery grants program, and the Canadian Institutes for Health Research (CIHR).

REFERENCES

[1] A. Beygelzimer, S. Kale, and H. Luo, "Optimal and adaptive algorithms for online boosting," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 2323–2331.

[2] C. M. Bishop, *Pattern Recognition and Machine Learning*. Berlin, Germany: Springer, 2006.

[3] P. Branco, L. Torgo, and R. Ribeiro, "A survey of predictive modelling under imbalanced distributions," arXiv:1505.01658, 2015.

[4] L. Breiman, "Bagging predictors," *Mach. Learn.*, vol. 24, no. 2, pp. 123–140, 1996.

[5] G. Cauwenberghs and T. Poggio, "Incremental and decremental support vector machine learning," *Advances Neural Inform. Process. Syst.*, vol. 13. pp. 409–415, 2001.

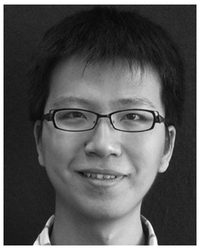
[6] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic minority over-sampling technique," *J. Artif. Intell. Res.*, vol. 16, no. 1, pp. 321–357, 2002.

[7] N. V. Chawla, A. Lazarevic, L. O. Hall, and K. W. Bowyer, "SMOTEBoost: Improving prediction of the minority class in boosting," in *Proc. Eur. Conf. Principles Practice Knowl. Discovery Databases*, 2003, pp. 107–119.

[8] S.-T. Chen, H.-T. Lin, and C.-J. Lu, "An online boosting algorithm with theoretical justifications," in *Proc. Int. Conf. Mach. Learn.*, 2012, pp. 1007–1014.

[9] S. Chen and H. He, "Nonstationary stream data learning with imbalanced class distribution," *Imbalanced Learning: Foundations, Algorithms, and Applications*. Hoboken, NJ, USA: Wiley, pp. 151–186, 2013.

[10] D. A. Cieslak, N. V. Chawla, and A. Striegel, "Combating imbalance in network intrusion datasets," in *Proc. IEEE Int. Conf. Granular Comput.*, 2006, pp. 732–737.
[11] A. Dal Pozzolo, O. Caelen, and G. Bontempi, "When is undersampling effective in unbalanced classification tasks?" in *Machine Learning and Knowledge Discovery in Databases*. Berlin, Germany: Springer, 2015, pp. 200–215.
[12] M. Denil, D. Matheson, and N. de Freitas, "Consistency of online random forests," *Proc. Int. Conf. Mach. Learn.*, 2013, pp. 1256–1264.
[13] C. Drummond and R. C. Holte, "Exploiting the cost (in)sensitivity of decision tree splitting criteria," in *Proc. Int. Conf. Mach. Learn.*, 2000, pp. 239–246.
[14] C. Drummond and R. C. Holte, "C4.5, class imbalance, and cost sensitivity: Why under-sampling beats over-sampling," in *Proc. Workshop Learn. Imbalanced Datasets II*, 2003, pp. 1–8.
[15] C. Elkan, "The foundations of cost-sensitive learning," in *Proc. Int. Joint Conf. Artif. Intell.*, 2001, pp. 973–978.
[16] T. Fawcett, "'In Vivo' spam filtering: A challenge problem for KDD," *ACM SIGKDD Explorations Newsletter*, vol. 5, no. 2, pp. 140–148, 2003.
[17] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *J. Comput. Syst. Sci.*, vol. 55, pp. 119–139, 1997.
[18] M. Galar, A. Fernández, E. Barrenechea, H. Bustince, and F. Herrera, "A review on ensembles for the class imbalance problem: Bagging-, boosting-, and hybrid-based approaches," *IEEE Trans. Syst. Man Cybern. Part C: Appl. Rev.*, vol. 42, no. 4, pp. 463–484, Jul.2012.
[19] J. Gotman, "Automatic seizure detection: Improvements and evaluation," *Electroencephalography Clinical Neurophysiology*, vol. 76, no. 4, pp. 317–324, 1990.
[20] H. Guo and H. L. Viktor, "Learning from imbalanced data sets with boosting and data generation: The DataBoost-IM approach," *ACM SIGKDD Explorations Newsletter*, vol. 6, no. 1, pp. 30–39, 2004.
[21] H. He and E. A. Garcia, "Learning from Imbalanced Data," *IEEE Trans. Knowl. Data Eng.*, vol. 21, no. 9, pp. 1263–1284, Sep. 2009.
[22] S. Hido, H. Kashima, and Y. Takahashi, "Roughly balanced bagging for imbalanced data," *Statistical Anal. Data Mining*, vol. 2, no. 5–6, pp. 412–426, 2009.
[23] T. R. Hoens, R. Polikar, and N. V. Chawla, "Learning from streaming data with concept drift and imbalance: An overview," *Progress Artif. Intell.*, vol. 1, no. 1, pp. 89–101, 2012.
[24] S. C. Hoi, R. Jin, P. Zhao, and T. Yang, "Online multiple kernel classification," *Mach. Learn.*, vol. 90, no. 2, pp. 289–316, 2013.
[25] M. V. Joshi, V. Kumar, and R. C. Agarwal, "Evaluating boosting algorithms to classify rare classes: Comparison and improvements," in *Proc. Int. Conf. Data Mining*, 2001, pp. 257–264.
[26] T. M. Khoshgoftaar, J. Van Hulse, and A. Napolitano, "Comparing boosting and bagging techniques with noisy and imbalanced data," *IEEE Trans. Syst. Man Cybern. Part A: Syst. Humans*, vol. 41, no. 3, pp. 552–568, May 2011.
[27] T.-K. Kim, S.-F. Wong, B. Stenger, J. Kittler, and R. Cipolla, "Incremental linear discriminant analysis using sufficient spanning set approximations," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2007, pp. 1–8.
[28] J. Kivinen, A. J. Smola, and R. C. Williamson, "Online learning with kernels," *IEEE Trans. Signal Process.*, vol. 52, no. 8, pp. 2165–2176, Aug. 2004.
[29] J. Langford, L. Li, and T. Zhang, "Sparse online learning via truncated gradient," *J. Mach. Learn. Res.*, vol. 10, pp. 777–801, 2009.
[30] P. Laskov, C. Gehl, S. Krüger, and K.-R. Müller, "Incremental support vector learning: Analysis, implementation and applications," *J. Mach. Learn. Res.*, vol. 7, pp. 1909–1936, 2006.
[31] S. S. Lee, "Noisy replication in skewed binary classification," *Comput. Statist. Data Anal.*, vol. 34, no. 2, pp. 165–191, 2000.
[32] Y. Lin, Y. Lee, and G. Wahba, "Support vector machines for classification in nonstandard situations," *Mach. Learn.*, vol. 46, no. 1–3, pp. 191–202, 2002.
[33] L.-P. Liu, Y. Jiang, and Z.-H. Zhou, "Least square incremental linear discriminant analysis," in *Proc. Int. Conf. Data Mining*, 2009, pp. 298–306.
[34] H. Masnadi-Shirazi and N. Vasconcelos, "Risk minimization, probability elicitation, and cost-sensitive SVMs," in *Proc. Int. Conf. Mach. Learn.*, 2010, pp. 759–766.
[35] L. L. Minku, A. P. White, and X. Yao, "The impact of diversity on online ensemble learning in the presence of concept drift," *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 5, pp. 730–742, May 2010.
[36] N. C. Oza and S. Russell, "Online bagging and boosting," in *Proc. Artif. Intell. Statist.*, 2005, pp. 105–112,.
[37] N. C. Oza, "Online ensemble learning," Ph.D. dissertation, Comput. Sci. Division, Univ. California, Berkeley, 2001.
[38] Y. Park, L. Luo, K. K. Parhi, and T. Netoff, "Seizure prediction with spectral power of EEG using cost-sensitive support vector machines," *Epilepsia*, vol. 52, no. 10, pp. 1761–1770, 2011.
[39] R. Polikar, "Ensemble based systems in decision making," *IEEE Circuits Syst. Mag.*, vol. 6, no. 3, pp. 21–45, Jul.-Sep. 2006.
[40] A. Saffari, C. Leistner, J. Santner, M. Godec, and H. Bischof, "Online random forests," in *Proc. Online Learn. Comput. Vis. Workshop*, 2009, pp. 1393–1400.
[41] C. Seiffert, T. M. Khoshgoftaar, J. Van Hulse, and A. Napolitano, "RUSBoost: A hybrid approach to alleviating class imbalance," *IEEE Trans. Syst., Man Cybern., Part A: Syst. Humans*, vol. 40, no. 1, pp. 185–197, Jan. 2010.
[42] V. S. Sheng and C. X. Ling, "Roulette sampling for cost-sensitive learning," in *Proc. Eur. Conf. Mach. Learn.*, 2007, pp. 724–731.
[43] Y. Sun, M. S. Kamel, A. K. Wong, and Y. Wang, "Cost-sensitive boosting for classification of imbalanced data," *Pattern Recognit.*, vol. 40, no. 12, pp. 3358–3378, 2007.
[44] K. M. Ting, "A comparative study of cost-sensitive boosting algorithms," in *Proc. Int. Conf. Mach. Learn.*, 2000, pp. 983–990.
[45] P. E. Utgoff, N. C. Berkman, and J. A. Clouse, "Decision tree induction based on efficient tree restructuring," *Mach. Learn.*, vol. 29, no. 1, pp. 5–44, 1997.
[46] H. Wang, W. Fan, P. S. Yu, and J. Han, "Mining concept-drifting data streams using ensemble classifiers," in *Proc. 9th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2003, pp. 226–235.
[47] J. Wang, P. Zhao, and S. C. Hoi, "Cost-sensitive online classification," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 10, pp. 2425–2438, Oct. 2014.
[48] S. Wang and X. Yao, "Diversity analysis on imbalanced data sets by using ensemble models," in *Proc. IEEE Symp. Comput. Intell. Data Mining*, 2009, pp. 324–331.
[49] G. Wu and E. Y. Chang, "Adaptive feature-space conformal transformation for imbalanced-data learning," in *Proc. Int. Conf. Mach. Learn.*, 2003, pp. 816–823.
[50] B. Zadrozny and C. Elkan, "Learning and making decisions when costs and probabilities are both unknown," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2001, pp. 204–213.
[51] P. Zhao, R. Jin, T. Yang, and S. C. Hoi, "Online AUC maximization," in *Proc. Int. Conf. Mach. Learn.*, 2011, pp. 233–240.
[52] M. Zinkevich, "Online convex programming and generalized infinitesimal gradient ascent," in *Proc. Int. Conf. Mach. Learn.*, 2003, pp. 928–936.



Boyu Wang received the BEng degree in electronic information engineering from Tianjin University, Tianjin, China, and the MSc degree in electrical and computer engineering from the University of Macau, Macau, China. He is currently a PhD student in the School of Computer Science, McGill University, Montreal, QC, Canada. His research interests include machine learning, brain signal analysis, and reinforcement learning.

Joelle Pineau received the BASc degree from the University of Waterloo, Canada, in 1998, and the PhD degree from Carnegie Mellon University, Pittsburgh, Pennsylvania, in 2004. She is a William Dawson Scholar and associate professor in the School of Computer Science, McGill University, Montreal, Canada. She is also a senior fellow of the Canadian Institute for Advanced Research (CIFAR). Her research focuses on developing models and algorithms for learning and decision-making in partially observable stochastic domains, and applying these results to complex problems in robotics and healthcare.