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 extensively, the effort in simultaneously dealing with these two issues is limited. Aiming at this challenge task, a novel learning framework is proposed in this paper. The key idea is based on the fusion of online ensemble algorithms and the state of the art batch mode cost-sensitive bagging/boosting algorithms. Within this framework, two separately developed research areas are bridged together, and a batch of theoretically sound online cost-sensitive bagging and online cost-sensitive boosting algorithms are first proposed. Unlike other online cost-sensitive learning algorithms lacking theoretical analysis of asymptotic properties, the convergence of the proposed algorithms is guaranteed under certain conditions, and the experimental evidence with benchmark data sets also validates the effectiveness and efficiency of the proposed methods.

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

arXiv:1310.8004v1 [cs.LG] 30 Oct 2013

1 Introduction

In many real world applications, such as medical diagnosis [38], network intrusion detection [10], spam filtering [15], the distribution between the classes of examples is not well balanced. In a binary classification scenario, such as automatically detecting whether an incoming individual is an intruder or not, we may have many more negative examples (i.e., non-intruders) than positive examples (i.e., intruders). The class with the most examples is typically referred to as the majority class, and the other one as the minority class. Conventional learning algorithms in machine learning and data mining typically do not work well for imbalanced class problems since their objective is to minimize the overall error rate, which implicitly treats all misclassification costs equally. As a result, these algorithms may produce trivial results, typically classifying all test examples as negative. Additionally, it is often the case that the positive (minority) class is the one of greater interest. For example, the expected cost of missing an intruder may be much higher than the cost of mis-identifying a non-intruder. For this reason, the class imbalance problem [21] has often been formulized within the cost-sensitive learning framework [13].

The class imbalance problem gets more challenging in the context of learning from data streams, which unfortunately is required in many practical classification problems. For example, in epileptic EEG/ECoG data analysis [19], one challenge is that seizures are relatively rare, compared to non-seizure brain activities, and the cost of missing a seizure is much higher than detecting seizure-free signal into a seizure. Meanwhile, in the clinical scenario, the EEG/ECoG data is usually collected incrementally over time, and the seizure detection system must respond to the signals in real time. Even though the entire data can be received from the start and the classifier can be trained in batch mode, it is still favorable if the model can adapt online to new data. Such data may come from the same subject but from different time, or even from different subjects. This is because the size of the entire data set can be too large to fit in memory or to train a classifier at once. Moreover, since the positive examples (seizures) are rather rare, each of them should be made full use of. However, in the batch learning scenario, the trained classifier is static and is not updated in the testing phase. Therefore, this problem should also be formulated within the incremental learning framework.[1] The main challenge of such a problem is how to perform well even with very few positive examples at the early stage of model building, and perform better as more examples are available, motivating us to develop effective incremental learning algorithms dealing with class imbalance problem.

Many techniques have been proposed to deal with class imbalance problem, among which ensemble learning approaches[2] have been proved more effective than other algorithms [17]. On the other hand, many methods for learning from data streams are also available in the literature on incremental/online learning. Although these two issues have been well studied independently in the past decades, the research on simultaneously

• *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.

1. In this paper, incremental learning is referred to the general technique learning from data streams, and online learning is characterized responding immediately to a new instance and then discarding it, except for SMOTE-based online learning algorithms, the positive examples are stored to create synthetic samples.

2. In this paper, we restrict ourselves to bagging and boosting approaches.

solving both of the problems is limited. In this paper, an online learning with class imbalance framework is proposed. Our main contributions can be summarized as follows.

1. We propose 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 version.
2. We theoretically analyze the consistency between the proposed algorithms and their batch mode counterparts, showing 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.
3. Two long separate research areas, cost-sensitive learning and incremental learning, are bridged together in this paper. As meta-learning techniques, the proposed framework can convert any existing cost-insensitive online learning algorithm into cost-sensitive one.
4. With some straightforward modifications, the proposed algorithms can also deal with the concept drift problem in non-stationary environments.

The performance of the proposed online cost-sensitive bagging and boosting algorithms is evaluated on the UCI data sets, and a comprehensive comparison study of online and batch cost-sensitive ensemble learning algorithms is presented. It should be emphasized that we mainly focus on the consistency between the proposed methods and their batch mode counterparts. The comparison among different ensembles for class imbalance or deciding the best one, which can be found in [17], [27], is beyond the scope of this paper. The remainder of this paper is organized as follows. Section 2 briefly reviews the related work of class imbalance learning and online learning. Section 3 presents technical background of our work. The online cost-sensitive ensemble learning framework is proposed in Section 4 followed by the theoretical analysis in Section 5. The experimental results are reported in Section 6, and some discussions are provided in Section 7. Section 8 extends the proposed framework to non-stationary environments. Section 9 completes the paper with conclusions.

2 Related Work

Existing approaches to dealing with class imbalanced problem can be roughly categorized into algorithm level approaches and data level approaches. The former directly modifies the 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 goal of a classifier is to minimize the cost instead of classification error, and therefore the classification algorithms will bias towards the small class. For example, cost-sensitive SVM can be derived by kernel modification [48], biased penalty [33], or loss function modification [35]. For decision tree, the cost sensitivity can be introduced by probabilistic estimate calibration [49] or using different pruning methods [12]. At the data level, a data preprocessing step is added to rebalance the class distribution by undersampling the negative class, oversampling the positive class, or creating synthetic positive examples [6], [7], and then conventional cost-insensitive algorithms are applied on the rebalanced data sets. Such approaches can also be regarded as meta-learning techniques, that is, they can convert cost-insensitive algorithms into cost-sensitive ones without modifying them. One advantage of this category of approaches is that it is independent of specific classifiers and therefore it is applicable to any existing cost-insensitive learning algorithm. The sampling techniques can be further incorporated with ensemble learning algorithms [40] and have received much attention recently due to their better performance on imbalanced data sets [17]. Ensemble based techniques for class imbalance problem inherit the good properties of ensemble learning algorithms, improving the generalization ability of learning algorithms via bias or/and variance reduction, as well as achieving cost sensitivity by sampling techniques. In addition, traditional ensemble learning algorithms themselves have sampling step in each iteration. Therefore, little extra learning cost is added when embedding the resampling step to rebalance the data set.

As for learning from data streams, some algorithms are naturally or can be easily extended to incremental version, including k-NN, naive Bayes classifier, binary linear discriminant analysis (LDA), quadratic discriminant analysis (QDA), and so on. In addition, the incremental/online versions of more sophisticated algorithms have been proposed in the literature, including but not limited to decision trees [46], random forests [41], [11] , multi-class LDA [34], [28], logistic regression [31], support vector machines [32], [5], and other kernel methods [29], [25]. Besides the base learning algorithms, the online versions of ensemble learning techniques, bagging and boosting, were also derived in [36] by approximating binominal distribution using a Poisson distribution.

3 Technical Background

In this section, we briefly review the standard bagging and boosting algorithms, as well as their online versions 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 idea here is to improve the generalization ability of individual classifiers by training them on different data sets, which

is motivated by bias-variance trade-off [3]. Averaging the outputs of base models tends to cancel the variance component or/and reduce the bias. On the other hand, to achieve good performance, the diversity, which is usually introduced by presenting different training data to different base models among classifiers, is an important characteristic [30]. Different approaches to averaging base models and obtaining desired diversity distinguish different ensemble methods. Two representative techniques among them are Bagging [4] and Boosting (AdaBoost) [16].

Given a data set $S = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ of size N, where $x_n \in X$, $y_n \in Y = \{0, 1\}$, M base models h_m, bagging constructs M classifiers with bootstrap replicas $\{S_m\}$ of S, where S_m is obtained by drawing examples from original the date 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 data set. After constructing ensembles, the prediction of the class of a new example is given by majority voting. The pseudo-code of Bagging is shown in Alg. 1.

Algorithm 1 Bagging Algorithm

Input: S, M

1: **for** $m = 1, \ldots, M$ **do**
2: $\quad S_m = Sample_with_replacement(S, N)$
3: $\quad$ 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 is another widely used ensemble learning algorithm. Unlike Bagging, which treats all of examples equally, AdaBoost focuses more on difficult examples. In particular, Adaboost sequentially constructs a series of base learner in such a way that examples that are misclassified by current base learner h_m are given more weights in the training set for the following learner h_{m+1}, whereas the correctly classified examples are given less weights. More specifically, the weights of all examples are initially equal, and then examples misclassified by h_m are given half the total weight for the following learner h_{m+1}, and the correctly classified examples are given the remaining half of the total weights. The pseudo-code of AdaBoost is shown in Alg. 2.

It should be emphasized that by using the update rule in step 6, the normalization step is avoided. That is, the summation of D_{m+1} will remain one after each update without normalization, which is crucial to designing the online boosting algorithm [36] and the proposed online cost-sensitive boosting algorithms, since the normalization factor is unavailable during the online learning process. After update, the reweighted examples can be either directly used to train the next base learner, or 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 folds: 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, reweighting on training set is not applicable to all learning algorithms.

Algorithm 2 Boosting (AdaBoost) Algorithm

Input: S, M

1: Initialize $D_1(n) = \frac{1}{N}$ for all $n \in \{1, \ldots, N\}$
2: **for** $m = 1, \ldots, M$ **do**
3: $\quad$ Train a base learner $h_m \rightarrow Y$ using S with distribution D_m
4: $\quad \epsilon_m = \sum_{n=1}^{N} D_m(n) I(h_m(x_n) \neq y_n)$
5: $\quad$ **for** $n = 1, \ldots, N$ **do**
6: $\quad\quad D_{m+1}(n) = D_m(n) \times \begin{cases} \frac{1}{2(1-\epsilon_m)}, & h_m(x_n) = y_n \\ \frac{1}{2\epsilon_m}, & h_m(x_n) \neq y_n \end{cases}$
7: $\quad$ **end for**
8: **end for**

Output: $H(x) = \arg\max_{y \in Y} \sum_{m=1}^{M} log(\frac{1-\epsilon_m}{\epsilon_m}) I(h_m(x) = y)$

3.2 Online bagging and boosting

The framework of online ensemble learning algorithms [36], [37] 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 $N \rightarrow \infty$, where the probability of success p in the binomial distribution is equivalent to $D(n)$ in bagging and boosting algorithms. For example, since $D(n) = \frac{1}{N}$ for all examples in bagging algorithm, the uniform sampling with replacement of bagging algorithms can be approximated by $Poisson(1)$. For online boosting, λ can be computed by tracking the total weights of correctly classified and misclassified examples for each base learner $(\lambda_m^{SC}, \lambda_m^{SW})$.The online bagging and boosting algorithms are described in Alg. 3 and Alg. 4 respectively. For more detailed description of the online ensemble learning framework, we refer the reader to [36], [37].

Algorithm 3 Online Bagging Algorithm

Input: S, M

1: **for** $n = 1, \ldots, N$ **do**
2: $\quad$ **for** $m = 1, \ldots, M$ **do**
3: $\quad\quad$ Let $k \sim Poisson(1)$
4: $\quad\quad$ Do k times
$\quad\quad\quad$ Train the base learner $h_m \rightarrow Y$ using (x_n, y_n)
5: $\quad$ **end for**
6: **end for**

Output: $H(x) = \arg\max_{y \in Y} \sum_{m=1}^{M} I(h_m(x) = y)$

3.3 Cost-sensitive bagging and boosting

Cost-sensitive ensemble learning is a meta-technology that takes different misclassification costs into consideration via biased resampling/reweighting methods before each iteration of bagging/boosting. As a result, cost-sensitive ensemble learning algorithms can turn any

Algorithm 4 Online Boosting (AdaBoost) Algorithm

Input: S, M

Initialize $\lambda_m^{SC} = 0, \lambda_m^{SW} = 0$ for all $m \in \{1, \ldots, M\}$
for $n = 1, \ldots, N$ **do**
  Set $\lambda = 1$
  for $m = 1, \ldots, M$ **do**
    Let $k \sim Poisson(\lambda)$
    Do k times
      Train the base learner $h_m \rightarrow Y$ using (x_n, y_n)
    if $h_m(x_n) = y_n$ **then**
      $\lambda_m^{SC} \leftarrow \lambda_m^{SC} + \lambda, \epsilon_m \leftarrow \frac{\lambda_m^{SW}}{\lambda_m^{SC} + \lambda_m^{SW}}, \lambda \leftarrow \frac{\lambda}{2(1-\epsilon_m)}$
    else
      $\lambda_m^{SW} \leftarrow \lambda_m^{SW} + \lambda, \epsilon_m \leftarrow \frac{\lambda_m^{SW}}{\lambda_m^{SC} + \lambda_m^{SW}}, \lambda \leftarrow \frac{\lambda}{2\epsilon_m}$
    end if
  end for
end for

Output: $H(x) = \arg\max_{y \in Y} \sum_{m=1}^{M} log(\frac{1-\epsilon_m}{\epsilon_m}) I(h_m(x) = y)$

cost-insensitive classifier into a cost-sensitive one with little extra learning cost, while preserving the good properties of standard ensemble learning algorithms (i.e., bias and/or variance reduction). The main difference between these techniques lies in the choice of resampling mechanism. From this point, we briefly review six popular cost-sensitive ensemble learning algorithms: UnderOverBagging [47], SMOTEBagging [47], AdaC2 [44], CSB2 [45], RUSBoost [42], and SOMOTEBoost [8]. The derivation of their online extensions is the main contribution of this paper and is described in the next section.

Given an imbalanced data set of N^+ minority class S^+ and N^- majority class S^-, one straightforward approach to implement bagging-based ensemble learning algorithms is to undersample the majority class or oversample the minority class, which gives UnderBagging and OverBagging respectively. UnderOverBagging [47] is a uniform approach combining both UnderBagging and OverBagging. In addition, the resampling rate ($a\%$) can be also varied over the bagging iterations, which further boosts the diversity among the base learners.

Alg. 5 shows the pseudo-code of UnderOverBagging, from which it can be observed 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 [47] (Alg. 6), the negative class is sampled with replacement at rate 100% (i.e., N^- negative examples are generated), while CN^+ positive examples are generated for each base learner, among which $a\%$ of them are created by resampling and the rest of the examples are created by the synthetic minority oversampling technique (SMOTE) [6] shown in Alg. 7. The main idea of SMOTE is to generating more new synthetic examples by interpolating the positive examples. As a result, all of the base learners are trained on a more

Algorithm 5 Batch UnderOverBagging Algorithm

Input: $S, M, C > 1$

for $m = 1, \ldots, M$ **do**
  $a = \frac{100m}{M}$
  $S_{train}^- = Sample_with_replacement(S^-, N^- a\%)$
  $S_{train}^+ = Sample_with_replacement(S^+, CN^+ a\%)$
  $S_m = S_{train}^+ + S_{train}^-$
  Train a base learner $h_m \rightarrow Y$ using S_m
end for

Output: $H(x) = \arg\max_{y \in Y} \sum_{m=1}^{M} I(h_m(x) = y)$

balanced and diverse data set. The diversity is further boosted by varying $a\%$ so that the ratios of bootstrap replicates and synthetic examples generated by SMOTE varies over the bagging iterations.

Algorithm 6 Batch SMOTEBagging Algorithm

Input: $S, M, k, C > 1$

for $m = 1, \ldots, M$ **do**
  $a = \frac{100m}{M}$
  $S_{train}^- = Sample_with_replacement(S^-, N^-)$
  $S_{train}^+ = Sample_with_replacement(S^+, CN^+ a\%)$
  $S_S^+ = SMOTE(S^+, C(1 - a\%), k)$
  $S_m = S_{train}^+ + S_{train}^- + S_S^+$
  Train a base learner $h_m \rightarrow Y$ using S_m
end for

Output: $H(x) = \arg\max_{y \in Y} \sum_{m=1}^{M} I(h_m(x) = y)$

Algorithm 7 Batch SMOTE Algorithm

Input: S, T, k

for $n = 1, \ldots, N$ **do**
  for $i = 1, \ldots, T$ **do**
    Randomly choose one of the k nearest neighbors of x_n, x'_n
    Calculate the difference between x_n and x'_n
    Generate a random number γ between 0 and 1
    Create a synthetic instance:
    $x_{S,n}^i = x_n + \gamma(x'_n - x_n)$
  end for
end for

Output: TN synthetic instances $\{x_{S,1}^1, \ldots, x_{S,N}^T\}$

In [47], the sampling rate C for both UnderOverBagging and SMOTEBagging was set as $C = \frac{N^-}{N^+}$ to obtain balanced class distribution. However, it has been reported that the optimal class ratio need not be $1:1$ [41]. To eliminate this effect, we vary the resampling rate to generate an ROC to compare different cost-sensitive algorithms and their online counterparts.

AdaC2 [44] is a boosting algorithm that takes the different misclassification costs into consideration when calculating the classifier weights, and updates the sample weights. In particular, AdaC2 increases more weights on the misclassified positive class, compared to misclassified negative ones. Similarly, it decreases less the weights on correctly classified positive examples compared to correctly classified negative examples. The pseudo-code for AdaC2 is shown in Alg. 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.

Algorithm 8 Batch AdaC2 Algorithm

Input: S, M, C_N, C_P
1: Initialize $D_1(n) = \frac{1}{N}$ for all $n \in \{1, \ldots, N\}$
2: **for** $m = 1, \ldots, M$ **do**
3: Train a base learner $h_m \to Y$ 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** $n = 1, \ldots, N$ **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: $H(x) = \arg\max_{y \in Y} \sum_{m=1}^{M} \alpha I(h_m(x) = y)$

The CSB2 algorithm [45] (Alg. 9) is a kind of compromise between AdaBoost and AdaC2. For correctly classified examples, CSB2 treats them in the same way as AdaBoost, while for misclassified examples, it does the same as AdaC2. In addition, the voting weight of each base learner in CSB2 is the same as AdaBoost. AdaC2 and CSB2 take 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 unequally treating the examples from different classes. Therefore, such approaches are categorized as cost-sensitive boosting [17].

Algorithm 9 Batch CSB2 Algorithm

Input: S, M, C_N, C_P
1: Initialize $D_1(n) = \frac{1}{N}$ for all $n \in \{1, \ldots, N\}$
2: **for** $m = 1, \ldots, M$ **do**
3: Train a base learner $h_m \to Y$ 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** $n = 1, \ldots, N$ **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: $H(x) = \arg\max_{y \in Y} \sum_{m=1}^{M} \alpha I(h_m(x) = y)$

Cost sensitivity can also be introduced 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 [42] and SMOTEBoost [8][3]. These two algorithms differ in the way they undersample the negative class or oversample the positive class by SMOTE. In particular, both algorithms rebalance the class distribution before representing the training data to base learners, yet the error estimate of base learners is still measured on the original data set. The RUSBoost and SMOTEBoost algorithms are shown in Alg. 10 and Alg. 11 respectively[4]. For each algorithm, there are at least three different implementations. Take RUSBoost for example, in RUSBoost1, the class ratio is fixed, which means the ratio of the weighted positive and negative examples are fixed, and then the modified training set are generated according to their weights. The proportion of examples of the two classes for each base learner is not necessarily fixed. The difference between RUSBoost2 and RUSBoost1 is that it is the ratio of unweighted examples of two classes that is fixed, which means that the proportion of examples of the two classes that pass through each base learner is fixed. RUSBoost3 is easier to understand: the sampling rate of negative class is $\frac{1}{C}$ of that of positive class. The three implementations of SMOTEBoost are analogous to the ones of RUSBoost. The only difference is that instead of undersampling the majority class, SMOTEBoost algorithms oversample the minority class by generating synthetic positive examples using SMOTE.

Algorithm 10 Batch RUSBoost Algorithm

Input: S, M, C
1: Initialize $D_1(n) = \frac{1}{N}$ for all $n \in \{1, \ldots, N\}$
2: **for** $m = 1, \ldots, M$ **do**
3: Modify the distribution D_m by undersampling the majority class and generate a new training set S'
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** $n = 1, \ldots, N$ **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 data 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)
Modify the distribution D_m by undersampling the majority class: Generate CN^+ and N^+ majority and minority examples respectively by sampling D_m.

RUSBoost3 (fix the sampling rate)
Modify the distribution D_m by undersampling the majority class: Generate N exmples according to distribution D_m, and then undersample majority class until $\frac{1}{C}$ of them left.

3. AdC2, CSB2 and RUSBoost, SMOTEBoost were respectively categorized into different families in [17]. Here we do not distinguish them since all of these methods can be formulated within the framework of combination of resampling techniques and conventional boosting algorithms, and we refer to all of them as cost-sensitive boosting algorithms

4. RUSBoost and SMOTEBoost are originally based on AdaBoost.M2, which is a variant of AdaBoost for multi-classification problems. As for binary classification problems, AdaBoost.M2 with base learner outputing 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 Algorithm

Input: S, M, C

Initialize $D_1(n) = \frac{1}{N}$ for all $n \in \{1, \ldots, N\}$
for $m = 1, \ldots, M$ **do**
  Modify the distribution D_m by creating synthetic examples from minority class using SMOTE
  Train a base learner $h_m \to Y$ using S with distribution D_m
  $\epsilon_m = \sum_{n=1}^{N} D_m(n) I(h_m(x_n) \neq y_n)$
  for $n = 1, \ldots, N$ **do**
    $D_{m+1}(n) = D_m(n) \times \begin{cases} \frac{1}{2(1-\epsilon_m)}, & h_m(x_n) = y_n \\ \frac{1}{2\epsilon_m}, & h_m(x_n) \neq y_n \end{cases}$
  end for
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)
Modify the distribution D_m 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)
Modify the distribution D_m 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)
Modify the distribution D_m 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

Our proposed framework for online cost-sensitive ensemble learning is based on the combination of online ensemble methods [36] and batch cost-sensitive ensembles described above. More specifically, the cost sensitivity of batch approaches is introduced by different resampling mechanism, while in the online ensembles it is achieved by manipulating the parameters of the Poisson distribution for different classes. In this section, the online extensions of all of the popular methods described in previous section are derived. Note that the proposed framework can also be applied to other batch cost-sensitive ensembles, such as RBBagging [22], CSRoulette [43], RareBoost [26], or DataBoost-IM [20].

The main challenge of generalizing the cost-sensitive ensemble algorithms to online mode is how to impose the cost setting on standard online ensemble algorithms. This is straightforward for some cases. In some other cases, however, it is not straightforward to embed costs into online ensembles, especially for boosting based algorithms. In particular, the weight update step in standard AdaBoost only involves the classification of each base learner without normalization, so the online boosting algorithm can be implemented by simply tracking the error and using it to update the Poisson parameter λ for each base learner. Therefore, the key point for us is to reformulate the batch cost-sensitive boosting algorithms in a way that there is no normalization step at each iteration, and then incrementally estimate the quantities embedded with the cost setting in the online learning scenario. Also, even though the asymptotic properties of standard online ensemble algorithms have been thoroughly studied in [37], the theoretical properties in online cost-sensitive learning scenario need to be reformulated and established.

4.1 Online Cost-Sensitive Bagging

The derivation of online cost-sensitive bagging algorithms is straightforward. Since the Poisson distribution parameter $\lambda = 1$ corresponds to sampling with probability $\frac{1}{N}$, sampling an example with probability $\frac{C}{N}$ can be obtained by presenting the instance to the base model $k \sim Poisson(C)$ times in online bagging. Therefore, online UnderOverBagging can be derived as in Alg. 12.

Algorithm 12 Online UnderOverBagging Algorithm

Input: $S, M, C > 1$

for $n = 1, \ldots, N$ **do**
  for $m = 1, \ldots, M$ **do**
    a = $\frac{m}{M}$
    If $y_n = 1$, $\lambda = aC$, else $\lambda = a$
    Let $k \sim Poisson(\lambda)$
    Do k times
      Train the base learner $h_m \to Y$ using (x_n, y_n)
  end for
end for

Output: $H(x) = \arg\max_{y \in Y} \sum_{m=1}^{M} I(h_m(x) = y)$

For the online SMOTEBagging, we have the similar generalization, which is shown in Alg. 13, together with online SMOTE in Alg. 14. It is worth noting that the online SMOTE may not be a good approximate of its batch

Algorithm 13 Online SMOTEBagging Algorithm

Input: $S, M, C > 1$

$X^+ = \{\}$
for $n = 1, \ldots, N$ **do**
  for $m = 1, \ldots, M$ **do**
    a = $\frac{m}{M}$
    if $y_n = 1$ **then**
      $X^+ = \{X^+; x_n\}$,
      $\lambda = aC, \lambda_{SMOTE} = (1-a)C$
      Let $k \sim Poisson(\lambda)$
      Do k times
        Train the base learner $h_m \to Y$ using (x_n, y_n)
      Let $k_{SMOTE} \sim Poisson(\lambda_{SMOTE})$
      Do k_{SMOTE} times
        $x_S = Online_SMOTE(X^+)$
        Train the base learner $h_m \to Y$ using (x_S, y_n)
    else
      Let $k \sim Poisson(\lambda)$
      Do k times
        Train the base learner $h_m \to Y$ using (x_n, y_n)
    end if
  end for
end for

Output: $H(x) = \arg\max_{y \in Y} \sum_{m=1}^{M} I(h_m(x) = y)$

counterpart, since at the early stage of the algorithm, the positive examples are extremely rare, and therefore the generated synthetic examples by online SMOTE can be much different from the ones by batch SMOTE. This is an intrinsic problem of online SMOTE, and the distributions of synthetic examples created by online SMOTE and batch SMOTE cannot be the same unless we can obtain a large number of minority examples from the very beginning. However, even with this problem, experimental results demonstrate reasonably good consistency between online and batch SMOTEBagging algorithms.

Algorithm 14 Online SMOTE Algorithm

Input: X

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 γ between 0 and 1
5: Create a synthetic instance:
$x_S = x + \gamma(x' - x)$

Output: a synthetic instances x_{SMOTE}

4.2 Online Cost-Sensitive Boosting

The standard online boosting framework can be generalized to be cost-sensitive by introducing different parameters of Poisson distribution for different classes. As stated at the beginning of this section, to implement online cost-sensitive boosting, the key step is to formulate new weight updates formulas of the batch algorithms that do not require the normalization step.

For AdaC2 algorithm, we have

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

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 weighed accuracy. If we reformulate the weight update step (step 6) of AdaC2 as

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

it can be observed that it is equivalent to step 6 of Alg. 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 unweighted sum of total Poisson distribution parameter for the mth base learner. The pseudo-code of online AdaC2 is shown in Alg. 15.

Algorithm 15 Online AdaC2 Algorithm

Input: S, M, C_N, C_P

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: $H(x) = \arg\max_{y \in Y} \sum_{m=1}^{M} log(\frac{wacc_m}{werr_m}) I(h_m(x) = y)$

By similar calculation, the update formula of 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 can be done by tracking $\lambda_m^{FP}, \lambda_m^{FN}, \lambda_m^{SW}$ and λ_m^{SUM} respectively. The pseudo-code of online CSB2 is shown in Alg. 16.

As shown in Alg. 10, the RUSBoost algorithm is the same as the standard boosting algorithm except that there is a preprocessing 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

Algorithm 16 Online CSB2 Algorithm

Input: S, M, C_N, C_P

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: $H(x) = \arg\max_{y \in Y} \sum_{m=1}^{M} log(\frac{1-\epsilon_m}{\epsilon_m}) I(h_m(x) = y)$

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 one of negative examples. Since the size of training examples is $(C+1)N^+$ after undersampling, we finally get the new weight of each example:

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

Therefore, to approximate these quantities in online RUSBoost1, we need to record the total weights of positive examples and negative examples, as well as the total number of them respectively, which can be done by four additional parameters: λ_m^{POS}, λ_m^{NEG}, n^+, and n^-. In particular, N^+ and N^- can be approximated by n^+, and n^-. Since $D_m(n)$ in batch boosting (Alg. 2) corresponds to the parameter λ in online boosting (Alg. 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 the total weight of them is larger 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 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 demonstrated in Alg. 17.

Algorithm 17 Online RUSBoost Algorithm

Input: S, M, C

1: Initialize $\lambda_m^{SC} = 0, \lambda_m^{SW} = 0, \lambda_m^{POS} = 0, \lambda_m^{NEG} = 0$, for all $m \in \{1, \ldots, M\}$, $n^+ = 0$, $n^- = 0$
2: **for** $n = 1, \ldots, N$ **do**
3: Set $\lambda = 1$
4: **for** $m = 1, \ldots, M$ **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 λ^{RUS} (step 21 – 23)
11: Let $k \sim Poisson(\lambda^{RUS})$
12: Do k times
13: Train the base learner $h_m \to Y$ 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 forOutput:** $H(x) = \arg\max_{y \in Y} \sum_{m=1}^{M} log(\frac{1-\epsilon_m}{\epsilon_m}) I(h_m(x) = y)$

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

The pseudo-code of the online SMOTEBoost algorithms are described in Alg. 18. In online SMOTEBoost1, the weights of synthetic examples are $\frac{1}{N'}$, where N' is the number of examples in the new data set. As a result, the probability that an example in the original data set is sampled is the same as its weight before renormalization, which means the parameter λ used in online SMOTEBoost1 is the same as the standard online boosting. Besides, synthetic examples are generated by uniformly sampling from positive examples and applying 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 data set, the probability of a positive example being chosen to generate synthetic

examples is proportional to its weight. Therefore, the parameter λ^{SMOTE} is given by $\lambda'(\frac{N^-}{CN^+}-1)$. The online SMOTEBoost3 is the same as standard online boosting, except for the additional parameter $\lambda^{SMOTE}=\lambda(C-1)$.

Algorithm 18 Online SMOTEBoost Algorithm

Input: S, M, C

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 λ' (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 λ^{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 forOutput:** $H(x)=\arg\max_{y\in Y}\sum_{m=1}^{M}log(\frac{1-\epsilon_m}{\epsilon_m})I(h_m(x)=y)$

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

5 Theoretical Analysis

In this section, we prove that under the similar conditions as for cost-insensitive online bagging and boosting [36], the online UnderOverBagging, AdaC2, CSB2 and RUSBoost converge to the same solution as their batch counterparts. SMOTEBagging and SMOTEBoost theoretically cannot converge to their batch mode counterparts since online SMOTE cannot converge to batch SMOTE. Because of the space limitation, we only give the statements of the main theorems.

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

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

Theorem 3. *As $N^+\rightarrow\infty$ and $N^-\rightarrow\infty$, if the base learners are naive Bayes classifiers, the online RUSBoost algorithms converge to their batch mode counterparts.*

The proofs of the theorems follows quite readily from the work by [37], but need to be generalized to different cost settings. The details of the proofs are in the Supplementary Materials.

6 Experiments

In this section, the proposed online cost-sensitive ensemble learning algorithms were compared with their batch mode counterparts using benchmark data sets, and the performances, measured by area under an ROC curve (AUC), were obtained by five-fold cross-validation. Since the performance of different batch ensemble algorithms under different conditions has been thoroughly compared in [17], [27], [44], [42], this paper mainly focuses on the performances of online algorithms, and the comparison between the two different types of learning approaches. We aim to answer the following questions:

1. Whether the relative performance of online cost-sensitive ensemble algorithms are consistent with that of their batch counterparts, and which ones achieve the best performance.
2. For each algorithm, how close is the performance of the online algorithm to its batch counterpart?
3. Whether the choice of base learners affects the performance and convergence of the online ensemble algorithms.
4. Though the asymptotic properties of some online ensemble algorithms have been proven, what can we observe empirically about convergence speed for the batch versus online algorithms?

6.1 Base Learners and Parameters

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 as much as possible. In Lemma 1, it is required that the base learners should be *lossless*, *proportional*, as well as *stable*, which means that given the same replicas or proportion of training examples, the base learners returned by batch and online learning algorithms should be the same, and the small changes in the distribution of the examples should not cause large changes in the base learners. In addition, the algorithms should be easy to implement and the computation cost of an update step should be low in the online learning scenario. Based on these considerations, three classifiers

TABLE 1: Parameter Setting

Algorithms	Parameters
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

TABLE 2: Summary of Data Sets Used in Experiments

Data Set	#ex	#fea	%pos; %neg	CR
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	2308	19	14.26, 85.74	6.01
Glass6	214	9	13.55, 86.45	6.38
Yeast3	1484	8	10.98, 89.02	8.11
Elico3	336	7	10.88, 89.12	8.19
Satimage	6435	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	1484	8	2.96, 97.04	32.78
Yeast6	1484	8	2.49, 97.51	39.15

are used in the experiments: linear discriminant analysis (LDA), quadratic discriminant analysis (QDA), and naive Bayes (NB) with Gaussian distribution for each feature. For all of the base learners, the parameters can be estimated incrementally and losslessly without storing the data stream [50]. Therefore, the differences of the performances of online and batch algorithms only come from the ensemble part.

There are some parameters that need to be specified. In particular, we use ten base learners for all algorithms as suggested in [17]. The other parameters of the ensemble algorithms are summarized in Table 1.

6.2 Data Sets

Eighteen data sets from the UCI repository[5] with different class ratios were selected to compare the performances of batch and online ensemble learning algorithms. Multiclass data sets were converted to binary class data sets by selecting one class to be the positive and the rest of the examples to be the negative. Table 2. summarizes the characteristics of the data sets, including the number of examples (#ex), number of features (#fea), the percentage of positive and negative examples (%pos; %neg), and the class ratio (CR).

6.3 Results

Fig. 1 demonstrates the average AUC with standard deviation for each algorithm and each base learner. At first glance, there is little difference between the different batch ensemble algorithms. Bagging algorithms achieve slightly better performance than boosting based approaches, which is consistent with the conclusions in [29]. On the other hand, the performance of online algorithms are much different. Roughly speaking the online algorithms can be divided into three groups. oUOB, oSB, oAC2, and oCSB2 with AUC larger than 0.87, oRUS2, oRUS3, oSBO1, oSBO2, oSBO3, with AUC between 0.84 and 0.87, and oRUS1 with AUC of 0.81.

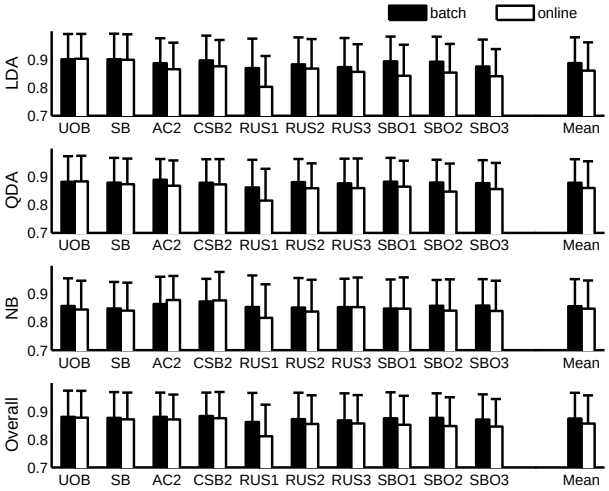

Fig. 1: Overall Performances with standard deviation

To further investigate the performance and consistency of the algorithms, the box-plot of the absolute difference of AUC between batch and online ensemble algorithms is shown in Fig. 2, and the ROCs of different algorithms with different base learners for a particular data set (satimage) are shown in Fig. 3. It can be observed that the performances of oUOB and oSB is much closer to their batch counterparts than any other algorithm. The ROCs of batch and online bagging algorithms are almost overlapped. Furthermore, this consistency exists across all data sets and all base learners. oAC2 and oCSB2 also show good consistency compared to other boosting algorithms (Note that the 75th percentiles of AdaC2 and CSB2 are smaller than for other boosting-based algorithms). We also observe that oRUS1 demonstrates worst consistency among the algorithms. Therefore, we can conclude that the relative performance of of online cost-sensitive ensemble algorithms are not necessarily consistent with the ones of their batch counterparts. While the performances of batch algorithms are similar to each other, the performances of the online ensembles appears to be mainly determined by the consistency with their batch mode counterparts.

From Fig. 1 and Fig. 2, it can be observed that LDA and QDA achieve slightly better performance in terms of average AUC than NB yet NB has a slightly better consistency. In addition, oUOB and oSB perform consistently better (in terms of convergence), yet oRUS1 consistently performs worse than other algorithms across all base

5. Download form http://sci2s.ugr.es/keel/datasets.php.

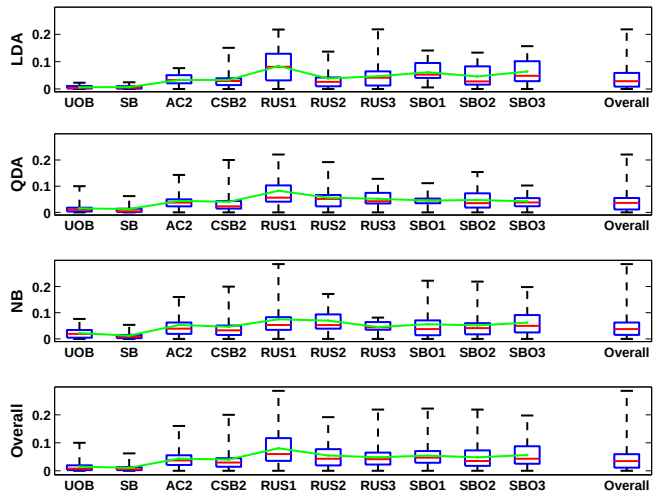

Fig. 2: Box-plot of absolute difference of AUC between batch and online ensemble algorithms

learners. Therefore, there is no strong evidence indicating the choice of base learners can significantly affect the performances or consistency of the online ensemble algorithms.

The last question is the speed of convergence of the proposed algorithms. Because of the better performances of oUOB, oSB, oAC2, oCSB2, and RUS3 in terms of both consistency and AUC, as well as the similarity between oUOB and oSB, oAC2 and oCSB2, we only discuss oUOB, oAC2 and RUS3 as the representatives of online bagging and boosting algorithms. We vary the number of examples of satimage data set from 5% to 90% used as training set, and the rest are used as testing set. The experiment for each algorithm is repeated five times, and average AUCs with standard deviation are shown in Fig. 4. It can be observed that the performance of oUOB converges to bUOB even with a very small amount of training examples. The consistency of online boosting based algorithms cannot be compared with that of UOB, but their performance gets closer to that of their batch counterparts as the number of training example grows, especially when the percentage of training examples is larger than 70%.

7 Discussion of the Results

The good consistency of bagging algorithms is not difficult to explain. The approximation of oUOB only comes from approximating a binomial distribution by a Poisson distribution. For AdaC2 and CSB2, there is another approximation, that is, the (weighted) accuracy and error. In the early stages of the learning process, the estimates of these variables can be unreliable, which can deteriorate the performances of the algorithms by improper weight update and base learner combination. For oRUS1, oRUS2, oSBO1, and oSBO2, the performance is further deteriorated by approximating N^+ and N^- by n^+ and n^-. Fig. 5 demonstrates the $werr$, $wacc$ of AdaC2 and error of RUSBoost1 on training examples during the learning process with QDA base learners for the satimage data set. The costs are set to be $C_N = 0.1$ for AdaC2, and $C = 1$ for RUSBoost1. Therefore, the performances in Fig. 5 correspond to the left upper points of the ROCs in Fig. 3. It can be observed that the performance of AdaC2 converges faster and better than that of RUSBoost1, which can explain their higher consistency for this data set.

In addition, the essential requirement of boosting is that the base learner is at least better than random guess. However, in RUSBoost and SMOTEBoost algorithms, the base learners are trained on the modified data set, but the weights are updated in the same way as the standard boosting algorithm. As a result, base learner trained on modified distribution may have an error larger than 0.5 on the original distribution. On the other hand, AdaC2 and CSB2 do not have this problem since the weights are updated in a way consistent with the target function of the base learner. Therefore, AdaC2 and CSB2 have lower chance to violate the requirement of the algorithms. Usually, for standard boosting, if the mth base learner violates the requirements, the algorithm stops and returns the ensemble of $m-1$ base learners. However, in online learning scenarios, the algorithms should not stop since as more training examples pass through the base learners, the error on training examples may become smaller than 0.5. Therefore, to make the experimental setting of batch algorithms consistent with that of online algorithms, the batch boosting algorithms were stopped even when the requirements were violated. As a result, if the error is larger than 0.5, the weights of correctly classified examples increase, whereas the weights of misclassified examples decrease[6]. This improper weight update, propagated through the base learners, may deteriorate the performances of both batch and online algorithms as well as the consistency. Fig. 6 shows the anytime performances of AdaC2 and RUSBoost3 on yeast6 with the same experimental setting as in Fig. 5. It is obvious that RUSBoost3 achieves the higher consistency than AdaC2. However, in AdaC2, there is no base learner violating the requirement $wacc > werr$, except for some base learners from the early stage of online learning, while several base learners in RUSBoost3 violate $\epsilon_m < 0.5$ in either batch or online (or even both) mode, which may hurt the performance of the algorithm.

In summary, the performances of online cost-sensitive ensemble learning algorithms are largely determined by the consistency to their batch mode counterparts and the performances of the batch ensemble algorithms. Overall, the bagging based algorithms beats all boosting based algorithms in terms of both performance and consistency. AdaC2 and CSB2 achieve comparable performance. The RUSBoost and SMOTEBoost algorithms have worse performance because of their less reliable approximation of the weight update formula, and higher chance of

6. For AdaC2 and CSB2, the requirements are $wacc_m > werr_m$ and $\frac{\epsilon_m^2}{1-\epsilon_m} < werr_m$ respectively.

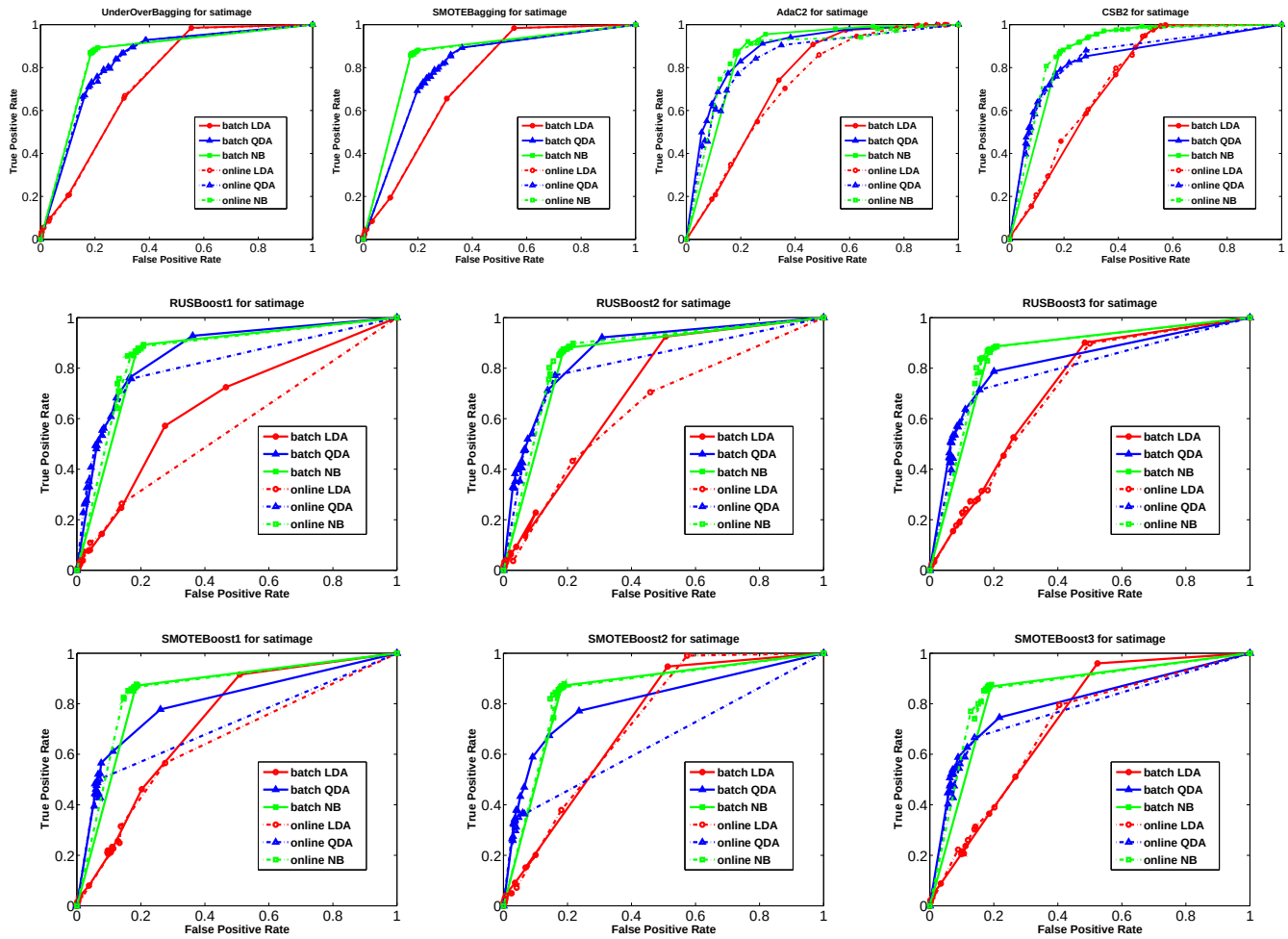

Fig. 3: ROCs of different algorithms with different base learner for satimage data set

violating the requirements of boosting than AdaC2 and CSB2.

8 Learning in Non-stationary Environments

In very recent years, learning of concept drifts from non-stationary imbalanced data streams has attracted increasing attention [18], [14], [9], [24], [23]. Besides the difficulties of standard online cost-sensitive learning, the main challenge of learning in non-stationary environments is that the temporal interval between two successive minority class examples can be quite large, and therefore they can be drawn from different distributions. As a result, it is more difficult to track the concept drifts and the evolution of decision boundary.

As a flexible framework, the online cost-sensitive ensemble learning algorithms can be easily generalized to deal with non-stationarity by leveraging existing algorithms for concept drift. In particular, these methods are developed at the data level, ensemble level, and base learner level. At the data level, the approaches select or reuse examples based on generative models or k-nearest neighbors [18], [9], [24] to select instances that are consistent with the current concept. At the ensemble level, the algorithms discard poorly performing base learners and build new ones to adapt to changes in the environments [41], [39]. Finally, at the base learner level, the online base learners themselves can be adaptive [1]. Note that the ensemble level and base learner level approaches are not originally developed for class imbalance problem, but they can be accommodated in our online cost-sensitive ensemble learning framework in a very natural way. In addition, the proposed framework can also combine with drift detection algorithms [2].

As an example, here we give a simple yet feasible solution based on online AdaC2 to learning in non-stationary environments. It should be emphasized that the purpose here is to demonstrate the flexibility of the proposed framework for dealing with non-stationarity and suggest the possible research directions in future work, rather than develop sophisticated algorithms. Therefore, the proposed method is quite straightforward and intuitive. The main idea is to put more weights on recent examples than past ones by using *forgetting factors*. For example, the mean μ and Σ of a Gaussian distribution can be

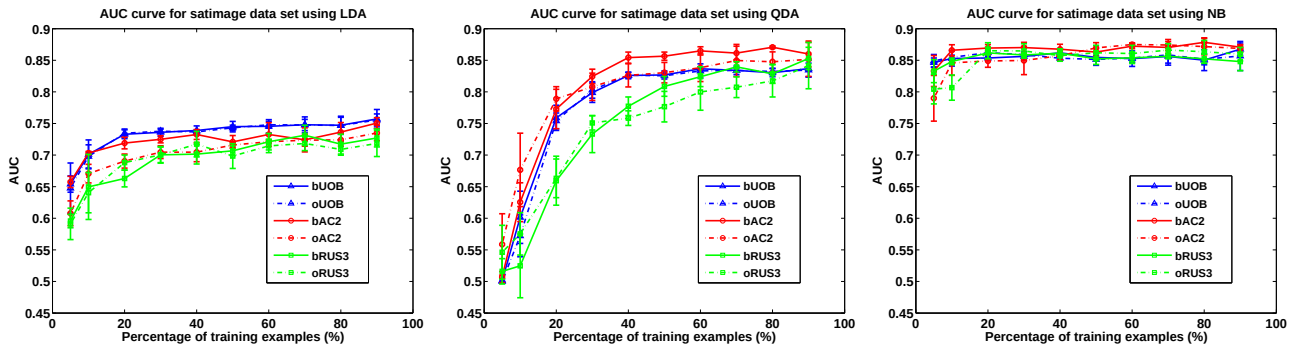

Fig. 4: Learning curves with different base learning for satimage data set

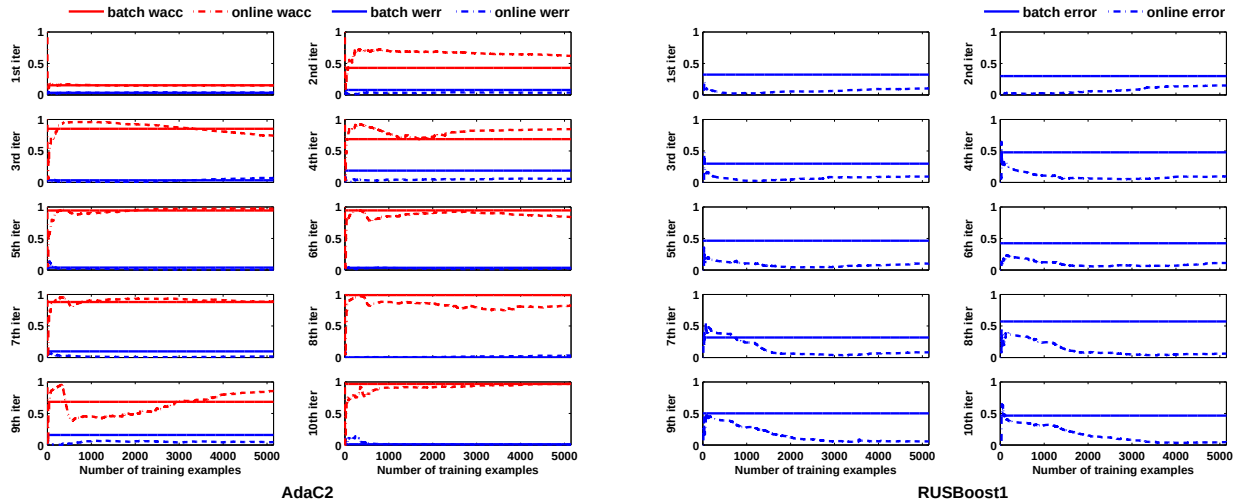

Fig. 5: Anytime performances on training examples for satimage data set

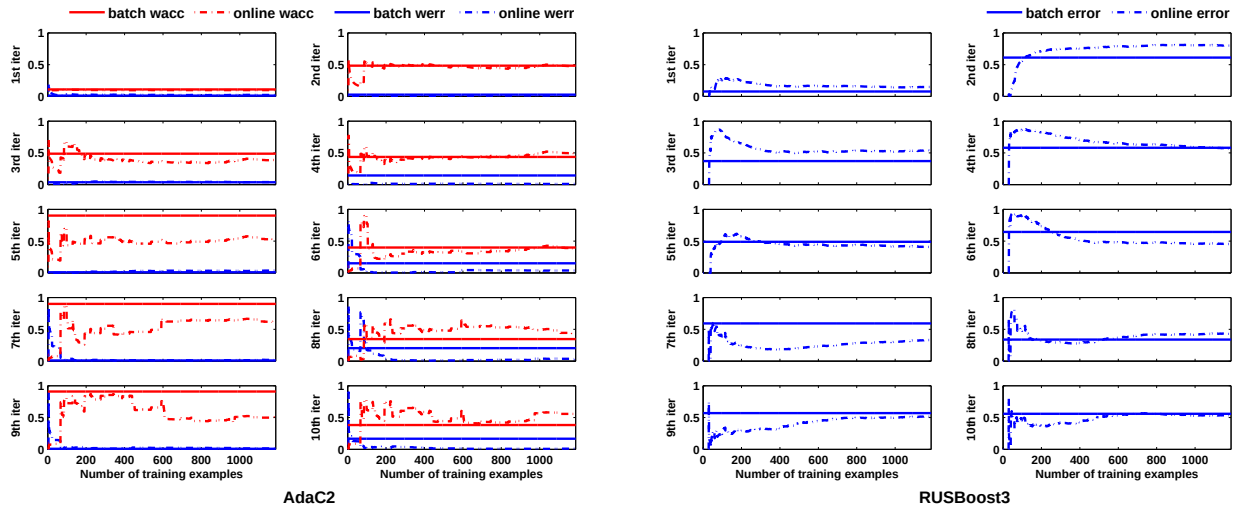

Fig. 6: Anytime performances on training examples for yeast6 data set

incrementally estimated as follows [1]:

$$\mu_n = \left(1 - \frac{1}{t_n}\right)\mu_{n-1} + \frac{1}{t_n}x_n, \quad x_0 = 0$$
$$\Pi_n = \left(1 - \frac{1}{t_n}\right)\Pi_{n-1} + \frac{1}{t_n}x_n x_n^T, \quad \Pi_0 = 0$$
$$\Sigma_n = \Pi_n - \mu_n \mu_n^T,$$

with $t_n = \beta t_{n-1} + 1$, where β is the forgetting factor. Correspondingly, we can also update the weighted Poisson parameters by $\lambda^{TP} \leftarrow \beta\lambda^{TP} + C_P\lambda$, $\lambda^{TN} \leftarrow \beta\lambda^{TN} + C_N\lambda$, $\lambda^{FP} \leftarrow \beta\lambda^{FP} + C_P\lambda$, and $\lambda^{FN} \leftarrow \beta\lambda^{FN} + C_N\lambda$ in steps 10 – 16 of Alg. 15

Three artificial data sets are used to evaluate the performance of the proposed method on three types of non-stationarity. The first two are SINE1 and SINE1G which are described in [2]. SINE1 is an unfrequent abrupt concept drift, noise-free data stream of length $N = 4000$. The data set has two relevant features with values uniformly distributed in [0,1]. In the first half of the data stream, points lying below the curve $y = sin(x)$ are classified as positive, otherwise are negative. After the midpoint, the classification is reversed. SINE1G models gradual concept drift in a noise-free data stream. The probability of selecting an example from old concept varies from 1 to 0 over the transition period of length $N = 2000$. The third data stream SINE1M is a mixture of the previous two of length $N = 4000$. The examples gradually drift from the old concept to the new one during the first half of the data stream. After the midpoint, the concept returns to the old one the process of the first half is repeated once more. In all of these three data sets, the class ratio is 90, which is highly imbalanced.

For each data stream, the experiment is repeated ten times. The average online AUC results ($\pm$ for standard deviation) are shown in Table 3, where the "ns-" algorithms are the non-stationary versions obtained by setting $\beta = 0.9$ The online AUC is generated by predicting each example before it is used for training the classifier. It can be observed that for all cases and all base learners, the simple modification of online AdaC2 can improve the overall performance, especially for SINE1. In summary, the preliminary results are encouraging, indicating that our proposed cost-sensitive online ensemble framework is sufficiently flexible to combine with other existing methods to deal with non-stationarity.

TABLE 3: Average Online AUC

	SINE1	SINE1G	SINE1M
LDA	.7747±.0408	.5716±.0975	.5547±.0435
nsLDA	.8885±.0157	.5833±.0680	.6348±.0372
QDA	.7434±.0444	.5439±.0607	.5214±.0339
nsQDA	.8720±.0229	.5944±.0637	.6048±.0355
NB	.5275±.0309	.4904±.0356	.5174±.0344
nsNB	.7270±.0436	.5498±.0598	.5571±.0368

9 Conclusions

A novel online cost-sensitive ensemble learning framework is proposed in this paper, which generalizes a number of batch cost-sensitive ensemble learning algorithms to their online version. Based on this framework, online learning and cost-sensitive learning are bridged together. The proposed framework is used to derive the online extensions of UnderOverBagging, SMOTEBagging, AdaC2, CSB2, RUSBoost, and SMOTEBoost. We demonstrate theoretically and experimentally that the proposed online cost-sensitive algorithms perform comparably with their batch mode counterparts. In particular, the bagging based algorithms achieve better performance than boosting based algorithms in terms of both consistency and AUC value. AdaC2 and CSB2 also achieve comparable performance. In addition, the proposed framework can also be combined with algorithms for concept drift to learn in non-stationary environments in a very natural way. The encouraging preliminary results on artificial data sets demonstrate that with simple extensions, the proposed algorithms can accommodate a variety of drift scenarios, regardless of whether it is abrupt, gradual, or even a mixture of two. Future work includes designing more efficient and effective algorithms for non-stationary environments with class imbalance and evaluating their performances more extensively, and applying the proposed algorithms to realistic large-scale problems.

Acknowledgment

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] C. Anagnostopoulos, D. K. Tasoulis, N. M. Adams, N. G. Pavlidis, and D. J. Hand, "Online Linear and Quadratic Discriminant Analysis with Adaptive Forgetting for Streaming Classification," *Statistical Analysis and Data Mining*, vol. 5, no. 2, pp. 139–166, 2012.

[2] M. Baena-García, J. del Campo-Ávila, R. Fidalgo, A. Bifet, R. Gavaldà, and R. Morales-Bueno, "Early Drift Detection Method," in *Proc. ECML PKDD Int'l Workshop Knowledge Discovery from Data Streams*.

[3] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[4] L. Breiman, "Bagging Predictors," *Machine Learning*, vol. 24, no. 2, pp. 123–140, 1996.

[5] G. Cauwenberghs and T. Poggio, "Incremental and Decremental Support Vector Machine Learning," *Advances in Neural Information Processing Systems*, 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. Artificial Intelligence Research*, vol. 16, no. 1, pp. 321–357, 2002.

[7] N. V. Chawla, N. Japkowicz, and A. Kotcz, "Editorial: Special Issue on Learning from Ibalanced Data Sets," *ACM SIGKDD Explorations Newsletter*, vol. 6, no. 1, pp. 1–6, 2004.

[8] N. V. Chawla, A. Lazarevic, L. O. Hall, and K. W. Bowyer, "SMOTEBoost: Improving Prediction of the Minority Class in Boosting," in *Proc. European Conf. Principles and Practice of Knowledge Discovery in Databases*, 2003, pp. 107–119.

[9] S. Chen and H. He, "Towards Incremental Learning of Nonstationary Imbalanced Data Stream: A Multiple Selectively Recursive Approach," *Evolving Systems*, vol. 2, no. 1, pp. 35–50, 2011.

[10] D. Cieslak, N. Chawla, and A. Striegel, "Combating Imbalance in Network Intrusion Datasets," in *IEEE Int'l. Conf. Granular Computing*, 2006, pp. 732–737.

[11] M. Denil, D. Matheson, and N. de Freitas, "Consistency of Online Random Forests," *Proc. Int'l Conf. Machine Learning*, 2013.

[12] C. Drummond and R. C. Holte, "Exploiting the Cost (In)Sensitivity of Decision Tree Splitting Criteria," in *Proc. Int'l Conf. Machine Learning*, 2000, pp. 239–246.

[13] C. Elkan, "The Foundations of Cost-Sensitive Learning," in *Proc. Int'l Joint Conf. Artificial Intelligence*, 2001, pp. 973–978.

[14] R. Elwell and R. Polikar, "Incremental Learning of Concept Drift in Nonstationary Environments," *IEEE Trans. Neural Networks*, vol. 22, no. 10, pp. 1517–1531, 2011.

[15] T. Fawcett, "'In Vivo' Spam Filtering: A Challenge Problem for KDD," *ACM SIGKDD Explorations Newsletter*, vol. 5, no. 2, pp. 140–148, 2003.

[16] Y. Freund and R. E. Schapire, "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting," in *J. Computer and System Sciences*, vol. 55.

[17] 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. Systems, Man and Cybernetics, Part C: Applications and Reviews*, vol. 42, no. 4, pp. 463–484, 2012.

[18] J. Gao, W. Fan, J. Han, and S. Y. Philip, "A General Framework for Mining Concept-Drifting Data Streams with Skewed Distributions," in *Proc. SIAM Int'l Conf. Data Mining*, 2007, pp. 3–14.

[19] J. Gotman, "Automatic Seizure Detection: Improvements and Evaluation," *Electroencephalography and 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. Knowledge and Data Engineering*, vol. 21, no. 9, pp. 1263–1284, 2009.

[22] S. Hido, H. Kashima, and Y. Takahashi, "Roughly Balanced Bagging for Imbalanced Data," *Statistical Analysis and 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 in Artificial Intelligence*, vol. 1, no. 1, pp. 89–101, 2012.

[24] T. R. Hoens and N. V. Chawla, "Learning in Non-Stationary Environments with Class Imbalance," in *Proc. ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining*, 2012, pp. 168–176.

[25] S. C. Hoi, R. Jin, P. Zhao, and T. Yang, "Online Multiple Kernel Classification," *Machine Learning*, vol. 90, no. 2, pp. 289–316, 2013.

[26] M. V. Joshi, V. Kumar, and R. C. Agarwal, "Evaluating Boosting Algorithms to Classify Rare Classes: Comparison and Improvements," in *Proc. Int'l Conf. Data Mining*, 2001, pp. 257–264.

[27] T. M. Khoshgoftaar, J. Van Hulse, and A. Napolitano, "Comparing Boosting and Bagging Techniques with Noisy and Imbalanced Data," *IEEE Trans. Systems, Man and Cybernetics, Part A: Systems and Humans*, vol. 41, no. 3, pp. 552–568, 2011.

[28] 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. Computer Vision and Pattern Recognition*, 2007, pp. 1–8.

[29] J. Kivinen, A. J. Smola, and R. C. Williamson, "Online Learning with Kernels," *IEEE Trans. Signal Processing*, vol. 52, no. 8, pp. 2165–2176, 2004.

[30] L. I. Kuncheva and C. J. Whitaker, "Measures of Diversity in Classifier Ensembles and Their Relationship with the Ensemble Accuracy," *Machine learning*, vol. 51, no. 2, pp. 181–207, 2003.

[31] J. Langford, L. Li, and T. Zhang, "Sparse Online Learning via Truncated Gradient," *J. Machine Learning Research*, vol. 10, pp. 777–801, 2009.

[32] P. Laskov, C. Gehl, S. Krüger, and K.-R. Müller, "Incremental Support Vector Learning: Analysis, Implementation and Applications," *J. Machine Learning Research*, vol. 7, pp. 1909–1936, 2006.

[33] Y. Lin, Y. Lee, and G. Wahba, "Support Vector Machines for Classification in Nonstandard Situations," *Machine Learning*, vol. 46, no. 1-3, pp. 191–202, 2002.

[34] L.-P. Liu, Y. Jiang, and Z.-H. Zhou, "Least Square Incremental Linear Discriminant Analysis," in *Proc. Int'l Conf. Data Mining*, 2009, pp. 298–306.

[35] H. Masnadi-Shirazi and N. Vasconcelos, "Risk Minimization, Probability Elicitation, and Cost-Sensitive SVMs," in *Proc. Int'l Conf. Machine Learning*, 2010, pp. 759–766.

[36] N. C. Oza and S. Russell, "Online Bagging and Boosting," in *Proc. Artificial Intelligence and Statistics*, pp. 105–112,.

[37] N. C. Oza, "Online Ensemble Learning," Ph.D. dissertation, University of 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] A. Pocock, P. Yiapanis, J. Singer, M. Luján, and G. Brown, "Online Non-Stationary Boosting," in *Proc. Int'l Workshop on Multiple Classifier Systems*, 2010, pp. 205–214.

[40] R. Polikar, "Ensemble Based Systems in Decision Making," *IEEE Circuits and Systems Magazine*, vol. 6, no. 3, pp. 21–45, 2006.

[41] A. Saffari, C. Leistner, J. Santner, M. Godec, and H. Bischof, "Online Random Forests," in *Proc. Online Learning for Computer Vision Workshop*, 2009, pp. 1393–1400.

[42] C. Seiffert, T. M. Khoshgoftaar, J. Van Hulse, and A. Napolitano, "RUSBoost: A Hybrid Approach to Alleviating Class Imbalance," *IEEE Trans. Systems, Man and Cybernetics, Part A: Systems and Humans*, vol. 40, no. 1, pp. 185–197, 2010.

[43] V. S. Sheng and C. X. Ling, "Roulette Sampling for Cost-Sensitive Learning," in *Proc. European Conf. Machine Learning*, 2007, pp. 724–731.

[44] Y. Sun, M. S. Kamel, A. K. Wong, and Y. Wang, "Cost-Sensitive Boosting for Classification of Imbalanced Data," *Pattern Recognition*, vol. 40, no. 12, pp. 3358–3378, 2007.

[45] K. M. Ting, "A Comparative Study of Cost-Sensitive Boosting Algorithms," in *Proc. Int'l Conf. Machine Learning*, 2000.

[46] P. E. Utgoff, N. C. Berkman, and J. A. Clouse, "Decision Tree Induction Based on Efficient Tree Restructuring," *Machine Learning*, vol. 29, no. 1, pp. 5–44, 1997.

[47] S. Wang and X. Yao, "Diversity Analysis on Imbalanced Data Sets by Using Ensemble Models," in *Proc. IEEE Symp. Computational Intelligence and Data Mining*, 2009, pp. 324–331.

[48] G. Wu and E. Y. Chang, "Adaptive Feature-Space Conformal Transformation for Imbalanced-Data Learning," in *Proc. Int'l Conf. Machine Learning*, 2003, pp. 816–823.

[49] B. Zadrozny and C. Elkan, "Learning and Making Decisions When Costs and Probabilities Are Both Unknown," in *Proc. ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining*, 2001, pp. 204–213.