From Online to Non-i.i.d. Batch Learning

Yufei Tao
Chinese University of Hong Kong
taoyf@cse.cuhk.edu.hk

Shangqi Lu
Chinese University of Hong Kong
sqlu@cse.cuhk.edu.hk

ABSTRACT

This paper initializes the study of *online-to-batch conversion* when the samples in batch learning are *not* i.i.d. Our motivation originated from two facts. First, sample sets in reality are seldom i.i.d., thus preventing the application of the existing conversions. Second, the online model of learning permits an *adversarial* stream of samples that almost for sure violates the i.i.d. assumption, raising the possibility of adapting an online algorithm effectively to learn from a non-i.i.d. sample set. We present a set of techniques to utilize an online algorithm as a *black box* to perform batch learning in the absence of the i.i.d. assumption. Our techniques are generic, and are applicable to virtually any online algorithms on classification. This provides strong evidence that the great variety of known algorithms in the online-learning literature can indeed be harnessed to learn from sufficiently-representative non-i.i.d. samples.

CCS CONCEPTS

• **Theory of computation → Machine learning theory**.

KEYWORDS

Online-to-Batch Conversion; Learning Algorithms; Classification

ACM Reference Format:
Yufei Tao and Shangqi Lu. 2020. From Online to Non-i.i.d. Batch Learning. In *Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '20), August 23–27, 2020, Virtual Event, CA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3394486.3403075

1 INTRODUCTION

Batch (a.k.a., offline) learning and online learning are *the* two most popular models in machine learning. In batch learning, a learner algorithm is provided a (reasonably large) training set *all at once*, and produces a model deployed to make predictions on future samples. Provided that the training set is *sufficiently representative* of the distribution from which future samples are drawn, the model output by a good algorithm should strike high accuracy in its predictions.

In online learning, on the other hand, a learner algorithm is given *one* sample at a time, but is required to make a prediction *right away*. Only after the prediction has been made will the algorithm be told the correct answer. In case the answer is different from the prediction, the algorithm makes a *mistake*. As the procedure continues, the algorithm must learn from the history — why mistakes were made on some samples but not others — in order to avoid future mistakes as much as possible.

These two forms of learning suit different sets of applications. Batch learning is appropriate when training data are abundant and the underlying distribution is relatively stable. One example is *face recognition*, where samples are easy to acquire and faces of the same individual do not incur drastic changes. In contrast, online learning should be performed when urgent decisions must be made before sufficient training data can be gathered, or the target distribution is drifting or even unknown. A typical scenario is *weather forecast*, where each day gives only one new sample, yet a prediction must be rendered before the answer becomes available, not to mention that the "distribution" of weather is still abstruse till this day.

The literature of machine learning harbors two families of techniques developed in each model. They are not "equal" because standard methods exist to *convert* an online algorithm to work in the batch model (the reverse is impossible [4]). Significantly, this makes the entire online family exploitable for batch learning. Many online algorithms are light weighted yet possess interesting theoretical guarantees; these nice properties are usually inherited by their batch versions. The *Perceptron* [37] and *winnow* [27] algorithms are perhaps the most classic examples of this kind. Originally designed for online *linear classification*, ideas of the two algorithms have permeated into a great variety of batch learning based on convex optimization and weighted voting. It has been argued recently [20] that online learning will gain even more momentum in the big data era as the need of real-time stream mining gets strengthened.

Unfortunately, the existing *online-to-batch conversions* (Section 3) all assume that the training set of batch learning is collected in an i.i.d. (independently and identically distributed) manner. The assumption is inherent in PAC-learning, although it is an open secret that training samples are seldom independent in practice. There has been a lack of efforts to remedy the issue. This is understandable because theoretical analysis in learning typically requires *generalization theorems* to relate a model's empirical error (on training samples) to its prediction error (on future samples). Traditionally, such theorems were established under the i.i.d. assumption, which ties the related studies to the assumption as well.

The situation has changed dramatically. Recent years have witnessed considerable progress in proving generalization theorems under the *non-i.i.d. model* (Sections 2.1 and 3), amid the substantial on-going efforts towards alleviating the i.i.d. assumption. However, the previous online-to-batch conversion methods all lose their mathematical rigor in the non-i.i.d. model. As a consequence, the large "treasure trove" of online algorithms is yet to be made applicable to batch learning with non-i.i.d. samples.

Online-to-batch conversion without the i.i.d. assumption is an intriguing topic on its own. Intuitively, the assumption does not seem necessary since, in the online model, the incoming samples are not required to be independent at all! In fact, they can even be

KDD '20, August 23–27, 2020, Virtual Event, CA, USA

ACM ISBN 978-1-4503-7998-4/20/08…$15.00
https://doi.org/10.1145/3394486.3403075

adversarial, namely, maliciously chosen to thwart the learner algorithm, whereas a robust algorithm must still work well nonetheless. This fact actually has not been reflected in the previous research because the i.i.d. assumption is an indigenous property of PAC-learning. Therefore, only in the non-i.i.d. model would the essence of online-to-batch conversion be revealed.

In this paper, we develop the first suite of techniques to convert an online algorithm for batch learning on a non-i.i.d. sample set. Our techniques are *generic* because they are applicable to any classification algorithm. In fact, the algorithm does not even need to possess strong theoretical guarantees; instead, it can be a heuristic algorithm, but as long as it works well on the *given* training data, it can be deployed for effective batch learning. This owes to a new concept called *worst-permutation mistake bound*, by virtue of which we prove instance-sensitive (i.e., stronger than worst-case) theoretical bounds. Besides ensuring rigorous guarantees, the proposed algorithms are easy to understand and implement, and perform very well in practice as demonstrated in our experimentation.

The rest of the paper is organized as follows. Section 2 sets up the stage for our study by formally describing the problem. Section 3 reviews the previous research directly related to this work. Sections 4-6 develop the proposed algorithms by gradually unfolding the technical details. Section 7 presents an extensive experimental evaluation using real data, while finally Section 8 concludes the paper with a summary of findings.

2 PROBLEM FORMALIZATION

We will start by reviewing in Section 2.1 the learning models assumed by this work. After that, we will formalize in Section 2.2 the framework of online-to-batch conversion in the absence of the i.i.d. assumption. Finally, we will instantiate in Section 2.3 our general framework into two representative problems, illustrating the relevant concepts in concrete contexts.

2.1 Models of learning

Let $\mathcal{X}$ be a set where each element is an *instance*. Define $\mathcal{Y}$ to be another set where each element is a *label*. We will refer to $\mathcal{X}$ as the *instance space*, and $\mathcal{Y}$ as the *label space*. Note that $\mathcal{X}$ may be uncountably infinite, but $\mathcal{Y}$ is a finite set.

A *hypothesis* (a.k.a. *concept* or *classifier*) h is a function from $\mathcal{X}$ to $\mathcal{Y}$. In other words, h maps each instance $x \in \mathcal{X}$ to a label $h(x) \in \mathcal{Y}$. Denote by $\mathbb{H}$ the set of all possible hypotheses; $\mathbb{H}$ can be uncountably infinite. The objective of learning is to discover a hypothesis in $\mathbb{H}$ that can predict labels from instances accurately. What this means and how it can be achieved depend on the learning model, as defined next.

The online model. In this model, an algorithm $\mathscr{A}$ interacts with an *oracle* (sometimes referred to as the *environment*) in rounds. $\mathscr{A}$ holds a classifier h_{now} at all times (the initial h_{now} is arbitrary), while in each round:

1. the oracle selects a pair (x, y) from a certain $\mathcal{U} \subseteq \mathcal{X} \times \mathcal{Y}$ where the subset $\mathcal{U}$ is known to $\mathscr{A}$
2. the oracle reveals x to $\mathscr{A}$
3. $\mathscr{A}$ outputs $\hat{y} = h_{now}(x)$ as the predicted label for x
4. the oracle then reveals the (real) label y
5. $\mathscr{A}$ changes h_{now} if $y \neq \hat{y}$

$\mathscr{A}$ is said to make a *mistake* if $y \neq \hat{y}$. The goal of $\mathscr{A}$ is to make the fewest mistakes as the rounds proceed indefinitely.

It is important to note that the oracle does not need to choose the next pair (x, y) independently of the previous pairs. In fact, the oracle may purposely harm $\mathscr{A}$ by selecting the pair maliciously. Ideally, $\mathscr{A}$ should do well even on the worst stream of pairs.

PAC-learning. In this model, the oracle chooses a distribution $\mathcal{D}$ over some $\mathcal{U} \subseteq \mathcal{X} \times \mathcal{Y}$. $\mathcal{U}$ is known to the algorithm, but $\mathcal{D}$ is hidden. For a hypothesis $h \in \mathbb{H}$, its *error* is

$$\mathrm{err}(h) \quad = \quad \Pr_{(x,y)\sim\mathcal{D}}[h(x) \neq y] \tag{1}$$

namely, if we *sample* a pair (x, y) according to $\mathcal{D}$, how often would h *mis-predict* the label y. The goal of an algorithm $\mathscr{A}$ is to find a hypothesis with a small error. $\mathscr{A}$ works by drawing from the oracle a set $S = \{(x_i, y_i) \mid 1 \leq i \leq n\}$ of *independent* samples according to $\mathcal{D}$, and trying to find a good hypothesis using S.

For each hypothesis $h \in \mathbb{H}$, define its *empirical error* on S as[1]

$$\hat{\mathrm{err}}_S(h) \quad = \quad \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}_{h(x_i)\neq y_i}. \tag{2}$$

It is known — by *generalization theorems* (see, for example, Chapters 2 and 3 of [35]) — that with a large probability, for *every* hypothesis $h \in \mathbb{H}$, the absolute difference $|\mathrm{err}(h) - \hat{\mathrm{err}}_S(h)|$ is bounded by an *additive* factor that approaches 0 as n tends to ∞.[2] This motivates the *empirical risk minimization* (ERM) method, which returns a hypothesis h that has a small empirical error $\hat{\mathrm{err}}_S(h)$.

The non-i.i.d. model. This model generalizes PAC-learning by dropping the requirement that S should be independent.

Consider again a distribution $\mathcal{D}$ on $\mathcal{U} \subseteq \mathcal{X} \times \mathcal{Y}$. The error $\mathrm{err}(h)$ of a hypothesis $h \in \mathbb{H}$ is still defined in (1). As before, an algorithm $\mathscr{A}$ may draw a set S of samples, each of which is distributed according to $\mathcal{D}$. The difference, however, is that the $n = |S|$ samples *are not guaranteed independent*. Instead, the generation of those samples obeys a certain set $\mathscr{R}$ of *dependence rules*, which in general governs how correlation can be injected, and is what differentiates specific learning problems. $\mathscr{A}$ is not aware of $\mathcal{D}$ and $\mathscr{R}$ (but $\mathscr{A}$ knows $\mathcal{U}$); its goal is still to find an h from S to minimize $\mathrm{err}(h)$.

Once S is ready, the empirical error $\hat{err}_S(h)$ of a hypothesis $h \in \mathbb{H}$ is still given by (2). Generalization theorems akin to those in PAC-learning have been established for many types of $\mathscr{R}$ (a survey will appear in Section 3). In other words, it holds with a large probability that, for every $h \in \mathbb{H}$, $\mathrm{err}(h)$ can be higher than $\hat{\mathrm{err}}_S(h)$ by only an additive factor that vanishes as n approaches ∞. This again justifies the ERM method of returning a hypothesis h with small $\hat{err}_S(h)$.

2.2 Online to non-i.i.d. batch learning

We are ready to formulate the main problem studied in this work. Fix an online algorithm $\mathscr{A}_{online}$ designed for a learning problem $\mathscr{P}_{online}$ characterized by $\mathcal{X}, \mathcal{Y}, \mathcal{U}$, and $\mathbb{H}$. Let $\mathscr{P}_{noniid}$ be a learning problem in the non-i.i.d. model characterized by (i) the same $\mathcal{X}, \mathcal{Y}, \mathcal{U}$, and $\mathbb{H}$ as $\mathscr{P}_{online}$, and (ii) its own distribution $\mathcal{D}$ and set of dependent rules $\mathscr{R}$. Let S be a set of n samples drawn for $\mathscr{P}_{noniid}$ (remember that

[1] In general, $\mathbb{1}_\pi$ equals 1 if the predicate π holds, or 0 otherwise.

[2] One provable factor of this kind is $\tilde{O}(\sqrt{\lambda/n})$ where λ is the so-called VC-dimension of $\mathbb{H}$, and a constant independent of n. The notation $\tilde{O}(.)$ hides polylogarithmic terms.

the samples may not be independent). We want to deploy $\mathscr{A}_{online}$ to find a hypothesis with small empirical error on S.

To continue the formulation, we introduce the concept of *worst-permutation mistake bound*, which will be important to our results. Fix an arbitrary permutation Π of the samples in S; suppose without loss of generality that Π is the sequence $(x_1, y_1), ..., (x_n, y_n)$. Imagine running $\mathscr{A}_{online}$ on Π for problem $\mathscr{P}_{online}$, i.e., *as if the oracle had chosen* Π *as the sequence of instance-label pairs.* Denote by M_Π the number of mistakes that are made by $\mathscr{A}_{online}$ on Π. Then:

Definition 1. *The* ***worst-permutation (WP) mistake bound*** *of* $\mathscr{A}_{online}$ *on* S *is* $\max_\Pi M_\Pi$*, where the maximization is over all the* $n!$ *permutations of* S*.*

We will use M to denote the above WP mistake bound (remember: M depends on the sample set S). M measures the *robustness* of $\mathscr{A}_{online}$ with respect to S. The problem studied in this work can be stated as:

Problem 1. *Without knowing the value of* M*, deploy* $\mathscr{A}_{online}$ *as a black box to find a hypothesis* $h \in \mathbb{H}$ *whose empirical error on* S *is provably small, provided that* $M \ll n$*.*

Remark 1 ($\mathscr{R}$-oblivious). Our formulation deals with S directly, and has nothing to do with the dependence rules in $\mathscr{R}$. A solution to the problem must be applicable to all learning problems in the non-i.i.d. model, regardless of how the samples are correlated. This is important as in practice $\mathscr{R}$ may not be known.

Remark 2 ($\mathscr{A}$ must be good). The requirement $M \ll n$ is indispensable. Consider the extreme case $M = n$; hence, $\mathscr{A}$ is no better than the *don't-care* algorithm that simply predicts all labels as 1. The don't-care algorithm is useless for batch learning, and so is $\mathscr{A}$.

Remark 3 ($\mathscr{A}$ can be empirically good). Every $\mathscr{A}_{online}$ has a WP mistake bound M on every S. This is true even if $\mathscr{A}_{online}$ is heuristic and promises no strong guarantees in general. A solution to Problem 1 permits us to leverage $\mathscr{A}_{online}$ *whenever* its bound M on the *given* S is small.

Remark 4 (computational efficiency). We will be interested in only solutions demanding polynomial computation time. Details will be elaborated when they become necessary.

2.3 Two representative instantiations

Online learning has been extensively studied for a great variety of problems (Section 3). Next, we will look at two problems, and discuss how the proposed framework gets instantiated in each.

Selection of the two problems is based on the following thoughts: (i) the first, *conjunction of boolean literals*, is one of the simplest online problems, and enables concept illustration with the least technical details; (ii) the second problem, *linear classification*, bears significant importance in machine learning; (iii) the first problem demonstrates *consistent* classification, while the second exemplifies *inconsistent* classification; and (iv) finally, their WP mistake bounds have drastically different styles.

Conjunction of boolean literals. Consider d boolean variables $x_1, x_2, ..., x_d$. A *literal* is as either x_i or its negation $\overline{x_i}$ where $i \in [1, d]$. An *expression* is the conjunction of a non-empty set of literals. For example, x_1, $\overline{x_2}$, and $x_1 \wedge \overline{x_2}$ are all expressions, but $x_1 \vee x_2$ is not, and neither is $(x_1 \vee x_2) \wedge x_3$.

Define the instance space $\mathcal{X} = \{0, 1\}^d$. Note that each element $x \in \mathcal{X}$ is a d-dimension boolean vector $x = (x_1, x_2, ..., x_d)$. Define the label space $\mathcal{Y} = \{-1, 1\}$, i.e., binary classification. Let $\mathbb{H}$ be the set of all expressions. For each expression (i.e., hypothesis) $h \in \mathbb{H}$, $h(x)$ equals 1 if evaluating the expression h on x gives 1, or -1 if the evaluation gives 0.

In the online model, the oracle (secretly) chooses an expression $h^* \in \mathbb{H}$, and sets $\mathcal{U} = \{(x, h^*(x)) \mid x \in \mathcal{X}\}$. In other words, every instance-label pair (x, y) that the oracle subsequently picks must be *consistent* with h^* (hence, *consistent* classification).

Consider an online algorithm in the above scenario. There are 3^d meaningful expressions (three choices per variable: itself, negation, or absence). A naive algorithm would make at most 3^d mistakes, before narrowing down to h^*. It is, however, possible to make at most $d + 1$ mistakes[3]; let $\mathscr{A}_{online}$ denote that algorithm.

We now turn attention to the non-i.i.d. model. The oracle chooses a distribution $\mathcal{D}$ over $\mathcal{U}$ and a certain set $\mathscr{R}$ of dependence rules. We draw a set S of (non-i.i.d.) samples from the oracle, and aim to use $\mathscr{A}_{online}$ to find an expression $h \in \mathbb{H}$ with small empirical error $\hat{err}_S(h)$. Clearly, $\mathscr{A}_{online}$ has a WP mistake bound of $M \leq d + 1$. When $d \ll n$, we must guarantee that $\hat{err}_S(h)$ be provably small.

As a remark, the bound $d + 1$ is *data-independent*, i.e., it holds for all sample sets S.

Linear classification. Consider $\mathbb{R}^d$ (with $\mathbb{R}$ being the real domain) where each element x is a d-dimensional point. We will represent x as a d-dimensional vector $\boldsymbol{x}$, which lists out the point's coordinates on dimensions 1, 2, ..., d, respectively (in this order). Denote by $|\boldsymbol{x}|$ the Euclidean norm of $\boldsymbol{x}$ (i.e., the distance between point x and the origin). Set the instance space $\mathcal{X} = \{\boldsymbol{x} \in \mathbb{R}^d \mid |\boldsymbol{x}| \leq 1\}$, and the label space $\mathcal{Y} = \{-1, 1\}$.

Define $\mathbb{H}$ as the set of *linear classifiers*. Specifically, each classifier (a.k.a., hypothesis) $h \in \mathbb{H}$ is described by a d-dimensional vector $\boldsymbol{w}$ such that $h(\boldsymbol{x})$ equals 1 if $\boldsymbol{w} \cdot \boldsymbol{x} \geq 0$, or -1 otherwise.

In the online model, the oracle sets $\mathcal{U}$ to the entire $\mathcal{X} \times \mathcal{Y}$. As such, no classifier in $\mathbb{H}$ is consistent with all the pairs in $\mathcal{U}$. In other words, for any $h \in \mathbb{H}$, there is at least one $(\boldsymbol{x}, y) \in \mathcal{U}$ such that $h(\boldsymbol{x}) \neq y$ (i.e., *inconsistent* classification).

No online algorithm can have a data-independent bound on the number of mistakes[4]. However, there exist algorithms achieving data-*dependent* bounds, namely, the number of mistakes is sensitive to the actual sequence Π of instance-label pairs received from the oracle. To explain, suppose that Π has length n, and that the i-th pair in Π is $(\boldsymbol{x}_i, y_i)$. Fix a d-dimensional vector $\boldsymbol{w} \neq \boldsymbol{0}$ and a real value $\gamma > 0$, each pair $(\boldsymbol{x}, y) \in \mathcal{U}$ defines a *hinge loss*:

$$L_{\boldsymbol{w},\gamma}(\boldsymbol{x}, y) \quad = \quad \max\left\{0, \gamma - y \cdot \frac{\boldsymbol{x} \cdot \boldsymbol{w}}{|\boldsymbol{w}|}\right\}. \tag{3}$$

The well-known algorithm *Perceptron* [37] makes a number of mistakes that can be bounded using the hinge losses of the points in Π. One upper bound (Theorem 5.11 of [7]) is:

$$\min_{\boldsymbol{w},\gamma}\left(\frac{1}{\gamma^2} + \frac{2}{\gamma}\sum_{i=1}^{n} L_{\boldsymbol{w},\gamma}(\boldsymbol{x}_i, y_i)\right) \tag{4}$$

[3] See the lecture notes at www.cs.cmu.edu/~avrim/ML98/lect0114

[4] The oracle can always make an algorithm err in every round, if the oracle knows the algorithm.

where the minimization is over all $\boldsymbol{w} \neq \boldsymbol{0}$ and all $\gamma > 0$. Note that the above bound is *insensitive* to the ordering of the points in Π. See also [13] for another similar bound.

In the non-i.i.d. model, after choosing a distribution $\mathcal{D}$ over $\mathcal{U} = \mathcal{X} \times \mathcal{Y}$ and certain dependence rules $\mathscr{R}$, the oracle provides us with a set S of (non-i.i.d.) n samples $(\boldsymbol{x}_1, y_1), ..., (\boldsymbol{x}_n, y_n)$. We want to deploy $\mathscr{A}_{online}$ = *Perceptron* to find a linear classifier h with small empirical error $\hat{err}_S(h)$. By the earlier discussion, $\mathscr{A}_{online}$ enjoys a WP mistake bound M that is at most (4). As before, when $M \ll n$, our mission is to ensure provably low $\hat{err}_S(h)$.

3 RELATED WORK

Online (classification) learning. The online model in Section 2.1 is known as the *mistake bound model* [27] (which is identical to the *equivalence query model* [1]). Theoretically, the chief objective is to understand how many mistakes are compulsory for the best algorithm designed to solve a given learning problem. Significant progress has been achieved — including algorithms with non-trivial upper bounds and proofs of lower bounds — for many problems of fundamental nature, most notably: *k-CNF* (the *conjunction of boolean literals* problem in Section 2.3 is 1-CNF) [17, 27], *k-DNF* [6, 17, 27, 43], *k-decision list* [18, 24, 29, 36, 38], *k-parities* [2, 3, 8, 24], *linear classification* [13, 16, 26, 27, 29, 40], and so on. See [5, 39] for additional problems on which interesting results are known.

Practically, the main focus of online (classification) learning has been to develop algorithms that function nicely on *realistic* streams of instance-label pairs (i.e., those likely to be encountered in reality), although not necessarily endowed with a strong theoretical bound on the worst stream. There is a very rich literature on solutions of this sort; we refer the interested readers to the excellent surveys of [20, 21, 30].

Online-to-batch conversion with the i.i.d. assumption. Such conversion has been well understood for PAC-learning, i.e., when the i.i.d. assumption *does* hold. Although the existing methods [9–12, 23, 28, 46] differ considerably in details, their high-level intuition can be grasped rather easily, as explained next.

Let S be a set of *i.i.d.* samples $(x_1, y_1), ..., (x_n, y_n)$. Given an online algorithm $\mathscr{A}_{online}$, let us run it on the sequence $(x_1, y_1), ..., (x_n, y_n)$. Recall from Section 2.1 that $\mathscr{A}_{online}$ maintains a hypothesis h_{now} at all times; therefore, the above execution produces $n+1$ hypotheses $h_0, h_1, ..., h_n$, where h_0 is the initial h_{now} before $\mathscr{A}_{online}$ starts, and h_i ($i \in [1, n]$) is the h_{now} after processing sample (x_i, y_i). As a crucial observation behind all the existing methods, the probability that h_i ($i \in [0, n-1]$) mis-predicts the next sample (x_{i+1}, y_{i+1}) is *exactly* $\text{err}(h_i)$ since every sample is i.i.d. Therefore, $\sum_{i=0}^{n-1} \text{err}(h_i)$ is the expected number of mistakes $\mathscr{A}_{online}$ makes on $(x_1, y_1), ..., (x_n, y_n)$. This relationship provides vital clues on how a good hypothesis can be picked from $h_0, ..., h_{n-1}$.

Unfortunately, the above observation no longer holds in the non-i.i.d. model. Overcoming the obstacle calls for new ideas, as is the technical motivation of our paper.

Learning in the non-i.i.d. model. The main difficulty in getting rid of the i.i.d. assumption lies in figuring out how to relax the assumption to such an extent that allows rigorous theoretical analysis, yet in the meantime can be satisfied by practical applications. Currently, the most successful relaxation is based on the theory of *stationary* random variables, on which various generalization theorems have been proved; see [15, 32–34, 44, 45] and the references therein. Relaxations departing from the stationary theory have also been attempted, e.g., [22, 31, 41, 42], although generalization theorems appeared only recently [25].

Any of the generalization theorems mentioned earlier can be combined with the results of this paper to upper bound the $\text{err}(h)$ of the hypothesis h returned by our techniques, as discussed in Section 2.1.

4 THE FIRST SOLUTION

This section will present our first solution to Problem 1. Remember that we are given (i) an online algorithm $\mathscr{A}_{online}$ designed for a problem characterized by $\mathcal{X}, \mathcal{Y}, \mathcal{U}$, and $\mathbb{H}$, and (ii) a non-i.i.d. sample set S of size n where each sample comes from $\mathcal{U}$. We propose the following algorithm to find a hypothesis $h \in \mathbb{H}$ using $\mathscr{A}_{online}$ and S:

```
Algorithm I
1. initialize $\mathscr{A}_{online}$
2. repeat
   /* an iteration */
3.     for each pair $(x, y) \in S$ do
4.         feed $(x, y)$ to $\mathscr{A}_{online}$ to obtain the predicted label $\hat{y}$
5.         if $y \neq \hat{y}$ then remove $(x, y)$ from $S$
6. until no mistake was made in this iteration
7. return the current $h_{now}$ of $\mathscr{A}_{online}$
```

The above strategy launches a series of *iterations*. Each iteration, except the last one, shrinks S by deleting some instance-label pairs. Specifically, at the beginning of an iteration, we order the pairs of S arbitrarily, and then run $\mathscr{A}_{online}$ on S by the ordering. For each pair $(x, y) \in S$, $\mathscr{A}_{online}$ makes a label prediction $\hat{y}$. If the prediction is correct, (x, y) is retained; otherwise, we delete the pair from S, shrinking $|S|$ by 1. The whole process finishes if $\mathscr{A}_{online}$ makes no mistakes in the current iteration; otherwise, a new iteration is performed. Throughout Algorithm I, $\mathscr{A}_{online}$ is initialized only once (at Line 1). In particular, the execution of $\mathscr{A}_{online}$ in a new iteration *continues* the state of $\mathscr{A}_{online}$ left over from the previous iteration. The output is the final hypothesis h_{now} held by $\mathscr{A}_{online}$.

Theorem 1. *The output hypothesis h of Algorithm I has empirical error $\hat{err}_S(h)$ at most M/n, where M is the WP mistake bound of $\mathscr{A}_{online}$ on S.*

Proof. Instance-label pairs are deleted from S in succession. Let Σ be the *sequence* by which those pairs are deleted. By how Algorithm I runs, we know that h (which is the h_{now} of $\mathscr{A}_{online}$ at the end) correctly predicts all the pairs of $S \setminus \Sigma$. In other words, $\hat{err}_S(h) \leq |\Sigma|/n$.[5]

It remains to prove $|\Sigma| \leq M$. Construct a permutation Π of S by concatenating Σ with an *arbitrary* permutation of $S \setminus \Sigma$. Consider executing $\mathscr{A}_{online}$ in the online model as if the oracle had chosen Π as the stream of instance-label pairs. Observe that, at the moment right after $\mathscr{A}_{online}$ has finished processing Σ, its hypothesis h_{now} is exactly h. This implies no more mistakes by $\mathscr{A}_{online}$ in the rest of the execution. Therefore, the total number of mistakes on Π is exactly $|\Sigma|$, which by Definition 1 is at most M. □

[5] Note that $\hat{err}_S(h)$ can be strictly less than $|\Sigma|/n$ because h may be correct on some of the pairs already removed.

Regarding computation time, the proof of Theorem 1 suggests that there can be at most $M + 1$ iterations, because each iteration except the last one increases $|\Sigma|$ by 1, while $|\Sigma|$ is bounded by M. Each iteration obviously processes at most $|S| \le n$ pairs. Therefore, the total CPU time is $O(TnM)$, where T is the amount of time required by $\mathscr{A}_{online}$ to process one pair. The time complexity is polynomial as long as T is polynomial.

5 A DUPLICATION TECHNIQUE

This section will present an algorithm with a more refined guarantee. Again, consider an online algorithm $\mathscr{A}_{online}$ for a problem characterized by $\mathcal{X}, \mathcal{Y}, \mathcal{U}$, and $\mathbb{H}$.

Definition 2. *Let*

- *M_0 be a fixed non-negative real value;*
- *M_1 be a function that maps $\mathcal{U}$ to non-negative real values.*

*$\mathscr{A}_{online}$ is **(M_0, M_1)-bounded** if, for any set S of instance-label pairs drawn from U, $\mathscr{A}_{online}$ has a WP mistake bound M satisfying*

$$M \quad \le \quad M_0 + \sum_{(x,y) \in S} M_1(x, y). \tag{5}$$

Phrased differently, if $\mathscr{A}_{online}$ is (M_0, M_1)-bounded, the WP mistake bound M of $\mathscr{A}_{online}$ on every (non-i.i.d.) sample set S is at most the sum of (i) value M_0, and (ii) a non-negative contribution $M_1(x, y)$ from each $(x, y) \in S$. The discussion below further illustrates this using the two representative problems from Section 2.3:

Example. For *conjunction of boolean literals*, we noted an algorithm $\mathscr{A}_{online}$ with a data-independent mistake bound $d + 1$ on any S. $\mathscr{A}_{online}$ is (M_0, M_1)-bounded with $M_0 = d + 1$, and $M_1(x, y) = 0$.

For *linear classification*, we noted that *Perceptron* has a data-dependent mistake bound given in (4). The algorithm is therefore (M_0, M_1)-bounded where M_0 and M_1 are defined by a d-dimensional vector $\boldsymbol{w} \neq \boldsymbol{0}$ and a real value $\gamma > 0$:

$$\begin{aligned} M_0 &= 1/\gamma^2 \\ M_1(x, y) &= L_{\boldsymbol{w},\gamma}(x, y) \end{aligned}$$

where $L_{\boldsymbol{w},\gamma}(x, y)$ is given in (3). Notice that any $(\boldsymbol{w}, \gamma)$ gives a pair of workable (M_0, M_1)! Therefore, for a *specific* S, one can actually select the *best* $(\boldsymbol{w}, \gamma)$ that yields the *lowest* $M_0 + \sum_{(x,y)\in S} M_1(x, y)$ to upper bound the algorithm's WP mistake bound. □

Our second solution to Problem 1 generalizes the one in Section 4.

```
Algorithm II
1. choose an integer $\phi \ge 1$
2. create a multi-set $S'$ from $S$ by duplicating each $(x, y) \in S$
   $\phi$ times
3. run Algorithm I on $S'$
4. return the hypothesis obtained by Algorithm I
```

Algorithm II does not require the knowledge of M_0 and M_1, yet ensures the following for all workable (M_0, M_1) *simultaneously*:

Theorem 2. *For any (M_0, M_1) such that $\mathscr{A}_{online}$ is (M_0, M_1)-bounded, Algorithm II outputs a hypothesis h' with empirical error $\hat{err}_S(h')$ at most*

$$\frac{1}{n}\Big(\frac{M_0}{\phi} + \sum_{(x,y)\in S} M_1(x, y)\Big). \tag{6}$$

Proof. Since $\mathscr{A}_{online}$ is (M_0, M_1)-bounded, its WP mistake bound M' on S' satisfies:

$$M' \le M_0 + \sum_{(x,y)\in S'} M_1(x, y) = M_0 + \phi \sum_{(x,y)\in S} M_1(x, y). \tag{7}$$

By Theorem 1, the hypothesis h' returned by Algorithm I at Line 3 has empirical error on S' at most $M'/|S'|$, or phrased differently, h' mis-predicts at most M' pairs $(x, y) \in S'$.

Observe that if h' mis-predicts a pair $(x, y) \in S$, h' mis-predicts all the ϕ copies of (x, y) in S'. Therefore, h' can mis-predict at most M'/ϕ pairs in S. The theorem thus follows from (7). □

Corollary 1. *Algorithm II outputs a hypothesis h' with empirical error $\hat{err}_S(h')$ at most*

$$\min_{(M_0,M_1)} \frac{1}{n}\Big(\frac{M_0}{\phi} + \sum_{(x,y)\in S} M_1(x, y)\Big).$$

where the minimum is taken over all (M_0, M_1) such that $\mathscr{A}_{online}$ is (M_0, M_1)-bounded.

Remark 1 (choice of ϕ). The CPU cost of Algorithm II is bounded by $O(\phi TnM)$. Therefore, any $\phi = \text{poly}(n, T)$ will guarantee termination in polynomial time. In Section 7.2, we will describe an incremental approach that avoids the need of tuning this parameter.

Remark 2 (no empirical error on data-independent bounds). For those problems such as *conjunction of boolean literals* that admit an online algorithm whose WP mistake bound is at most a data-independent value c, we can set $M_0 = c$ and $M_1(x, y) = 0$, as demonstrated in the earlier example. By Theorem 2, we can ensure empirical error *strictly less* than $1/n$ by setting $\phi = c + 1$. Such empirical error can only be 0 as $|S| = n$.

6 IMPROVING THE COMPETITIVE RATIO

This section will explain how to leverage an online algorithm to obtain a hypothesis whose empirical error is competitive with respect to the *best* hypothesis with the smallest empirical error.

Again, let $\mathscr{A}_{online}$ be an online algorithm for a problem characterized by $\mathcal{X}, \mathcal{Y}, \mathcal{U}$, and $\mathbb{H}$. We have obtained a set S of non-i.i.d. samples. Denote by M the WP mistake bound of $\mathscr{A}_{online}$ on S. Let h^* be the hypothesis in $\mathbb{H}$ with the lowest empirical error on S. Denote by m^* the number of instance-label pairs $(x, y) \in S$ mis-predicted by h^*. Thus, $\hat{err}_S(h^*) = m^*/n$, which is either 0 or at least $1/n$.

For any hypothesis $h \in \mathbb{H}$, we define its *competitive ratio* to be

$$\frac{\max\{\hat{err}_S(h), 1/n\}}{\max\{\hat{err}_S(h^*), 1/n\}}$$

where the presence of $1/n$ makes sure (i) we never divide by 0, and (ii) the competitive ratio is at least 1. A trivial claim following Theorem 1 is that the hypothesis returned by Algorithm I has a competitive ratio of M.

Next, we establish a stronger claim by introducing *conditional* WP mistake bounds. For each hypothesis $h \in \mathbb{H}$, define

$$S(h) \quad = \quad \{(x, y) \in S \mid h(x) = y\}$$

namely, the set of instance-pair pairs in S correctly predicted by h.

Definition 3. *Fix an arbitrary hypothesis $h \in \mathbb{H}$. The **WP mistake bound** of $\mathscr{A}_{online}$ **conditioned on** h is defined to be the WP mistake bound of $\mathscr{A}_{online}$ on $S(h)$.*

Phrased differently, imagine shrinking S to $S(h)$ by evicting all the pairs mis-predicted by h from S. Now, get the WP mistake bound of $\mathscr{A}_{online}$ on $S(h)$; this is the conditional WP mistake bound of $\mathscr{A}_{online}$ under h.

We will denote the above conditional bound as M_h. The property $M_h \leq M$ follows immediately from the fact $S(h) \subseteq S$, and the definition of WP mistake bound. For some learning problems, M_h can be *far less* than M. An important example is *linear classification*:

Example. Let $\mathscr{A}_{online}$ be the *Perceptron* algorithm. As explained before, a classifier $h \in \mathbb{H}$ is described by a d-dimensional vector $\boldsymbol{w}$. That h is correct on all $(x, y) \in S(h)$ means $y(\boldsymbol{w} \cdot \boldsymbol{x}) \geq 0$. Define $\gamma = \min_{(x,y) \in S(h)} y \frac{\boldsymbol{w} \cdot \boldsymbol{x}}{|\boldsymbol{w}|}$, i.e., the so-called *margin* of h. If $\gamma > 0$, *Perceptron* makes at most $1/\gamma^2$ mistakes on $S(h)$ [37]; hence, $M_h \leq 1/\gamma^2$. This can be considerably smaller than the bound in (4). □

6.1 A weak algorithm

For the time being, let us focus on an *arbitrary* hypothesis $h^+ \in \mathbb{H}$, and assume a known upper bound Z of M_{h^+}. We now give an algorithm to find a hypothesis whose empirical error can be bounded using the empirical error of h^+ and Z.

Algorithm III-weak

```
/* $Z$ is an input integer */
1. $S_1 = S$
2. for $i = 1, 2, \ldots$ do
   /* a super iteration */
3.     run Algorithm I on $S_i$ until either (a) the algorithm
       terminates normally, or (b) $|S_i|$ has decreased by $Z + 1$
4.     if (a) happened then
           return the hypothesis output by Algorithm I
5.     $S_{i+1} = S_i$ (what remains from Line 3)
```

The algorithm runs in *super iterations* (to be distinguished with the *iterations* in Algorithm I). Super iteration $i \geq 1$ executes Algorithm I *from scratch* on a set S_i. The sets $S_1, S_2, \ldots$ satisfy $S = S_1 \supset S_2 \supset S_3 \supset \ldots$ Note that super iteration i does not necessarily run Algorithm I in full: it forces the algorithm to terminate as soon as $Z + 1$ pairs have been removed from S_i. The *remaining* S_i is then used as S_{i+1} for the next super iteration. If Algorithm I terminates normally in the current super iteration, Algorithm III-weak finishes; furthermore, if Algorithm I outputs hypothesis h, we return h as the hypothesis found.

Lemma 1. *The number of super iterations is at most* $1 + |S \setminus S(h^+)|$.

Proof. We argue that, in every super iteration $i \geq 1$ except the last, at least one instance-label pair (x, y) wrongly predicted by h^+ must have disappeared from S_i. The lemma will then follow.

Collect the $Z + 1$ pairs that have been removed from S_i in this super iteration, and arrange them into a sequence Σ by the order they were deleted by Algorithm I. Running $\mathscr{A}_{online}$ on Σ will force $\mathscr{A}_{online}$ to make $Z + 1$ mistakes. If h^+ is correct on all the pairs in Σ, it means $\Sigma \subseteq S(h^+)$. But then, by Definition 3, $\mathscr{A}_{online}$ can make at most $M_{h^+} \leq Z$ mistakes on Σ, giving a contradiction. □

Corollary 2. *The hypothesis returned by Algorithm III-weak has empirical error at most* $(\hat{err}(h^+) + 1/n) \cdot (Z + 1)$.

Proof. Since $Z + 1$ pairs are removed in each super iteration, by Lemma 1, in total at most $(1 + |S \setminus S(h^+)|)(Z + 1)$ pairs can be removed from S. The hypothesis h returned by Algorithm III-weak correctly predicts all the remaining pairs in S. Hence, $\hat{err}_S(h)$ is bounded by $(1 + |S \setminus S(h^+)|)(Z + 1)/n = (1/n + \hat{err}(h^+)) \cdot (Z + 1)$. □

Remark 1 (polynomial time). The CPU time of Algorithm III-weak is at most $O(n/(Z + 1))$ times higher than that of Algorithm I because $O(n/(Z + 1))$ is a trivial bound on the number of super iterations.

Remark 2 (hypotheses captured). It is important to note that Corollary 2 *simultaneously* holds for all $h^+ \in \mathbb{H}$ satisfying $M_{h^+} \leq Z$. Indeed, we can substitute any such h^+ in the above analysis.

6.2 Making the algorithm strong

We now drop all the assumptions behind Algorithm III-weak by invoking it $O(\log n)$ times:

Algorithm III

```
1. for $i = 1, 2, \ldots, \lceil \log_2 n \rceil$ do
2.     run Algorithm III-weak with $Z = 2^i$
3.     $h_i \leftarrow$ the hypothesis obtained from Line 2
4. return the hypothesis among $h_1, \ldots, h_{\lceil \log_2 n \rceil}$ with the
   smallest empirical error
```

Theorem 3. *Algorithm III returns a hypothesis with empirical error at most*

$$\min_{h^+ \in \mathbb{H}} \left(\hat{err}(h^+) + \frac{1}{n} \right) \cdot (2M_{h^+} + 1).$$

Proof. Consider any $h^+ \in \mathbb{H}$. Let i be the smallest integer satisfying $2^i \geq M_{h^+}$. By Corollary 2, $\hat{err}(h_i) \leq (1/n + \hat{err}(h^+)) \cdot (2^i + 1) \leq (1/n + \hat{err}(h^+)) \cdot (2M_{h^+} + 1)$. The theorem follows from Line 4 of the pseudocode. □

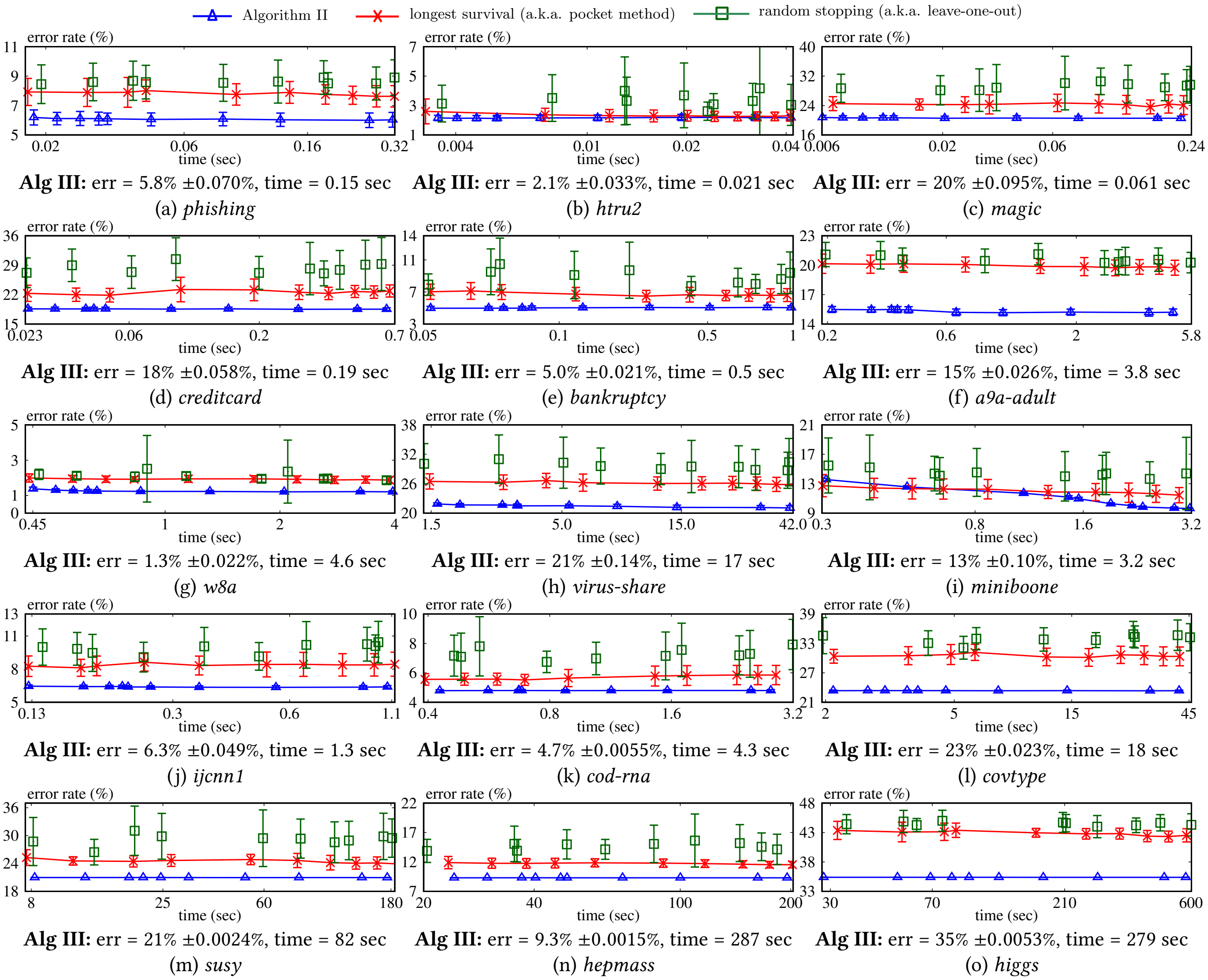
Recall that h^* was defined earlier to be the hypothesis with the smallest empirical error. The above theorem implies that the hypothesis output by Algorithm III has a competitive ratio at most $2M_{h^*} + 1$. The constant 2 can be made arbitrarily close to 1 by increasing Z with a smaller factor each time at Line 2. It can even be made 1 by attempting all $Z = 1, 2, \ldots, n$, although this will increase CPU cost n times (nevertheless still polynomial).

7 EXPERIMENTS

This section contains an empirical evaluation of the proposed algorithms. We will first describe the experimentation settings in Section 7.1 before presenting the results in Section 7.2.

7.1 Setup

Competing methods. Existing online-to-batch conversion methods are all designed for i.i.d. samples. For a comparison, we suggest an approach to adapt an i.i.d.-method to work on a non-i.i.d. sample set S. The idea is to *generate* an i.i.d. stream Σ by repeatedly and independently taking an instance-label pair from S uniformly at random (with replacement). We can then use an i.i.d. conversion method to obtain a good hypothesis h from Σ. As a sufficiently long Σ accurately represents S, h should work well on S too.



Alg III: err = 5.8% ±0.070%, time = 0.15 sec
(a) *phishing*

Alg III: err = 2.1% ±0.033%, time = 0.021 sec
(b) *htru2*

Alg III: err = 20% ±0.095%, time = 0.061 sec
(c) *magic*

Alg III: err = 18% ±0.058%, time = 0.19 sec
(d) *creditcard*

Alg III: err = 5.0% ±0.021%, time = 0.5 sec
(e) *bankruptcy*

Alg III: err = 15% ±0.026%, time = 3.8 sec
(f) *a9a-adult*

Alg III: err = 1.3% ±0.022%, time = 4.6 sec
(g) *w8a*

Alg III: err = 21% ±0.14%, time = 17 sec
(h) *virus-share*

Alg III: err = 13% ±0.10%, time = 3.2 sec
(i) *miniboone*

Alg III: err = 6.3% ±0.049%, time = 1.3 sec
(j) *ijcnn1*

Alg III: err = 4.7% ±0.0055%, time = 4.3 sec
(k) *cod-rna*

Alg III: err = 23% ±0.023%, time = 18 sec
(l) *covtype*

Alg III: err = 21% ±0.0024%, time = 82 sec
(m) *susy*

Alg III: err = 9.3% ±0.0015%, time = 287 sec
(n) *hepmass*

Alg III: err = 35% ±0.0053%, time = 279 sec
(o) *higgs*

Figure 1: Empirical error vs. execution time: linear classifiers

We applied the idea to two popular i.i.d. conversion methods. Recall that running an online algorithm $\mathscr{A}_{online}$ on Σ produces $|\Sigma| + 1$ hypotheses $h_0, ..., h_{|\Sigma|}$.[6] *Random stopping* — also called *leave-one-out* [19] — chooses a hypothesis from $\{h_0, ..., h_{|\Sigma|-1}\}$ uniformly at random. *Longest survival* — also called the *pocket method* [14] — picks the hypothesis that survived for the longest streak (a hypothesis h survived for a streak of length ℓ if $h = h_i = h_{i+1} = ... = h_{i+\ell}$ for some i). Both methods have rigorous theoretical guarantees; see Ch. 5.9 of [7] for a nice discussion.

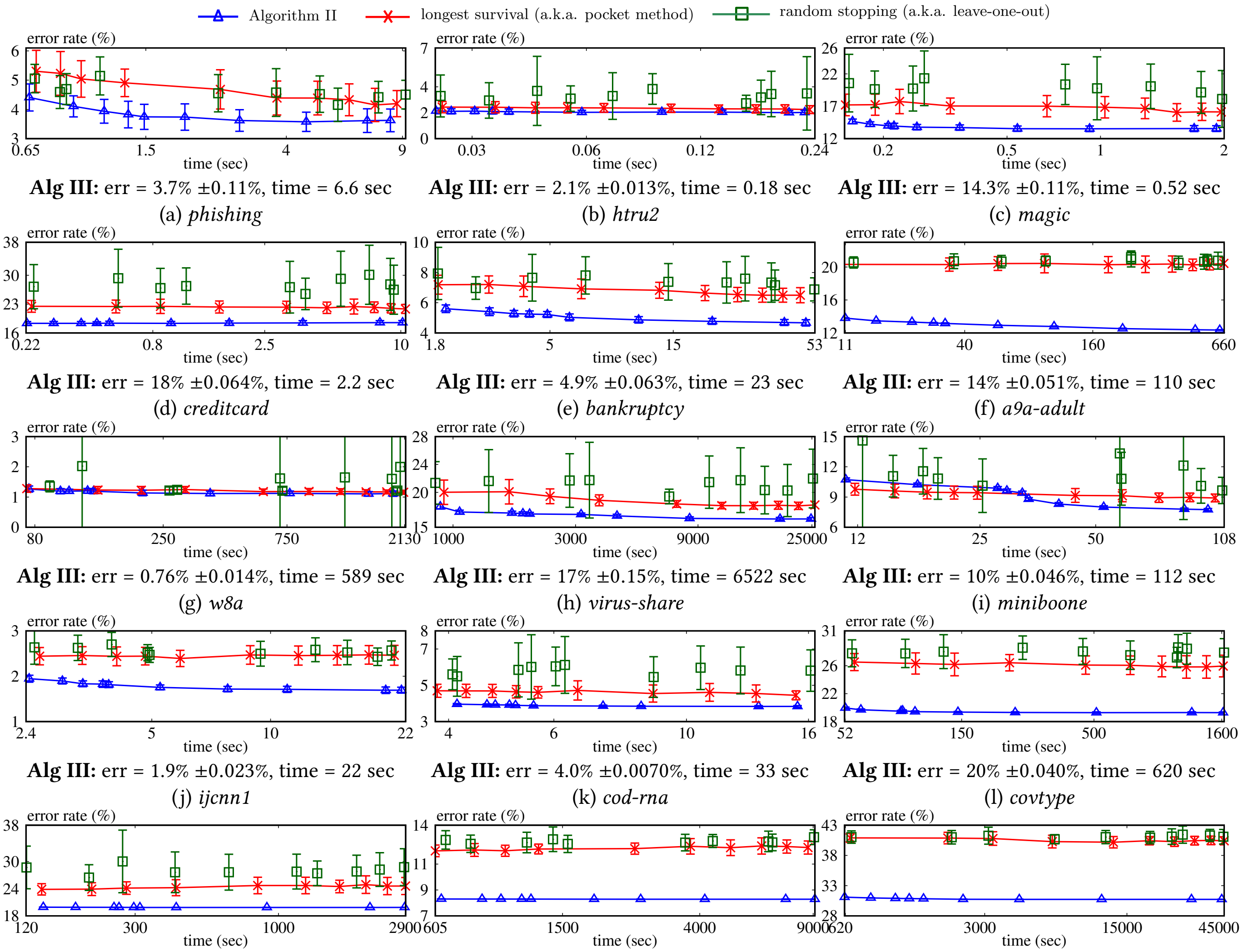
The learning problems and $\mathscr{A}_{online}$. Our techniques are applicable to any online algorithm $\mathscr{A}_{online}$ on any classification problem. We experimented with $\mathscr{A}_{online}$ = *Perceptron* on *linear classification* for three reasons. First, linear classification is arguably the most important problem in machine learning, and the foundation of non-linear classification, the kernel method, neural networks, etc.

id	name	cardinality n	dim. d	source
1	*phishing*	11,055	68	LS
2	*htru2*	17,898	8	UCI
3	*magic*	19,020	10	UCI
4	*creditcard*	30,000	23	UCI
5	*bankruptcy*	43,405	64	UCI
6	*a9a-adult*	48,842	123	LS
7	*w8a*	64,700	300	LS
8	*virus-share*	107,856	482	UCI
9	*miniboone*	130,064	50	UCI
10	*ijcnn1*	141,691	22	LS
11	*cod-rna*	488,565	8	LS
12	*covtype*	581,012	54	LS
13	*susy*	5,000,000	18	LS
14	*hepmass*	10,500,000	27	UCI
15	*higgs*	11,000,000	28	LS

Table 1: The datasets deployed in the experiments

Second, *Perceptron* is by far the most influential online algorithm for linear classification. Third, *Perceptron* and linear classification are both simple enough to avoid unexpected technical factors affecting

[6] Specifically, h_0 is the initial hypothesis, while h_i ($i \geq 1$) is the hypothesis of $\mathscr{A}_{online}$ after the i-th sample in Σ.



Alg III: err = 3.7% ±0.11%, time = 6.6 sec
(a) *phishing*

Alg III: err = 2.1% ±0.013%, time = 0.18 sec
(b) *htru2*

Alg III: err = 14.3% ±0.11%, time = 0.52 sec
(c) *magic*

Alg III: err = 18% ±0.064%, time = 2.2 sec
(d) *creditcard*

Alg III: err = 4.9% ±0.063%, time = 23 sec
(e) *bankruptcy*

Alg III: err = 14% ±0.051%, time = 110 sec
(f) *a9a-adult*

Alg III: err = 0.76% ±0.014%, time = 589 sec
(g) *w8a*

Alg III: err = 17% ±0.15%, time = 6522 sec
(h) *virus-share*

Alg III: err = 10% ±0.046%, time = 112 sec
(i) *miniboone*

Alg III: err = 1.9% ±0.023%, time = 22 sec
(j) *ijcnn1*

Alg III: err = 4.0% ±0.0070%, time = 33 sec
(k) *cod-rna*

Alg III: err = 20% ±0.040%, time = 620 sec
(l) *covtype*

Alg III: err = 20% ±0.0070%, time = 1093 sec
(m) *susy*

Alg III: err = 8.3% ±0.0030%, time = 6147 sec
(n) *hepmass*

Alg III: err = 31% ±0.030%, time = 5819 sec
(o) *higgs*

Figure 2: Empirical error vs. execution time: quadratic classifiers

our comparison, yet interesting enough to admit a sound theory with mathematical rigor.

We also experimented with *quadratic classification* as an extension of *linear classification*, still using *Perceptron* as the choice of $\mathscr{A}_{online}$ (Appendix C).

Data. We deployed 15 real datasets as summarized in Table 1.[7] See Appendix A for how the data were preprocessed.

Implementation. Heuristics incorporated in implementing the proposed algorithms are discussed in Appendix B. In particular, random permutation was adopted in Algorithm I which thus became randomized (and, so did Algorithms II and III).

Measurement. For our solutions, we concentrated on Algorithms II and III because Algorithm I is a special case of the above two. Every measurement was repeated at least 20 times with the mean (if crucial, also the standard deviation) reported.

Environment. Our machine had an Intel CPU at 2.20GHz and 16GB memory. The OS was CentOS Linux 7.

7.2 Results

Linear classification. We ran Algorithm II by changing its parameter ϕ *incrementally*. To start with, ϕ is set to 1 initially, but when Algorithm II finishes, we *continue* its execution by raising ϕ to 2, which can be easily achieved by adding another copy of the dataset *on the fly* (in fact, increasing the counter of each instance-label pair by 1; see Appendix B). Likewise, when the algorithm finishes again, another copy is added, effectively bumping ϕ to 3, and so on. In practice, the above approach removes the need of parameter tuning because when the algorithm has returned a hypothesis h on some ϕ, one can check the empirical error of h before deciding whether to increase ϕ. At the end, the best h of all ϕ is returned.

The blue curves in Figure 1 show the empirical error of Algorithm II as a function of running time. Markers on each curve indicate

[7] LS is at *http://www.csie.ntu.edu.tw/∼cjlin/libsvmtools/datasets*, while UCI is at *http://archive.ics.uci.edu/ml/datasets.html*.

the empirical errors of the hypotheses obtained at ϕ = 1, 2, 3, 4, 5, 10, 20, 40, 80, and 100, respectively. Each marker is associated with a vertical bar representing the amount of standard deviation (from at least 20 runs). The x-coordinate of the marker is the mean elapsed time until the algorithm terminated on the designated ϕ value (x-axis in log scale). Algorithm II exhibited excellent performance, even at $\phi = 1$. Increasing ϕ *did* help to improve the accuracy, sometimes quite significantly, e.g., in Figure 1(i). The standard deviation in empirical error is exceedingly small, suggesting very stable performance.

The result of Algorithm III is presented under each diagram in texts. Its empirical error is in the form *mean ± standard dev.*; and the time shown is the mean. The accuracy was very similar to that of Algorithm II, and was also extremely stable, whereas the running time was on average that of Algorithm II with $\phi = 60$.

The green markers[8] and red curves illustrate respectively the results for *random stopping* and *longest survival*, whose running time depends on the length of Σ (Section 7.1). Their empirical errors were in general much higher, and suffered quite heavily from large deviation (especially so for *random stopping*).

Quadratic classification. Figure 2 shows the results on quadratic classification in the same style. Overall, the earlier observations from Figure 1 are re-confirmed in this context. Note that Algorithm II benefited more significantly from the increase of ϕ.

8 CONCLUSIONS

Traditional online-to-batch conversion is based on the assumption that the samples in batch learning are i.i.d. In reality the assumption seldom holds, in which case existing conversion methods loose their mathematical rigor, creating a serious issue in analyzing the quality of learning. This paper remedies the issue by proposing a suite of techniques to carry out the conversion on non-i.i.d. samples with strong theoretical guarantees. Our techniques leverage an online algorithm as a black box and applies to all classification problems, even if the algorithm is only empirically efficient on that problem. The practical usefulness of the proposed solutions is confirmed by an extensive experimentation evaluation with real data.

REFERENCES

[1] Dana Angluin. 1987. Queries and Concept Learning. *Machine Learning* 2, 4 (1987), 319–342.

[2] Arnab Bhattacharyya, Ameet Gadekar, and Ninad Rajgopal. 2015. On learning k-parities with and without noise. *CoRR* abs/1502.05375 (2015).

[3] Arnab Bhattacharyya, Ameet Gadekar, and Ninad Rajgopal. 2018. Improved Learning of k-Parities. In *Proceedings of International Conference on Computing and Combinatorics (COCOON).* 542–553.

[4] Avrim Blum. 1994. Separating Distribution-Free and Mistake-Bound Learning Models over the Boolean Domain. 23, 5 (1994), 990–1000.

[5] Avrim Blum. 1996. On-line Algorithms in Machine Learning. In *Online Algorithms, The State of the Art.* 306–325.

[6] Avrim Blum, Lisa Hellerstein, and Nick Littlestone. 1995. Learning in the Presence of Finitely or Infinitely Many Irrelevant Attributes. *JCSS* 50, 1 (1995), 32–40.

[7] Avrim Blum, John Hopcroft, and Ravindran Kannan. 2018. *Foundations of Data Science.* Book available at http://www.cs.cornell.edu/jeh/book.pdf.

[8] Harry Buhrman, David García-Soriano, and Arie Matsliah. 2010. Learning parities in the mistake-bound model. *IPL* 111, 1 (2010), 16–21.

[9] Nicolo Cesa-Bianchi, Alex Conconi, and Claudio Gentile. 2004. On the Generalization Ability of On-Line Learning Algorithms. *IEEE Trans. Information Theory* 50, 9 (2004), 2050–2057.

[10] Nicolo Cesa-Bianchi and Claudio Gentile. 2008. Improved Risk Tail Bounds for On-Line Algorithms. *IEEE Trans. Information Theory* 54, 1 (2008), 386–390.

[11] Ofer Dekel. 2008. From Online to Batch Learning with Cutoff-Averaging. In *NIPS.* 377–384.

[12] Ofer Dekel and Yoram Singer. 2005. Data-Driven Online to Batch Conversions. In *NIPS.* 267–274.

[13] Yoav Freund and Robert E. Schapire. 1999. Large Margin Classification Using the Perceptron Algorithm. *Machine Learning* 37, 3 (1999), 277–296.

[14] S.I. Gallant. 1986. Optimal linear discriminants. In *International Conference on Pattern Recognition.* 849–852.

[15] Wei Gao, Xin-Yi Niu, and Zhi-Hua Zhou. 2016. Learnability of Non-I.I.D. In *Proceedings of Asian Conference on Machine Learning (ACML).* 158–173.

[16] Claudio Gentile. 2001. A New Approximate Maximal Margin Classification Algorithm. *JMLR* 2 (2001), 213–242.

[17] David Haussler. 1988. Quantifying Inductive Bias: AI Learning Algorithms and Valiant's Learning Framework. *Artif. Intell.* 36, 2 (1988), 177–221.

[18] David P. Helmbold, Robert H. Sloan, and Manfred K. Warmuth. 1990. Learning Nested Differences of Intersection-Closed Concept Classes. *Machine Learning* 5 (1990), 165–196.

[19] David P. Helmbold and Manfred K. Warmuth. 1995. On Weak Learning. *JCSS* 50, 3 (1995), 551–573.

[20] Steven C. H. Hoi, Doyen Sahoo, Jing Lu, and Peilin Zhao. 2018. Online Learning: A Comprehensive Survey. *CoRR* abs/1802.02871 (2018).

[21] Lakhmi C. Jain, Manjeevan Seera, Chee Peng Lim, and Pagavathigounder Balasubramaniam. 2014. A review of online learning in supervised neural networks. *Neural Computing and Applications* 25, 3-4 (2014), 491–509.

[22] Rajeeva L Karandikar and M. Vidyasagar. 2002. Rates of uniform convergence of empirical means with mixing processes. *Statistics & Probability Letters* 58, 3 (2002), 297–307.

[23] Norbert Klasner and Hans Ulrich Simon. 1995. From Noise-Free to Noise-Tolerant and from On-line to Batch Learning. In *COLT.* 250–257.

[24] Adam R. Klivans and Rocco A. Servedio. 2006. Toward Attribute Efficient Learning of Decision Lists and Parities. *JMLR* 7 (2006), 587–602.

[25] Vitaly Kuznetsov and Mehryar Mohri. 2017. Generalization bounds for non-stationary mixing processes. *Machine Learning* 106, 1 (2017), 93–117.

[26] Yi Li and Philip M. Long. 2002. The Relaxed Online Maximum Margin Algorithm. *Machine Learning* 46, 1-3 (2002), 361–387.

[27] Nick Littlestone. 1987. Learning Quickly When Irrelevant Attributes Abound: A New Linear-threshold Algorithm. *Machine Learning* 2, 4 (1987), 285–318.

[28] Nick Littlestone. 1989. From On-Line to Batch Learning. In *COLT.* 269–284.

[29] Philip M. Long and Rocco A. Servedio. 2006. Attribute-efficient learning of decision lists and linear threshold functions under unconcentrated distributions. In *NIPS.* 921–928.

[30] Viktor Losing, Barbara Hammer, and Heiko Wersing. 2018. Incremental on-line learning: A review and comparison of state of the art algorithms. *Neurocomputing* 275 (2018), 1261–1274.

[31] Dharmendra S. Modha and Elias Masry. 1998. Memory-Universal Prediction of Stationary Random Processes. *IEEE Trans. Information Theory* 44, 1 (1998), 117–133.

[32] Mehryar Mohri and Afshin Rostamizadeh. 2007. Stability Bounds for Non-i.i.d. Processes. In *NIPS.* 1025–1032.

[33] Mehryar Mohri and Afshin Rostamizadeh. 2008. Rademacher Complexity Bounds for Non-I.I.D. Processes. In *NIPS.* 1097–1104.

[34] Mehryar Mohri and Afshin Rostamizadeh. 2010. Stability Bounds for Stationary phi-mixing and beta-mixing Processes. *JMLR* 11 (2010), 789–814.

[35] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. 2018. *Foundations of machine learning.* MIT Press.

[36] Ziv Nevo and Ran El-Yaniv. 2002. On Online Learning of Decision Lists. *JMLR* 3 (2002), 271–301.

[37] Frank Rosenblatt. 1958. The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. *Psychological Review* 65, 6 (1958), 386–408.

[38] Rocco A. Servedio. 2000. Computational Sample Complexity and Attribute-Efficient Learning. *JCSS* 60, 1 (2000), 161–178.

[39] Shai Shalev-Shwartz. 2012. Online Learning and Online Convex Optimization. *Foundations and Trends in Machine Learning* 4, 2 (2012), 107–194.

[40] Shai Shalev-Shwartz and Yoram Singer. 2005. A New Perspective on an Old Perceptron Algorithm. In *COLT.* 264–278.

[41] Ingo Steinwart and Andreas Christmann. 2009. Fast Learning from Non-i.i.d. Observations. In *NIPS.* 1768–1776.

[42] Ingo Steinwart, Don R. Hush, and Clint Scovel. 2009. Learning from dependent observations. *J. Multivariate Analysis* 100, 1 (2009), 175–194.

[43] Ryuhei Uehara, Kensei Tsuchida, and Ingo Wegener. 1997. Optimal Attribute-Efficient Learning of Disjunction, Parity and Threshold Functions. In *Proceedings of European Conference on Computational Learning Theory.* 171–184.

[44] Mathukumalli Vidyasagar. 2002. *A theory of learning and generalization: with applications to neural networks* (2nd ed.). Springer.

[45] Bin Yu. 1994. Rates of Convergence for Empirical Processes of Stationary Mixing Sequences. *Annals of Probability* 22, 1 (1994), 94–116.

[46] Tong Zhang. 2005. Data Dependent Concentration Bounds for Sequential Prediction Algorithms. In *COLT.* 173–187.

[8] No line connects the green markers because they do not demonstrate obvious patterns.

APPENDIX

A DATA PREPROCESSING

Each dataset contains d-dimensional points (d shown in Table 1) each of which is associated with a label. All labels are binary, except *virus-share*, where a label is a numeric value between 0 and 1. We changed each label in *virus-share* to 1 if it was at least 0.5, or -1 otherwise.

Normalization was carried out on each dataset S. For dimension $i \in [1, d]$, we first obtained the mean $mean_i$ and standard deviation std_i of the coordinates in S. Then, each point $p \in S$ was normalized to the point p' with $p'[i] = (p[i] - mean_i)/std_i$, for all $i \in [1, d]$.

Finally, we added one more dimension where every point had coordinate 1, as is a standard step in linear classification.

B HEURISTICS FOR ALGORITHMS I, II, III

I. The input set S is randomly permuted before the algorithm starts.

II. As discussed in Section 5, S' unions ϕ copies of S, but this does not imply a ϕ-fold increase in space consumption. It suffices keep a *counter* for every instance-label pair $(x, y) \in S$ to record how many copies of (x, y) still remain in S'. At the beginning, all counters are ϕ. When a copy of (x, y) is removed, its counter decreases; when the counter reaches 0, no more copies of (x, y) exist in S'.

Algorithm II permits the ϕ to be increased at any time *during* execution. Specifically, to increase ϕ from ϕ_1 to ϕ_2, it suffices to add $\phi_2 - \phi_1$ to each counter. No other state of the algorithm needs to be altered.

III. In the experiments we doubled Z from $n/1000$ to (the first value at least) n. This runs Algorithm III-weak roughly 10 ($\approx \log_2 1000$) times.

C QUADRATIC CLASSIFICATION IMPLEMENTATION

Linear classification can be integrated with the kernel method to perform non-linear classification. Define $\mathcal{X} = \mathbb{R}^d$, i.e., the entire d-dimensional space, $\mathcal{Y} = \{-1, 1\}$, and $\mathcal{U} = \mathcal{X} \times \mathcal{Y}$.

A kernel function $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ has the property that, for any points $\boldsymbol{p}$, $\boldsymbol{q}$ in $\mathbb{R}^d$, $K(\boldsymbol{p}, \boldsymbol{q})$ equals the dot product of two d'-dimensional vectors $\Phi(\boldsymbol{p}), \Phi(\boldsymbol{q})$, where $\Phi : \mathbb{R}^d \to \mathbb{R}^{d'}$ is a non-linear transformation that converts a d-dimensional point into some d'-dimensional *feature space*.

For example, a *polynomial kernel* has the form:

$$K(\boldsymbol{p}, \boldsymbol{q}) \quad = \quad (\boldsymbol{p} \cdot \boldsymbol{q} + 1)^c$$

for some constant $c \geq 1$. The corresponding feature space has dimensionality $d' = \binom{d+c}{c}$. To illustrate, consider $d = 2$ and $c = 2$, for which:

$$\begin{aligned}
(\boldsymbol{p} \cdot \boldsymbol{q} + 1)^2 \quad &= \quad (p[1]q[1] + p[2]q[2] + 1)^2 \\
&= \quad p[1]^2 q[1]^2 + p[2]^2 q[2]^2 + 2(p[1]p[2])(q[1]q[2]) \\
&\qquad +2p[1]q[1] + 2p[2]q[2] + 1 \\
&= \quad \Phi(\boldsymbol{p}) \cdot \Phi(\boldsymbol{q})
\end{aligned}$$

where $\Phi(\boldsymbol{x})$ converts a 2D point $\boldsymbol{x}$ to a 6D one:

$$(x[1]^2, x[2]^2, \sqrt{2}x[1]x[2], \sqrt{2}x[1], \sqrt{2}x[2], 1). \tag{8}$$

Given a set S of samples from $\mathcal{U}$, define $\Phi(S) = \{\phi(\boldsymbol{x}) \mid \boldsymbol{x} \in S\}$, i.e., the set of converted points. The learning on $\Phi(S)$ returns a linear classifier in the feature space characterized by a d'-dimensional vector $\boldsymbol{w'}$. This yields a non-linear classifier in the *original* space.

For instance, in the example of (8), consider $\boldsymbol{w'} = (1, 2, 3, 4, 5, 6)$, which determines the following quadratic classifier h in the original 2D space: $h(\boldsymbol{x}) = 1$ if

$$x[1]^2 + 2x[2]^2 + 3\sqrt{2}x[1]x[2] + 4\sqrt{2}x[1] + 5\sqrt{2}x[2] + 6 \quad \geq \quad 0$$

or -1, otherwise.

In our experiments with quadratic classification, $c = 2$, and $\boldsymbol{w'}$ was obtained by running *Perceptron* on $\Phi(S)$. Naive generation of $\Phi(S)$ requires $d' = \Theta(\binom{d}{2}) = \Theta(d^2)$ space for each point, which is prohibitive for large d. We avoided the issue by generating the coordinates of $\phi(p)$ on demand for each $p \in S$. It then suffices to store S itself and the weight vector $\boldsymbol{w'}$ maintained by *Perceptron* in the feature space.