On-line Learning with Delayed Label Feedback

Chris Mesterharm

mesterha@cs.rutgers.edu
Rutgers Computer Science Department
110 Frelinghuysen Road
Piscataway, NJ 08854

Abstract. We generalize on-line learning to handle delays in receiving labels for instances. After receiving an instance x, the algorithm may need to make predictions on several new instances before the label for x is returned by the environment. We give two simple techniques for converting a traditional on-line algorithm into an algorithm for solving a delayed on-line problem. One technique is for instances generated by an adversary; the other is for instances generated by a distribution. We show how these techniques effect the original on-line mistake bounds by giving upper-bounds and restricted lower-bounds on the number of mistakes.

1 Introduction

In this paper, we consider the problem of label feedback in on-line learning. On-line learning is composed of trials. Each trial t can be broken up into three steps. First, the algorithm receives an instance from a set X. Second, the algorithm predicts a label from a finite set Y. Last, the algorithm receives the correct label from the environment. The goal of the algorithm is to minimize the number of mistakes. [1] We are interested in the last step. For many practical problems, the algorithm may not receive the label feedback in a timely matter.

Delayed labels are a realistic assumption for many potential on-line learning problems. Consider spam email filtering. The filtering algorithm often allows the user to train the algorithm using labeled emails. [2] In between training, many emails may arrive that need to be classified. Anytime successive predictions need to be made without receiving a label, it is a delayed learning problem. Another example is webpage prefetching. This is useful for speeding up the performance of low bandwidth Internet connections. Learning which links to preload is a useful optimization[3], however, the label feedback might be delayed until it is determined that a prefetched webpage will not be used. As a final example, a doctor may want to predict health problems in a patient in order to start treatment as soon as possible. A more definitive test may be prescribed to confirm the diagnosis; this test provides delayed feedback.

To solve this problem, we propose the delayed model of on-line learning. This model is identical to the traditional on-line learning except that the environment can return the label feedback any number of trials after the arrival of the instance. This amounts to changing the last step of on-line learning to

receive possibly multiple labels from the current or previous trials. For every on-line learning problem, there are matching delayed learning problems that receive delayed labels.

The main contribution of this paper is to give two ways to transform a traditional on-line algorithm to an algorithm that works with delayed labels. We give upper-bounds on the number of mistakes of these algorithms, where the bounds are given as a function of the bounds for the original on-line algorithm. We assume two different techniques for instance generation. First, we assume the instances are generated by an adversary. This is a common assumption used when analyzing many on-line algorithms. [1] Second, we assume the instances are generated by a distribution. While this assumption is less common [4], it can give tighter bounds for a large class of practical problems. In both cases, the bounds are robust; the bounds allow noisy instances that do not correspond to a target function [4], and the bounds allow tracking a target function that is allowed to change over the trials. [5, 6] We also give lower-bounds on restricted forms of these two instance generation techniques. We show these lower-bounds are close to our upper-bounds for these restricted problems.

2 Notation

All of our transformations take an existing traditional on-line algorithm and convert it to handle delayed instances. Let algorithm B be our traditional on-line algorithm. The pseudo-code for algorithm B is given in Fig. 1. On trial t, the algorithm accepts instance $x_t \in X$ and returns a distribution $\hat{y}_t \in [0,1]^{|Y|}$ over the possible output labels. The algorithm predicts by sampling from this distribution. [1] The algorithm then receives feedback on the correct label in $y_t \in Y$. It can use this information to update the current state of the algorithm to improve performance on future instances.

Initialization
  $t \leftarrow 0$ is the trial number.
  Initialize algorithm state $s \leftarrow s_0$.
Trials
  $t \leftarrow t + 1$.
  Instance: $\mathbf{x}_t$.
  Prediction: $\hat{y}_t \leftarrow \text{Pred}(s, \mathbf{x}_t)$.
  Update: Let y_t be the correct label.
    $s \leftarrow \text{Update}(s, \mathbf{x}_t, y_t, \hat{y}_t)$.

Fig. 1. on-line algorithm B

The transformed algorithms use the same procedures as algorithm B for updates and predictions. The prediction procedure of algorithm B accepts two

[1] While it is common to just let $\hat{y}_t$ be the predicted label, our notation is needed to help describe one of the algorithm transformations.

parameters: the instance $\mathbf{x}_t$ for prediction and the current state of the algorithm, s. The state of the algorithm encodes the value of all the memory used by the algorithm that can have an effect on future predictions. The initial state is represented as s_0. The prediction procedure returns a probability distribution for the label. We use the notation $\hat{y}_t[i]$ to determine the probability that the predicted label is i on trial t. For a deterministic algorithm, a single label will have value 1. The update procedure accepts four parameter: the state of the algorithm, the instance, the label returned by the environment, and the predicted label distribution. The update procedure returns two outputs: the new algorithm state and boolean variable `new_state` that is `TRUE` if the algorithm state has changed because of the update and `FALSE` otherwise. We ignore the `new_state` variable if it is not used by a particular algorithm.

The only difference in delayed on-line learning is that the label for instance x_t may not be returned at the end of trial t. Therefore, we need notation to represent when the label feedback is returned by the environment. We use $y_{a,b}$ to refer to the label of an instance where the attributes arrive on trial a and the label arrives right before the start of trial b. Each instance arrives at a unique trial, but labels may arrive during the same trial. Therefore, when we want to specify an instance, we will refer to the trial where the attributes arrive. We define the delay of a particular instance, with label $y_{a,b}$, as $b - a$. Let k be the maximum delay over all instances. In traditional on-line learning, all instances have a delay of 1 and have labels of the form $y_{t,t+1}$. In delayed on-line learning, each instance may have an arbitrary positive delay.

In the rest of the paper, we use the following notation. Let $E[\text{Mist}(B, s)]$ be the expected number of mistakes B makes on s, a sequence of instances, and let $E[\text{Change}(B, s)]$ be the expected number of times B changes its state on sequence s. The expectation is taken with respect to any randomization used by the algorithm. If the instances are generated by an adversary, let $E[\text{Mist}(B)]$ be the maximum expected mistakes made by algorithm B over a set of instance sequences. If the instance are generated by a distribution, let let $E[\text{Mist}(B)]$ be the expected number of mistakes. It should be clear from context whether we are dealing with an adversary or a distribution. In the case of a distribution, the expectation is taken with respect to the generation of instances and any randomization of the algorithm. Let *Opt* be the algorithm that minimizes $E[\text{Mist}(B)]$. Since we are interested in taking existing on-line algorithms and converting them to the delayed on-line model, the restrictions on instances will come from the on-line algorithm. If the B algorithm is deterministic we can drop the expectation from the notation in the adversarial case.

3 Instances Generated by an Adversary

In this section, we give algorithms and upper-bounds on mistakes when instances are generated by an adversary. First, we need to define what we mean by an adversary generating instances. Later we will allow the adversary to set delays for these instances. Let $\mathcal{A}$ be a nonempty set of instances (x, y) where $x \in X$

and $y \in Y$. These are the instances that an adversary can pick during a trial. We also define a function $\eta(x, y)$ that measures the amount of noise in an instance. This is used to control what instances the adversary can pick but is not revealed to the learning algorithm.

The adversary can pick any instance for the current trial that has zero noise. These non-noisy instances correspond to the target function. The adversary must have restrictions on the number of noisy instances it can generate otherwise learning would be impossible. A common bound on the noisy instances is to allow only a fixed total amount of noise, where the total noise is computed by summing the $\eta()$ amounts from the generated instances. However, our result generalizes any traditional adversarial on-line algorithm to the delayed setting and inherits whatever noise assumption is made by the original on-line algorithm including the values for the $\eta()$ function.

In this paper, we allow the adversary to track a moving target function. Let $\Phi = [(\mathcal{A}_1, \eta_1()), (\mathcal{A}_2, \eta_2()), \cdots]$ be a sequence of instance sets and noise functions. We continually allow the adversary two choices; the adversary can either generate a trial by selecting an instance, or the adversary can increment the instance selection to the next element of Φ. Instance generation starts at $(\mathcal{A}_1, \eta_1())$, and the adversary is not allowed to go backwards in the sequence. We are interested in the worst-case performance of the algorithm over a set of possible Φ. This is a general model of instance generation that includes fixed concepts [1], by having only a single element in each Φ, and concept tracking [5, 6], by using instance sets and noise functions that correspond to different target functions.

For delayed on-line learning, we need to let the adversary delay the label feedback. To make this as general as possible, we will let the adversary pick the delay for an instance from a multi-set $\mathcal{D}$ of positive numbers. More formally let $d_i \in R \cup \infty$ be the maximum number of instances that have a delay of i trials; $\mathcal{D} = \cup_{i=1}^{\infty} \left(\cup_{j=1}^{d_i} i \right)$. For example, $\mathcal{D}$ may only contain the number 5 an infinite amount of times. In this case, the adversary must give each instance a delay of 5. Our bound will be based on values of the various d_i; this allows us to model a wide range of problems. For example, in the medical problem explained earlier, each patient may take a different amount of time to get the lab test needed for the label; a few patients may never take the lab test and have an infinite delay.

Before we give our main bound, we need a lemma to help us work with the delay multi-set $\mathcal{D}$. We want to place as many elements from $\mathcal{D}$ into a list L with the restriction that the first C numbers must be at least 1, the next C numbers must be at least 2, and so on where the mth block of C numbers must be at least m. We call this the ordered class selection problem. If one uses the greedy algorithm of always placing the smallest number remaining in $\mathcal{D}$ into the next position in the list then the total number of elements of value i that are in this greedy list is $r_i = \min \left(d_i, iC - \sum_{j=1}^{i-1} r_j \right)$.

Lemma 1. *Let $F(\mathcal{D}, C)$ be the maximum number of elements that can be placed from $\mathcal{D}$ in the ordered class selection problem. This maximum is obtained by the greedy algorithm, and $F(\mathcal{D}, C) = \sum_{j=1}^{\infty} r_j$.*

Proof. We break the proof into two cases. First, assume that $F(\mathcal{D}, C) = \infty$. Based on the definition of the ordered class problem, there must be no upper-bound on the elements in D. Therefore, the greedy algorithm will also generate an infinite list.

Second, assume that $F(\mathcal{D}, C)$ is finite. Let l_o be a list of elements that satisfy the ordered class selection problem with the number of elements in l_o equal to $F(\mathcal{D}, C)$. Let l_g be the list generated by the greedy algorithm. We will compare each element of l_g with l_o and show that the lists must have the same length.

Start at the beginning of each list and compare elements. If $l_o(0) = l_g(0)$ then go to the next element. If $l_o(0) > l_g(0)$ then find the next index, i, in list l_o such that $l_o(i) = l_g(0)$. If index i exists then, in list l_o, swap values $l_o(0)$ and $l_o(i)$. This still gives a legal list. If there is no such index then the number of elements with value $l_g(0)$ used in list l_o must be less then $d_{l_g(0)}$. Therefore we can just assign $l_o(0)$ to value $l_g(0)$. The new l_o list is still a valid list and still has length $F(\mathcal{D}, C)$.

We can repeat this procedure for each pair of elements from list l_o and l_g. Let i_e be the last element in list l_g. At this point both lists are identical up to index i_e. Any additional elements in l_o must have a value of at least $l_g(i_e)$ since otherwise the l_g list would not be greedy. However, if the additional elements have values of at least $l_g(i_e)$ then l_g would not end at index i_e. Therefore the new l_o list and l_g list must be the same length. Since the length of the modified l_o has not changed from the original length, the length of l_g is $F(\mathcal{D}, C)$. Based on the greedy algorithm, the length of l_g is also equal to $\sum_{j=1}^{\infty} r_j$. This proves the lemma. □

3.1 Algorithm

We call the first transformed algorithm $OD1$-B. This algorithm is similar to algorithm B except that it potentially skips some of the updates. The pseudo-code for $OD1$-B is in Fig. 2. We use a stack U to store the instances that are ready for updates. We store instances waiting for labels in a hash table.

$OD1$-B keeps track of a `last` trial and only performs updates using instances that are more recent than this trial. After the algorithm performs an update, if the update either changes the state of the algorithm, or if the update is based on an instance that could have caused a mistake, given the state used for the update, then the algorithm increases `last` to the trial of the instance used for the update. This ensures that the changes to the algorithm occur in the same order as the instance arrival times. Changes occurring out of order can cause problems when the concept is shifting.

The computational cost of the $OD1$-B algorithm is similar to the cost of the B algorithm. The number of updates is at most the same as algorithm B and the number of predictions is at most double. There is an extra cost based on sorting the instances in U. This cost depends on the number of labels returned per trial. Let γ be the maximum number of label returned during a trial. Using merge sort gives an amortized cost of at most $O(\ln(\gamma))$ per trial. Because $F(\mathcal{D}, 1)$ is the maximum number of instances that have arrived but have not yet received labels,

Initialization
$t \leftarrow 0$ is the trial number.
`last` $\leftarrow 0$ is the last instance used for an update.
$U \leftarrow$ `null` is a stack that stores instances that are ready for updates.
Initialize algorithm to state $s \leftarrow s_0$.
Trials
$t \leftarrow t + 1$.
Instance: Store $\mathbf{x}_t$ using t as the key.
Prediction: $\hat{y}_t \leftarrow \text{Pred}(s, \mathbf{x}_t)$.
Update:
For all returned labels $y_{a,t}$
If $a >$ `last` then
add instance $(a, x_a, y_{a,t})$ to U in sorted order based on a.
For $i = 1$ to $|U|$
$(a, x_a, y_{a,t}) \leftarrow \text{pop}(U)$
$\hat{y} \leftarrow \text{Pred}(s, x_a)$
$(s,$ `new_state`$) \leftarrow \text{Update}(s, x_a, y_{a,t}, \hat{y})$
If `new_state` $= 1$ or $\hat{y}[y_{a,t}] \neq 1$ then
`last` $\leftarrow a$.
Remove x_a.
Periodically remove all instances older than `last`.

Fig. 2. delayed on-line algorithm $OD1$-B

we have that $\gamma \leq F(\mathcal{D}, 1) \leq k$. The $OD1$-B algorithm needs extra storage for at most $F(\mathcal{D}, 1)$ instances. By keeping space for $2F(\mathcal{D}, 1)$ instances, the algorithm can periodically remove any old instances that missed an update with only a constant amortized increase in the cost per trial.

3.2 Upper-bound on Mistakes

Here is our main result for learning against an adversary. Recall that $\text{Change}(B, s)$ is a random variable for the number of times algorithm B changes its state on instance sequence s.

Theorem 1. *Assume B is a traditional on-line algorithm, and assume s is a sequence of instances generated by an adversary. The expected number of mistakes of the OD1-B algorithm is at most $E[Mist(B)] + E[F(\mathcal{D}, Change(B, s))] - E[Change(B, s)]$ mistakes in the delayed on-line model.*

Proof. Consider all the instances that change the variable `last` in algorithm $OD1$-B. The updates on these instances are in trial order. Call this sequence of instances u. If we pass these instances to algorithm B, giving each instance a delay of 1, we can expect at most $E[\text{Mist}(B)]$ mistakes on algorithm B, since sequence u corresponds to a sequence that could be generated by the adversary. Notice that the algorithm states for $OD1$-B on subsequence u of s is identical to the states of algorithm B on u. For algorithm $OD1$-B, all the other trials in s just copy the state from the previous trial.

Next, it is useful to partition the sequence s into two sets. Let Q_1 be the set of instances $\mathbf{x}_t$ such that, when $OD1$-B updates the label $y_{t,t+k}$, the state at trial $t+k$ has not changed since trial t. Let Q_2 be all other instances. We can divide the instances from Q_1 into two groups. Let g_1 be the instances from Q_1 that are in u. Let g_2 be all other instances in Q_1. Assume that x is an instance from g_1. The probability of a mistake by $OD1$-B on x must equal the probability of a mistake in the equivalent instance from u on B since the state when the label arrived is the same as the state when the instance arrived. For the instances in g_2, the $OD1$-B algorithm must have a zero probability of making a mistake otherwise the instance would be in u. Therefore the expected number of mistakes by algorithm $OD1$-B for instances from Q_1 must be at most $E[\text{Mist}(B)]$.

Next consider Q_2. There is a limit on the number of instances in Q_2 based on the number of times the state changes and the number of instances with specific delay values. This number is primarily determined by the solution to the multi-set problem in lemma 1. However, for each state change at least one of the delayed instances from the multi-set solution must cause the update that changes the state. Therefore the expected number of elements in Q_2 is at most $E[F(\mathcal{D}, \text{Change}(B, s))] - E[\text{Change}(B, s))]$. Since each instance in Q_2 can cause at most one mistake, this proves the theorem. □

At this point, the bound is not very intuitive because of the complexity of the F function. It is interesting to see how this function varies for different $\mathcal{D}$ multi-sets. If all instances have delays of at most k then $E[F(\mathcal{D}, \text{Change}(B, s))] \leq kE[\text{Change}(B, s)]$. In addition, if we add m delays to $\mathcal{D}$ of any value then the value of $E[F(\mathcal{D}, \text{Change}(B, s))]$ can increase by at most m. This give a rough idea of how the bound depends on the delays of the instances. A more precise analysis will depend on a specific multi-set $\mathcal{D}$. In the remainder of the paper, we do not want to dwell on different choices for the $\mathcal{D}$ multi-set. Therefore, we use the fact that $E[F(\mathcal{D}, \text{Change}(B, s))] \leq kE[\text{Change}(B, s)]$ to simply our results. However, it is possible to generalize using the F function.

Corollary 1. *Assume B is a traditional on-line algorithm, and s is a sequence of instances generated by an adversary. If the maximum delay of any instance is k then the expected number of mistakes of the OD1-B algorithm is at most $E[Mist(B)] + (k-1)E[Change(B, s)]$ in the delayed on-line model.*

Proof. Combine $E[F(\mathcal{D}, \text{Change}(B, s))] \leq kE[\text{Change}(B, s)]$ with Theorem 1. □

In order to get a good bound, we need to use a B algorithm that makes few mistakes and that changes its state few times. Fortunately, deterministic mistake-driven algorithms fulfill these criteria. A mistake-driven algorithm is an algorithm that only updates its state when it makes a mistake. [1] In Sect. 5, we will show that that converting a deterministic algorithm B to a mistake-driven form, MD-B, does not increase the mistake bound for a class of adversaries that includes the adversary used in this section.

To handle randomized algorithms, we use the fact that any randomized learning algorithm can be converted to a deterministic learning algorithm with a similar mistake bound. On every trial, this new deterministic algorithm just predicts the highest probability label from the randomized algorithm. The deterministic algorithm makes at most double the expected number of mistakes of the randomized algorithm. [7] Given a learning algorithm B, we call *DR-B* the derandomized learning algorithm.

Theorem 2. *Assume B is an on-line algorithm. If the maximum delay of any instance is k then the number of mistakes of the OD1-MD-DR-B algorithm is at most $2kE[Mist(B)]$ in the delayed on-line model.*

Proof. Assume $E[\text{Mist}(B)] = M$. Using the derandomized algorithm, we get $\text{Mist}(DR\text{-}B) \leq 2M$. Next, we make the algorithm mistake-driven. Theorem 5 shows that $\text{Mist}(MD\text{-}DR\text{-}B) \leq 2M$. Last, we use Corollary 1, and the fact that for any sequence of instances s, $E[\text{Change}(B, s)] \leq E[\text{Mist}(B)]$ for a mistake-driven algorithm. □

3.3 Lower-bound on Mistakes

A natural question is whether a different transformations can make fewer mistakes. Assume that $\mathcal{D}$ has an infinite number of delays of value k or greater. To help with the lower bound, we use another algorithm transformation. This transformation converts a delayed on-line learning algorithm C into a traditional on-line algorithm $DO\text{-}C(k)$. We use the k notation because we have a different transformation for each value of k.

The $DO\text{-}C(k)$ algorithm solves a traditional on-line problem. The $DO\text{-}C(k)$ algorithm receives instance $\mathbf{x}_1$ and creates k copies of the instance, $(\mathbf{x}'_1 = \mathbf{x}_1, \ldots, \mathbf{x}'_k = \mathbf{x}_1)$. These k copies are used as the first k instances of algorithm C. When the label y_1 is received, it is copied to k labels. The labels are spaced to give a delay of at least k to each instance. This continues for every trial of $DO\text{-}C(k)$ algorithm, creating k instances and labels for input into the C algorithm. The prediction for $\mathbf{x}_t$ by the $DO\text{-}C(k)$ algorithm is just the random majority prediction of the C algorithm over the k identical instances. By random, we mean the algorithm predicts 1 with a probability equal to the ratio of the k instances that predict 1.

Lemma 2. *Assume you have a delayed on-line learning problem where $\mathcal{D}$ has an infinite number of delays of value k or greater. Let C be a delayed on-line learning algorithm. When running algorithm $DO\text{-}C(k)$ on the same learning problem with all the delays set to 1, $E[Mist(DO\text{-}C(k))] \leq E[Mist(C)]/k$.*

Proof. Consider of sequence of instances, s, for the on-line problem under consideration, where each instance has a delay of 1. Copy each instance k times and give each instance a delay of k or greater. Call this new sequence s'. Running algorithm $DO\text{-}C(k)$ on s is related to running algorithm C on s'. Every instance from s' that is predicted incorrectly increases the probability that

DO-C(k) will make a mistake on that instance by $1/k$ because of the random majority algorithm. Therefore, for all sequences s generated by the adversary, $E[\text{Mist}(DO\text{-}C(k), s)] = E[\text{Mist}(C, s')]/k \leq E[\text{Mist}(C)]/k$. Since this is true for all legal sequences s, $E[\text{Mist}(DO\text{-}C(k))] \leq E[\text{Mist}(C)]/k$. □

The previous lemma implies how the bound for a delayed on-line learning algorithm must grow with k.

Theorem 3. *Assume $E[Mist(Opt)] = M$ for a traditional on-line learning problem. For any delayed learning algorithm C, on the same learning problem, when $\mathcal{D}$ has an infinite number of delays of value k or greater, $E[Mist(C)] \geq kM$.*

Proof. If $E[Mist(C)] < kM$ then we can use Lemma 2 to create a traditional on-line algorithm, *DO-C*(k), where $E[\text{Mist}(DO\text{-}C(k))] \leq E[\text{Mist}(C)]/k < M$. This contradicts the definition of *Opt*. □

There are learning problems that show this lower bound is tight. In addition, we can also give forms of the lower-bound that work for general delay multi-sets. This general form involves the $F(\mathcal{D}, C)$ function. Because of space constraints, we will save these results for a later full version of this paper.

4 Instances Generated by a Distribution

In this section, we give an algorithm transformation for delayed on-line learning when the instances are generated by a shifting distribution. This shifting includes both the target function and the probability of a particular instance. A shifting distribution is a realistic model for many on-line learning problems. Often the learning environment is not trying to maximize the number of mistakes, instead the instances are generated by a distribution that is infrequently or slowly changing. For example, in our hypothetical medical learning problem, the population may be slowly changing dietary habits which could effect the target function.

Let $\Psi = (\mathcal{W}_1, \eta_1()), (\mathcal{W}_2, \eta_2()), \cdots$ be a sequence of distributions and noise functions over $X \times Y$. This model is similar to the adversary model in Sect. 3. The noise function $\eta_i()$ maps the instances to a measure of noise, and the function may change for different algorithms. When using a shifting distribution for the traditional on-line model, we repeatedly allow the environment to either pick an instance from the current distribution or advance to the next distribution. While this is still partly adversarial, it does not give the environment as much freedom since the instance is picked from a distribution.

For the delayed on-line model, the environment selects the delays of the instances from a multi-set $\mathcal{D}$. We assume the environment must select a delay before picking an instance from the distribution. Allowing the environment to select the delays is a worst-case assumption, but it is possible to refine the analysis to allow a distribution to generate the delays.

A key component of the bound is the total amount the distribution changes over the trials. We use variational distance to measure the change between two

distributions. [8] Given discrete distributions $\mathcal{W}_1$ and $\mathcal{W}_2$ over sample space H, let the probability of an element x be $p_1(x)$ for $\mathcal{W}_1$ and $p_2(x)$ for $\mathcal{W}_2$. The variational distance is $V(\mathcal{W}_1, \mathcal{W}_2) = \frac{1}{2}\sum_{x \in H} |p_1(x) - p_2(x)|$. The total variational distance over all trials is $\Psi = \sum_{i=1}^{\infty} V(\mathcal{W}_{i+1}, \mathcal{W}_i)$. This definition generalizes to arbitrary probability measures.

4.1 Algorithm

In this section, we transform an on-line algorithm B to perform well in the delayed on-line model when instances are generated by a shifting distribution. We call the transformed algorithm $\overline{OD3}$-$B(k')$. The $\overline{OD3}$-$B(k')$ algorithm does an update with every instance, and it does the updates in trial order, the same order as B. In other words, $\overline{OD3}$-$B(k')$ can only update the instance from trial $t+1$ after the update for trial t occurs. Also only a single update is allowed per trial, so if multiple labels arrive at the start of the trial, only one can be used for the update. The remaining labels must wait for another trial to perform their update. Therefore, $\overline{OD3}$-$B(k')$ computes the same hypotheses as algorithm B, but it will use them in different trials. There is an exception to the above scheme based on the single parameter k'. This parameter controls the maximum delay $\overline{OD3}$-$B(k')$ will allow for any instance. If an instance has not received its label after $k'+1$ trials then the algorithm pretends the instance does not exist for the purpose of updates. For some problems, this technique is important since otherwise a single early instance with an infinite delay can prevent all later updates. The pseudo-code for $\overline{OD3}$-$B(k')$ is given in Fig. 3. We use a heap U to store the instances that are ready for updates.

The computational cost of the $\overline{OD3}$-$B(k')$ algorithm is similar to the cost of the B and $OD1$-B algorithms. The number of updates is at most the same as algorithm B and the number of predictions is at most double. There is an extra cost based on using the heap. The cost to insert and remove instances from the heap adds an amortized cost of $O(\ln(\min(k', k)))$ per trial. The $\overline{OD3}$-$B(k')$ algorithm needs space for at most $\min(k', k)$ instances. By keeping space for $2\min(k', k)$ instances, it can periodically remove any old instances with only a constant amortized increase in the cost per trial.

4.2 Upper-bound on Mistakes

First, we prove a lemma that bounds the number of trials that do not perform an update. We need the following notation; let μ be the maximum number of instances with a delay greater than k'.

Lemma 3. *When running the $\overline{OD3}$-$B(k')$ algorithm, there are at most $\mu + \min(k', k)$ trials that do not perform an update.*

Proof. Assume that trial t is the $\min(k', k) + \mu + 1$ trial that does not perform an update. Therefore only $t - \min(k', k) - \mu - 1$ labels have been used for updates. Looking at instance x_1 to instance $x_{t-\min(k',k)}$, all of the labels from these

Initialization
$t \leftarrow 0$ is the trial number.
`current` $\leftarrow 0$ is the next instance for an update.
$U \leftarrow$ `null` is a heap that stores instances that are ready for updates.
Initialize algorithm to state $s \leftarrow s_0$.
Trials
$t \leftarrow t + 1$.
Instance: Store $\mathbf{x}_t$ using t as the key.
Prediction: $\hat{y}_t \leftarrow \text{Pred}(s, \mathbf{x}_t)$.
Update:
If $t -$ `current` $= k' + 1$ then
`current` $\leftarrow$ `current` $+ 1$.
For all returned labels $y_{a,t}$
If $a \geq$ `current` then
add instance $(a, x_a, y_{a,t})$ to U.
$a \leftarrow \text{top}(U)$.
If $a =$ `current`
$(a, x_a, y_{a,t}) \leftarrow \text{extract-min}(U)$
$\hat{y} \leftarrow \text{Pred}(s, x_a)$
$s \leftarrow \text{Update}(s, x_a, y_{a,t}, \hat{y})$
`current` $\leftarrow$ `current` $+ 1$.
Remove x_a.
Periodically remove all instances older than `current`.

Fig. 3. delayed on-line algorithm $\overline{OD3}$-$B(k')$

instances that have a delay of at most $\min(k', k)$ must have been returned by trial t. Therefore the minimum number of labels from these instances that have been received is $t - \min(k', k) - \mu$. This means there must be at least one label from this sequence of instances that has not been used for an update. By trial t, this label has been placed on the heap for an update. This is a contradiction since trial t does not perform an update. □

Theorem 4. *Assume the expected number of mistakes of on-line algorithm B is at most M when instances are generated by Ψ. The expected number mistakes made by the $\overline{OD3}$-$B(k')$ algorithm is at most $M + (\mu + \min(k', k) - 1)(\Psi + 1)$ in the delayed distribution model.*

Proof. Let u be the sequence of instances that cause updates in algorithm $\overline{OD3}$-$B(k')$. If we use sequence u on algorithm B with a delay of 1 given to each instance then we expect to make at most M mistakes. This is because the distribution of sequence u can be generated by Ψ.

For algorithm B on sequence u, let h_i be the hypothesis that is used for prediction in trial i, and let X_i be a random variable that is 1 if algorithm B makes a mistake on trial i and 0 otherwise. Let Y_i be a random variable that is 1 if algorithm $\overline{OD3}$-$B(k')$ makes a mistake on trial i and 0 otherwise.

The hypotheses used by $\overline{OD3}$-$B(k')$ are the same as the hypotheses used by algorithm B. The only difference is that hypotheses are shifted a certain positive

number of trials since the instances have to wait for their labels. This will force the $\overline{OD3}$-$B(k')$ algorithm to occasionally use the same hypothesis for multiple trials as the algorithm waits for a label. Based on Lemma 3, the maximum number of trials that do not perform an update for algorithm $\overline{OD3}$-$B(k')$ is $\mu + \min(k', k)$. Since the first trial never performs an update, this means that $\mu + \min(k', k) - 1$ of the hypothesis from algorithm B are reused. Let $r = \mu + \min(k', k) - 1$.

Consider the shifted hypothesis of algorithm $\overline{OD3}$-$B(k')$. When the same hypothesis is used on two related distributions, the accuracies will be similar. The difference in accuracy comes from the amount the distribution changes between the trials. Let $v_t = V(D_{t+1}, D_t)$. Based on this metric, the error-rate of hypothesis h_t may, in the worst case, increase by v_t if used during trial $t+1$. Since each hypothesis is shifted by at most r trials, the error-rate of hypothesis h_t can increase by at most $\sum_{i=t}^{t+r-1} v_i$. We can use this to bound the expected number of mistakes.

$$E[Mist(\overline{OD3}\text{-}B(k'))] = r + \sum_{i=1}^{\infty} E[Y_i] \ .$$

Assuming mistakes on repeated hypotheses and taking into account the number of trials the hypothesis from B are shifted, the above is

$$\leq r + \sum_{i=1}^{\infty} \left(E[X_i] + \sum_{j=i}^{i+r-1} v_j \right) \leq r + E\left[\sum_{i=1}^{\infty} X_i\right] + r \sum_{i=i}^{\infty} v_i \leq r + M + r\Psi \ .$$

This proves the result. □

A possible modification to algorithm $\overline{OD3}$-$B(k')$ is to predict with a random coin flip on any repeated hypothesis. This will lower the upper-bound on mistakes to $M + (\mu + \min(k', k) - 1)(\Psi + 1/2)$. In practice, one may want to restrict a coin flip prediction to repeated hypotheses near the start of the trials, since later repeated hypotheses may have a high accuracy.

4.3 Lower-bound on Mistakes

For a shifting adversary, a trivial lower bound, for the delayed learning problem, is the bound of the optimal algorithm on the traditional on-line learning problem. This lower-bound bound is good when $(\mu + \min(k', k) - 1)(\Psi + 1)$ is small. For example, consider $\Psi = 0$. This forces the distribution to be fixed. Here the bound for $\overline{OD3}$-$Opt(k')$ is $\text{Mist}(Opt) + \mu + \min(k', k) - 1$. The term $\mu + \min(k', k) - 1$ comes from the assumption that $\overline{OD3}$-$Opt(k')$ makes a mistake on all the repeated hypotheses. In the worst-case, these repeated hypothesis can be forced to be the beginning trials of the sequence. Since the algorithm will have no information about the labels of these beginning trials, the probability of a mistake will depend on the algorithm being able to select a initial hypothesis that will guarantee good performance no matter what learning problem the adversary

selects. For many learning problems, this will not be possible, and the best the algorithm can do is to predict with an unbiased coin.

In the full version of the paper, we give an algorithm for learning fixed distributions that slightly improves the bound and removes the parameter k'. This new algorithm's upper-bound on mistakes does not depend on k but instead depends on q, the maximum number of trials before the first label is received. It has an expected bound of $\text{Mist}(Opt) + (q-1)/2$ which, as explained, for many problems is optimal.

We do not have a good lower bound when $\Psi > 0$, however notice that a shifting distribution can duplicate an adversary by using distributions that place all the weight on particular instances. Therefore, the bounds for shifting distributions also covers adversaries. Unfortunately, the distribution based bounds get considerably weaker when the distribution changes frequently. Therefore, when dealing with a problem that is more adversarial, this bound will be quite poor. The distribution bound is most relevant for problems where Ψ is small.

5 Mistake-driven Algorithms

In this section, we restrict ourselves to deterministic traditional on-line algorithms. We prove that a simple transformation for converting any on-line algorithm into a mistake-driven algorithm does not effect the mistake bound for a wide range of adversaries. A mistake-driven algorithm is an algorithm that only changes its internal state when it makes a prediction mistake on an instance.

In a paper by Littlestone [1], a transformation is given to convert an on-line learning algorithm to a mistake-driven algorithm with the same mistake-bound. However, this transformation only applies to learning fixed concepts without noise. We consider a simpler transformation that skips any instances that are correctly classified. The only instances that can effect the state of the algorithm are instances that cause mistakes. The technique was originally used for converting Bayesian algorithms into algorithms that perform well against adversaries.[9, 10] Notationally, we will add the prefix *MD* to any algorithm to show that it has been converted into the mistake-driven form.

We prove that this simple transformation retains the existing mistake bound for a specific type of adversary. We call these special adversaries subset adversaries.

Definition 1. *An adversary is a subset adversary if, for every sequence s of instances that the adversary can generate, the adversary can also generate every subsequence of s.*

The next theorem is used in Sect. 3 to help give a bound on algorithm $OD1$-B when instances are generated by an adversary.

Theorem 5. *For a traditional on-line learning problem with instances generated by a subset adversary, if B is a deterministic on-line algorithm then $Mist(MD\text{-}B) \leq Mist(B)$.*

Proof. Since the instances are generated by a subset adversary, there must exist a sequence of instances s that maximizes the number of mistakes for algorithm *MD-B* where all the mistakes occur at the beginning of the sequence. Up to a certain trial m, both algorithm B and *MD-B* must make identical predictions, updates, and mistakes on instances sequence s. Since *MD-B* makes no further mistakes past trial m, $\text{Mist}(\textit{MD-B}) = Mist(\textit{MD-B}, s) \leq Mist(B, s) \leq \text{Mist}(B)$. □

To understand the limits of a subset adversary, we need a general definition for an adversary. A definition used in many papers is to define an adversary with the set of sequences it is allowed to generate. [7] Call this set of sequences $\mathcal{S}$. Anytime $\mathcal{S}$ is not closed with respect to subsets then the adversary is not a subset adversary. A non-subset adversary must generate correctly classified instances for some algorithm/problem combinations in order to maximize the mistake bound. If these instances contain new information about the target function, they can help lower the mistake bound.

As an example of an non-subset adversary, consider a shifting concept where the concept is forced to shift at specific trials. This is not a subset adversary since all possible subsequences of instances are not allowed. This may force the adversary to generate an instance that will be predicted correctly. However, if we assume that there are default instances that the adversary can always generate that will not give much information about the target concept[2] then mistake-driven algorithms will still give close to the best bounds.

It is an open question as to whether any types of non-subset adversaries are useful for modeling learning problems. One purpose of an adversarial analysis is to show that an algorithm performs well even given the worst-case assumption of an adversary. Since an adversary can always be extended to a subset adversary by adding instance sequences, a subset adversary extends this notion of worst-case. In addition, being a subset adversary is only a sufficient condition for Theorem 5. For many problems, $\text{Mist}(\textit{MD-B}) \leq \text{Mist}(B)$ is true even if the adversary is not a subset adversary.

We want to stress that many practical on-line problems will not have an adversary generating the instances. In these cases, a more aggressive algorithm that sometimes updates on correct predictions can improve performance. [11, 12] Still the algorithm must be careful to avoid extra updates that increase the number of mistakes.

6 Conclusion

In this paper, we give algorithms and mistake bounds for delayed on-line learning. In general, when dealing with an adversary generating the instances, the new bounds can be poor. If the instances all have a delay of k trials then the mistake

[2] This instance could be a previously given instance or an instance that has a known value based on the set of target functions, such as the instance of all zeros for monotone disjunctions.

bound can grow by a factor of $2k$ over the normal on-line learning bounds. We show this bound is within a factor of 2 from optimal. We also give a more general analysis for problems where the adversary must select the delay of instances from a multi-set $\mathcal{D}$. This upper-bound depends on a particular combinatorial property of $\mathcal{D}$ and gives insight to how the algorithm behaves with a range of instance delays. Things are more hopeful when dealing with a distribution generating the instances. In this case, if all the instances have a delay k then the expected mistake bound only increases by $k - 1$. Slightly shifting the distribution also performs well with a penalty based on the amount of shifting.

In the full version of the paper, we will include two additional algorithms and experiments to show how the delayed algorithms perform on a shifting concept using a form of the Winnow algorithm. [13] These additional algorithms fill in the gaps with the numbering convention used in naming our delayed on-line algorithm transformations.

References

1. Littlestone, N.: Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. Machine Learning **2** (1988) 285–318
2. Androutsopoulos, I., Koutsias, J., Chandrinos, K., Paliouras, G., Spyropoulos, C.: An evaluation of naive bayesian anti-spam filtering (2000)
3. Padmanabhan, V.N., Mogul, J.C.: Using predictive prefetching to improve world wide web latency. ACM SIGCOMM Computer Communication Review **26** (1996) 22–36
4. Littlestone, N.: Redundant noisy attributes, attribute errors, and linear-threshold learning using winnow. In: Proceedings of the Third Annual Conference on Computational Learning Theory. (1991) 147–156
5. Helmbold, D.P., Long, P.M.: Tracking drifting concepts using random examples. In: Proceedings of the Third Annual Conference on Computational Learning Theory. (1991) 13–23
6. Kuh, A., Petsche, T., Rivest, R.L.: Learning time-varying concepts. In: Neural Information Processing Systems Three, Morgan Kaufmann Publishers, Inc. (1991) 183–189
7. Auer, P., Warmuth, M.K.: Tracking the best disjunction. In: Proceedings of the 36th annual symposium on foundations of computer science, IEEE Computer Society Press (1995) 312–321
8. Devroye, L., Györfi, L., Lugosi, G.: A Probabilistic Theory of Pattern Recognition. Springer, New York (1991)
9. Littlestone, N.: Comparing several linear-threshold learning algorithms on tasks involving superfluous attributes. In: Proceeding of the Twelve International Conference on Machine Learning. (1995) 353–361
10. Littlestone, N., Mesterharm, C.: An apobayesian relative of winnow. In: Neural Information Processing Systems Nine, MIT Press (1997) 204–210
11. Minsky, M.L., Papert, S.A.: Perceptrons. MIT Press, Cambridge, MA (1969)
12. Li, Y., Long, P.: The relaxed online maximum margin algorithm. In: Neural Information Processing Systems Twelve, MIT Press (2000) 498–504
13. Mesterharm, C.: Tracking linear-threshold concepts with winnow. Journal of Machine Learning Research **4** (2003) 819–838