Regret Minimization With Concept Drift

Koby Crammer[*]
The Technion
koby@ee.technion.ac.il

Eyal Even-Dar
Google Research
evendar@google.com

Yishay Mansour[†]
Tel Aviv University
mansour@cs.tau.ac.il

Jennifer Wortman Vaughan[‡]
Harvard University
jenn@seas.harvard.edu

Abstract

In standard online learning, the goal of the learner is to maintain an average loss close to the loss of the best-performing function in a fixed class. Classic results show that simple algorithms can achieve an average loss arbitrarily close to that of the best function in retrospect, even when input and output pairs are chosen by an adversary. However, in many real-world applications such as spam prediction and classification of news articles, the best target function may be drifting over time. We introduce a novel model of concept drift in which an adversary is given control of both the distribution over input at each time step and the corresponding labels. The goal of the learner is to maintain an average loss close to the 0/1 loss of the best slowly changing sequence of functions with no more than K large shifts. We provide regret bounds for learning in this model using an (inefficient) reduction to the standard no-regret setting. We then go on to provide and analyze an efficient algorithm for learning d-dimensional hyperplanes with drift. We conclude with some simulations illustrating the circumstances under which this algorithm outperforms other commonly studied algorithms when the target hyperplane is drifting.

1 Introduction

Consider the classical problem of online learning. At each time step, the learner is given a new data instance (for example, an email) and must output a prediction of its label (for example, "spam" or "not spam"). The true label is then revealed, and the learner suffers a loss based on both the label and its prediction. Generally in this setting, the goal of the learner is to achieve an average loss that is "not too big" compared to the loss it would have received if it had always chosen to predict according to the best-performing function from a fixed class $\mathcal{F}$. It is well-known that as the number of time steps grows, very simple aggregation algorithms are able to achieve an average loss arbitrarily close to that of the best function in retrospect. Furthermore, such guarantees hold even if the input and output pairs are chosen in a fully adversarial manner with no distributional assumptions [6].

Despite the extensive literature on no-regret learning and the impressive guarantees that can be made, competing with the best fixed function is not always good enough. In many real-world applications, the true target function is not fixed, but is slowly changing over time. Consider a classifier designed to identify news articles related to China. Over time, the most relevant topics might drift from the Olympics to exports to finance to human rights. When this drift occurs, the classifier itself must also change in order to remain relevant. Similarly, the very definition of spam is changing over time as spammers become more creative and deviant. Any useful spam filter must evolve to keep up with this drift.

[*]Some of this research was completed while KC was at University of Pennsylvania. KC is a Horev Fellow, supported by the Taub Foundations.

[†]This work was supported in part by a grant from the Ministry of Science (grant No. 3-6797), by a grant from the Israel Science Foundation (grant No. 709/09), by grant No. 2008-321 from the United States-Israel Binational Science Foundation (BSF), and by the IST Programme of the European Community, under the PASCAL2 Network of Excellence, IST-2007-216886. This publication reflects the authors' views only.

[‡]Some of this research was completed while Vaughan was at Google. Vaughan is supported by NSF under grant CNS-0937060 to the CRA for the CIFellows Project. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors alone.

With such applications in mind, we develop a new theoretical model for regret minimization with concept drift. Here the goal of the algorithm is no longer to compete well with a single function, but to maintain an average loss close to that of the best slowly changing sequence of functions with no more than K large shifts. In order to achieve this goal, it is necessary to restrict the adversary in some way — in classification, if the adversary is given full power over the choice of input and output, it can force any algorithm to suffer a constant regret simply by choosing the direction of drift at random and selecting input points near the decision boundary, even when $K = 0$. To address this problem, early models of drift assumed a fixed input distribution [10] or a joint distribution over input and output that changes slowly over time [1, 15]. More recently, Cavallanti et al. [5] addressed this problem by bounding the number of mistakes made by the algorithm not in terms of the number of mistakes made by the adversary, but in terms of the adversary's hinge loss. (Recall that the hinge loss would assign a positive loss to observations near the decision boundary even if no error is made.) We take a different approach, requiring more traditional regret bounds in terms of the adversary's 0/1 loss while still endowing the adversary with a significant amount of power. We allow the adversary to specify not a single point but a distribution over points at each time. The distributions D_t and D_{t+1} at consecutive times need not be close in any usual statistical sense, and can even have disjoint supports. However, the adversary is prevented from choosing distributions that put too much weight on small regions of input space where pairs of "similar" functions disagree.

Our first algorithmic result shows that learning in this model can be reduced to learning in the standard adversarial online setting. Unfortunately, the resulting algorithms are generally not efficient, in some cases requiring updating an exponential number of weights. Luckily, specialized algorithms can be designed for efficiently learning particular function classes. To gain intuition, we start by providing a simple algorithm for learning one-dimensional threshold functions with drift. We then analyze the performance of the Modified Perceptron algorithm of Blum et al. [4], showing that it can be used to efficiently learn d-dimensional hyperplanes with concept drift.

We conclude with some simulations illustrating the circumstances under which the Modified Perceptron outperforms other algorithms when the target hyperplane is drifting. We find that the Modified Perceptron performs best relative to other algorithms when the underlying dimension of the data is small, even if the data is projected into a high dimensional space. When the underlying dimension of the data is large, the standard Perceptron is equally capable of handling drifting targets. This phenomenon is not explained by our theoretical results, and would be an interesting direction for future research.

2 Related Work

The first model of concept drift was proposed by Helmbold and Long [10]. In their model, at each time t, an input point x_t is drawn from a fixed, unknown distribution D and labeled by a target function f_t, where for each t, the probability that f_t and f_{t+1} disagree on the label of a point drawn from D is less than a fixed value Δ. They showed that a simple algorithm achieves an average error of $\tilde{O}((\Delta d)^{1/2})$ where d is the VC dimension of the function class, or $\tilde{O}((\Delta d)^{1/3})$ in the unrealizable setting. Kuh et al. [13, 14] examined a similar model and provided an efficient algorithm for learning two-dimensional half-planes through the origin and intersections of half-planes through the origin.

Bartlett [1] introduced a more general agnostic model of drift. In this model, the sequence of input and output pairs is generated according to a sequence of joint distributions $P_1, \cdots, P_T$, such that for each t, P_t and P_{t+1} have total variation distance less than Δ. It is easy to verify that the earlier drifting model described above is a special case of this model. Long [15] showed that one can achieve a similar error rates of $O((\Delta d)^{1/2})$ (or $O((\Delta d)^{1/3})$ in the unrealizable setting) in this model, and Barve and Long [3] provided additional upper and lower bounds. Freund and Mansour [8] showed that improvements are possible if the joint distribution is changing at a constant rate. Bartlett et al. [2] also studied a variety of drifting settings, including one in which the target may change arbitrarily but only infrequently.

Most of these models assume a fixed or slowly changing input distribution. At the other extreme lie models in which the input points may be chosen in an arbitrary, adversarial manner. Herbster and Warmuth [11] studied a setting in which the time sequence is partitioned into k segments. The goal of the algorithm is to compete with the best expert in each segment for the best segmentation in retrospect. They later studied algorithms for tracking the best linear predictor with drift [12].

Cavallanti et al. [5] analyzed a variant of the Perceptron algorithm for learning d-dimensional hyperplanes with drift. They bounded the number of mistakes made by the algorithm in terms of the hinge loss of the best sequence of hyperplanes and the amount of drift in a fully adversarial setting. As we briefly discuss in Section 3, there is no way to obtain a result such as theirs in a fully adversarial setting if we wish to measure the regret with respect to the 0/1 loss of the drifting sequence rather

than the hinge loss. We have no choice but to limit the power of the adversary in some way. Finally, Hazan and Seshadhri [9] study drift in the more general online convex optimization setting, providing bounds in terms of the maximum regret (to a single optimal point) achieved over any contiguous time interval. This captures the notion of drift because the optimal point can vary across different time intervals.

Our model falls between these two extremes. On the one hand, we make no requirements on how quickly the distribution over input points can change from one time step to the next, and in fact allow the scenario in which the support of D_t and the support of D_{t+1} do not overlap on any points. However, unlike the purely adversarial models, we require that the distribution chosen at each time step not place "too much" weight on points where pairs of nearby functions disagree. This added requirement gives us the ability to produce bounds in terms of 0/1 loss in situations in which it is provably impossible to learn in a fully adversarial setting.

3 A New Model of Drift

Let $\mathcal{F}$ be a hypothesis class mapping elements in a set $\mathcal{X}$ to elements in a set $\mathcal{Y}$, and let `near` be an arbitrary binary relation over elements in $\mathcal{F}$. For example, if $\mathcal{F}$ is the class of linear separators, we might define $\texttt{near}(f, f')$ to hold if and only if the weight vectors corresponding to f and f' are sufficiently close to each other. At a high level, the `near` relation is meant to encapsulate some notion of similarity between functions. Our model implicitly assumes that it is common for the target to drift from one function to another function near it from one time step to the next.

We say that a sequence of functions $f_1, \cdots, f_T$ is a *K-shift legal sequence* if $\texttt{near}(f_t, f_{t+1})$ holds for at least $T - K$ time steps $t < T$. Unlike standard online learning where the goal is to have low regret with respect to the best single function, the goal in our model is to have low regret with respect to the best K-shift legal sequence. Regret is defined in terms of a loss function $\mathcal{L}$, which is assumed to satisfy the triangle inequality, be bounded in $[0, 1]$, and satisfy $\mathcal{L}(x, x) = 0$ for all x.

In order to achieve this goal, some restrictions must be made. We cannot expect an algorithm to be able to compete in a fully adversarial setting with drift. To understand why, consider for example the problem of online classification with drifting hyperplanes. Here the adversary can force any algorithm to have an average loss of 1/2 by simply randomizing the direction of drift at each time step and choosing input points near the decision boundary, even when $K = 0$. As such, we work in a setting in which the adversary may specify not a single point but a distribution over points. In particular, the adversary may specify any distribution that is "good" in the following precise sense.[1]

Definition 1 *A distribution D is λ-**good** for loss function $\mathcal{L}$ and binary relation **near** if for all pairs $f, f' \in \mathcal{F}$ such that $\textbf{near}(f, f')$, $E_{x \sim D}\left[\mathcal{L}(f(x), f'(x))\right] \leq \lambda$.*

For most of this paper, we restrict our attention to the problem of online classification with $\mathcal{Y} = \{+1, -1\}$ and define $\mathcal{L}$ to be 0/1 loss. In this case, D is λ-good if for every $f, f' \in \mathcal{F}$ such that $\texttt{near}(f, f')$, $\Pr_{x \sim D}\left(f(x) \neq f'(x)\right) \leq \lambda$. Restricting the input distribution in this way ensures that the adversary cannot place too much weight on areas of the input space where pairs of `near` functions disagree, while still providing the adversary with the power to select arbitrarily different distributions from one time step to the next. Note that the larger the space of `near` pairs is, the smaller the set of λ-good distributions, and vice versa. When the `near` space is empty, every distribution is λ-good. At the other one extreme, the set of λ-good distributions might be empty (for example, if the function that classifies all points as positive and the function that classifies all points as negative are defined to be `near`). We restrict our attention only to triples ($\mathcal{F}$, `near`, λ) such that at least one λ-good distribution exists.

The majority of our results hold in the following adversarial setting. Fix a value of λ and definition of `near`. At each time $t \in \{1, \cdots, T\}$, the learning algorithm chooses a (possibly randomized) hypothesis h_t. The adversary then chooses an arbitrary λ-good distribution D_t and an arbitrary function $f_t \in \mathcal{F}$. The algorithm is presented with a point x_t distributed according to D_t, learns the label $f_t(x_t)$, and receives a loss $\mathcal{L}(h_t(x_t), f_t(x_t))$. Let K be the number of time steps t for which $\texttt{near}(f_t, f_{t+1})$ does not hold. (We are usually interested in the case in which K is a small constant.) Then by definition, $f_1, \cdots, f_T$ is a K-shift legal sequence. The goal of the learning algorithm is to maintain low expected regret with respect to the best K-shift legal sequence (where the expectation is taken with respect to the random sequence of input points and any internal randomization of the algorithm), which is equivalent to maintaining a small expected average loss since a perfect

[1] Of course it could be possible to obtain results by restricting the adversary in other ways, such as requiring that points be chosen to respect a minimum margin assumption. However, some restriction is needed.

K-shift legal sequence is guaranteed to exist. The adversary's choice of D_t and f_t may depend on any available historical information, including the sequence of input points $x_1, \cdots, x_{t-1}$, and on the algorithm itself. The algorithm has no knowledge of the number of shifts K or the times at which these shifts occur. We refer to this scenario as the *realizable* setting.

We also briefly consider an *unrealizable* setting in which the adversary is not required to choose f_t on the fly each time step. Instead, the adversary selects an arbitrary λ-good distribution D_t (again for a fixed value of λ) and a distribution over labels y_t conditioned on x_t. The algorithm is presented with a point x_t distributed according to D_t and a label y_t distributed according to the distribution chosen by the adversary and receives a loss of $\mathcal{L}(h_t(x_t), y_t)$. In this setting, the goal is to maintain low expected regret with respect to the best K-shift legal sequence in retrospect for a fixed value of K, where the expectation is taken with respect to the random input sequence, random labels, and any randomization of the algorithm, and the regret is defined as

$$\sum_{t=1}^{T} \mathcal{L}(h_t(x_t), y_t) - \min_{f_1, \cdots, f_T \in \Phi_K} \sum_{t=1}^{T} \mathcal{L}(f_t(x_t), y_t),$$

where Φ_K is the set of all K-shift legal sequences $f_1, \cdots, f_T$.

Note that unlike the standard online learning setting, we should not expect the regret per round to go to zero, even in the realizable setting. Suppose that the target is known perfectly at some time t. It is still possible for the algorithm to make an error at time $t+1$ because the target can move. This uncertainty never goes away, even as the number of time steps grows very large, so we should expect to see a dependence on λ in the average regret that does not diminish over time. This inherent uncertainty is the very heart of the drifting problem.

4 A General Reduction

We now provide general upper bounds for learning finite function classes in the model. We show via a simple reduction that it is possible to achieve an expected average per time step regret of $O((\lambda \log N)^{1/3})$, and that this regret can be reduced to $O(\sqrt{\lambda \log N})$ in the realizable setting.[2] However, the algorithm used is not always efficient when the function class is very large or infinite. The subsequent sections are devoted to efficient algorithms for particular function classes.

The results rely on the following lemma.

Lemma 2 *Let $\mathcal{L}$ be any loss function with $\mathcal{L}(x, x) = 0$ for all x that satisfies the triangle inequality. For any 0-shift legal sequence $f_1, \cdots, f_\ell$, and any sequence of joint distributions $P_1, \cdots, P_\ell$ over pairs $\{x_1, y_1\}, \cdots, \{x_\ell, y_\ell\}$ such that the marginal distributions $D_1, \cdots, D_\ell$, over $x_1, \cdots, x_\ell$ are λ-good, there exists a function $f \in \mathcal{F}$ such that $\sum_{t=1}^{\ell} E_{\{x_t, y_t\} \sim P_t} [\mathcal{L}(f(x_t), y_t)] \leq \sum_{t=1}^{\ell} E_{\{x_t, y_t\} \sim P_t} [\mathcal{L}(f_t(x_t), y_t)] + (\ell - 1)^2 \lambda / 2$.*

Proof: We first show by induction that for any λ-good distribution D and any 0-shift legal sequence $f_1, \cdots, f_\ell$, $\mathrm{E}_{x \sim D} [\mathcal{L}(f_1(x), f_\ell(x))] \leq (\ell - 1)\lambda$. This clearly holds for $\ell = 1$. Suppose that $\mathrm{E}_{x \sim D} [\mathcal{L}(f_1(x), f_{\ell-1}(x))] \leq (\ell - 2)\lambda$. Since the functions form a 0-shift legal sequence and D is λ-good, we must have $\mathrm{E}_{x \sim D} [\mathcal{L}(f_{\ell-1}(x), f_\ell(x))] \leq \lambda$. By the triangle inequality and linearity of expectation, $\mathrm{E}_{x \sim D} [\mathcal{L}(f_1(x), f_\ell(x))] \leq \mathrm{E}_{x \sim D} [\mathcal{L}(f_1(x), f_{\ell-1}(x)) + \mathcal{L}(f_{\ell-1}(x), f_\ell(x))] \leq (\ell - 1)\lambda$.

This implies that for any t, $\mathrm{E}_{x_t \sim D_t} [\mathcal{L}(f_1(x_t), y_t)] \leq \mathrm{E}_{x_t \sim D_t} [\mathcal{L}(f_t(x_t), y_t) + \mathcal{L}(f_1(x_t), f_t(x_t))] \leq \mathrm{E}_{x_t \sim D_t} [\mathcal{L}(f_t(x_t), y_t)] + (t - 1)\lambda$. Summing over all t yields the lemma. ■

The following theorem provides a general upper bound for the unrealizable setting.

Theorem 3 *Let $\mathcal{F}$ be a finite function class of size N and* `near` *be any binary relation on $\mathcal{F}$ that yields a non-empty set of λ-good distributions. There exists an algorithm for learning $\mathcal{F}$ that achieves an average expected regret of $O\left((\lambda \ln N)^{1/3}\right)$ when $T \geq (\ln N)^{1/3} \lambda^{-2/3}$ for any $K \leq \lambda T$, even in the unrealizable setting.*

Proof: Let $\mathcal{A}$ be any regret minimization algorithm for $\mathcal{F}$ that is guaranteed to have regret at most $r(m)$ over m time steps. We can construct an algorithm for learning $\mathcal{F}$ using $\mathcal{A}$ as a black box. The algorithm simply divides the sequence of T time steps into $\lceil T/m \rceil$ consecutive subsequences of length at most m and runs $\mathcal{A}$ on each of these subsequences.

[2] Here and throughout this paper, we consider the asymptotic behavior of functions as $\lambda \to 0$ (or equivalently, as $1/\lambda \to \infty$). This implies, for example, that an error of $O(\sqrt{\lambda})$ is preferred to an error of $O(\lambda^{1/3})$.

The regret of the algorithm with respect to the best function in $\mathcal{F}$ on each subsequence is at most $r(m)$. Furthermore, by Lemma 2, the best function in $\mathcal{F}$ on this subsequence has a regret of no more than $m^2\lambda$ with respect to any legal sequence and thus also the best legal sequence. Combining these facts with the fact that the error can be no more than m on any subsequence yields a bound of

$$\left\lceil \frac{T}{m} \right\rceil \left(r(m) + \frac{m^2\lambda}{2} \right) + Km \leq \left(\frac{T}{m} + 1 \right) \left(r(m) + \frac{m^2\lambda}{2} \right) + Km$$

on the total expected regret of the algorithm with respect to the best K-shift legal sequence.

There exist well-known algorithms for finite classes with regret $r(m) = O(\sqrt{m \ln N})$ [6]. Letting $\mathcal{A}$ be one of these algorithms and setting $m = (\ln N)^{1/3}\lambda^{-2/3}$ yields the bound. ■

The following theorem shows that in the realizable setting, it is possible to obtain an average loss of $O(\sqrt{\lambda \ln N})$ as long as T is sufficiently large and K sufficiently small compared with T. This is an improvement on the previous bound whenever the bound is not trivial, i.e., when $\lambda \ln N < 1$.

Theorem 4 *Let $\mathcal{F}$ be a finite function class of size N and* `near` *be any binary relation on $\mathcal{F}$ such that the set of λ-good distributions is not empty. There exists an algorithm for learning $\mathcal{F}$ in the realizable setting that achieves an expected average per time step loss of $O(\sqrt{\lambda \ln N})$ when $T \geq \sqrt{\ln N/\lambda}$ for any $K \leq T\lambda$.*

Proof Sketch: The proof is nearly identical to the proof of Theorem 3. The only differences are that the regret minimization algorithm employed must guarantee a regret of $O(\sqrt{L^* \ln N} + \ln N)$ where L^* is the loss of the best expert (see Cesa-Bianchi and Lugosi [6] for examples), and m must be set to $\sqrt{\ln N/\lambda}$. The proof then relies on the fact that in the realizable setting, on any period of length m during which no shift occurs, $L^* \leq m^2\lambda$ (from Lemma 2). ■

The results are easily extended to the case in which $\mathcal{F}$ is not finite but has finite VC dimension d, but hold only under certain restricted definitions of `near`.

5 Efficient Algorithms for Drifting Thresholds and Hyperplanes

The reductions described in the previous section are not efficient in general and may require maintaining an exponential number of weight for infinite function classes. In this section, we analyze an efficient Perceptron-style algorithm for learning drifting d-dimensional hyperplanes. To promote intuition, we begin by describing and analyzing a simple specialized algorithm for learning one-dimensional thresholds. The analysis of the Perceptron-style algorithm uses similar ideas.

5.1 One-Dimensional Thresholds

We denote by $\tau_t \in [0, 1]$ the threshold corresponding to the target function f_t; thus $f_t(x) = 1$ if and only if $x \geq \tau_t$. For any two functions f and f' with corresponding thresholds τ and τ', we say that the relation $\texttt{near}(f, f')$ holds if and only if $|\tau - \tau'| \leq \gamma$ for some fixed $\gamma \leq \lambda$. By definition, this implies that any λ-good input distribution D_t can place at most weight λ on any interval of width γ.

A simple algorithm can be used to achieve optimal error bounds in the realizable setting. At each time t, the algorithm keeps track of an interval I_t of threshold values corresponding to all functions that could feasibly be the current target if no major shift has recently occurred. When the input point x_t is observed, the algorithm predicts the label selected by the majority of the threshold values in I_t (that is, 0 if x_t is closer to the lower border of I_t and 1 if it is closer to the upper border).

To start, I_1 is initialized to the full interval $[0, 1]$ since any threshold is possible. At each subsequent time t, one of three things happens. If $x_t \notin I_t$, then the entire feasible set agrees on the label of x_t. If the predicted label is correct, then to allow for the possibility of concept drift, the algorithm sets I_{t+1} to be I_t increased by γ on each side. If the predicted label is incorrect, then it must be the case that a shift has recently occurred, and I_{t+1} is reset to the full interval $[0, 1]$. (Note that this can happen at most K times.) On the other hand, if $x_t \in I_t$, then there is disagreement in the shift-free interval about the label of the point and the algorithm learns new information about the current threshold by learning the label. In this case, all infeasible thresholds are removed from the shift-free feasible set and then again γ is added on each side to account for possible concept drift. Namely, if $x_t \in I_t = (a, b)$ then I_{t+1} is either $(a - \gamma, x_t + \gamma)$ or $(x_t - \gamma, b + \gamma)$. The next theorem shows that this algorithm results in error $O(\sqrt{\lambda})$ as long as T is sufficiently large.

Theorem 5 *Let $\mathcal{F}$ be the class of one-dimensional thresholds and let* `near` *be defined as above. The expected average error of the algorithm described above in the realizable setting is no more than $K/T + \sqrt{2(K+1)\lambda/(T\gamma)} + \sqrt{5\lambda}$.*

Proof: Let $\texttt{inc}_t$ be a random variable that is 1 if the label of the input point at time t is inconsistent with all hypotheses in the version space and 0 otherwise; if $\texttt{inc}_t = 1$, then the algorithm described above makes a mistake at time t and sets I_{t+1} to the full interval $[0, 1]$. Note that $\sum_{t=1}^{T} \texttt{inc}_t \leq K$.

Let $\texttt{err}_t$ be a random variable that is 1 if the label of the input at time t is consistent with some hypothesis in the version space but the algorithm makes an error anyway and 0 if this is not the case. For any positive-width interval I and λ-good distribution D, $D(I) \leq \lceil |I|/\gamma \rceil \lambda \leq (|I|/\gamma + 1)\lambda$, where $|I|$ is the length of interval I. Hence, at every time step t, $\Pr(\texttt{err}_t = 1 | I_t, f_t, D_t) \leq D_t(I_t) \leq |I_t|\lambda/\gamma + \lambda$, and so $|I_t| \geq (\gamma/\lambda)\Pr(\texttt{err}_t = 1 | I_t, f_t, D_t) - \gamma$.

Since the algorithm predicts according to the majority of the (shift-free) feasible set, it eliminates at least half of the hypotheses in this set on each consistent mistake. However, at every time step, the feasible set can grow by γ on each side. Thus,

$$\begin{aligned}
\mathrm{E}\left[|I_{t+1}| \mid I_t, f_t, D_t\right] &\leq \Pr(\texttt{err}_t = 1 | I_t, f_t, D_t)\left(|I_t|/2 + 2\gamma\right) \\
&\quad + \left(1 - \Pr(\texttt{err}_t = 1 | I_t, f_t, D_t)\right)\left(|I_t| + 2\gamma\right) + \Pr(\texttt{inc}_t = 1 | I_t, f_t, D_t) \cdot 1 \\
&= |I_t| + 2\gamma - (|I_t|/2)\Pr(\texttt{err}_t = 1 | I_t, f_t, D_t) + \Pr(\texttt{inc}_t = 1 | I_t, f_t, D_t) \\
&\leq |I_t| + 5\gamma/2 - (\gamma/(2\lambda))\Pr(\texttt{err}_t = 1 | I_t, f_t, D_t)^2 + \Pr(\texttt{inc}_t = 1 | I_t, f_t, D_t),
\end{aligned}$$

where the final step follows from the lower bound on $|I_t|$ given above. Taking the expectation of both sides with respect to $\{I_t, f_t, D_t\}$ gives us that for any t,

$$\begin{aligned}
\mathrm{E}\left[|I_{t+1}|\right] &= \mathrm{E}\left[|I_t|\right] + 5\gamma/2 - (\gamma/(2\lambda))\mathrm{E}\left[\Pr(\texttt{err}_t = 1 | I_t, f_t, D_t)^2\right] + \mathrm{E}\left[\Pr(\texttt{inc}_t = 1 | I_t, f_t, D_t)\right] \\
&\leq \mathrm{E}\left[|I_t|\right] + 5\gamma/2 - (\gamma/(2\lambda))\left(\Pr(\texttt{err}_t = 1)\right)^2 + \Pr(\texttt{inc}_t = 1),
\end{aligned}$$

where the last step follows from the convexity of x^2. Summing over all time steps gives us

$$\sum_{t=1}^{T} \mathrm{E}\left[|I_{t+1}|\right] \leq \sum_{t=1}^{T} \mathrm{E}\left[|I_t|\right] + 5\gamma T/2 - (\gamma/(2\lambda)) \sum_{t=1}^{T} \left(\Pr(\texttt{err}_t = 1)\right)^2 + \sum_{t=1}^{T} \Pr(\texttt{inc}_t = 1) \ .$$

Noting that $\mathrm{E}\left[|I_1|\right] = 1$ and $\mathrm{E}\left[|I_{T+1}|\right] \geq 0$, multiplying both sides by $2\lambda/(\gamma T)$, and rearranging terms gives us

$$\frac{1}{T} \sum_{t=1}^{T} \left(\Pr(\texttt{err}_t = 1)\right)^2 \leq \frac{2\lambda}{\gamma T} + 5\lambda + \frac{2\lambda}{\gamma T} \sum_{t=1}^{T} \Pr(\texttt{inc}_t = 1) \leq \frac{2\lambda}{\gamma T}(K + 1) + 5\lambda \ . \tag{1}$$

The last inequality holds because $\sum_{t=1}^{T} \Pr(\texttt{inc}_t = 1) = \sum_{t=1}^{T} \mathrm{E}\left[\texttt{inc}_t\right] = \mathrm{E}\left[\sum_{t=1}^{T} \texttt{inc}_t\right] \leq K$. Applying Jensen's inequality to the left-hand side and taking the square root of both sides, we get

$$\frac{1}{T} \sum_{t=1}^{T} \Pr(\texttt{err}_t = 1) \leq \sqrt{\frac{2\lambda(K+1)}{T\gamma} + 5\lambda} \leq \sqrt{\frac{2\lambda(K+1)}{T\gamma}} + \sqrt{5\lambda} \ .$$

This allows us to bound the expected average error with

$$\mathrm{E}\left[\frac{1}{T} \sum_{t=1}^{T} \left(\texttt{err}_t + \texttt{inc}_t\right)\right] = \frac{1}{T} \sum_{t=1}^{T} \Pr(\texttt{err}_t = 1) + \frac{1}{T} \sum_{t=1}^{T} \Pr(\texttt{inc}_t = 1) \leq \sqrt{\frac{2\lambda(K+1)}{T\gamma}} + \sqrt{5\lambda} + \frac{K}{T} \ .$$

■

The following theorem shows that the dependence on λ cannot be significantly improved. The proof is in the appendix.

Theorem 6 *Any algorithm for learning one-dimensional thresholds in the realizable setting (i.e., $K = 0$) under the definition of* ***near*** *stated above must suffer error $\Omega(\sqrt{\lambda})$.*

5.2 Hyperplanes

We now move on to the more interesting problem of efficiently learning hyperplanes with drift. For any two normalized vectors u and u', let $\theta(u, u') = \arccos(u \cdot u')$ denote the angle between u and u'. We define $\texttt{near}(u, u')$ to hold if and only if $\theta(u, u') \leq \gamma$ for some fixed parameter $\gamma \in (0, \pi/2)$.

At each time step t, the adversary selects an arbitrary λ-good distribution D_t over unit-length d-dimensional points and a unit-length weight vector u_t such that the set $u_1, \cdots, u_T$ forms a K-shift legal sequence.[3] The input point x_t is then drawn from D_t and assigned the label $\text{sign}(u_t \cdot x_t)$.

We analyze the Modified Perceptron algorithm originally proposed by Blum et al. [4] and later studied by Dasgupta et al. [7] in the context of active learning. This algorithm maintains a current weight vector w_t. The initial weight vector w_1 can be selected arbitrarily. At each time step t, when the algorithm observes the point x_t, it predicts the label $\text{sign}(w_t \cdot x_t)$. If the algorithm makes a mistake at time t, it sets $w_{t+1} = w_t - 2(w_t \cdot x_t)x_t$, otherwise no update is made and $w_{t+1} = w_t$. The factor of 2 in the update rule enforces that $||w_t|| = 1$ for all t as long as $||x_t|| = 1$. Note that unlike the algorithm for thresholds, this algorithm *does not* require any knowledge of γ.

We begin with a lemma which extends Lemma 3 of Dasgupta et al. [7] from a uniform distribution to a λ-good distribution. The intuition behind the proofs is similar. At a high level, we need to show that the adversary cannot place too much weight on points close to the algorithm's current decision boundary. Thus if the algorithm's probability of making a mistake is high, then there is a significant probability that the mistake will be on a point far from the boundary and significant progress will be made. In this lemma and the results that follow, let $\texttt{err}_t$ be a random variable that is 1 if the algorithm makes an error at time t and 0 otherwise.

Lemma 7 *Consider the Modified Perceptron. At every time t, $w_{t+1} \cdot u_t \geq w_t \cdot u_t$. Furthermore, there exists a positive constant $c \leq 10$ such that for all t, for any $\eta \in (0, 1/2)$, if $Pr(\texttt{err}_t | w_t, u_t, D_t) \geq 2c\eta\lambda/\gamma + 4\lambda$, then with probability at least $Pr(\texttt{err}_t | w_t, u_t, D_t) - (2c\eta\lambda/\gamma + 4\lambda)$, we have $1 - w_{t+1} \cdot u_t \leq \left(1 - \eta^2/d\right)(1 - w_t \cdot u_t)$.*

The proof relies on the following fact about λ-good distributions under the current definition of `near`.

Lemma 8 *There exists a positive constant $c \leq 10$ such that for any $\eta \in (0, 1/2)$, for any d-dimensional vector w such that $||w|| = 1$, for any λ-good distribution D, $Pr_{x \sim D}(|w \cdot x| \leq \eta/\sqrt{d}) \leq c\eta\lambda/\gamma + 2\lambda$.*

The proof of this lemma is based on the following intuition. Consider the pair of hyperplanes corresponding to any two weight vectors w_1 and w_2 such that the angle between w_1 and w_2 is at most γ. Let Δ be the set of points x on which these hyperplanes disagree, i.e., all x such that $\text{sign}(w_1 \cdot x) \neq \text{sign}(w_2 \cdot x)$. Since D is λ-good, $D(\Delta) \leq \lambda$. The idea of the proof is to cover at least half of the points x such that $|w \cdot x| \leq \eta/\sqrt{d}$ using k sets like Δ, implying that the total weight D assigns to these points is at most $k\lambda$. In particular, we show that it is possible to construct such a cover with $k \leq 5\eta/\gamma + 1$, implying that the total probability D can place on points x such that $|w \cdot x| \leq \eta/\sqrt{d}$ is bounded by $10\eta\lambda/\gamma + 2\lambda$. The full proof appears in the appendix.

We are now ready to prove Lemma 7.

Proof of Lemma 7: The first half of the lemma is trivial if no mistake is made since $w_{t+1} = w_t$ in this case. If a mistake is made, then $w_{t+1} \cdot u_t = w_t \cdot u_t - 2(w_t \cdot x_t)(x_t \cdot u_t)$. Since there was an error, $\text{sign}(w_t \cdot x_t) \neq \text{sign}(x_t \cdot u_t)$ and $2(w_t \cdot x_t)(x_t \cdot u_t) < 0$.

For the second half, by Lemma 8 and the union bound, $\Pr\left(|w_t \cdot x_t||u_t \cdot x_t| \leq \eta^2/d\right) \leq 2c\eta\lambda/\gamma + 4\lambda$. Thus if $\Pr(\texttt{err}_t | w_t, u_t, D_t) > 2c\eta\lambda/\gamma + 4\lambda$, then the probability that an error is made and $|w_t \cdot x_t||u_t \cdot x_t| > \eta^2/d$ is at least $\Pr(\texttt{err}_t | w_t, u_t, D_t) - (2c\eta\lambda/\gamma + 4\lambda)$. Suppose this is the case. Then, as desired,

$$\begin{aligned} 1 - w_{t+1} \cdot u_t &= 1 - w_t \cdot u_t + 2(w_t \cdot x_t)(x_t \cdot u_t) = 1 - w_t \cdot u_t - 2|w_t \cdot x_t||x_t \cdot u_t| \\ &\leq 1 - w_t \cdot u_t - \frac{2\eta^2}{d} \leq 1 - w_t \cdot u_t - \frac{2\eta^2}{d}\frac{1 - w_t \cdot u_t}{2} = (1 - w_t \cdot u_t)\left(1 - \frac{\eta^2}{d}\right). \end{aligned}$$

■

Corollary 9 below follows from a simple application of technical properties of the cosine function.

Corollary 9 *There exists a constant $c \leq 10$ such that for any $\eta \in (0, 1/2)$, if $Pr(\texttt{err}_t | w_t, u_t, D_t) > 2c\eta\lambda/\gamma + 4\lambda$, then with probability at least $Pr(\texttt{err}_t | w_t, u_t, D_t) - (2c\eta\lambda/\gamma + 4\lambda)$,*

$$\theta(w_{t+1}, u_t) \leq \sqrt{1 - \frac{\eta^2}{d}}\theta(w_t, u_t) \leq \left(1 - \frac{\eta^2}{2d}\right)\theta(w_t, u_t) \ .$$

[3] The assumption that $||u_t|| = ||x_t|| = 1$ simplifies our presentation of results and nothing more. By modifying the definition of `near` and the update rule in a straight-forward manner, all of the results in this section can be extended to hold when the assumption is not true.

Finally, the following theorem uses these lemmas to bound the average error of the Modified Perceptron. The evolution of the angle between w_t and u_t is analyzed over time, similarly to how the evolution of $|I_t|$ was analyzed in the proof of Theorem 5. The result for general values of K is obtained by breaking the sequence down into (no more than) $K+1$ subsequences on which there is no shift, applying a similar analysis on each subsequence, and summing the error bounds. This analysis is possible only if the time steps at which the shifts occur are fixed in advance, though it does *not* require that the algorithm is aware of the shifts in advance. The optimal setting of η and a brief interpretation of the resulting bounds are given below.

Theorem 10 *Let $\mathcal{F}$ be the class of hyperplanes and let $\texttt{near}(u, u')$ hold if and only if $\arccos(u \cdot u') \leq \gamma$ for a fixed parameter $\gamma \leq \lambda$. There exists a constant $c \leq 10$ such that when $K = 0$, for any $\eta \in (0, 1/2)$, the expected average error of the Modified Perceptron algorithm is no more than*

$$\left(1 + \frac{2\pi}{T\gamma} + \frac{\eta^2}{2d}\right) \frac{\lambda d}{q\eta^2} + 2q$$

where $q = (c\eta\lambda/\gamma + 2\lambda)$.

If the adversary chooses the time steps t at which a shift will occur in advance (yet, unknown to the learner), then for any K, for any $\eta \in (0, 1/2)$, the expected average error of the Modified Perceptron algorithm is no more than

$$\frac{K+1}{T} + \left(1 + \frac{2\pi(K+1)}{T\gamma} + \frac{\eta^2}{2d}\right) \frac{\lambda d}{q\eta^2} + 2q \ .$$

The bounds stated in this theorem can be difficult to interpret. Before jumping into the proof, let us take a moment to examine them in more detail to understand what this theorem really means. Setting $\eta = (d/\lambda)^{1/4}\gamma^{1/2}$ in Theorem 10, we obtain that when $T >> (K+1)/\gamma$, the average error is bounded by $O(\lambda^{1/4} d^{1/4} \sqrt{\lambda/\gamma})$). If we think of γ as a constant fraction of λ, then this bound is essentially $O(\lambda^{1/4} d^{1/4})$. We do not know if it is possible to improve this bound to achieve an error of $O(\sqrt{\lambda d})$, which would match the bound of the inefficient algorithm presented in Section 4. Certainly such a bound would be desirable. However, this bound tells us that for hyperplanes, some amount of drift-resistance is possible with an efficient algorithm.

Note that in order for the bound to be non-trivial, γ must be small compared to $1/d$, in which case η is less than $1/2$.

Proof of Theorem 10: We first prove the result for $K = 0$ and then briefly discuss the how to extend the proof to cover general values of K.

Let $\theta_t = \theta(w_t, u_t)$. By definition, $\Pr(\texttt{err}_t = 1 | w_t, u_t, D_t) \leq (\theta_t/\gamma + 1)\lambda$, and $\theta_t \geq \Pr(\texttt{err}_t = 1 | w_t, u_t, D_t)\,\gamma/\lambda - \gamma$. By Lemma 7, for all t, $w_{t+1} \cdot w_t \geq w_t \cdot u_t$. Thus $\theta(w_{t+1}, u_t) \leq \theta(w_t, u_t)$, and since we have assumed no shifts, $\theta_{t+1} \leq \theta_t + \gamma$. We will show that this implies that for any t,

$$\mathrm{E}\left[\theta_{t+1} | w_t, u_t, D_t\right] \leq \theta_t + \left(1 + \frac{\eta^2}{2d}\right)\gamma - \left(\Pr(\texttt{err}_t = 1 | w_t, u_t, D_t) - 2q\right) \frac{\eta^2 q \gamma}{d\lambda} \ , \tag{2}$$

where $q = (c\eta\lambda/\gamma + 2\lambda)$. This clearly holds if $\Pr(\texttt{err}_t = 1 | w_t, u_t, D_t) \leq 2q$ since η^2 and d are positive and in this case the last term is negative. Suppose instead that $\Pr(\texttt{err}_t = 1 | w_t, u_t, D_t) > 2q = 2(c\eta\lambda/\gamma + 4\lambda)$. By Corollary 9 and the bounds above,

$$\begin{aligned}
&\mathrm{E}\left[\theta_{t+1} | w_t, u_t, D_t\right] \\
&\leq \left(\Pr(\texttt{err}_t = 1 | w_t, u_t, D_t) - 2q\right)\left(1 - \frac{\eta^2}{2d}\right)\theta_t + \left(1 - \left(\Pr(\texttt{err}_t = 1 | w_t, u_t, D_t) - 2q\right)\right)\theta_t + \gamma \\
&\leq \theta_t + \gamma - \left(\Pr(\texttt{err}_t = 1 | w_t, u_t, D_t) - 2q\right)\frac{\eta^2}{2d}\left(\frac{\Pr(\texttt{err}_t = 1 | w_t, u_t, D_t)\,\gamma}{\lambda} - \gamma\right) \\
&\leq \theta_t + \gamma - \left(\Pr(\texttt{err}_t = 1 | w_t, u_t, D_t) - 2q\right)\frac{\eta^2}{2d}\left(\frac{2q\gamma}{\lambda} - \gamma\right)
\end{aligned}$$

which implies Equation 2. Now, taking the expectation over $\{w_t, u_t, D_t\}$ of both sides of Equation 2, we get that for any t

$$\mathrm{E}\left[\theta_{t+1}\right] \leq \mathrm{E}\left[\theta_t\right] + \left(1 + \frac{\eta^2}{2d}\right)\gamma - \left(\Pr(\texttt{err}_t) - 2q\right)\frac{\eta^2 q \gamma}{d\lambda}.$$

Summing over time steps, we then have that

$$\sum_{t=1}^{T} \mathrm{E}\left[\theta_{t+1}\right] \leq \sum_{t=1}^{T} \mathrm{E}\left[\theta_t\right] + \left(1 + \frac{\eta^2}{2d}\right)\gamma T - \frac{\eta^2 q\gamma}{d\lambda}\left(\sum_{t=1}^{T} \Pr\left(\mathtt{err}_t\right) - 2qT\right).$$

Since $\theta_t \in [0, 2\pi)$ for all t, this implies that

$$0 \leq 2\pi + \left(1 + \frac{\eta^2}{2d}\right)\gamma T - \frac{\eta^2 q\gamma}{d\lambda}\left(\sum_{t=1}^{T} \Pr\left(\mathtt{err}_t\right) - 2qT\right).$$

Rearranging terms and multiplying both sides by $d\lambda/(\eta^2 q\gamma)$ yields

$$\sum_{t=1}^{T} \Pr\left(\mathtt{err}_t\right) \leq \frac{2\pi d\lambda}{\eta^2 q\gamma} + \left(1 + \frac{\eta^2}{2d}\right)\frac{d\lambda}{\eta^2 q}T + 2qT\ .$$

Dividing both sides by T gives the desired bound on error.

To get the bound for general K, note that the analysis leading up to this past equation can be applied to all subsequences during which there is no shift. Summing the above bound for all such subsequences (where T is replaced by the length of the subsequence) with $\Pr\left(\mathtt{err}_t\right)$ pessimistically bounded by 1 whenever a shift occurs between times t and $t+1$ leads to the bound. ■

6 Simulations

In this section, we discuss a series of simulations on synthetic data designed to illustrate the effectiveness of different algorithms for learning drifting hyperplanes. In particular, we compare the performance of the standard Perceptron algorithm [16], the Shift Perceptron [5], the Randomized Budget Perceptron [5], and the Modified Perceptron [4, 7] analyzed above. Like the Perceptron algorithm, the Shift Perceptron maintains a vector of weights, but each time a mistake is made, the Shift Perceptron shrinks its current weight vector towards zero in a way that depends on both the current number of mistakes and a parameter λ. The Randomized Budget Perceptron is similar but additionally tracks the set of examples that contribute to its current weight vector. If the size of this set exceeds a predetermined budget B, one example is randomly removed and the weight vector is updated by removing the contribution of this example.

For each experiment, the sequence $\mathbf{x}_1, \mathbf{x}_2, \cdots$ of synthetic data points was generated as follows. We first generated 5000 d-dimensional random points $\mathbf{z}_1, \cdots, \mathbf{z}_{5000}$ drawn from a zero-mean unit-covariance Gaussian distribution. We then generated a random $D \times d$ linear transformation matrix A, and used this to project each d-dimensional point $\mathbf{z}_t$ to a D-dimensional vector $\mathbf{x}_t = A\mathbf{z}_t$. The resulting data points were thus D-dimensional points with a true underlying dimension of d. We fixed $D = 1000$ and experimented with various values of d between 5 and 500.

We generated each sequence of randomly drifting target weight vectors $\mathbf{u}_1, \mathbf{u}_2, \cdots$ as follows. To start, $\mathbf{u}_1$ was drawn from a zero-mean unit-covariance D-dimensional Gaussian distribution. In the first set of experiments, which we refer to as the *random drift* experiments, each subsequent target $\mathbf{u}_t$ was set to $\mathbf{u}_{t-1} + \delta_t$ where $\delta_t \sim \mathcal{N}(\mathbf{0}, \sigma I)$ for $\sigma = 0.1$. In the second set of experiments, which we refer to as the *linear drift* experiments, each $\mathbf{u}_t$ was set to $\mathbf{u}_{t-1} + \delta$ for a fixed random vector δ. Each set of experiments was repeated 1000 times.

Both the Shift Perceptron and the Randomized Budget Perceptron are tuned using a single parameter (denoted by λ and B respectively). While we originally planned to tune these parameters using additional random draws of the data, the best values of these parameters simply reduced each algorithm to the original Perceptron. Instead, we set $\lambda = 0.01$ or $\lambda = 0.0001$, and $B = 300$, as these values resulted in fairly typical behavior for each of the algorithms.

The results are summarized in Figure 1. The two left plots show the results for the random drift experiments with $d = 5$. The two right plots show the results for the linear drift experiments with $d = 50$. The two top plots show the cumulative number of mistakes made by each of the four algorithms averaged over 1000 runs, while the bottom two plots show *difference* between the cumulative number of mistakes made by the Perceptron and the cumulative number of mistakes made by each algorithm. (Values above 0 indicate that an algorithm made more mistakes than the Perceptron, while values below 0 indicate than an algorithm made fewer.) The error bars correspond to the 95% confidence interval over the 1000 runs.

Consider the top-left plot summarizing the results of the random drift experiments. We see that all algorithms made between 250 and 300 mistakes, but the Modified Perceptron (green circles) made (statistically significantly) fewer mistakes than the others. This difference is easier to see in

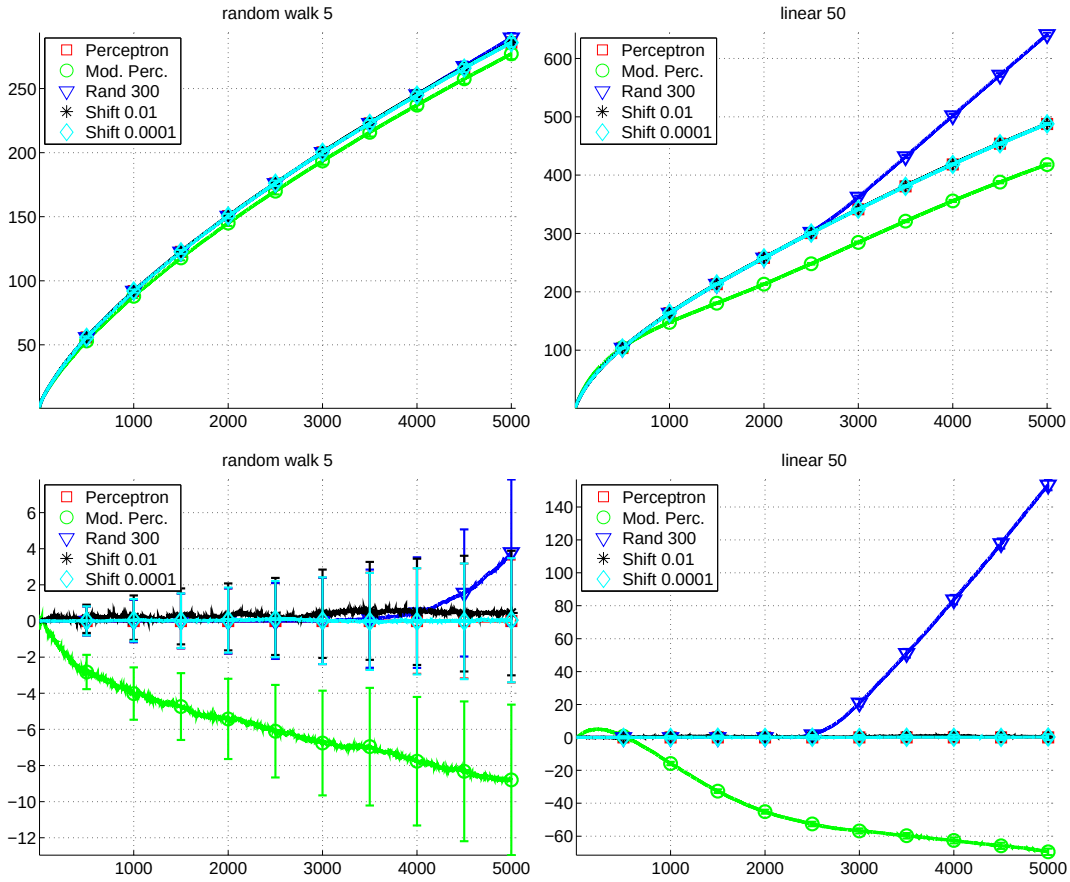

Figure 1: Cumulative number of mistakes (top) and difference between the cumulative number of mistakes and the cumulative number of mistakes made by the Perceptron algorithm (bottom) averaged over 1000 runs for the random drift experiments (left) and linear drift experiments (right). Four algorithms are evaluated: The standard Perceptron (red squares, largely hidden behind the Shift Perceptron), the Modified Perceptron (analyzed above, green circles), the Random Budget Perceptron with $B = 300$ (blue triangles), and the Shift Perceptron with $\lambda = 0.1$ (black stars) and $\lambda = 0.0001$ (teal diamonds). The bars indicate 95% confidence intervals.

the bottom-left plot. The Shift Perceptron made about 2% more mistakes than the Perceptron throughout the entire run. The Randomized Budget Perceptron was identical to the Perceptron algorithm until its budget of examples is exceeded, but overall made 1.2% more mistakes. Finally, during a prefix of training the Modified Perceptron made more mistakes than the Perceptron, but after about 500 training examples, the Modified Perceptron outperformed the Perceptron, making about 4% fewer mistakes.

The two right plots show qualitatively similar results for the linear drift experiments. During the first 500 examples the Modified Perceptron made more mistakes then the Perceptron algorithm, but it eventually performs better, ending with 15% fewer mistakes. As before, the performance of the Randomized Budget Perceptron started to degrade after its number of mistakes exceeded the budget. Finally, the Shift Perceptron made 4% more mistakes than the Perceptron after 1000 examples.

The total number of mistakes made by each of the four algorithms for various values of d are shown in the top two panels of Figure 2. As before, the bottom panels show the performance relative to the Perceptron, with values above 0 corresponding to more mistakes than the Perceptron. The two left plots show the results for the random drift experiments and the two right plots for the linear drift.

Comparing the top plots we observe that the random drift setting is slightly harder than the linear drift setting for lower values of d. For example, for $d = 20$ most algorithms made about

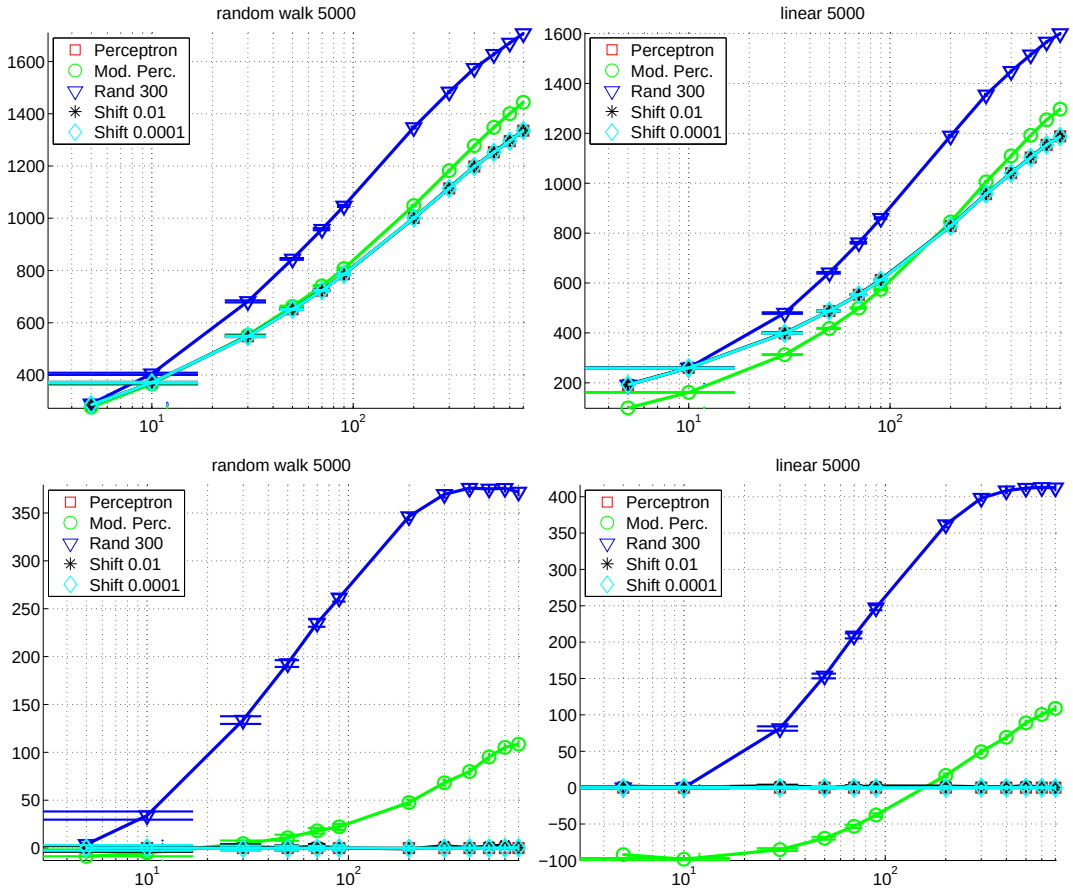

Figure 2: Total number of mistakes (top) and difference between the total number of mistakes and the number of mistakes made by the Perceptron algorithm (bottom) for different values of the underlying dimension d of the data for the random drift experiments (left) and the linear drift experiments (right). Again, the bars indicate 95% confidence intervals. (The elongation of these bars toward the edge of the plot is only an artifact of the log-scale axis.)

550 errors in the random drift setting but only 400 mistakes in the linear drift setting. For high values of d this gap is reduced. For small values of d, the Modified Perceptron outperformed the other algorithms, especially in the linear drift setting. For example, it made 100 fewer mistakes than the Perceptron algorithm with $d = 5$. When the underlying dimension d was large it made more mistakes compared to the other algorithms. The break-even point is about $d = 15$ for random drift and $d = 150$ for linear drift. The reason for this phenomenon is not clear to us. Note, however, that the underlying dimension d plays a major role in determining the difficulty of each problem, while the actual dimension D matters less. (This is not apparent from the experiments presented here, but we found it to be true when experimenting with different values of D.) This observation is in line with the dimension-independent bounds commonly published in the literature.

References

[1] P. L. Bartlett. Learning with a slowly changing distribution. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, 1992.

[2] P. L. Bartlett, S. Ben-David, and S. Kulkarni. Learning changing concepts by exploiting the structure of change. *Machine Learning*, 41:153–174, 2000.

[3] R. D. Barve and P. M. Long. On the complexity of learning from drifting distributions. *Information and Computation*, 138(2):101–123, 1997.

[4] A. Blum, A. Frieze, R. Kannan, and S. Vempala. A polynomial-time algorithm for learning noisy linear threshold functions. *Algorithmica*, 22:35–52, 1998.

[5] G. Cavallanti, N. Cesa-Bianchi, and C. Gentile. Tracking the best hyperplane with a simple budget perceptron. *Machine Learning*, 69(2/3):143–167, 2007.

[6] N. Cesa-Bianchi and G. Lugosi. *Prediction, learning, and games*. Cambridge University Press, 2006.

[7] S. Dasgupta, A. Tauman Kalai, and C. Monteleoni. Analysis of perceptron-based active learning. In *Proceedings of the 18th Annual Conference on Learning Theory*, 2005.

[8] Y. Freund and Y. Mansour. Learning under persistent drift. In *Proceedings of EuroColt*, pages 109–118, 1997.

[9] E. Hazan and C. Seshadhri. Efficient learning algorithms for changing environments. In *Proceedings of the 26th International Conference on Machine Learning*, 2009.

[10] D. P. Helmbold and P. M. Long. Tracking drifting concepts by minimizing disagreements. *Machine Learning*, 14(1):27–46, 1994.

[11] M. Herbster and M. Warmuth. Tracking the best expert. *Machine Learning*, 32(2):151–78, 1998.

[12] M. Herbster and M. Warmuth. Tracking the best linear predictor. *Journal of Machine Learning Research*, 1:281–309, 2001.

[13] A. Kuh, T. Petsche, and R. L. Rivest. Learning time-varying concepts. In *Advances in Neural Information Processing Systems 3*, 1991.

[14] A. Kuh, T. Petsche, and R. L. Rivest. Incrementally learning time-varying half-planes. In *Advances in Neural Information Processing Systems 4*, pages 920–927, 1992.

[15] P. M. Long. The complexity of learning according to two models of a drifting environment. *Machine Learning*, 37:337–354, 1999.

[16] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–407, 1958. (Reprinted in *Neurocomputing* (MIT Press, 1988).).

A Additional Proofs

A.1 Proof of Theorem 6

We describe a strategy that the adversary can employ in order to force any learning algorithm to make a mistake with probability at least $(1-1/e)/2$ every $2/\sqrt{\lambda}$ time steps, resulting in an average error of $\Omega(\sqrt{\lambda})$.

The strategy for the adversary is simple. At time $t=0$, the adversary sets f_1 such that $\tau_t = 1/2$. The adversary then chooses a random bit b which is 1 with probability $1/2$ and 0 with probability $1/2$. If $b=1$, then the adversary gradually shifts the threshold to the right, increasing it by γ each time step until it reaches $1/2+\gamma/\sqrt{\lambda}$, and then shifts it back again. On the other hand, if $b=0$, then the adversary fixes $\tau_t = 1/2$ for $t=1$ to $2/\sqrt{\lambda}$. In either case, at each of time step, the adversary sets D_t to be any λ-good distribution for which weight $\sqrt{\lambda}$ is spread uniformly over the region $[1/2, 1/2+\gamma/\sqrt{\lambda}]$. After time $2/\sqrt{\lambda}$, the process repeats with a new random bit b.

Let us consider the probability that the learning algorithm makes at least one mistake during the first $2/\sqrt{\lambda}$ time steps. Because the learning algorithm does not know the random bit b, the algorithm cannot know whether the target is shifting or fixed, even if it is aware of the adversary's strategy. Therefore, the first time that the algorithm sees a point x_t in $(1/2, 1/2+t\gamma)$ for $t \in \{1, \cdots, 1/\sqrt{\lambda}\}$ or a point x_t in $(1/2, 1/2+(2/\sqrt{\lambda}-t)\gamma)$ for $t \in \{1/\sqrt{\lambda}+1, \cdots, 2/\sqrt{\lambda}\}$, it makes a mistake with probability $1/2$. The algorithm will see at least one such point with probability $1-\prod_{t=1}^{1/\sqrt{\lambda}}(1-t\lambda)^2 = 1-\mathrm{e}^{\sum_{t=1}^{1/\sqrt{\lambda}} 2\ln(1-t\lambda)} \geq 1-\mathrm{e}^{\sum_{t=1}^{1/\sqrt{\lambda}} -2t\lambda} \geq 1-\mathrm{e}^{-1}$, where the first inequality follows from the fact that

$\ln(x) \leq x-1$ and the second from the fact that $\sum_{t=1}^{1/\sqrt{\lambda}} t \geq 1/(2\lambda)$. This implies that the probability that the algorithm makes at least one mistake during one of the first $2/\sqrt{\lambda}$ time steps is $(1-1/e)/2$.

The same analysis can be repeated to show that this is true of each consecutive interval of $2/\sqrt{\lambda}$ steps. Thus the error rate of any algorithm is at least $(1-1/e)\sqrt{\lambda}/4$.

A.2 Proof of Lemma 8

Let U be the uniform distribution over d-dimensional unit vectors. For any d-dimensional vector x such that $||x|| = 1$, $\Pr_{z\sim U}(|z \cdot x| > 1/(2\sqrt{d})) \geq 1/2$ (see Dasgupta et al. [7]). Let I be an indicator function that is 1 if its input is true and 0 otherwise. For any distribution Q over d-dimensional unit vectors,

$$\begin{aligned}\sup_{z:||z||=1} \Pr_{x\sim Q}(|z \cdot x| > 1/(2\sqrt{d})) &\geq \mathrm{E}_{z\sim U}\left[\mathrm{E}_{x\sim Q}\left[I(|z \cdot x| > 1/(2\sqrt{d}))\right]\right] \\ &= \mathrm{E}_{x\sim Q}\left[\Pr_{z\sim U}(|z \cdot x| > 1/(2\sqrt{d}))\right] \geq 1/2.\end{aligned}$$

This implies that for any distribution Q there exists a vector z such that $\Pr_{x\sim Q}(|z \cdot x| > 1/(2\sqrt{d})) \geq 1/2$. For the remainder of the proof we let Q be the distribution D conditioned on $|w \cdot x| \leq \eta/\sqrt{d}$, and define z to be any vector satisfying the property above for Q.

Let $w^+ = w + 2\eta z$ and $w^- = w - 2\eta z$. Let X be the set of all unit vectors x such that $|z \cdot x| > 1/(2\sqrt{d})$ and $|w \cdot x| \leq \eta/\sqrt{d}$. It is easy to verify that for all $x \in X$, $\mathrm{sign}(w^+ \cdot x) \neq \mathrm{sign}(w^- \cdot x)$. Furthermore, we can construct a sequence of unit vectors $w_0, \ldots, w_k$ such that $w_0 = w^+/\|w^+\|$ and $w_k = w^-/\|w^-\|$, $\arccos(w_i \cdot w_{i+1}) < \gamma$, and $k \leq 5\eta/\gamma + 1$. To see how, let θ be the angle between w_0 and w_k. Then $\cos(\theta) = (w^+ \cdot w^-)/(\|w^+\|\,\|w^-\|) \geq (1-4\eta^2)/(1+4\eta^2) > 1-8\eta^2$, where the first inequality follows from the fact that $\|w^+\|\|w^-\| = \sqrt{(1+4\eta^2+4\eta(w \cdot z))(1+4\eta^2-4\eta(w \cdot z))} = \sqrt{(1+4\eta^2)^2 - 16\eta^2(w \cdot z)^2} \leq 1+4\eta^2$.

Since $\eta < 1/2$ we have that $\theta \in [0, \pi/2]$. It can be shown that for any $\theta \in [0, \pi/2]$, $(4/\pi^2)\theta^2 \leq 1-\cos(\theta)$. This implies that $\theta < \sqrt{2}\pi\eta < 5\eta$. Since the angle between each pair w_i and w_{i+1} can be γ, we can create a sequence of vectors satisfying the property above with $k \leq \lceil 5\eta/\gamma \rceil \leq 5\eta/\gamma + 1$.

We have established that for every $x \in X$, $\mathrm{sign}(w^+ \cdot x) \neq \mathrm{sign}(w^- \cdot x)$. This implies that for every x there is an i such that $\mathrm{sign}(w_i \cdot x) \neq \mathrm{sign}(w_{i+1} \cdot x)$. Thus to bound the weight of X under D, it suffices to bound the weight of the regions $\Delta_i = \{x : \mathrm{sign}(w_i \cdot x) \neq \mathrm{sign}(w_{i+1} \cdot x)\}$. Since, by construction, the angle between each adjacent pair of vectors is at most γ and D is λ-good, D places weight no more than λ on each set Δ_i, and no more than $k\lambda \leq 5\eta\lambda/\gamma + \lambda$ on the set X.

Finally, since we have shown that $\Pr_{x\sim Q}(|z \cdot x| > 1/(2\sqrt{d})) \geq 1/2$, it follows that $\Pr_{x\sim D}(|w \cdot x| \leq \eta/\sqrt{d}) \leq 2D(X) \leq 10\eta\lambda/\gamma + 2\lambda$. ■