The Perceptron algorithm vs. Winnow: linear vs. logarithmic mistake bounds when few input variables are relevant

Jyrki Kivinen*
Manfred K. Warmuth†

UCSC-CRL-95-44
October 6, 1995

Baskin Center for
Computer Engineering & Information Sciences
University of California, Santa Cruz
Santa Cruz, CA 95064 USA

ABSTRACT

We give an adversary strategy that forces the Perceptron algorithm to make $(N - k + 1)/2$ mistakes when learning k-literal disjunctions over N variables. Experimentally we see that even for simple random data, the number of mistakes made by the Perceptron algorithm grows almost linearly with N, even if the number k of relevant variable remains a small constant. In contrast, Littlestone's algorithm Winnow makes at most $O(k \log N)$ mistakes for the same problem. Both algorithms use thresholded linear functions as their hypotheses. However, Winnow does multiplicative updates to its weight vector instead of the additive updates of the Perceptron algorithm.

---

*Supported by the Academy of Finland. This work was done while the author was visiting University of California, Santa Cruz.
Address: Department of Computer Science, P.O. Box 26 (Teollisuuskatu 23), FIN-00014 University of Helsinki, Finland, jkivinen@cs.helsinki.fi.

†Supported by NSF grant IRI-9123692.
Address: Computer and Information Sciences, University of California, Santa Cruz, Santa Cruz, CA 95064, USA, manfred@cse.ucsc.edu.

1 Introduction

This paper addresses the familiar problem of predicting with a *linear classifier*. The *instances*, for which one tries to predict a binary classification, are N-dimensional real vectors. A linear classifier is represented by a pair $(\mathbf{w}, \theta)$, where $\mathbf{w} \in \mathbf{R}^N$ is an N-dimensional *weight vector* and $\theta \in \mathbf{R}$ is a *threshold*. The linear classifier represented by the pair $(\mathbf{w}, \theta)$ has the value 1 on an instance $\mathbf{x}$ if $\mathbf{w} \cdot \mathbf{x} \geq \theta$, and the value 0 otherwise. Each instance $\mathbf{x} \in \mathbf{R}^N$ can be thought of as a value assingment for N input variables: x_i is the value for the ith input variable.

Monotone disjunctions are a special case of linear classifiers. The monotone k-literal disjunction $X_{i_1} \vee \ldots \vee X_{i_k}$ corresponds to the linear classifier represented by the pair $(\mathbf{w}, 1/2)$ where $w_{i_1} = \cdots = w_{i_k} = 1$ and $w_j = 0$ for $j \notin \{ i_1, \ldots, i_k \}$. For a given disjunction, the variables in the disjunction are called *relevant* and the remaining variables *irrelevant*. In this paper we study the performance of two known learning algorithms for linear classifiers when they are applied to learning monotone disjunctions in which the number k of relevant variables is much smaller than the total number N of variables.

We analyze the algorithms in the following simple on-line prediction model of learning. The learning proceeds in trials. In trial t, the learning algorithm is given an instance $\mathbf{x}_t$ and produces its prediction $\hat{y}_t$ using its current hypothesis, which is a linear classifier given by the algorithm's current weight vector $\mathbf{w}_t$ and threshold θ_t. The algorithm then receives a binary outcome y_t and may update its weight vector and threshold to $\mathbf{w}_{t+1}$ and θ_{t+1}. If the outcome differs from the prediction, we say that the algorithm made a mistake. Following Littlestone [Lit89, Lit88], our goal is to minimize the total number of mistakes that the learning algorithm makes for certain sequences of trials.

The standard on-line algorithm for learning with linear classifiers is the simple Perceptron algorithm of Rosenblatt [Ros58]. An alternate algorithm called Winnow was introduced by Littlestone [Lit89, Lit88]. To see how the algorithms work, consider a binary vector $\mathbf{x}_t \in \{ 0, 1 \}^N$ as an instance, and assume that the algorithm predicted 0 while the outcome was 1. Then both algorithms increment those weights $w_{t,i}$ for which the corresponding input $x_{t,i}$ was 1, but do not change the weights $w_{t,i}$ with $x_{t,i} = 0$. Thus, $\mathbf{w}_{t+1} \cdot \mathbf{x}_t > \mathbf{w}_t \cdot \mathbf{x}_t$, and the dot product increases as it should. The difference between the algorithms is in how they increment the weights in question. The Perceptron algorithm *adds* a positive constant to each of them, whereas Winnow *multiplies* each of them by a constant that is larger than one. Similarly, if the prediction was 1 and the outcome 0, the weights $w_{t,i}$ with $x_{t,i} = 1$ are decremented either by subtracting a positive constant or dividing by a constant larger than one. The choice of the constants for the updates, as well as the initial weights and thresholds, can significantly affect the performance of the algorithms. We call choosing these parameters *tuning*.

If there is a linear classifier $(\mathbf{u}, \psi)$ such that for all t we have $y_t = 1$ if and only if $\mathbf{u} \cdot \mathbf{x}_t \geq \psi$, we say that the trial sequence is *consistent* with the classifier $(\mathbf{u}, \psi)$ and say that the classifier $(\mathbf{u}, \psi)$ is a *target* of the trial sequence. It is easy to tune Winnow so that it makes at most $O(k \log N)$ mistakes [Lit88, Lit91] on any sequence with a k-literal disjunction as a target. If the tuning is allowed to depend on k, the tighter bound $O(k + k \log(N/k))$ is obtainable. This upper bound is optimal to within a constant factor since the Vapnik-Chervonenkis (VC) dimension [VC71, BEHW89] of the class of k-literal disjunctions is $\Omega(k + k \log(N/k))$ [Lit88] and this dimension is always a lower bound for the optimal mistake bound. The best upper bound we know for learning k-literal monotone disjunctions with the Perceptron

algorithm is $O(kN)$ mistakes. This bound comes from the Perceptron Convergence Theorem [DH73], and we suspect it is not very tight for our case, particularly when k is large.

The main result of this paper is to give a simple adversary strategy that forces the Perceptron algorithm to make $N - k + 1$ mistakes, assuming that the initial weight vector $\mathbf{w}_1$ of the algorithm is zero. If the Perceptron algorithm is allowed to choose an arbitrary initial weight vector, we can still prove a lower bound of $(N - k + 1)/2$. The lower bound of $(N - k + 1)/2$ actually holds for all algorithms with the property that the weight vector of the algorithm is obtained by adding to the initial weight vector a linear combination of the instances seen thus far, i.e.,

$$\mathbf{w}_t = \mathbf{w}_1 + \sum_{j=1}^{t-1} \alpha_{t,j}\mathbf{x}_j \tag{1.1}$$

for some scalars $\alpha_{t,j} \in \mathbf{R}$. We call such algorithms *additive*. The class of additive algorithms includes the classical Perceptron algorithm as well as a more complicated algorithm that is based on the ellipsoid method for linear programming [MT94]. Additive algorithms include all algorithms that do not change their predictions if the instances are rotated, i.e., if each instance $\mathbf{x}_t$ is replaced by $A\mathbf{x}_t$ where $A \in \mathbf{R}^{N\times N}$ is orthonormal. In contrast, the weight vectors of Winnow satisfy

$$w_{t,i} = w_{1,i} \prod_{j=1}^{t-1} \beta_{t,j}^{x_{j,i}} \tag{1.2}$$

for some positive scalars $\beta_{t,j}$, so Winnow could be called a *multiplicative* algorithm. The Perceptron algorithm has the special property that for it the scalars $\alpha_{t,j}$ in (1.1) satisfy $\alpha_{t,j} = \alpha_{j+1,j}$ for all $t > j$. Similarly, the scalars $\beta_{t,j}$ for Winnow in (1.2) satisfy $\beta_{t,j} = \beta_{j+1,j}$ for all $t > j$. Hence, for these two algorithms the new weight vector $\mathbf{w}_{t+1}$ can be obtained from $\mathbf{w}_t$ and $\mathbf{x}_t$ without knowledge of the past instances $\mathbf{x}_j$ where $j < t$.

When k is small, the mistake bound of the Perceptron algorithm is *exponential* in the optimal mistake bound (in this case essentially the VC dimension). This may be seen as an instance of what is called the *curse of dimensionality* in the literature of neural networks and statistics [DH73]. It may seem surprising that the Perceptron algorithm performs so badly for monotone disjunctions even though these concepts are simple linear classifiers with small weights and threshold. The difference in the performances of the algorithms shows that the algorithms have a different bias in their search for a good hypothesis. Intuitively, Winnow favors weight vectors that are in some sense sparse, and wins if the target weight vector is sparse ($k \ll N$ in the disjunction case). If the target weight vector is dense ($k = \Omega(N)$ in the disjunction case), the advantage of Winnow in the worst-case bounds becomes much smaller. In experiments we have seen that the Perceptron algorithm actually makes fewer mistakes than Winnow when the target is dense and the learning rate and threshold is set as in the theorems that guarantee the $O(k + k\log(N/k))$ mistake bound. In these experiments we made the instances sparse, i.e., for each instance $\mathbf{x}_t$ most of the components $x_{t,i}$ were set to zero. This assures that roughly half of the instances are still positive. We believe that the sparseness of the instances is advantageous for the Perceptron algorithm. Note that if it is known that k is close to N, Winnow can be tuned so that it simulates the classical elimination algorithm for learning disjunctions [Val84], in which case it makes at most $N-k$ mistakes for k literal monotone disjunctions but is not robust against noise.

The trade-off in which Winnow is able to take advantage of sparse targets and dense instances and the Perceptron algorithm is able to take advantage of sparse instances and

dense targets is similar to the situation in on-line linear regression [KW94]. In the regression problem, the classical Gradient Descent algorithm makes Perceptron-style additive updates, and a new family of Exponentiated Gradient algorithms makes multiplicative Winnow-style updates. Again, the Exponentiated Gradient algorithms win for sparse targets and dense instances, while the gradient descent algorithm wins for dense targets and sparse instances. In the regression problem the incomparability of the algorithm is brought out more clearly than in the classification problem. The proven worst-case loss bounds are incomparable as well as the experimental performance on simple artificial data.

Our lower bounds for the Perceptron algorithm are proven by an adversary argument in which the instances are obtained by a simple transformation from the rows of a Hadamard matrix. Since the rows of a Hadamard matrix are orthogonal, and the Perceptron algorithm updates its weight vector $\mathbf{w}_t$ by adding to it some multiple of the current instance $\mathbf{x}_t$, the result is that the predictions of the Perceptron algorithm are independent of the preceding trials. Similar proofs have been devised for linear regression [LLW91]. The proofs presented here for the case of linear classification are slightly more involved.

We also wish to point out that the behavior similar to that predicted by the worst-case lower bounds already takes place on random data. In this case, too, the number of mistakes made by the Perceptron algorithms gets close to N, while it is significantly lower for Winnow if the number k of relevant variables is small. Possibly this is explained by fact that for large N, the dot product between two random N-dimensional binary vectors is likely to be close to the fixed value $N/4$.

One might argue that random inputs are not natural, either. However, even a few genuine random inputs may lead to a large number of pseudo-random variables when inputs are expanded to form new variables. In this case the number of mistakes of the Perceptron algorithm may grow almost linearly in the number of pseudo-random variables.

The Perceptron algorithm's susceptibility to irrelevant input variables is also present in the batch setting. If the Perceptron algorithm is trained on the $N/2$ instances derived from the first half of the rows of an $N \times N$ Hadamard matrix, it will still be wrong roughly half of the time on the instances derived from the second half of the Hadamard matrix.

Winnow opens up a new venue of algorithm design. Since additional irrelevant input variables do not degrade the algorithm's prediction performance too badly, one can extend the algorithm's capabilities to nonlinear prediction by introducing as additional inputs the values for a large number of nonlinear basis functions. Ignoring for the moment computational efficiency, we could as an extreme case consider learning DNF using Winnow. Introducing one new input variable for each of the possible 3^N conjunctions in the DNF formula, one gets a worst-case bound of $O(kN)$ mistakes for k-term DNF formulas. Note that this mistake bound is very good in light of the fact that the logarithm of the number of such formulas is $O(kN)$. However, the computational cost of maintaining the 3^N weights soon becomes prohibitive. Instead of introducing all the new variables at once, one might as a heuristic first introduce only a few. Later one could use the weights of the tested variables as a guide for choosing variables for the next iteration.

We did some experiments with the above method for learning DNF and noticed that for random instances with respect to the uniform distribution the number of mistakes made by Winnow did not even get close to its theoretical bound. For this simple artificial data the Perceptron algorithm actually outperformed Winnow that was tuned according to the upper bound theorems of Littlestone [Lit88].

We introduce the details of the on-line prediction model and the algorithms we consider in Section 2. Section 3 gives our adversarial lower bound constructions for the class of additive algorithms. Our experimental results are presented in Section 4.

2 The prediction model and algorithms

2.1 The basic setting

We use a pair $(\mathbf{u}, \psi)$ to represent a *linear classifier* with the *weight vector* $\mathbf{u} \in \mathbf{R}^N$ and the *threshold* ψ. The classifier represented by $(\mathbf{u}, \psi)$ is denoted by $\Phi_{\mathbf{u},\psi}$ and defined for $\mathbf{x} \in \mathbf{R}^N$ by $\Phi_{\mathbf{u},\psi}(\mathbf{x}) = 1$ if $\mathbf{u} \cdot \mathbf{x} \geq \psi$ and $\Phi_{\mathbf{u},\psi}(\mathbf{x}) = 0$ otherwise. We are mostly concerned with the special case when $\mathbf{x} \in \{0,1\}^N$.

An *N-dimensional trial sequence* is a game played between two players, the learner and the teacher. For the purposes of the present paper, we restrict ourselves to learners that predict using linear classifiers, in a manner we shall soon describe in more detail. The game has ℓ rounds, or *trials*, for some positive integer ℓ. In a trial sequence, trial t for $t = 1, \ldots, \ell$ proceeds as follows:

1. The learner chooses its *hypothesis* $(\mathbf{w}_t, \theta_t)$, with $\mathbf{w}_t \in \mathbf{R}^N$ and $\theta_t \in \mathbf{R}$.
2. The teacher presents the *instance* $\mathbf{x}_t \in \{0,1\}^N$.
3. The learner's *prediction* is now defined to be $\hat{y}_t = \Phi_{\mathbf{w}_t,\theta_t}(\mathbf{x}_t)$.
4. The teacher presents the *outcome* $y_t \in \{0,1\}$.

After the last trial, the teacher must present a *target* $(\mathbf{u}, \psi)$, with $\mathbf{u} \in \mathbf{R}^N$ and $\psi \in \mathbf{R}$, such that $\Phi_{\mathbf{u},\psi}(\mathbf{x}_t) = y_t$ for all t. The goal of the learner is to minimize the number of *mistakes*, i.e., trials with $y_t \neq \hat{y}_t$. The teacher, on the other hand, tries to force the learner to make many mistakes.

This seemingly very strong worst-case model of prediction, with an adversarial teacher, can be justified by the fact that there are algorithms that can be guaranteed to make a reasonable number of mistakes as learners in this model. We soon introduce two such algorithms, the Perceptron algorithm and Winnow, and their mistake bounds. The model could be made even more adversarial by allowing the teacher a given number of *classification errors*, i.e., trials with $\Phi_{\mathbf{u},\psi}(\mathbf{x}_t) \neq y_t$. On the other hand, we often restrict the teacher by restricting the target. In this paper we consider the case where the target is required to be a monotone k-literal disjunction, i.e., to have $\psi = 1/2$ and $\mathbf{u} \in \{0,1\}^N$ with exactly k components u_i with value 1.

An *on-line linear prediction algorithm* is a deterministic algorithm that can act as the learner in the game described above. A general on-line prediction algorithm would be allowed to choose as its hypothesis any mapping from $\{0,1\}^N$ to $\{0,1\}$ instead of a linear classifier. For the restriction to linear classifiers to be effective, it is essential that the learner is required to fix its hypothesis before the instance $\mathbf{x}_t$ is given. Otherwise, the learner could emulate an arbitrary on-line algorithm by choosing its hypothesis to be either the constant threshold function $(\mathbf{0}, -1)$ or $(\mathbf{0}, 1)$ depending on what the prediction of that algorithm would be on the instance $\mathbf{x}_t$.

We use the term *trial sequence* for the sequence $S = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_\ell, y_\ell))$ that gives the teacher's part of the game. Given a fixed deterministic learning algorithm, the learner's part is completely determined by the trial sequence.

2.2 The algorithms

Both for the Perceptron algorithm and Winnow, the new hypothesis $(\mathbf{w}_{t+1}, \theta_{t+1})$ depends only on the old hypothesis $(\mathbf{w}_t, \theta_t)$ and the observed instance $\mathbf{x}_t$ and outcome y_t. We call this dependence the *update rule* of the algorithm. In addition to the update rule, we must also give the *initial hypothesis* $(\mathbf{w}_1, \theta_1)$ to characterize an algorithm. The most usual initial weight vectors $\mathbf{w}_1$ are the zero vector $\mathbf{0} = (0, \ldots, 0)$ and the vector $\mathbf{1} = (1, \ldots, 1)$. Note that the definition of a linear on-line prediction algorithm allows the new hypothesis $(\mathbf{w}_{t+1}, \theta_{t+1})$ to depend on earlier instances $\mathbf{x}_i$ and outcomes y_i, $i < t$, and there are indeed some more sophisticated algorithms with such dependencies.

The Perceptron algorithm and Winnow are actually families of algorithms, both parameterized by the initial hypothesis and a *learning rate* $\eta > 0$. To give the update rules of the algorithms, let us first denote by σ_t the sign of the prediction error at trial t, that is, $\sigma_t = \hat{y}_t - y_t$. In their basic forms, both the Perceptron algorithm and Winnow maintain a fixed threshold, i.e., $\theta_t = \theta_1$ for all t. Given an instance $\mathbf{x}_t \in \{0,1\}^N$, the sign σ_t, and a learning rate η, the update of the Perceptron algorithm can be written componentwise as

$$w_{t+1,i} = w_{t,i} - \eta \sigma_t x_{t,i} \tag{2.1}$$

and the update of Winnow as

$$w_{t+1,i} = w_{t,i} e^{-\eta \sigma_t x_{t,i}} \quad . \tag{2.2}$$

Note that this basic version of Winnow (the algorithm Winnow2 of [Lit88]) only uses positive weights (assuming that the initial weights are positive). The algorithm can be generalized for negative weights by a simple reduction [Lit88]. See Littlestone [Lit89] for a discussion on the learning rates and other parameters of Winnow. Here we just point out the standard method of allowing the threshold to be fixed to 0 at the cost of increasing the dimensionality of the problem by one. To do this, each instance $\mathbf{x} = (x_1, \ldots, x_N)$ is replaced by $\mathbf{x}' = (x_1, \ldots, x_N, 1)$. Then a linear classifier $(\mathbf{w}, \theta)$ with a nonzero threshold can be replaced by $(\mathbf{w}', 0)$ where $\mathbf{w}' = (w_1, \ldots, w_N, -\theta)$. This useful technique gives a method for effectively updating the threshold together with the components of the weight vector.

It is known that if the target is a monotone k-literal disjunction, Winnow makes $O(k \log N)$ mistakes [Lit88]. There are several other algorithms that make multiplicative weight updates and achieve similar mistake bounds [Lit89]. The best upper bound we know for the Perceptron algorithm comes from the Perceptron Convergence Theorem given, e.g, by Duda and Hart [DH73, pp. 142–145]. Assuming that the target is a monotone k-literal disjunctions and the instances $\mathbf{x}_t \in \{0,1\}^N$ satisfy $\sum_i x_{t,i} \leq X$ for some value X, the bound is $O(kX)$ mistakes. Note that always $X \leq N$.

As Maass and Turán [MT94] have pointed out, several linear programming methods can be transformed into efficient linear on-line prediction algorithms. Most notably, this applies to Khachiyan's ellipsoid algorithm [Kha79] and to a newer algorithm due to Vaidya [Vai89]. Vaidya's algorithm achieves an upper bound of $O(N^2 \log N)$ mistakes for an arbitrary linear classifier as the target when the instances are from $\{0,1\}^N$. The Perceptron algorithm and Winnow are not suitable for learning arbitrary linear classifiers over the domain $\{0,1\}^N$. Maass and Turán show that in the worst case the number of mistakes of both algorithms is exponential in N. The proof of the $O(N^2 \log N)$ mistake bound for general linear classifiers is based on first observing that arbitrary real weights in a linear classifier can be replaced with integer weights no larger than $O(N^{O(N)})$ without changing the classification of any

point in $\{0,1\}^N$. For monotone disjunctions, all the weights u_i and the threshold ψ can directly be chosen from $\{0,1,2\}$, which leads to the better bound of $O(N \log N)$ mistakes.

In what follows we assume that the arithmetic operations of the various algorithms can be performed exactly, without rounding errors.

2.3 Special classes of algorithms

The main theoretical results of this paper are lower bounds for the class of *additive* algorithms.

Definition 1: *A linear on-line prediction algorithm is* additive *if for all t, the algorithm's tth weight vector $\mathbf{w}_t$ can be written as*

$$\mathbf{w}_t = \mathbf{w}_1 + \sum_{j=1}^{t-1} \alpha_{t,j} \mathbf{x}_j \tag{2.3}$$

for some fixed initial weight vector $\mathbf{w}_1$ and for some coefficients $\alpha_{t,j} \in \mathbf{R}$.

As we are considering on-line prediction algorithms, the coefficients $\alpha_{t,j}$ in (2.3) of course depend only on the instances $x_{t,i}$ and outcomes y_i for $i < t$. However, the lower bounds we shall prove would be valid even if the algorithm were allowed to set the coefficients $\alpha_{t,j}$ to their best possible values using the future instances and even the future outcomes.

The Perceptron algorithm is additive. By comparing (2.1) and (2.3) we see that we can take $\alpha_{t,j} = -\eta\sigma_j$ for the Perceptron algorithm.

Consider now Winnow with initial weights $\mathbf{w}_1 = \mathbf{1}$, learning rate $\eta = \ln 2$, and threshold $\theta_1 = N = 3$. Let $\mathbf{x}_1 = (1,1,0)$, $\mathbf{x}_2 = (1,0,1)$, and $y_1 = y_2 = 1$. This is consistent with the target $((1,0,0), 1/2)$, and gives $\mathbf{w}_3 = (4,2,2)$. As the vector $\mathbf{w}_3 - \mathbf{w}_1 = (3,1,1)$ is not in the span of $\{\mathbf{x}_1, \mathbf{x}_2\}$, we see that Winnow is not additive.

Recall that a square matrix $A \in \mathbf{R}^{m \times m}$ is *orthogonal* if its columns are orthogonal to each other, and *orthonormal* if it is orthogonal and its columns have Euclidean norm 1. Thus, for an orthogonal matrix A the product $A^T A$ is a diagonal matrix, and for an orthonormal matrix $A^T A = I$ where I is the $m \times m$ identity matrix.

Consider an orthonormal matrix $A \in \mathbf{R}^{m \times m}$. If we think of a vector $\mathbf{x} \in \mathbf{R}^m$ as a list of coordinates of some point in m-dimensional space, then $A\mathbf{x}$ can be considered the list of coordinates of the same point in a new coordinate system. The basis vectors of the new coordinate system are represented in the original coordinate system by the column vectors of A. Thus, orthonormal matrices represent rotations of the coordinate system. Let us write $\widetilde{\mathbf{x}} = A\mathbf{x}$. Rotations preserve angles: $\widetilde{\mathbf{w}} \cdot \widetilde{\mathbf{x}} = (A\mathbf{w})^T A\mathbf{x} = \mathbf{w}^T (A^T A)\mathbf{x} = \mathbf{w} \cdot \mathbf{x}_t$. In a situation in which this geometric interpretation is meaningful, it would be natural to assume that the choice of coordinate system is irrelevant, i.e., nothing changes if one systematically replaces $\mathbf{x}$ by $\widetilde{\mathbf{x}}$ everywhere.

Definition 2: *A linear on-line prediction algorithm is* rotation invariant *if for all orthonormal matrices $A \in \mathbf{R}^{N \times N}$ and all trial sequences $S = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_\ell, y_\ell))$, the predictions made by the algorithm given the trial sequence S are the same as its predictions given the trial sequence $\widetilde{S} = ((A\mathbf{x}_1, y_1), \ldots, (A\mathbf{x}_\ell, y_\ell))$.*

In general, being rotation invariant is not necessarily a natural property of an algorithm. For instance, the components $x_{t,i}$ of the instances often represent some physical quantities that for different i may have entirely different units. It is also common to scale the instances to make, for example, $-1 \le x_{t,i} \le 1$ hold for all t and i. In such cases, the original coordinate system clearly has a special meaning. However, there are several common algorithms that are rotation invariant.

The Perceptron algorithm is rotation invariant. The linear on-line prediction algorithm one obtains by applying the reduction given by Maass and Turán to the ellipsoid method for linear programming is also rotation invariant. This is because the initial ellipsoid used by the algorithm is a ball centered at the origin, and the updates of the ellipsoid are done in a rotation invariant manner. If one uses Vaidya's algorithm for the linear programming in the reduction, one gets an algorithm that is not rotation invariant. Vaidya's algorithm uses a polytope that is updated in a rotation invariant manner, but the initialization of the polytope cannot be rotation invariant.

Winnow is not rotation invariant, either. To see this, consider a two-dimensional trial sequence with $\mathbf{x}_1 = (1,0)$, $\mathbf{x}_2 = (0,1)$, and $y_1 = y_2 = 1$. Assume that Winnow uses the inital weight vector $\mathbf{w}_1 = \mathbf{1}$ and a threshold such that $\hat{y}_1 = \hat{y}_2 = 0$. Then after the two trials, Winnow has the weight vector $\mathbf{w}_3 = (e^\eta, e^\eta)$. Consider now the orthonormal matrix

$$A = 2^{-1/2} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} .$$

After seeing the counterexamples $(A\mathbf{x}_1, 1)$ and $(A\mathbf{x}_2, 1)$, Winnow has the hypothesis $\widetilde{\mathbf{w}}_3 = (e^{\eta\sqrt{2}}, 1)$. As $\mathbf{w}_3$ is linear in e^η and $\widetilde{\mathbf{w}}_3$ is not, it is clear that Winnow cannot be rotation invariant. To be specific, consider the instance $\mathbf{x}_3 = (r, -r)$ for some $r \in \mathbf{R}$. Then $\mathbf{w}_3 \cdot \mathbf{x}_3 = 0$, while $\widetilde{\mathbf{w}}_3 \cdot A\mathbf{x}_3 = r\sqrt{2}$. Therefore, for some values of r the predictions of Winnow are not the same for the rotated and the original instances.

We have the following general result.

Theorem 3: *If a linear on-line prediction algorithm is rotation invariant, then it is an additive algorithm with zero initial weight vector.*

Proof Let $(\mathbf{w}_{t+1}, \theta_{t+1})$ be the hypothesis of a rotation invariant algorithm after it has seen the instances $\mathbf{x}_1, \ldots, \mathbf{x}_t$ and outcomes $y_1, \ldots, y_t$. We claim that $\mathbf{w}_{t+1}$ is in the subspace spanned by the set $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_t\}$. It is easy to construct an orthonormal matrix $A \in \mathbf{R}^{N \times N}$ such that $A\mathbf{x}_i = \mathbf{x}_i$ for $i = 1, \ldots, t$, and $A\mathbf{x} = -\mathbf{x}$ for any vector $\mathbf{x}$ that is orthogonal to X. Since $A\mathbf{x}_t = \mathbf{x}_t$, the definition of a rotation invariant algorithm implies for all $\mathbf{x} \in \mathbf{R}^N$ that $\mathbf{w}_{t+1}^T \mathbf{x} \ge \theta_{t+1}$ if and only if $\mathbf{w}_{t+1}^T A\mathbf{x} \ge \theta_{t+1}$. Therefore, $\mathbf{w}_{t+1}^T \mathbf{x} = \mathbf{w}_{t+1}^T A\mathbf{x}$ for all $\mathbf{x}$. If we choose a vector $\mathbf{x}$ that is orthogonal to X, we have $\mathbf{w}_{t+1}^T \mathbf{x} = \mathbf{w}_{t+1}^T A\mathbf{x} = -\mathbf{w}_{t+1}^T \mathbf{x}$. Hence, $\mathbf{w}_{t+1}$ is in the subspace spanned by X. □

Conversely, consider an algorithm that is additive and has zero initial weight vector. If further the algorithm's thresholds θ_t and the coefficients $\alpha_{t,j}$ in (2.3) depend only on the outcomes and the dot products $\mathbf{x}_i \cdot \mathbf{x}_j$, then the algorithm is easily seen to be rotation invariant.

3 Lower bounds for additive algorithms

Given two vectors $\mathbf{p} \in \{-1,1\}^N$ and $\mathbf{q} \in \{-1,1\}^N$, we denote by $D(\mathbf{p},\mathbf{q})$ their Hamming distance, i.e., the number of indices i such that $p_i \neq q_i$.

In the proofs we use some basic properties of *Hadamard matrices*. A Hadamard matrix is an orthogonal matrix with its element in $\{-1,1\}$. Multiplying a row or a column of a Hadamard matrix by -1 leaves it a Hadamard matrix. Note that if $\mathbf{p}$ and $\mathbf{q}$ are two different rows in an $N \times N$ Hadamard matrix, we have $D(\mathbf{p},\mathbf{q}) = N/2$. The following definition gives the most straightforward way of obtaining high-dimensional Hadamard matrices.

Definition 4: *When $n = 2^d$ for some d, let H_n be the $n \times n$ Hadamard matrix obtained by the recursive construction $H_1 = (1)$,*

$$H_{2n} = \begin{pmatrix} H_n & H_n \\ H_n & -H_n \end{pmatrix} .$$

We also have the following result.

Lemma 5: *For $n = 2^d$ where d is a positive integer, let H_n be the $n \times n$ Hadamard matrix defined in Definition 4. Then for any vector $\mathbf{p} \in \{-1,1\}^n$ there is an index j such that $D(\mathbf{p},\mathbf{q}) \geq N/2$ holds if $\mathbf{q}$ is the jth column of H_n.*

Consider now an additive algorithm and its hypothesis given in (2.3). Its prediction on the instance $\mathbf{x}_t$ can depend only on the dot products $\mathbf{w}_1 \cdot \mathbf{x}_t$ and $\mathbf{x}_i \cdot \mathbf{x}_t$ where $i < t$. Thus, for an adversary it would be helpful to have for $\mathbf{x}_t$ two different candidates $\mathbf{z}'$ and $\mathbf{z}''$ for which these dot products do not differ. This motivates the following definition.

Definition 6: *Let $B = ((\mathbf{z}'_1,\mathbf{z}''_1),\ldots,(\mathbf{z}'_\ell,\mathbf{z}''_\ell))$, where $\mathbf{z}'_t$ and $\mathbf{z}''_t$ are in $\{0,1\}^N$ for all t. We say that B is a sequence with* pairwise constant dot products *if for $1 \leq i < t \leq \ell$ we have $\mathbf{z}'_i \cdot \mathbf{z}'_t = \mathbf{z}'_i \cdot \mathbf{z}''_t$ and $\mathbf{z}''_i \cdot \mathbf{z}'_t = \mathbf{z}''_i \cdot \mathbf{z}''_t$.*

Our basic idea is to form a sequence with pairwise constant dot products by choosing $\mathbf{z}'_t$ to be the tth row of an $\ell \times \ell$ Hadamard matrix, and $\mathbf{z}''_t = -\mathbf{z}'_t$, but a simple transformations is necessary to make the instances binary.

Merely having pairwise constant dot products is not sufficient for generating mistakes. The adversary needs a target $(\mathbf{u},\psi)$ that is suitably different from the algorithm's initial hypothesis. To get an idea about this, consider two instance candidates $\mathbf{z}'_t$ and $\mathbf{z}''_t$ with, say, $\mathbf{w}_t \cdot \mathbf{z}'_t \leq \mathbf{w}_t \cdot \mathbf{z}''_t$. Depending on the algorithm's threshold θ_t, the algorithm may either predict $\hat{y}_t = 0$ for both $\mathbf{x}_t = \mathbf{z}'_t$ and $\mathbf{x}_t = \mathbf{z}''_t$, predict $\hat{y}_t = 0$ for $\mathbf{x}_t = \mathbf{z}'_t$ and $\hat{y}_t = 1$ for $\mathbf{x}_t = \mathbf{z}''_t$, or predict $\hat{y}_t = 1$ for both $\mathbf{x}_t = \mathbf{z}'_t$ and $\mathbf{x}_t = \mathbf{z}''_t$. If the target $(\mathbf{u},\psi)$ now is such that $\mathbf{u} \cdot \mathbf{z}''_t < \psi < \mathbf{u} \cdot \mathbf{z}'_t$, then by choosing either $\mathbf{z}'_t$ or $\mathbf{z}''_t$ for the tth instance $\mathbf{x}_t$ the adversary can force the algorithm to make a mistake regardless of its choice of θ_t. Note that if the adversary is choosing its instances from a sequence with pairwise constant dot products and the algorithm is additive, the condition $\mathbf{w}_t \cdot \mathbf{z}'_t \leq \mathbf{w}_t \cdot \mathbf{z}''_t$ is equivalent with $\mathbf{w}_1 \cdot \mathbf{z}'_t \leq \mathbf{w}_1 \cdot \mathbf{z}''_t$ and hence independent of the updates made by the algorithm. This leads to the following definition.

Definition 7: *Let $B = ((\mathbf{z}'_1,\mathbf{z}''_1),\ldots,(\mathbf{z}'_\ell,\mathbf{z}''_\ell))$, where $\mathbf{z}'_t$ and $\mathbf{z}''_t$ are in $\{0,1\}^N$ for all t. Let $\mathbf{w} \in \mathbf{R}^N$ be a weight vector and $(\mathbf{u},\psi) \in \mathbf{R}^N \times \mathbf{R}$ a linear classifier. We say that the weight vector $\mathbf{w}$ and the classifier $(\mathbf{u},\psi)$* differ *at trial t on the sequence B if either $\mathbf{w}_1 \cdot \mathbf{z}'_t \leq \mathbf{w}_1 \cdot \mathbf{z}''_t$ and $\mathbf{u} \cdot \mathbf{z}'_t > \psi > \mathbf{u} \cdot \mathbf{z}''_t$, or $\mathbf{w}_1 \cdot \mathbf{z}'_t \geq \mathbf{w}_1 \cdot \mathbf{z}''_t$ and $\mathbf{u} \cdot \mathbf{z}'_t < \psi < \mathbf{u} \cdot \mathbf{z}''_t$.*

Using the basic idea given above, one can now prove the following result.

Theorem 8: *Let B be a sequence with pairwise constant dot products. Consider an additive linear on-line prediction algorithm with the initial weight vector $\mathbf{w}_1$. For any target $(\mathbf{u}, \psi)$, the adversary can choose the instances $\mathbf{x}_t$ in such a way that the algorithm makes a mistakes at all trials at which $\mathbf{w}_1$ and $(\mathbf{u}, \psi)$ differ on B.*

Proof Consider a trial sequence $S = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_\ell, y_\ell))$, in which $y_t = \Phi_{\mathbf{u},\psi}(\mathbf{x}_t)$ for $t = 1, \ldots, \ell$. Assume that for $i = 1, \ldots, t-1$ we have $\mathbf{x}_i \in \{\, \mathbf{z}'_i, \mathbf{z}''_i \,\}$ where $B = ((\mathbf{z}'_1, \mathbf{z}''_1), \ldots, (\mathbf{z}'_\ell, \mathbf{z}''_\ell))$ is a sequence with pairwise constant dot products. Let $(\mathbf{w}_t, \theta_t)$ be the hypothesis of an additive linear on-line prediction algorithm before trial t. Write $\mathbf{w}_t = \mathbf{w}_1 + \sum_{j=1}^{t-1} \alpha_{t,j} \mathbf{x}_j$, and assume that the initial weight vector $\mathbf{w}_1$ and the target $(\mathbf{u}, \psi)$ differ at trial t on the sequence B.

Consider first the case with $\mathbf{w}_1 \cdot \mathbf{z}'_t \leq \mathbf{w}_1 \cdot \mathbf{z}''_t$ and $\mathbf{u} \cdot \mathbf{z}'_t > \psi > \mathbf{u} \cdot \mathbf{z}''_t$. Since B has pairwise constant dot products, we also have $\mathbf{w}_t \cdot \mathbf{z}'_t \leq \mathbf{w}_t \cdot \mathbf{z}''_t$. If $\theta_t \leq \mathbf{w}_t \cdot \mathbf{z}'_t$, the adversary chooses $\mathbf{x}_t = \mathbf{z}''_t$. In this case $\hat{y}_t = 1$ and $y_t = 0$, so the algorithm makes a mistake. Otherwise, the adversary chooses $\mathbf{x}_t = \mathbf{z}'_t$, so $\hat{y}_t = 0$ and $y_t = 1$ and again the algorithm makes a mistake. The case $\mathbf{w}_1 \cdot \mathbf{z}'_t \geq \mathbf{w}_1 \cdot \mathbf{z}''_t$ and $\mathbf{u} \cdot \mathbf{z}'_t < \psi < \mathbf{u} \cdot \mathbf{z}''_t$ is similar. □

Thus, proving lower bounds is reduced to finding for a given initial weight vector a sequence with pairwise constant dot products and a target such that the initial weight vector and the target differ sufficiently often. The sequence we use is given in the following definition.

Definition 9: *Let $N = 2^d + k - 1$ for some positive integers d and k, and write $\ell = N - k + 1$. Let H_ℓ be the $\ell \times \ell$ Hadamard matrix defined in Definition 4, and for $t = 1, \ldots, \ell$, let $\mathbf{h}_t$ be the tth row of H_ℓ. We define a sequence $B_{\mathrm{H}} = ((\mathbf{z}'_1, \mathbf{z}''_1), \ldots, (\mathbf{z}'_\ell, \mathbf{z}''_\ell))$ by setting*

$$\begin{aligned} \mathbf{z}'_t &= ((h_{t,1} + 1)/2, \ldots, (h_{t,\ell} + 1)/2, 0, \ldots, 0) \\ \mathbf{z}''_t &= ((-h_{t,1} + 1)/2, \ldots, (-h_{t,\ell} + 1)/2, 0, \ldots, 0) \ . \end{aligned}$$

Lemma 10: *The sequence B_{H} defined in Definition 9 has pairwise constant dot products.*

Proof Follows from the facts that $\mathbf{h}_t \cdot \mathbf{h}_{t'} = 0$ for $t \neq t'$ and $\sum_{i=1}^{N} h_{t,i} = -\sum_{i=1}^{N} h_{t,i} = 0$ for $t \geq 2$. □

Lemma 11: *Let $N = 2^d + k - 1$ for some positive integers d and k, and write $\ell = N - k + 1$, and let B_{H} be as in Definition 9. There is a monotone k-literal disjunction $(\mathbf{u}, 1/2)$ such that the zero vector $\mathbf{0}$ and $(\mathbf{u}, 1/2)$ differ at every trial on the sequence B_{H}.*

Proof Let $\mathbf{u} \in \{\, 0, 1 \,\}^N$ be the vector with $u_i = 1$ for $i = 1$ and $\ell + 1 \leq i \leq N$, and $u_i = 0$ for $2 \leq i \leq \ell$. For all t we have $\mathbf{0} \cdot \mathbf{z}'_t = \mathbf{0} \cdot \mathbf{z}''_t = 0$. For the sequence B_{H} we have $\mathbf{u} \cdot \mathbf{z}''_t = 0 < 1/2 < 1 = \mathbf{u} \cdot \mathbf{z}'_t$. □

Lemma 12: *Let $N = 2^d + k - 1$ for some positive integers d and k, and write $\ell = N - k + 1$, and let B_{H} be as in Definition 9. For any vector $\mathbf{w} \in \mathbf{R}^N$ there is a monotone k-literal disjunction $(\mathbf{u}, \psi)$ such that $\mathbf{w}$ and $(\mathbf{u}, \psi)$ differ at at least $\ell/2$ trials on the sequence B_{H}.*

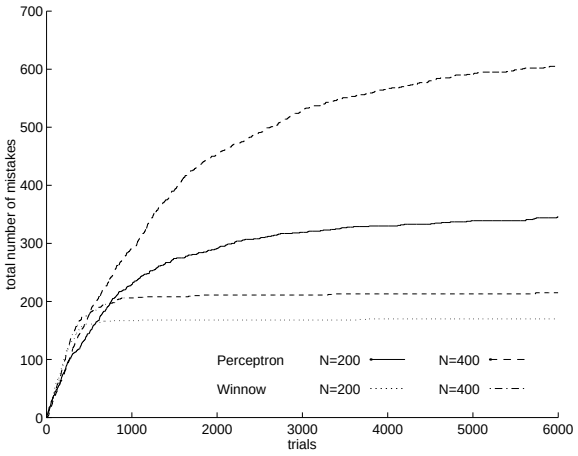

Figure 4.1: Cumulative mistake counts for the Perceptron algorithm and Winnow with random instances and a monotone 20-literal disjunction as target.

Proof Define a vector $\mathbf{p} \in \{-1, 1\}^\ell$ by $p_t = -1$ if $\mathbf{w} \cdot \mathbf{z}'_t \leq \mathbf{w} \cdot \mathbf{z}''_t$, and $p_t = 1$ otherwise. According to Lemma 5, we can choose an index i such that $D(\mathbf{p}, \mathbf{q}) \geq \ell/2$ when $\mathbf{q}$ is the ith column of the Hadamard matrix H_ℓ. We now choose $\mathbf{u}$ with $u_j = 1$ if $j = i$ or $\ell + 1 \leq j \leq N$, and $u_j = 0$ otherwise. By the construction of B_{H}, we have $\mathbf{u} \cdot \mathbf{z}'_t = 0$ and $\mathbf{u} \cdot \mathbf{z}''_t = 1$ when $q_i = -1$, and $\mathbf{u} \cdot \mathbf{z}'_t = 1$ and $\mathbf{u} \cdot \mathbf{z}''_t = 0$ when $q_i = 1$. Therefore, for $\psi = 1/2$ the vector $\mathbf{w}$ and the k-literal disjunction $(\mathbf{u}, \psi)$ differ at trial t on B whenever $p_t \neq q_t$. □

By combining Theorem 8 and Lemma 10 with Lemmas 11 and 12 we now get our main results.

Theorem 13: *Let $N = 2^d + k - 1$ for some positive integers d and k, and write $\ell = N - k + 1$.*

For any additive linear on-line prediction algorithm there is an N-dimensional trial sequence with a monotone k-literal disjunction as a target such that the algorithm makes $\ell/2$ mistakes on the trial sequence.

For any additive linear on-line prediction algorithm with a zero initial weight vector $\mathbf{w}_1 = \mathbf{0}$ there is an N-dimensional trial sequence with a monotone k-literal disjunction as a target such that the algorithm makes ℓ mistakes on the trial sequence.

By the comments made in Section 2, Theorem 13 gives a lower bound of $N - k + 1$ mistakes for the ellipsoid algorithm and for the Perceptron algorithm with zero as its initial weight vector. We also obtain a lower bound of $(N - k + 1)/2$ mistakes for the Perceptron algorithm with arbitrary initial weight vectors. Both of the above lower bounds for the Perceptron algorithm allow the algorithm to use arbitrary thresholds in each trial.

4 Experiments

In this section we describe some simple experiments on the Perceptron algorithm and Winnow. The experiments are not intended as a thorough empirical evaluation of the algorithms. Rather, they illustrate the fact that in certain circumstances the actual behavior of the algorithms is qualitatively similar to what could be expected from considering the worst-case bounds.

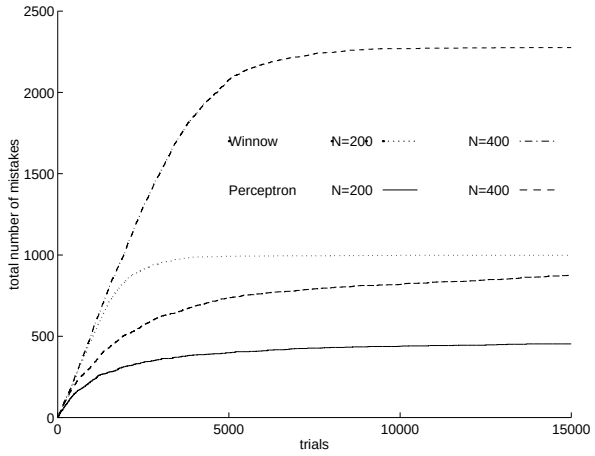

Figure 4.2: Cumulative mistake counts for the Perceptron algorithm and Winnow with sparse random instances and $k = 2N/5$ literals in the target.

Figure 4.1 shows the results of experiments in which the target was a monotone disjunction with $k = 20$ literals. The instances were generated according to the uniform distribution on $\{0,1\}^N$. The number N of variables was 200 in the first experiment and 400 in the second one. We used for Winnow the parameters $\mathbf{w}_1 = (1,\ldots,1)$, $\eta = \ln(3/2)$, and $\theta_t = N/k$ for all t. With this tuning Winnow is guaranteed to make at most $O(k+\log(N/k))$ mistakes if the target is a k-literal monotone disjunction, and the algorithm is even robust against noise [Lit91]. For the Perceptron algorithm we have used zero initial weights and eliminated the threshold by the transformation given in Section 2. In this case the choice of the learning rate of the Perceptron algorithm makes no difference.

As we see, in this sparse case Winnow made clearly fewer mistakes. Even more notable is how little Winnow's performance degraded when the dimensionality of the instances was doubled. On the other hand, the number of mistakes of the Perceptron algorithm increased much more drastically.

Figure 4.2 shows the results of experiments in which the number k of literals in the target monotone disjunction was $2N/5$ for N-dimensional instances. Thus, the targets were much denser that the ones used for Figure 4.1. Again, we made experiments with $N = 200$ and $N = 400$. The instances were chosen from $\{0,1\}^N$ at random with each component $x_{t,i}$ having probability $1 - 2^{-1/k}$ of having value 1. This makes the probability of having a positive example $1/2$. With the values k and N we considered, this gave approximately 1.7 as the average number of input variables with value one; that is, $\frac{1}{\ell}\sum_{t=1}^{\ell}(\sum_{i=1}^{N} x_{t,i}) \approx 1.7$.

In the experiments of Figure 4.2 the Perceptron algorithm outperformed Winnow. Note that we used the tuning that gives for Winnow the $O(k + \log(N/k))$ worst-case bound and makes it robust against noise. For the tuning $\theta = 1$ and η very large, the algorithm would make at most $N - k$ mistakes in our noise-free setting. This bound is lower than the number of mistakes of the Perceptron algorithm in the experiments of Figure 4.2.

There are several issues that need to be addressed in further experimental work. The experiments reported here were made with noiseless data. Both Winnow and the Perceptron algorithm tolerate noise reasonably well, but the presence of noise could affect the comparison. Finally, it remains to see whether the algorithm's behavior on data from actual applications is similar to their behavior on random data.

5 Open problems

So far the evaluation of our algorithms on random data is only experimental. However, it seems possible to obtain closed formulas for the expected total number of mistakes of the Perceptron algorithm on some thermodynamic limit (see, e.g, [SST91, WRB93]). We wish to study how these closed formulas relate to the worst-case upper bounds and the adversary lower bounds. Studying this behavior will lead to a deeper understanding of how high dimensionality hurts the Perceptron algorithm and other additive algorithms. A more extensive experimental comparison of various on-line algorithms for learning disjunctions in the presence of attribute noise has recently been done by Littlestone [Lit95].

Bounds on the worst-case number of mistakes have earlier been obtained for both the Perceptron algorithm and Winnow. Both of these upper bound proofs can be interpreted as using amortized analysis with a potential function. Different potential functions are used: a generalized version of the entropy function for Winnow [Lit91, AW95] and the squared Euclidean distance in the Perceptron Convergence Theorem [DH73]. Our observations are analogous to the case of on-line linear regression. Again, there are two algorithms, and worst-case loss bounds for them can be proved using the two potential functions [KW94]. In the case of linear regression it is even possible to derive the updates from the two potential functions in addition to using them in a worst-case analysis. It is an open problem to devise a framework for deriving updates from the potential functions in the linear classification case.

The Perceptron algorithm is not specialized to learn disjunctions. However, the purpose of this paper is to show that the number of mistakes made by this and any other additive algorithm can grow linearly in the number of variables even for simple linear classifiers such as small disjunctions. If only few variables are relevant then additive algorithms seem to use all the dimensions in a futile search for a good predictor. A cleaner comparison between the related algorithms has been obtained in the linear regression case [KW94].

When the targets are small disjunctions, the ellipsoid method also exhibits similar linear growth of its number of mistakes as the computationally trivial Perceptron algorithm. We do not know whether Vaidya's algorithm for linear programming exhibits the linear growth. Applying linear programming methods to learning simple disjunctions remains a subject for further study. There are some implementation issues that need to be addressed to make the general linear programming methods perform optimally on this very restricted problem. In any case, these algorithms require significantly more computation time per update than the Perceptron algorithm or Winnow.

In this paper we considered only the case in which there is a target disjunction that is consistent with the trial sequence. If the instances or outcomes are corrupted by noise, such a target does not usually exist. The noisy case has been considered by Littlestone [Lit91] and more recently by Auer and Warmuth [AW95]. Auer and Warmuth also prove worst-case loss bounds for the situation in which the target may change over time.

References

[AW95] P. Auer and M. K. Warmuth. Tracking the best disjunction. In *Proc. 36th Symposium on the Foundations of Comp. Sci.* IEEE Computer Society Press, Los Alamitos, CA, 1995.

[BEHW89] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *J. ACM*, 36(4):929–965, 1989.

[DH73] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis*. Wiley, 1973.

[Kha79] L. G. Khachiyan. A polynomial algorithm in linear programming (in Russian). *Doklady Akademii Nauk SSSR*, 244:1093–1096, 1979. (English translation: *Soviet Mathematics Doklady* 20:191–194, 1979.)

[KW94] J. Kivinen and M. K. Warmuth. Exponentiated gradient versus gradient descent for linear predictors. Report UCSC-CRL-94-16, University of California, Santa Cruz, June 1994. An extended abstract appeared in STOC '95.

[Lit88] N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Mach. Learning*, 2:285–318, 1988.

[Lit89] N. Littlestone. *Mistake Bounds and Logarithmic Linear-threshold Learning Algorithms*. PhD thesis, Report UCSC-CRL-89-11, University of California Santa Cruz, 1989.

[Lit91] N. Littlestone. Redundant noisy attributes, attribute errors, and linear threshold learning using Winnow. In *Proc. 4th Workshop on Comput. Learning Theory*, pages 147–156. Morgan Kaufmann, 1991.

[Lit95] N. Littlestone. Comparing several linear-threshold learning algorithms on tasks involving superfluous attributes. In *Proc. 12th International Machine Learning Conference (ML-95)*, pages 353–361, 1995.

[LLW91] N. Littlestone, P. M. Long, and M. K. Warmuth. On-line learning of linear functions. In *Proc. 23rd ACM Symposium on Theory of Computing*, pages 465–475, 1991.

[MT94] W. Maass and G. Turán. How fast can a threshold gate learn. In *Computational Learning Theory and Natural Learning Systems*, Volume I, pages 381–414. MIT Press, 1994.

[Ros58] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psych. Rev.*, 65:386–407, 1958. (Reprinted in *Neurocomputing* (MIT Press, 1988).).

[SST91] H. Sompolinsky, H. S. Seung, and N. Tishby. Learning curves in large neural networks. In *Proc. 4th Workshop on Comput. Learning Theory*, pages 112–127. Morgan Kaufmann, 1991.

[Vai89] P. M. Vaidya. A new algorithm for minimizing convex functions over convex sets. In *Proc. 30th Symposium on Foundations of Computer Science*, pages 338–343. IEEE Computer Society, 1989.

[Val84] L. G. Valiant. A theory of the learnable. *Commun. ACM*, 27(11):1134–1142, 1984.

[VC71] V. N. Vapnik and A. Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probab. and its Applications*, 16(2):264–280, 1971.

[WRB93] T. L. H. Watkin, A. Rau, and M. Biehl. The statistical mechanics of learning a rule. *Rev. Mod. Phys.*, 65:499–556, 1993.