SIAM J. COMPUT.
Vol. 34, No. 3, pp. 640–668

A SECOND-ORDER PERCEPTRON ALGORITHM*

NICOLÒ CESA-BIANCHI†, ALEX CONCONI†, AND CLAUDIO GENTILE‡

Abstract. Kernel-based linear-threshold algorithms, such as support vector machines and Perceptron-like algorithms, are among the best available techniques for solving pattern classification problems. In this paper, we describe an extension of the classical Perceptron algorithm, called second-order Perceptron, and analyze its performance within the mistake bound model of on-line learning. The bound achieved by our algorithm depends on the sensitivity to second-order data information and is the best known mistake bound for (efficient) kernel-based linear-threshold classifiers to date. This mistake bound, which strictly generalizes the well-known Perceptron bound, is expressed in terms of the eigenvalues of the empirical data correlation matrix and depends on a parameter controlling the sensitivity of the algorithm to the distribution of these eigenvalues. Since the optimal setting of this parameter is not known a priori, we also analyze two variants of the second-order Perceptron algorithm: one that adaptively sets the value of the parameter in terms of the number of mistakes made so far, and one that is parameterless, based on pseudoinverses.

Key words. pattern classification, mistake bounds, Perceptron algorithm

AMS subject classifications. 68Q32, 68W40, 68T10

DOI. 10.1137/S0097539703432542

1. Introduction.

Research in linear-threshold classifiers has been recently revamped by the popularity of kernel methods [1, 12, 36], a set of mathematical tools used to efficiently represent complex nonlinear decision surfaces in terms of linear classifiers in a high-dimensional feature space defined by kernel functions. To some extent, statistical learning theories have been able to explain why kernel methods do not suffer from the "curse of dimensionality"—that is, why they exhibit a remarkable predictive power despite the fact that the kernel-induced feature space has very many (possibly infinite) dimensions. However, these statistical results are often based on quantities, like the "margin," that provide only a superficial account of the way the predictive power is affected by the geometry of the feature space.

A different approach to the analysis of linear-threshold classifiers is the mistake bound model of on-line learning [28]. In this model, similarly to the framework of competitive analysis, the learning algorithm must be able to sequentially classify each sequence of data points making a number of mistakes not much bigger than those made by the best fixed linear-threshold classifier in hindsight (the reference predictor). The power of this approach resides in the fact that the excess loss of the learning algorithm with respect to the reference predictor can be nicely bounded in terms of the geometrical properties of any individual sequence on which the algorithm is run. Furthermore, as shown in [9], mistake bounds have corresponding statistical risk bounds that are not worse, and sometimes better, than those obtainable with a direct statistical approach.

*Received by the editors July 31, 2003; accepted for publication (in revised form) September 21, 2004; published electronically March 17, 2005. A preliminary version of this paper appeared in *Proceedings of the* 15*th Annual Conference on Computational Learning Theory (COLT'*02*)*, pp. 121–137. This research was partially supported by the European Commission under KerMIT project IST-2001-25431.

http://www.siam.org/journals/sicomp/34-3/43254.html

†Department of Information Sciences, Università di Milano, Italy (cesa-bianchi@dsi.unimi.it, conconi@dti.unimi.it).

‡DICOM, Università dell'Insubria, Varese, Italy (gentile@dsi.unimi.it).

So far, the best known mistake bound for kernel-based linear-threshold classifiers was essentially the one achieved by the classical[1] Perceptron algorithm [7, 32, 34]. In this paper we introduce an extension of the standard Perceptron algorithm, called *second-order Perceptron*.

The standard Perceptron algorithm is a popular greedy method for learning linear-threshold classifiers. Since the early sixties, it has been known that the performance of the Perceptron algorithm is governed by simple geometrical properties of the input data. As the Perceptron algorithm is essentially a gradient descent (first-order) method, we improve on it by introducing sensitivity to second-order data information. Our second-order algorithm combines the Perceptron's gradient with a sparse (and incrementally computed) version of the data correlation matrix. Our analysis shows that the second-order Perceptron algorithm is able to exploit certain geometrical properties of the data which are missed by the first-order algorithm. In particular, our bounds for the second-order algorithm depend on the distribution of the eigenvalues of the correlation matrix for the observed data sequence, as opposed to the bound for the first-order algorithm, relying only on trace information. A typical situation where the second-order bound is substantially smaller than the first-order bound is when the data lie on a flat ellipsoid, so that the eigenvalues of the correlation matrix have sharply different magnitudes, whereas the trace is dominated by the largest one.

In its basic form, the second-order Perceptron algorithm is parameterized by a constant $a > 0$, which rules the extent to which the algorithm adapts to the "warpedness" of the data. In the limit as a goes to infinity our algorithm becomes the first-order Perceptron algorithm and, in that limit, the second-order bound reduces to the first-order one. The value of a affects performance in a significant way. The best choice of a depends on information about the learning task that is typically not available ahead of time. We develop two variants of our algorithm and prove corresponding mistake bounds. The first variant is an adaptive parameter version, while the second variant eliminates parameter a and replaces standard matrix inversion with pseudo-inversion. Again, the bounds we prove are able to capture the spectral properties of the data.

In the next section, we take the classical Perceptron algorithm as a starting point for defining and motivating our second-order extension. Our description steps through an intermediate algorithm, called the *whitened Perceptron* algorithm, which serves as an illustration of the kind of spectral behavior we mean. The section is concluded with a precise definition of the learning model and with a description of the basic notation used throughout the paper.

2. Perceptron and whitened Perceptron. The classical Perceptron algorithm processes a stream of examples $(\boldsymbol{x}_t, y_t)$ one at a time in trials. In trial t the algorithm receives an instance vector $\boldsymbol{x}_t \in \mathbb{R}^n$ and predicts the value of the unknown label $y_t \in \{-1, +1\}$ associated with $\boldsymbol{x}_t$. The algorithm keeps a weight vector $\boldsymbol{w}_t \in \mathbb{R}^n$ representing its internal state, and its prediction at time t is given by[2]

[1] Several kernel-based linear-threshold algorithms have been proposed and analyzed in recent years, including relaxation methods [14], ROMMA [27], Alma [17], and variants thereof. All these on-line algorithms are Perceptron-like, and their mistake bounds coincide with the one achieved by the standard Perceptron algorithm.

[2] Here and throughout, superscript $^\top$ denotes transposition. Also, SGN denotes the signum function $\mathrm{SGN}(z) = 1$ if $z \geq 0$ and $\mathrm{SGN}(z) = -1$ otherwise (we conventionally give a positive sign to zero).

$\widehat{y}_t = \text{SGN}(\boldsymbol{w}_t^\top \boldsymbol{x}_t) \in \{-1, 1\}$. We say that the algorithm has made a *mistake* at trial t if the prediction $\widehat{y}_t$ and the label y_t disagree. In such a case the algorithm updates its internal state according to the simple additive rule $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + y_t \boldsymbol{x}_t$, i.e., by moving the old weight vector along the direction of the instance $\boldsymbol{x}_t$ on which the algorithm turned out to be wrong (with the "right" orientation determined by $y_t = -\widehat{y}_t$). If $\widehat{y}_t = y_t$, no weight update is made (thus the number of mistakes is always equal to the number of weight updates).

A sequence of examples is linearly separable if the instance vectors $\boldsymbol{x}_t$ are consistently labeled according to whether they lie on the positive ($y_t = +1$) or the negative ($y_t = -1$) side of an unknown target hyperplane with normal vector $\boldsymbol{u} \in \mathbb{R}^n$ and passing through the origin. The well-known Perceptron convergence theorem [7, 32] states that the Perceptron algorithm is guaranteed to make at most $(R/\gamma)^2$ mistakes on any number t of examples in a linearly separable sequence, where

$$R = \max_{1 \le s \le t} \|\boldsymbol{x}_s\|,$$
$$\gamma = \min_{1 \le s \le t} \left|\boldsymbol{u}^\top \boldsymbol{x}_t\right|.$$

This implies that cycling through any sequence of linearly separable examples will eventually lead the Perceptron algorithm to compute a weight vector $\boldsymbol{w} \in \mathbb{R}^n$ classifying all examples correctly, i.e., such that $y_t = \text{SGN}(\boldsymbol{w}^\top \boldsymbol{x}_t)$ for all t in the sequence. The convergence speed is thus critically influenced by the degree of linear separability of the examples, as expressed by the ratio $(R/\gamma)^2$. Hence, those vectors $\boldsymbol{x}_t$ which have both a small projection component over $\boldsymbol{u}$ (i.e., they are "almost" orthogonal to $\boldsymbol{u}$) and have a large norm $\|\boldsymbol{x}_t\|$ are the hardest ones for the Perceptron algorithm. The situation is illustrated in Figure 2.1. Consider the case when the algorithm observes instance vectors $\boldsymbol{x}_t$ lying in the dotted circles on opposite sides of that figure. Such vectors have small components along the direction of $\boldsymbol{u}$; hence they are intuitively irrelevant to the target vector. Still, such $\boldsymbol{x}_t$'s might significantly mislead the Perceptron algorithm (thereby slowing down its convergence). Trial t depicted in Figure 2.1 is a mistaken trial for the algorithm, since $\text{SGN}(\boldsymbol{w}_t^\top \boldsymbol{x}_t) \neq \text{SGN}(\boldsymbol{u}^\top \boldsymbol{x}_t) = y_t = -1$. Now, the new weight vector $\boldsymbol{w}_{t+1}$ computed by the algorithm has a larger projection[3] onto $\boldsymbol{u}$ than the old vector $\boldsymbol{w}_t$. But the algorithm's prediction is based only upon the direction of its current weight vector. Hence the step $\boldsymbol{w}_t \to \boldsymbol{w}_{t+1}$ does not seem to make satisfactory progress toward $\boldsymbol{u}$.

Let us now turn to an algorithm which is related to both the first-order and the second-order Perceptron algorithm (described in section 3). We call this algorithm the *whitened Perceptron* algorithm. Strictly speaking, this is not an incremental algorithm. In fact, it assumes that all the instance vectors $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_T \in \mathbb{R}^n$ are preliminarily available, and only the labels $y_1, y_2, \ldots, y_T$ are hidden. For the sake of simplicity, assume that these vectors span $\mathbb{R}^n$. Therefore $T \ge n$, the correlation matrix $M = \sum_{t=1}^{T} \boldsymbol{x}_t \boldsymbol{x}_t^\top$ is full-rank, and M^{-1} exists. Also, since M is positive definite, $M^{-1/2}$ exists as well (see, e.g., [31, Chap. 3]). The whitened Perceptron algorithm is simply the standard Perceptron algorithm run on the transformed (whitened) sequence $(M^{-1/2}\boldsymbol{x}_1, y_1), (M^{-1/2}\boldsymbol{x}_2, y_2), \ldots, (M^{-1/2}\boldsymbol{x}_T, y_T)$. The transformation $M^{-1/2}$ is called the whitening transform (see, e.g., [14]) and has the

[3] In its simplest form, the Perceptron convergence theorem exploits the measure of progress $\boldsymbol{u}^\top \boldsymbol{w}_t$ and is based on the fact that under linear separability assumptions this measure steadily increases through mistaken trials.

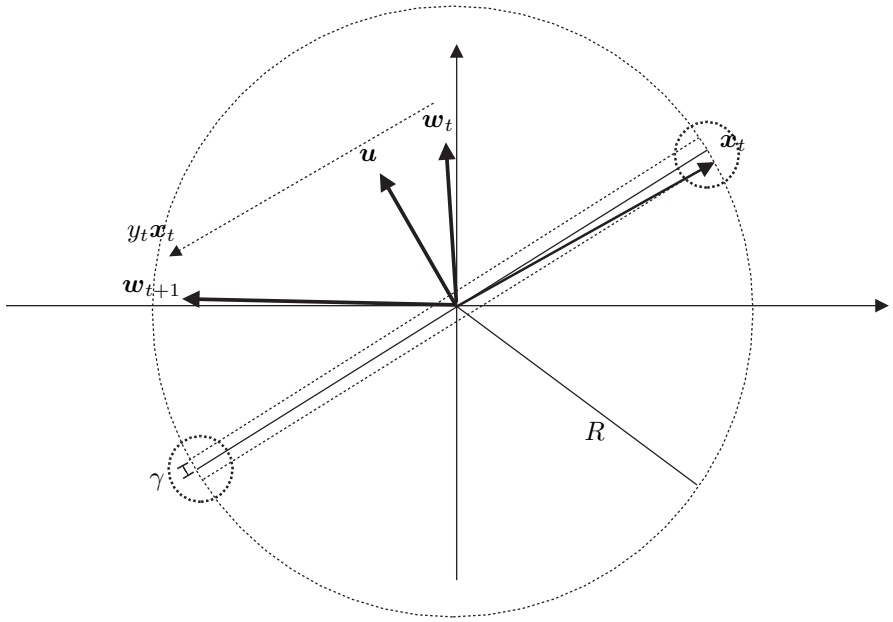

FIG. 2.1. *Behavior of the Perceptron algorithm on extreme (though linearly separable) cases. Here* $\boldsymbol{u}$ *denotes the hidden target vector,* $\boldsymbol{w}_t$ *is the weight vector maintained by the algorithm at the beginning of trial* t*, and* $\boldsymbol{x}_t$ *is the instance vector observed in that trial. We are assuming that all instances have bounded (Euclidean) length* R *and that the examples are linearly separable with margin* $\gamma > 0$ *(so that no vector in the sequence of examples can lie within the two dotted lines running parallel to the decision boundary of* $\boldsymbol{u}$*). Since the angle between* $\boldsymbol{u}$ *and* $\boldsymbol{x}_t$ *is (slightly) larger than* 90 *degrees, the label* y_t *is assigned the value* -1*. On the other hand,* $\boldsymbol{w}_t^\top \boldsymbol{x}_t > 0$ *holds; hence the algorithm makes a mistake. Now,* $\boldsymbol{x}_t$ *lies exactly on one of the two dotted lines and has maximal length* R*, but it has also a small projection along the direction of* $\boldsymbol{u}$*, meaning that the direction marked by* $\boldsymbol{x}_t$ *is (almost) irrelevant to* $\boldsymbol{u}$*. However, the simple additive rule of the Perceptron algorithm makes the new weight vector* $\boldsymbol{w}_{t+1}$ *farther from* $\boldsymbol{u}$ *than the old one.*

effect of reducing the correlation matrix of the transformed instances to the identity matrix I_n. In fact,

$$\begin{aligned}\sum_{t=1}^{T}\left(M^{-1/2}\boldsymbol{x}_t\right)\left(M^{-1/2}\boldsymbol{x}_t\right)^\top &= \sum_{t=1}^{T} M^{-1/2}\boldsymbol{x}_t\,\boldsymbol{x}_t^\top M^{-1/2}\\ &= M^{-1/2} M\, M^{-1/2}\\ &= I_n.\end{aligned}$$

Again for the sake of simplicity, suppose that the original sequence of examples is linearly separable: $\gamma = \min_t y_t\,\boldsymbol{u}^\top\boldsymbol{x}_t > 0$ for some unit norm vector $\boldsymbol{u}$. Then the whitened sequence is also linearly separable. In fact, hyperplane $\boldsymbol{z} = M^{1/2}\boldsymbol{u}$ separates the whitened sequence with margin $\gamma/\left\|M^{1/2}\boldsymbol{u}\right\|$. By the aforementioned convergence theorem, the number of mistakes made by the Perceptron algorithm on the whitened sequence is thus at most

$$\frac{1}{\gamma^2}\left(\max_t\left\|M^{-1/2}\boldsymbol{x}_t\right\|^2\right)\left\|M^{1/2}\boldsymbol{u}\right\|^2 = \frac{1}{\gamma^2}\max_t\left(\boldsymbol{x}_t^\top M^{-1}\boldsymbol{x}_t\right)\left(\boldsymbol{u}^\top M\boldsymbol{u}\right). \tag{2.1}$$

To appreciate the potential advantage of whitening the data, note that when the instance vectors $\boldsymbol{x}_1,\ldots,\boldsymbol{x}_T$ are very correlated the quadratic form $\boldsymbol{x}_t^\top M^{-1}\boldsymbol{x}_t$ tends

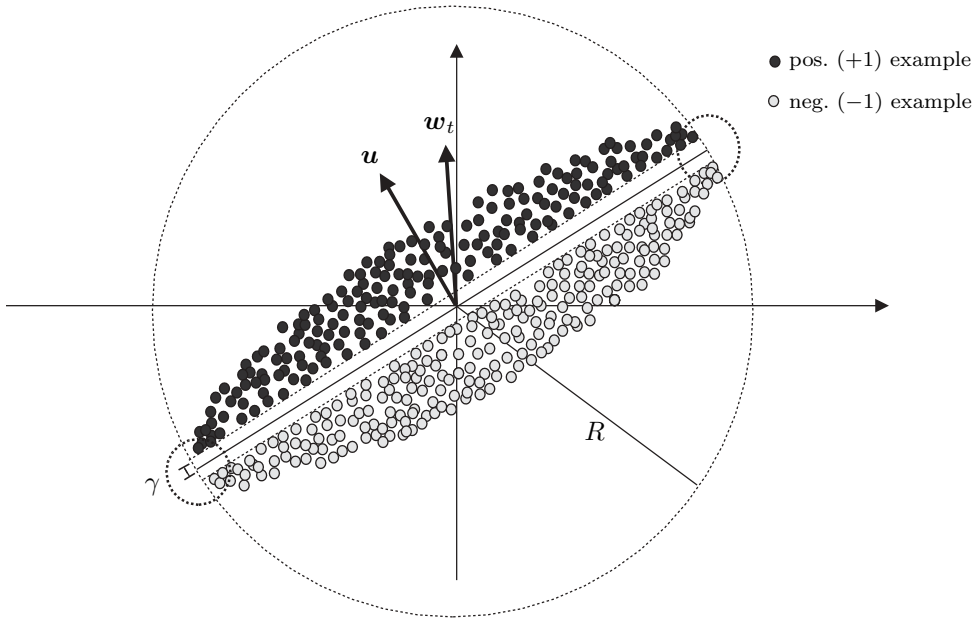

FIG. 2.2. *Scattering of data which the whitened (and the second-order) Perceptron algorithm can take advantage of. Here all instance vectors lie on a flat ellipsoid enclosed in a ball of radius R. The examples are linearly separable with margin $\gamma > 0$ via a hyperplane whose normal vector $\boldsymbol{u}$ is aligned with the small axis of the ellipse. Thus for any instance vector $\boldsymbol{x}_t$ the projection $|\boldsymbol{u}^\top \boldsymbol{x}_t|$ onto $\boldsymbol{u}$ is "small" (though not smaller than γ). This does not make any difference for the standard Perceptron algorithm, whose worst-case behavior is essentially ruled by the* norm *of the instances lying in the two dotted circles (recall Figure* 2.1*).*

to be quite small (the expression $\max_t \left(\boldsymbol{x}_t^\top M^{-1} \boldsymbol{x}_t\right)$ might actually be regarded as a measure of correlation of the instance vectors). Also, if the instances look like those displayed in Figure 2.2, where the separating hyperplane vector $\boldsymbol{u}$ is strongly correlated with a nondominant eigenvector of M (i.e., if all instances have a small projected component onto $\boldsymbol{u}$), then the bound in the right-hand side of (2.1) can be significantly smaller than the corresponding mistake bound $\max_t \|\boldsymbol{x}_t\|^2/\gamma^2 = R^2/\gamma^2$ for the classical (nonwhitened) Perceptron algorithm.

For the sake of clarity, consider the degenerate case when all data points in Figure 2.2 are evenly spread over the two parallel lines at margin γ (so that the ellipse sketched in that figure is maximally squashed along the direction of $\boldsymbol{u}$). It is not hard to argue that this symmetric scattering of data leads to the following eigenstructure of matrix M: Let $\lambda_{\min}$ and $\lambda_{\max}$ be the minimal and the maximal eigenvalues of M, respectively. We have that the first eigenvector of M (the one associated with $\lambda_{\min}$) is aligned with $\boldsymbol{u}$, while the second eigenvector (associated with $\lambda_{\max}$) is orthogonal to $\boldsymbol{u}$ (i.e., it is parallel to the two lines mentioned above). Let us denote by $\boldsymbol{u}^\perp$ a unit norm vector orthogonal to $\boldsymbol{u}$. Now, since $\boldsymbol{u}$ has unit norm, we have

$$\lambda_{\min} = \boldsymbol{u}^\top M \boldsymbol{u} = \boldsymbol{u}^\top \left(\sum_{t=1}^{T} \boldsymbol{x}_t \, \boldsymbol{x}_t^\top \right) \boldsymbol{u} = \sum_{t=1}^{T} (\boldsymbol{u}^\top \boldsymbol{x}_t)^2 = \gamma^2 \, T. \tag{2.2}$$

Also, since $\lambda_{\min} + \lambda_{\max} = \sum_{t=1}^{T} \|\boldsymbol{x}_t\|^2$ (see, e.g., section 3.1) and since, for all $\boldsymbol{x} \in \mathbb{R}^n$,

$(\boldsymbol{u}^\top \boldsymbol{x})^2 + ((\boldsymbol{u}^\perp)^\top \boldsymbol{x})^2 = \|\boldsymbol{x}\|^2$, for each $t = 1, \ldots, T$ we can write

$$\boldsymbol{x}_t M^{-1} \boldsymbol{x}_t = \frac{(\boldsymbol{u}^\top \boldsymbol{x}_t)^2}{\lambda_{\min}} + \frac{((\boldsymbol{u}^\perp)^\top \boldsymbol{x}_t)^2}{\lambda_{\max}} = \frac{\gamma^2}{\gamma^2 T} + \frac{\|\boldsymbol{x}_t\|^2 - \gamma^2}{\sum_{t=1}^T \|\boldsymbol{x}_t\|^2 - \gamma^2 T}, \tag{2.3}$$

where in the first step we used the singular value decomposition (SVD) of M (see Appendix D). Hence, combining (2.2) and (2.3) as in (2.1) and simplifying, we conclude that in the extreme case we sketched, the number of mistakes made by the whitened Perceptron algorithm can be bounded by

$$\frac{1}{\gamma^2} \max_t \left(\boldsymbol{x}_t^\top M^{-1} \boldsymbol{x}_t\right) \left(\boldsymbol{u}^\top M \boldsymbol{u}\right) = 1 + \frac{R^2 T - \gamma^2 T}{\sum_{t=1}^T \|\boldsymbol{x}_t\|^2 - \gamma^2 T}.$$

Note that this bound approaches 2 as the norm of the instance vectors $\boldsymbol{x}_t$ approaches R (which is just the hardest case for the standard Perceptron algorithm). Thus in this case, unlike the standard Perceptron bound R^2/γ^2, the whitened Perceptron bound tends to be a *constant*, independent of both the margin γ and the radius R of the ball containing the data.

Learning model and notation. In the last part of this section we precisely describe the learning model and introduce our notation (some of this notation has been used earlier). The formal model we consider is the well-known mistake bound model of incremental learning, introduced by Littlestone [28] (see also [2]) and further investigated by many authors (see, e.g., [4, 10, 11, 20, 21, 25, 26, 29, 30, 39] and the references therein).

In incremental or sequential models, learning proceeds in trials. In each trial $t = 1, 2, \ldots$, the algorithm observes an *instance* vector $\boldsymbol{x}_t \in \mathbb{R}^n$ (all vectors here are understood to be column vectors) and then guesses a binary label $\widehat{y}_t \in \{-1, 1\}$. Before seeing the next vector $\boldsymbol{x}_{t+1}$, the true label $y_t \in \{-1, 1\}$ associated with $\boldsymbol{x}_t$ is revealed and the algorithm knows whether its guess $\widehat{y}_t$ for y_t was correct or not. In the latter case we say that the algorithm has made a *mistake*, and we call t a *mistaken trial*. Each pair $(\boldsymbol{x}_t, y_t)$ we call an *example*, and a *sequence of examples* is any sequence $\mathcal{S} = ((\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_T, y_T))$. No assumptions are made on the mechanism generating the sequence of examples. Similarly to competitive analyses of on-line algorithms [8], the goal of the learning algorithm is to minimize the amount by which its total number of mistakes on an arbitrary sequence $\mathcal{S}$ exceeds some measure of the performance of a fixed classifier (in a given *comparison class*) on the same sequence $\mathcal{S}$.

The comparison class we consider here is the set of linear-threshold classifiers parametrized by the unit norm vectors $\{\boldsymbol{u} \in \mathbb{R}^n : \|\boldsymbol{u}\| = 1\}$. Thus we speak of the linear-threshold classifier $\boldsymbol{u}$ to mean the classifier $h(\boldsymbol{x}) = \textsc{sgn}(\boldsymbol{u}^\top \boldsymbol{x})$. For technical reasons, the number of mistakes of the learning algorithm will be compared to the cumulative *hinge loss* (see, e.g., [19]) of the best linear-threshold classifier in the comparison class. For any $\gamma > 0$, the hinge loss of the linear-threshold classifier $\boldsymbol{u}$ on example $(\boldsymbol{x}, y)$ at margin γ is $D_\gamma(\boldsymbol{u}; (\boldsymbol{x}, y)) = \max\{0, \gamma - y\, \boldsymbol{u}^\top \boldsymbol{x}\}$. Also, for a given sequence of examples $\mathcal{S} = ((\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_T, y_T))$, let $D_\gamma(\boldsymbol{u}; \mathcal{S}) = \sum_{t=1}^T D_\gamma(\boldsymbol{u}; (\boldsymbol{x}_t, y_t))$. Note that $D_\gamma(\boldsymbol{u}; \mathcal{S})/\gamma$ is a strict upper bound on the number of mistakes made by $\boldsymbol{u}$ on the same sequence $\mathcal{S}$. Moreover, if $\boldsymbol{u}$ linearly separates $\mathcal{S}$ with margin γ, i.e., if $y_t\, \boldsymbol{u}^\top \boldsymbol{x}_t \geq \gamma$ for $t = 1, \ldots, T$, then $D_\gamma(\boldsymbol{u}; \mathcal{S}) = 0$.

We will prove bounds on the number of mistakes having the general form

$$\texttt{number of mistakes on } \mathcal{S} \quad \leq \quad \inf_{\gamma > 0} \inf_{\boldsymbol{u}} \left(\frac{D_\gamma(\boldsymbol{u}; \mathcal{S})}{\gamma} + \frac{\texttt{spectral}(\boldsymbol{u}; \mathcal{S})}{\gamma} \right),$$

where $\mathcal{S}$ is any sequence of examples and $\texttt{spectral}(\boldsymbol{u};\mathcal{S})$ measures certain spectral properties arising from the interaction between $\boldsymbol{u}$ and $\mathcal{S}$. The mistake bound shown above reveals how the algorithm is able to optimally trade off between the terms $D_\gamma(\boldsymbol{u};\mathcal{S})$ and $\texttt{spectral}(\boldsymbol{u};\mathcal{S})$ using a *single* pass over an *arbitrary* data sequence. This aggressive adaptation can be successfully exploited in settings different from the mistake bound model. In fact, as mentioned in the conclusions, the linear-threshold classifiers generated by the second-order Perceptron algorithm can be easily shown to have a probability of mistake (risk) in the statistical model of pattern classification (see, e.g., [13]) which is tightly related to the algorithm's mistake bound. These risk bounds are actually among the best achievable by any algorithm for learning linear-threshold classifiers.

The rest of the paper is organized as follows. The next section describes and analyzes the basic form of the second-order Perceptron algorithm and also shows how to formulate the algorithm in dual form, thereby allowing the use of kernel functions. In section 4 we analyze two variants of the basic algorithm: the first variant has an adaptive parameter, and the second variant computes the pseudoinverse of the correlation matrix instead of the standard inverse. Some toy experiments are reported in section 5. Section 6 contains conclusions, final remarks, and open problems.

3. Second-order Perceptron, basic form. The second-order Perceptron algorithm might be viewed as an incremental variant of the whitened Perceptron algorithm; this means that the instances in the data sequence are not assumed to be known beforehand. Besides, the second-order Perceptron algorithm is sparse; that is, the whitening transform applied to each new incoming instance is based only on a possibly very small subset of the instances observed so far.

In its basic form, the second-order Perceptron algorithm (described in Figure 3.1) takes an input parameter $a > 0$. To compute its prediction in trial t the algorithm uses an n-row matrix X_{k-1} and an n-dimensional weight vector $\boldsymbol{v}_{k-1}$, where subscript $k-1$ indicates the number of times matrix X and vector $\boldsymbol{v}$ have been updated in the first $t-1$ trials. Initially, the algorithm sets $X_0 = \emptyset$ (the empty matrix) and $\boldsymbol{v}_0 = \mathbf{0}$. Upon receiving the tth instance $\boldsymbol{x}_t \in \mathbb{R}^n$, the algorithm builds the augmented matrix $S_t = [\, X_{k-1} \;\; \boldsymbol{x}_t \,]$ (where $\boldsymbol{x}_t$ is intended to be the last column of S_t) and predicts the label y_t of $\boldsymbol{x}_t$ with $\widehat{y}_t = \textsc{sgn}\left(\boldsymbol{v}_{k-1}^\top \left(aI_n + S_t\, S_t^\top\right)^{-1} \boldsymbol{x}_t\right)$, with I_n being the $n \times n$ identity matrix (the addition of I_n guarantees that the above inverse always exists). If $\widehat{y}_t \neq y_t$, then a mistake occurs and the algorithm updates[4] both $\boldsymbol{v}_{k-1}$ and X_{k-1}. Vector $\boldsymbol{v}_{k-1}$ is updated using the Perceptron rule $\boldsymbol{v}_k = \boldsymbol{v}_{k-1} + y_t\, \boldsymbol{x}_t$, whereas matrix X_{k-1} is updated using $X_k = S_t = [\, X_{k-1} \;\; \boldsymbol{x}_t \,]$. Note that this update implies $X_k X_k^\top = X_{k-1} X_{k-1}^\top + \boldsymbol{x}_t\, \boldsymbol{x}_t^\top$. The new matrix X_k and the new vector $\boldsymbol{v}_k$ will be used in the next prediction. If $\widehat{y}_t = y_t$, no update takes place, and hence the algorithm is mistake driven [28]. Also, just after an update, matrix X_k has as many columns as the number of mistakes made by the algorithm so far. Note that, unlike the Perceptron algorithm, here $\boldsymbol{w}_t$ does depend on $\boldsymbol{x}_t$ (through S_t). Therefore the second-order Perceptron algorithm, as described in Figure 3.1, is *not* a linear-threshold predictor.

Our algorithm might be viewed as an adaptation to on-line binary classification of the ridge regression [23] method. Indeed, our analysis is inspired by the analysis of a variant of ridge regression recently introduced by Vovk [40] and further studied by Azoury and Warmuth [5] and Forster and Warmuth [15]. This variant is an instance of Vovk's general *aggregating algorithm* and is called the "forward algorithm" in [5]. Both

[4] In the degenerate case when $\boldsymbol{x}_t = \mathbf{0}$ no update is performed.

Parameter: $a > 0$.
Initialization: $X_0 = \emptyset$; $\boldsymbol{v}_0 = \mathbf{0}$; $k = 1$.
Repeat for $t = 1, 2, \ldots$:

1. get instance $\boldsymbol{x}_t \in \mathbb{R}^n$;
2. set $S_t = [\, X_{k-1} \; \boldsymbol{x}_t \,]$;
3. predict $\widehat{y}_t = \text{SGN}(\boldsymbol{w}_t^\top \boldsymbol{x}_t) \in \{-1, +1\}$, where $\boldsymbol{w}_t = \left(a I_n + S_t \, S_t^\top\right)^{-1} \boldsymbol{v}_{k-1}$;
4. get label $y_t \in \{-1, +1\}$;
5. if $\widehat{y}_t \neq y_t$, then:

$$\begin{aligned} \boldsymbol{v}_k &= \boldsymbol{v}_{k-1} + y_t \, \boldsymbol{x}_t, \\ X_k &= S_t, \\ k &\leftarrow k + 1. \end{aligned}$$

FIG. 3.1. *The second-order Perceptron algorithm with parameter* $a > 0$.

our algorithm and the forward algorithm predict with a weight vector $\boldsymbol{w}_t$ given by the product of the inverse correlation matrix from past trials and a linear combination of past instance vectors. In both cases the current instance $\boldsymbol{x}_t$ is incorporated into the current correlation matrix before prediction. However, unlike the forward algorithm, we only keep track of past trials where the algorithm made a mistake. To complete this qualitative analogy, we note that the analysis of our algorithm uses tools developed in [5] for the forward algorithm.

The reader might wonder whether this nonlinear dependence on the current instance is somehow necessary. As a matter of fact, if in Figure 3.1 we predicted using X_{k-1} instead of the augmented matrix S_t, then the resulting algorithm would be a linear-threshold algorithm.[5] It is easy to show that the margin value $y_t \, \boldsymbol{w}_t^\top \boldsymbol{x}_t$ achieved by the latter algorithm is always equal to the margin value of the algorithm in Figure 3.1 multiplied by a *positive* quantity depending on all instances observed

[5] In this case, the weight vector $\boldsymbol{w}_t$ computed when the algorithm is run on a sequence of examples $((\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_{t-1}, y_{t-1}))$ can be defined by

$$\boldsymbol{w}_t = \operatorname*{argmin}_{\boldsymbol{v}} \left(\sum_{s \in \mathcal{M}_{t-1}} \left(\boldsymbol{v}^\top \boldsymbol{x}_s - y_s \right)^2 + a \, \|\boldsymbol{v}\|^2 \right),$$

where $\mathcal{M}_{t-1}$ is the set of all trials $s \leq t-1$ such that $y_s \, \boldsymbol{w}_s^\top \boldsymbol{x}_s < 0$, and $\boldsymbol{w}_0 = \mathbf{0}$. This way of writing the update rule shows that this version of the second-order Perceptron algorithm can also be viewed as a sparse variant of an on-line ridge regression algorithm, whose prediction rule is $\widehat{y}_t = \text{SGN}(\boldsymbol{w}_t^\top \boldsymbol{x}_t)$, where the weight vector $\boldsymbol{w}_t$ is just

$$\boldsymbol{w}_t = \operatorname*{argmin}_{\boldsymbol{v}} \left(\sum_{s=1}^{t-1} \left(\boldsymbol{v}^\top \boldsymbol{x}_s - y_s \right)^2 + a \, \|\boldsymbol{v}\|^2 \right).$$

This is actually the approach to binary classification taken in [33], where it is called the regularized least-squares classification (RLSC) method. The reader might also want to check [37] for an approach having a similar flavor. This comparison also stresses the role played by the sparsity: First, our algorithm is more efficient than RLSC, as we have only to store a (potentially small) submatrix of the data correlation matrix—the algorithm becomes significantly more efficient on the "easy" sequences where it makes very few mistakes. Second, the mistake-driven property, which causes sparsity, is a key feature when deriving spectral bounds on the number of mistakes that hold for arbitrary data sequences: no such bounds are currently known for RLSC.

so far. Therefore, unlike the linear regression framework considered in [5, 40], for binary classification the inclusion of the current instance makes no difference: running the two algorithms on the same sequence of examples would produce the same sequence of predictions. This is true even for the second-order Perceptron algorithm with pseudoinverse described in Figure 4.2 (section 4.2).

Hence, the algorithms in Figures 3.1 and 4.2 can be equivalently viewed as generators of linear-threshold classifiers. The reason we did not formulate our algorithms directly in this way is threefold. First, the above equivalence does not hold in general when parameter a changes with time, as in Figure 4.1 (section 4.1). Thus, in order to maintain a uniform style of exposition, we preferred to keep the difference between Figures 3.1, 4.2, and 4.1 as little as possible. Second, including the current instance for prediction seems to be naturally suggested by the way we analyzed the algorithms. Third, in scenarios where the margin plays a crucial role, such as the information filtering problems studied in [9], including the current instance in the margin computation seems to give a slight improvement in performance.

3.1. Analysis. We now claim the theoretical properties of our algorithm. The following theorem is proved in Appendix A.

THEOREM 3.1. *The number m of mistakes made by the second-order Perceptron algorithm of Figure* 3.1*, run on any finite sequence $\mathcal{S} = ((\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \dots)$ of examples, satisfies*

$$m \;\le\; \inf_{\gamma>0} \min_{\|\boldsymbol{u}\|=1} \left(\frac{D_\gamma(\boldsymbol{u};\mathcal{S})}{\gamma} + \frac{1}{\gamma} \sqrt{(a + \boldsymbol{u}^\top X_m\, X_m^\top \boldsymbol{u}) \sum_{i=1}^{n} \ln\left(1 + \lambda_i/a\right)} \right),$$

where $\lambda_1, \dots, \lambda_n$ are the eigenvalues of $X_m\, X_m^\top$.

Some remarks are in order.

First, observe that the quantity $\boldsymbol{u}^\top X_m\, X_m^\top \boldsymbol{u}$ in the above bound always lies between $\min_i \lambda_i$ and $\max_i \lambda_i$. In particular, $\boldsymbol{u}^\top X_m\, X_m^\top \boldsymbol{u} = \lambda_i$ when $\boldsymbol{u}$ is aligned with the eigenvector associated with λ_i. This fact entails a trade-off between the hinge loss term and the square-root term in the bound.

Second, the larger a gets, the more similar the "warping" matrix $(aI_n + S_t\, S_t^\top)^{-1}$ becomes to the diagonal matrix $a^{-1} I_n$. In fact, as the reader can see from Figure 3.1, in the limit as $a \to \infty$ the second-order Perceptron algorithm becomes the classical Perceptron algorithm, and the bound in Theorem 3.1 takes the form

$$m \le \inf_{\gamma>0} \min_{\|\boldsymbol{u}\|=1} \left(\frac{D_\gamma(\boldsymbol{u};\mathcal{S})}{\gamma} + \frac{1}{\gamma} \sqrt{\sum_{i=1}^{n} \lambda_i} \right).$$

Now, since the trace of a matrix equals the sum of its eigenvalues and the nonzero eigenvalues of $X_m\, X_m^\top$ coincide with the nonzero eigenvalues of $X_m^\top\, X_m$, we can immediately see that $\sum_{i=1}^{n} \lambda_i = \mathrm{trace}(X_m^\top\, X_m) = \sum_{t\in\mathcal{M}} \|\boldsymbol{x}_t\|^2 \le m \left(\max_{t\in\mathcal{M}} \|\boldsymbol{x}_t\|^2\right)$, where $\mathcal{M} \subseteq \{1, 2, \dots\}$ is the set of indices of mistaken trials. Thus, setting $R^2 = \max_{t\in\mathcal{M}} \|\boldsymbol{x}_t\|^2$, we can solve the resulting bound for m. This gives

$$m \le \inf_{\gamma>0} \min_{\|\boldsymbol{u}\|=1} \left(\frac{R^2}{2\,\gamma^2} + \frac{D_\gamma(\boldsymbol{u};\mathcal{S})}{\gamma} + \frac{R}{\gamma} \sqrt{\frac{D_\gamma(\boldsymbol{u};\mathcal{S})}{\gamma} + \frac{R^2}{4\,\gamma^2}} \right),$$

which is the Perceptron bound in the general nonseparable case [18]. This shows that, in a certain sense, the second-order Perceptron algorithm strictly generalizes the

classical Perceptron algorithm by introducing an additional parameter a. In general, the larger a becomes, the more the algorithm in Figure 3.1 resembles the Perceptron algorithm.

Third, in the linearly separable case the mistake bound for the Perceptron algorithm is $(R/\gamma)^2$. This bound is determined by the aforementioned trace inequality $\sum_{i=1}^n \lambda_i \le R^2\, m$. The second-order algorithm, on the other hand, has the bound $\sqrt{(a + \boldsymbol{u}^\top X_m\, X_m^\top \boldsymbol{u}) \sum_{i=1}^n \ln(1+\lambda_i/a)}\,/\,\gamma$, which is determined by the core spectral quantity $\left(a + \boldsymbol{u}^\top X_m\, X_m^\top \boldsymbol{u}\right) \sum_{i=1}^n \ln(1+\lambda_i/a)$. A quick comparison of the two bounds suggests that data sequences that are linearly separable by hyperplanes whose normal vectors are nearly aligned with eigenvectors having small eigenvalues should be advantageous for the second-order Perceptron algorithm (see Figure 2.2).

A more precise quantitative comparison between the two bounds might go as follows. We first note that in order to carry out this comparison, we need to somehow renounce the second-order dependence on the eigenvalues λ_i. Introduce the notation $\lambda_{\boldsymbol{u}} = \boldsymbol{u}^\top X_m\, X_m^\top \boldsymbol{u}$. We have

$$\begin{aligned}(a+\lambda_{\boldsymbol{u}}) \sum_{i=1}^n \ln(1+\lambda_i/a) &\le \max_{\lambda_1,\ldots,\lambda_n\,:\,\sum_{i=1}^n \lambda_i \le R^2 m} (a+\lambda_{\boldsymbol{u}}) \sum_{i=1}^n \ln(1+\lambda_i/a) \\ &= (a+\lambda_{\boldsymbol{u}})\; n\, \ln\left(1+\frac{R^2 m}{n\, a}\right),\end{aligned}$$

since the maximum is achieved when all λ_j are equal to $R^2 m/n$. Now, finding conditions on the data for which the second-order algorithm is advantageous is reduced to finding conditions on a, $\lambda_{\boldsymbol{u}}$, and $r = R^2 m/n$ such that

$$(a+\lambda_{\boldsymbol{u}})\, \ln\left(1+\frac{r}{a}\right) \le r \tag{3.1}$$

is satisfied. The next lemma, proved in Appendix B through a simple derivative argument, shows that if $\lambda_{\boldsymbol{u}} < r/2$, then $a = r\lambda_{\boldsymbol{u}}/(r-2\lambda_{\boldsymbol{u}})$ makes (3.1) a strict inequality. With this setting of a, the smaller $\lambda_{\boldsymbol{u}}$ is when compared to $r/2$, the smaller the left-hand side of (3.1) is when compared to the right-hand side, that is, the smaller the second-order bound is when compared to the first-order one. Thus, as we expected, if the data tend to be "flat" and $\boldsymbol{u}$ has irrelevant components along the direction of large instance vectors (as in Figure 2.2), then it is convenient to pick a small value of a. In particular, when $\lambda_{\boldsymbol{u}}$ is "small" the above setting of a becomes $a \simeq \lambda_{\boldsymbol{u}}$ (independent of r). On the other hand, the lemma also shows that whenever $\lambda_{\boldsymbol{u}} \ge r/2$, then (3.1) is satisfied (as an equality) only for $a \to \infty$. Thus when $\lambda_{\boldsymbol{u}}$ is "not small" the best thing to do is to resort to the first-order algorithm.

LEMMA 3.2. *Let $f(a,\lambda,r) = (a+\lambda)\ln\left(1+\frac{r}{a}\right)$, where $a,\lambda,r > 0$. Then the following hold:*

1. *$\lim_{a\to\infty} f(a,\lambda,r) = r$; moreover, if λ and r are such that $\lambda \ge r/2$, then $f(a,\lambda,r) \ge r$ for all $a > 0$.*
2. *If λ and r are such that $\lambda < r/2$, then setting $a = a(\lambda,r) = \frac{r\lambda}{r-2\lambda}$ gets $f(a,\lambda,r) < r$ and $\lim_{2\lambda/r\to 0} f(a,\lambda,r)/r = 0$.*

We finally observe that the running time per trial of the second-order algorithm is $\Theta(n^2)$, since by the well-known Sherman–Morrison formula (see, e.g., [24, Chap. 0]) if $\boldsymbol{x}$ is a vector and A is a positive definite matrix, then so is $B = A + \boldsymbol{x}\boldsymbol{x}^\top$ and

$$B^{-1} = A^{-1} - \frac{\left(A^{-1}\boldsymbol{x}\right)\left(A^{-1}\boldsymbol{x}\right)^\top}{1+\boldsymbol{x}^\top A^{-1}\boldsymbol{x}}. \tag{3.2}$$

This formula allows us to perform matrix inversion incrementally. In particular, since the $n \times n$ positive definite matrix $aI_n + S_t\, S_t^\top$ equals $aI_n + X_{k-1}\, X_{k-1}^\top + \boldsymbol{x}_t \boldsymbol{x}_t^\top$, one can compute in time $\Theta(n^2)$ the n-dimensional vector $\left(aI_n + X_{k-1}\, X_{k-1}^\top\right)^{-1} \boldsymbol{x}_t$ and use it along with the Sherman–Morrison formula to obtain $\left(aI_n + S_t\, S_t^\top\right)^{-1}$, again in time $\Theta(n^2)$.

3.2. The algorithm in dual variables and the use of kernel functions. In this section we show that the second-order Perceptron algorithm can be equivalently formulated in dual variables. This formulation allows us to run the algorithm efficiently in any given reproducing kernel Hilbert space. As a consequence, we are able to derive a kernel version of Theorem 3.1 where the mistake bound is expressed in terms of the eigenvalues of the kernel Gram matrix.

We recall that a *kernel* function (see, e.g., [12, 36, 38]) is a nonnegative function $K : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ satisfying

$$\sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i\, \alpha_j\, K(\boldsymbol{x}_i, \boldsymbol{x}_j) \geq 0$$

for all $\alpha_1, \ldots, \alpha_m \in \mathbb{R}$, $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m \in \mathbb{R}^n$, and $m \in \mathbb{N}$ (such functions are also called positive definite). Given a kernel K, we can define the linear space

$$\mathcal{V}_K = \left\{ f(\cdot) = \sum_{i=1}^{m} \alpha_i\, K(\boldsymbol{x}_i, \cdot) \,:\, \alpha_i \in \mathbb{R},\, \boldsymbol{x}_i \in \mathbb{R}^n,\, i = 1, \ldots, m,\, m \in \mathbb{N} \right\},$$

with norm defined by

$$\|f\|_K = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i\, \alpha_j\, K(\boldsymbol{x}_i, \boldsymbol{x}_j)}\ .$$

If this space is completed by adding all limit points of sequences $f_1, f_2, \ldots \in \mathcal{V}_K$ that are convergent in the norm $\|f\|_K$, the resulting space, denoted by $\mathcal{H}_K$, is called the *reproducing kernel Hilbert space* (induced by the kernel K). Classical examples of kernel functions include the so-called polynomial kernel $K(\boldsymbol{x}, \boldsymbol{y}) = (1 + \boldsymbol{x}^\top \boldsymbol{y})^d$, where d is a positive integer, and the Gaussian kernel $K(\boldsymbol{x}, \boldsymbol{y}) = \exp(-\|\boldsymbol{x} - \boldsymbol{y}\|^2 / 2\sigma^2)$, $\sigma > 0$.

In practice, any algorithm depending on the instance vectors $\boldsymbol{x}_i$ only through inner products $\boldsymbol{x}_i^\top \boldsymbol{x}_j$ can be turned into a more general kernel version just by replacing the standard inner products $\boldsymbol{x}_i^\top \boldsymbol{x}_j$ throughout by the kernel inner products $K(\boldsymbol{x}_i, \boldsymbol{x}_j)$.

The following theorem is a slight modification of the dual formulation of the ridge regression algorithm derived in [35].

THEOREM 3.3. *With the notation of Figure* 3.1*, let* $\widetilde{\boldsymbol{y}}_t$ *be the* k*-component vector whose first* $k-1$ *components are the labels* y_i *where the algorithm has made a mistake up to trial* $t-1$ *and whose last component is* 0*. Then, for all* $\boldsymbol{x}_t \in \mathbb{R}^n$*, we have*

$$\boldsymbol{v}_{k-1}^\top \left(aI_n + S_t\, S_t^\top\right)^{-1} \boldsymbol{x}_t = \widetilde{\boldsymbol{y}}_t^\top \left(aI_k + G_t\right)^{-1} \left(S_t^\top\, \boldsymbol{x}_t\right),$$

where $G_t = S_t^\top S_t$ *is a* $k \times k$ *(Gram) matrix and* I_k *is the* k*-dimensional identity matrix.*

Proof. Recalling Figure 3.1, we have $\boldsymbol{v}_{k-1} = S_t\, \widetilde{\boldsymbol{y}}_t$. This implies

$$\boldsymbol{v}_{k-1}^\top \left(aI_n + S_t\, S_t^\top\right)^{-1} = \widetilde{\boldsymbol{y}}_t^\top\, S_t^\top \left(aI_n + S_t\, S_t^\top\right)^{-1}.$$

Thus we need only prove that

$$S_t^\top \left(aI_n + S_t\, S_t^\top\right)^{-1} = \left(aI_k + G_t\right)^{-1} S_t^\top$$

holds. But this follows from, e.g., part (d) of Exercise 17 in [6, Chap. 1]. □

From a computational standpoint, we remark that the update rule of this algorithm can exploit a known adjustment formula for partitioned matrices (see, e.g., [24, Chap. 0]). This formula relates the inverse of a matrix to the inverses of some of its submatrices. In our simple case, given a $(k-1) \times (k-1)$ positive definite matrix A, a $(k-1)$-dimensional column vector $\boldsymbol{b}$, and a scalar c, we have

$$\begin{bmatrix} A & \boldsymbol{b} \\ \boldsymbol{b}^\top & c \end{bmatrix}^{-1} = \begin{bmatrix} A^{-1} + \boldsymbol{v}\boldsymbol{v}^\top/d & -\boldsymbol{v}/d \\ -\boldsymbol{v}^\top/d & 1/d \end{bmatrix},$$

where $\boldsymbol{v} = A^{-1}\boldsymbol{b}$ and[6] $d = c - \boldsymbol{b}^\top \boldsymbol{v}$ can be computed with $\Theta(k^2)$ multiplications. Using the notation of Theorem 3.3 it is easy to see that

$$aI_k + G_t = \begin{bmatrix} aI_{k-1} + X_{k-1}^\top X_{k-1} & X_{k-1}^\top \boldsymbol{x}_t \\ \boldsymbol{x}_t^\top X_{k-1} & a + \boldsymbol{x}_t^\top \boldsymbol{x}_t \end{bmatrix}.$$

Thus, using the above formula for the inverse of partitioned matrices, it follows that the k-dimensional matrix $\left(aI_k + G_t\right)^{-1}$ can be computed incrementally from the $(k-1)$-dimensional matrix $\left(aI_{k-1} + X_{k-1}^\top X_{k-1}\right)^{-1}$ with only $\Theta(k^2)$ extra dot products. Also, sweeping through a sequence of T examples needs only $\Theta(m^2\, T)$ dot products, where m is upper bounded as in Theorem 3.1.

The following result is a kernel version of Theorem 3.1. The hinge loss of any function $f \in \mathcal{H}_K$ is defined by $D_\gamma(f; (\boldsymbol{x}, y)) = \max\{0, \gamma - y\, f(\boldsymbol{x})\}$.

COROLLARY 3.4. *The number m of mistakes made by the dual second-order Perceptron algorithm with kernel K, run on any finite sequence $\mathcal{S} = ((\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \dots)$ of examples, satisfies*

$$m \;\le\; \inf_{\gamma > 0} \min_{\|f\|_K = 1} \left(\frac{D_\gamma(f; \mathcal{S})}{\gamma} + \frac{1}{\gamma} \sqrt{\left(a + \sum_{t \in \mathcal{M}} f(\boldsymbol{x}_t)^2\right) \sum_i \ln\left(1 + \lambda_i / a\right)} \right).$$

The numbers λ_i are the nonzero eigenvalues of the kernel Gram matrix with entries $K(\boldsymbol{x}_i, \boldsymbol{x}_j)$, where $i, j \in \mathcal{M}$ and $\mathcal{M}$ is the set of indices of mistaken trials.

Considerations similar to those made after the statement of Theorem 3.1 apply here as well. Note that the number of nonzero eigenvalues λ_i of the kernel Gram matrix is, in general, equal to m, since the very nature of kernel functions makes the dimension of the space $\mathcal{H}_K$ very large, possibly infinite (hence the kernel Gram matrix is likely to be full-rank). Note also that if a linear kernel $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{x}_i^\top \boldsymbol{x}_j$ is used in Corollary 3.4, then the mistake bound of Theorem 3.1 is recovered exactly. To see this observe that $\lambda_{\boldsymbol{u}} = \boldsymbol{u}^\top X_m\, X_m^\top \boldsymbol{u} = \sum_{t \in \mathcal{M}} (\boldsymbol{u}^\top \boldsymbol{x}_t)^2$ and also that the nonzero eigenvalues of the matrix $X_m\, X_m^\top$ coincide with the nonzero eigenvalues of the Gram matrix $X_m^\top X_m$.

[6] The quantity d is called the Schur complement of the augmented matrix $\left[\begin{smallmatrix} A & \boldsymbol{b} \\ \boldsymbol{b}^\top & c \end{smallmatrix}\right]$ with respect to matrix A. From the positive definiteness of both A and $\left[\begin{smallmatrix} A & \boldsymbol{b} \\ \boldsymbol{b}^\top & c \end{smallmatrix}\right]$ it follows that $d > 0$ (see, e.g., [24, Chap. 7]).

4. Getting rid of parameter a. A major drawback of the algorithm described in Figure 3.1 is that its input parameter a is fixed ahead of time. It turns out that in any practical application the value of this parameter significantly affects performance. For instance, a bad choice of a might make the second-order Perceptron algorithm similar to the first-order one, even in cases when the former should perform far better.

In this section we analyze two variants of the basic algorithm: The first variant (section 4.1) is an adaptive parameter version of the algorithm in Figure 3.1; the second variant (section 4.2) eliminates the need for the trade-off parameter a by replacing the "warping" matrix $(aI_n + S_t\,S_t^\top)^{-1}$ with the matrix $(S_t\,S_t^\top)^+$, i.e., the *pseudoinverse* of $S_t\,S_t^\top$.

4.1. Second-order Perceptron with adaptive parameter. In this section we refine the arguments of section 3.1 by motivating and analyzing an adaptive parameter version of the algorithm in Figure 3.1. Ideally, we would like the resulting algorithm to be able to learn on the fly the "best" a for the given data sequence, such as the value of a that minimizes the bound in Theorem 3.1. The optimal a clearly depends on unknown quantities, like the spectral structure of data and the unknown target $\boldsymbol{u}$. This ideal choice of a would be able to automatically turn the second-order algorithm into a first-order one when the actual scattering of data has, say, spherical symmetry. This is a typical eigenstructure of data which a second-order algorithm cannot take advantage of.

To make our argument, we first note that the value of a in Theorem 3.1 affects the bound only through the quantity

$$(a + \lambda_{\boldsymbol{u}}) \sum_{i=1}^{n} \ln\left(1 + \lambda_i/a\right). \tag{4.1}$$

In turn, both $\lambda_{\boldsymbol{u}}$ and the λ_i's depend only on the examples where the algorithm has made a mistake. Therefore it seems reasonable to let a change only in mistaken trials (this keeps the algorithm mistake driven).

Our second-order algorithm with adaptive parameter is described in Figure 4.1. The algorithm is the same as the one in Figure 3.1, except that we now feed the algorithm with an *increasing* sequence of parameter values $\{a_k\}_{k=1,2,\ldots}$, indexed by the current number of mistakes k. The algorithm is analyzed in Theorem 4.1 below. From the proof of that theorem (given in Appendix C) the reader can see that any strictly increasing sequence $\{a_k\}$ results in a bound on the number of mistakes. In Theorem 4.1 we actually picked a sequence which grows *linearly* with k. This choice seems best according to the following simple upper bounding argument. Consider again (4.1). As we noted in section 3.1,

$$(4.1) \le (a + \lambda_{\boldsymbol{u}})\, n\, \ln\left(1 + \frac{R^2 m}{n\, a}\right), \tag{4.2}$$

where $R = \max_{t\in\mathcal{M}} \|\boldsymbol{x}_t\|$. Now, from the Cauchy–Schwarz inequality one gets $\lambda_{\boldsymbol{u}} = \sum_{t\in\mathcal{M}} (\boldsymbol{u}^\top \boldsymbol{x}_t)^2 \le R^2 m$. On the other hand, if the data sequence is linearly separable with margin $\gamma > 0$, then $\lambda_{\boldsymbol{u}} = \sum_{t\in\mathcal{M}} (\boldsymbol{u}^\top \boldsymbol{x}_t)^2 \ge \gamma^2 m$. Hence, in many interesting cases, $\lambda_{\boldsymbol{u}}$ is linear in m. A further glance at (4.2) allows us to conclude that, viewed as a function of m, the right-hand side of (4.2) cannot grow less than linearly. Moreover, this minimal growth speed is achieved when a is a linear[7] function of m.

[7] The reader will note that if a grows faster than linearly, then (4.2) is still linear in m. However, the resulting (multiplicative) constants in (4.2) would be larger.

Parameter sequence: $\{a_k\}_{k=1,2,\ldots}$, $a_{k+1} > a_k > 0$, $k = 1, 2, \ldots$.
Initialization: $X_0 = \emptyset$; $\boldsymbol{v}_0 = \mathbf{0}$; $k = 1$.
Repeat for $t = 1, 2, \ldots$:

1. get instance $\boldsymbol{x}_t \in \mathbb{R}^n$;
2. set $S_t = [\, X_{k-1}\ \boldsymbol{x}_t \,]$;
3. predict $\widehat{y}_t = \textsc{sgn}(\boldsymbol{w}_t^\top \boldsymbol{x}_t) \in \{-1, +1\}$, where $\boldsymbol{w}_t = \left(a_k I_n + S_t\, S_t^\top\right)^{-1} \boldsymbol{v}_{k-1}$;
4. get label $y_t \in \{-1, +1\}$;
5. if $\widehat{y}_t \neq y_t$, then

$$\begin{aligned} \boldsymbol{v}_k &= \boldsymbol{v}_{k-1} + y_t\, \boldsymbol{x}_t, \\ X_k &= S_t, \\ k &\leftarrow k + 1. \end{aligned}$$

FIG. 4.1. *The second-order Perceptron algorithm with increasing parameter sequence* $\{a_k\}_{k=1,2,\ldots}$.

It is important to point out that this qualitative argument does not depend on the number of nonzero eigenvalues of matrix $X_m\, X_m^\top$. For example, our conclusions would not change even if all instances $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots$ lived in a small subspace of $\mathbb{R}^n$ or, alternatively, if we mapped the same instances into a high-dimensional feature space via a kernel function (see also the discussion after Corollary 3.4).

Theorem 4.1 below uses[8] $a_k = c\, R^2 k$, where c is a small positive constant. This tuning captures the "right" order of growth of a_k, up to a multiplicative constant c, whose best choice depends again on unknown quantities.

THEOREM 4.1. *If the second-order Perceptron algorithm of Figure* 4.1 *is run with parameter sequence* $a_k = c\, R^2 k$, *where* $c > 0$, *on any finite sequence* $\mathcal{S} = ((\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots)$ *of examples such that* $\|\boldsymbol{x}_t\| \leq R$, *then the total number* m *of mistakes satisfies*

$$m \leq \inf_{\gamma > 0} \min_{\|\boldsymbol{u}\| = 1} \left[\frac{D_\gamma(\boldsymbol{u}; \mathcal{S})}{\gamma} + \frac{1}{\gamma} \sqrt{(a_m + \lambda_{\boldsymbol{u}}) \left(B(c, m) + \sum_{i=1}^{n} \ln\left(1 + \frac{\lambda_i}{a_m}\right)\right)} \right], \tag{4.3}$$

where $\lambda_1, \ldots, \lambda_n$ *are the eigenvalues of* $X_m\, X_m^\top$, $a_m = c\, R^2 m$, *and*

$$B(c, m) = \frac{1}{c}\, \ln \frac{m + 1/c}{1 + 1/c}.$$

To see why the algorithm with adaptive parameter might be advantageous over the one with fixed parameter, recall the tuning argument around (3.1). There, it was shown that the second-order algorithm is able to exploit the spectral properties of data when $\lambda_{\boldsymbol{u}}$ is "small." In such a case, a good tuning of a is $a = r\lambda_{\boldsymbol{u}}/(r - 2\lambda_{\boldsymbol{u}})$, which, for small values of $\lambda_{\boldsymbol{u}}$, becomes $a \simeq \lambda_{\boldsymbol{u}}$. Now, for the sake of concreteness, let us focus on the linearly separable case. We have already observed that in this case $\lambda_{\boldsymbol{u}} = c' R^2 m$, where $\gamma^2/R^2 \leq c' \leq 1$ and $\gamma \in (0, R]$ is the minimal margin on the data. We emphasize that the tuning $a \simeq \lambda_{\boldsymbol{u}} = c'\, R^2 m$ matches the one mentioned in Theorem 4.1 up to a scaling factor. In particular, after comparing the bounds in

[8] The presence of the scaling factor R^2 simplifies the analysis in Appendix C.

Theorems 3.1 and 4.1, we see that Theorem 4.1 replaces a with $a_m = c\,R^2 m$, i.e., with a function having a linear dependence on m. The price we pay for this substitution is a further additive term $B(c, m)$ which has a mild (logarithmic) dependence on m. The resulting bound has an inconvenient implicit form. An upper bound on an explicit solution can be clearly computed, but the calculations get overly complicated and do not add any new insights into the heart of the matter. Therefore we feel justified in omitting any further analytical detail.

As in section 3.2, one can derive a dual formulation of the algorithm in Figure 4.1 and prove a result similar to the one contained in Corollary 3.4.

In general, the total number of mistakes made by the algorithm conveys relevant information about the specific dataset at hand. Using a parameter that scales with this number allows one to partially exploit this information. Note, for instance, that if the second-order algorithm of Figure 4.1 is making "many" mistakes (and c is not too small), then the algorithm tends to behave as a first-order algorithm, since a_k is growing "fast." In other words, we might view the algorithm as being able to detect that its second-order structure is not suitable to the data it is processing, thereby trying to turn itself into a first-order algorithm. On the other hand, if the algorithm is making only a few mistakes, then a_k tends to remain small, meaning that the current second-order structure of the algorithm appears to be the right one for the dataset at hand. This approach to parameter tuning is similar to the *self-confident* tuning adopted in [3].

We finally note that, unlike the algorithm with fixed a, here incremental matrix inversion looks a bit more troublesome. In fact, making a change from trial to trial results in a matrix update which is no longer a low-rank adjustment. Therefore the algorithm in Figure 4.1 (as well as its dual version) does not seem to have an update rule requiring only a quadratic number of operations per trial.

4.2. Second-order Perceptron with pseudoinverse. A more radical way of dealing with the trade-off parameter a is to set it to zero and replace the inverse of $aI_n + S_t S_t^\top$ with the pseudoinverse $(S_t S_t^\top)^+$. This is done in the algorithm of Figure 4.2. The matrix $(S_t S_t^\top)^+$ does always exist and coincides with $(S_t S_t^\top)^{-1}$ in the case when $S_t S_t^\top$ is nonsingular. In Appendix D we collected some relevant information about pseudoinverses and their connection to the SVD. A classical reference in which to learn about them is [6].

The following result is proved in Appendix E.

THEOREM 4.2. *If the second-order Perceptron algorithm of Figure* 4.2 *is run on any finite sequence* $\mathcal{S} = ((\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \dots)$ *of examples such that* $\|\boldsymbol{x}_t\| \le R$, *then the total number* m *of mistakes satisfies*

(4.4)
$$m \le \inf_{\gamma > 0} \min_{\|\boldsymbol{u}\|=1} \left[\frac{D_\gamma(\boldsymbol{u}; \mathcal{S})}{\gamma} + \frac{1}{\gamma} \sqrt{\lambda_{\boldsymbol{u}} \left(r + \frac{1}{2} r(r+1) \ln \left(1 + \frac{2\,R^2}{\lambda^*} \frac{m}{r(r+1)} \right) \right)} \right],$$

where $r = \operatorname{rank}(X_m\, X_m^\top) \le n$ *and* λ^* *is the minimum among the* smallest *positive eigenvalues of the matrices* $X_k\, X_k^\top$, $k = 1, \dots, m$, *produced by the algorithm during its run.*

Inequality (4.4) depends on a lower bound on the positive eigenvalues of the correlation matrices produced by the algorithm. In a sense, this dependence substitutes the dependence on a in the bound of Theorem 3.1. In fact, adding the matrix aI_n to

Initialization: $X_0 = \emptyset$; $\boldsymbol{v}_0 = \mathbf{0}$; $k = 1$.
Repeat for $t = 1, 2, \ldots$:

1. get instance $\boldsymbol{x}_t \in \mathbb{R}^n$;
2. set $S_t = [\, X_{k-1} \; \boldsymbol{x}_t \,]$;
3. predict $\widehat{y}_t = \text{SGN}(\boldsymbol{w}_t^\top \boldsymbol{x}_t) \in \{-1, +1\}$, where $\boldsymbol{w}_t = \left(S_t \, S_t^\top\right)^+ \boldsymbol{v}_{k-1}$;
4. get label $y_t \in \{-1, +1\}$;
5. if $\widehat{y}_t \neq y_t$, then:

$$\begin{aligned} \boldsymbol{v}_k &= \boldsymbol{v}_{k-1} + y_t \, \boldsymbol{x}_t, \\ X_k &= S_t, \\ k &\leftarrow k + 1. \end{aligned}$$

FIG. 4.2. *The second-order Perceptron algorithm with pseudoinverse.*

the current correlation matrix of the data might be viewed as a way of setting a lower bound on the positive eigenvalues of the resulting matrix.

As we have observed in section 4.1, when the data are linearly separable with margin $\gamma > 0$ the quantity $\lambda_{\boldsymbol{u}}$ is linear in m. Thus, once we set $\lambda_{\boldsymbol{u}} = c' R^2 m$, with $\gamma^2/R^2 \leq c' \leq 1$, and simplify, bound (4.4) takes the form

$$m \leq \frac{c' R^2}{\gamma^2} \left(r + \frac{1}{2} r(r+1) \, \ln \left(1 + \frac{2\, R^2}{\lambda^*} \frac{m}{r(r+1)} \right) \right). \tag{4.5}$$

As for Theorem 4.1, the two bounds (4.4) and (4.5) are in implicit form with respect to m. The bounds can be made explicit at the cost of adding a few logarithmic terms. Again, we decided not to carry out such calculations since they are not very insightful.

Comparing bounds (4.4) and (4.5) to the one in Theorem 3.1, one can see that the second-order algorithm with pseudoinverse might be advantageous over the basic version when the effective dimension r of the data is relatively small. Also, the bounds (4.4) and (4.5) are nonvacuous only when $r < m$.

Actually, as for the algorithm in Figure 3.1, it is not hard to turn the algorithm in Figure 4.2 into an equivalent dual form. Again, one can exploit known methods to incrementally compute the pseudoinverse (see, e.g., the so-called Greville's method in [6]), reducing the computational cost per trial from cubic to quadratic.

Unfortunately, the bounds (4.4) and (4.5) are generally not useful in the presence of kernel functions[9] since the bounds have a linear[10] dependence on r which cannot be avoided in the worst case. This is due to the simple fact that each time the new instance $\boldsymbol{x}_t$ lies outside the column space of the previous matrix X_{k-1} the pseudo-inverse $(S_t \, S_t^\top)^+$ maps $\boldsymbol{x}_t$ to the orthogonal space of this column space, so that the algorithm has a degenerate margin $\boldsymbol{w}_t^\top \boldsymbol{x}_t = 0$.

5. Simulations on a toy problem. With the purpose of empirically verifying our theoretical results, we ran our second-order Perceptron algorithm on two sets of

[9] In the kernel case, r is likely to be equal to m, giving rise to a vacuous bound.

[10] Note that the quadratic dependence on r is essentially fictitious here since the $r(r+1)$ factor occurs both at the numerator and at the denominator inside the logarithm.

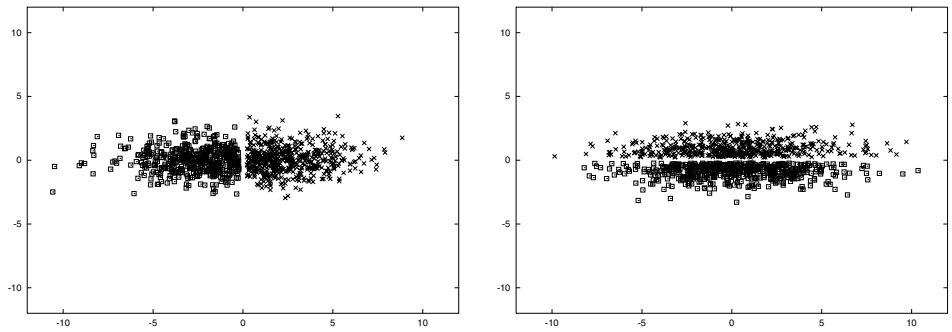

FIG. 5.1. *Projection on the two most relevant coordinates of the* 100*-dimensional datasets used in our simulations.*

TABLE 5.1

Algorithm	Mistakes, 1st dataset	Mistakes, 2nd dataset
Perceptron	30.20 (6.24)	29.80 (8.16)
second-order Perceptron, $a = 1$	9.60 (2.94)	5.60 (2.80)
second-order Perceptron, $a = 10$	10.60 (2.58)	3.20 (1.47)
second-order Perceptron, $a = 50$	14.00 (4.36)	10.40 (6.05)

linearly separable data with 100 attributes. The two datasets are generated by sampling a Gaussian distribution whose correlation matrix has a single dominant eigenvalue, the remaining eigenvalues having the same size (in the sample realizations, the dominant eigenvalue is about eight times bigger than the others). In the first dataset (leftmost plot in Figure 5.1), labels are assigned so that the separating hyperplane is orthogonal to the eigenvector associated with the dominant eigenvalue. In the second dataset (rightmost plot in Figure 5.1), labels are assigned so that the separating hyperplane is orthogonal to the eigenvector associated with the first nondominant eigenvalue in the natural ordering of the coordinates.

According to the remarks following Theorem 3.1, we expect the Perceptron algorithm to perform similarly on both datasets (as the radius of the enclosing ball and the margin do not change between datasets), whereas the second-order Perceptron algorithm is expected to outperform the Perceptron algorithm on the second dataset. This conjecture is supported by the results in Table 5.1. These results were obtained by running the Perceptron algorithm and the second-order Perceptron algorithm (for different values of parameter a) for two epochs on a training set of 9000 examples, and then saving the final classifier generated by each algorithm after its last training mistake. The numbers in the table are the average number of mistakes made by these classifiers on a test set of 3000 examples, where the averages are computed over 5 random permutations of the training set (standard deviations are shown in parentheses). Normalizing the instances did not alter significantly Perceptron's performance.

To offer a familiar context for the reader experienced in empirical comparisons of learning algorithms, the predictive performance in our experiments has been evaluated using the standard test error measure. However, we could have drawn the same conclusions using, instead of the test error, the number of mistakes made by the two algorithms during training. This quantity, which is the one we bound in our theoretical results, is different from the standard training error, since in the on-line model each new mistake is made by a different classifier. A theory accounting for the

relationship between the fraction of mistakes in the on-line model and the test error is developed in [9].

6. Conclusions and open problems. We have introduced a second-order Perceptron algorithm, a new on-line binary classification algorithm for learning linear-threshold functions. The algorithm is able to exploit certain spectral properties of the data sequence, expressed as an interaction between the underlying target vector and the eigenvalues of the correlation matrix of the data.

The second-order Perceptron algorithm retains the standard Perceptron's key properties of sparsity and efficient dual variable representation. This allows us to efficiently run the algorithm in any reproducing kernel Hilbert space. We have proved for this algorithm the best known mistake bound for efficient kernel-based linear-threshold classifiers to date.

Since the performance of our algorithm is critically influenced by an input parameter a, whose optimal setting depends on the whole training set, we have developed two variants of our basic algorithm. The first variant increases the parameter a as a function of the number of mistakes made. The second variant avoids altogether the parameter a by replacing the inverse correlation matrix $(aI + X\,X^\top)^{-1}$ with the pseudoinverse $(X\,X^\top)^+$.

We have also run a simple experiment on synthetic data to give some evidence of the theoretical properties of our algorithm (in its basic form).

Our second-order Perceptron algorithm might be seen as a new on-line classification technique. As such, this technique could be combined with previous techniques, such as the shifting target technique [4, 22] and the approximate on-line large margin technique (see, e.g., [17, 27]).

As shown in [9], our analysis can be used to derive linear-threshold classifiers whose statistical risk can be bounded, in probability, by quantities directly related to the mistake bounds of Theorems 3.1, 4.1, and 4.2. The resulting data-dependent generalization bounds are similar in spirit, though not readily comparable, to the bounds given in [41, Thm. 5.2]. In fact, the results in [41] are derived, via involved covering numbers arguments, in terms of the correlation matrix $X\,X^\top$ of the *whole* sequence of examples. More precisely, these results are expressed in terms of all the "large" eigenvalues of $X\,X^\top$, taken in decreasing order of magnitude up to the "effective number of dimensions."[11] In contrast to that, our bounds are in terms of all the eigenvalues of the submatrix of $X\,X^\top$ made up of instances where the algorithm has made a mistake. In this sense, the sparsity of the solution produced by our algorithm is directly reflected by the magnitude of the eigenvalues of the above submatrix. To see this, go back to the primal variable form of Theorem 3.1 and observe that adding rank-one matrices of type $\boldsymbol{x}\,\boldsymbol{x}^\top$ to a correlation matrix $X\,X^\top$ results in a new correlation matrix whose eigenvalues can only be larger. Hence few mistakes are equivalent to high sparsity, which, in turn, is equivalent to small eigenvalues.

We also observe that the application of mistake bounds to the statistical learning setting is not straightforward for the whitened Perceptron algorithm described in section 2. Since the whitening matrix $M^{-1/2}$ depends on the whole training data, the whitening transformation $\boldsymbol{x} \to M^{-1/2}\boldsymbol{x}$ does not preserve stochastic independence among the (whitened) instance vectors $M^{-1/2}\boldsymbol{x}_1, \ldots, M^{-1/2}\boldsymbol{x}_T$.

There are several directions in which this work can be extended. In the following we briefly mention three of them.

[11] The "effective number of dimensions" depends, for instance, on the margin of the data.

First, it might be possible to refine the analysis of the second-order Perceptron algorithm with adaptive parameter (Theorem 4.1). The linear growth of a_k is certainly a first step toward on-line parameter adaptation, though somewhat unsatisfactory since it leaves the leading coefficient of a_k unspecified. Moreover, we are not aware of any incremental updating scheme for this algorithm (neither in primal nor in dual form). Second, it might be possible to refine the analysis of the second-order Perceptron with the pseudoinverse (Theorem 4.2) so as to eliminate the dependence on λ^*. Third, we would like to combine the second-order classification technology with dual-norm algorithms such as the p-norm algorithms [18, 20] and the Winnow/weighted majority algorithms [28, 29, 30]. This would probably give rise to new attractive algorithms for learning sparse linear-threshold functions.

Appendix A. Proof of Theorem 3.1. Fix an arbitrary finite sequence $\mathcal{S} = ((\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \dots)$, and let $\mathcal{M} \subseteq \{1, 2, \dots\}$ be the set of trials where the algorithm of Figure 3.1 made a mistake. Let $A_0 = aI_n$ and $A_k = aI_n + X_k X_k^\top$. We study the evolution of $\boldsymbol{v}_k^\top A_k^{-1} \boldsymbol{v}_k$ over mistaken trials. Let $t = t_k$ be the trial where the kth mistake occurred. We have

$$\begin{aligned}
\boldsymbol{v}_k^\top A_k^{-1} \boldsymbol{v}_k &= (\boldsymbol{v}_{k-1} + y_t \boldsymbol{x}_t)^\top A_k^{-1} (\boldsymbol{v}_{k-1} + y_t \boldsymbol{x}_t) \\
&\quad (\text{since } \widehat{y}_t \neq y_t \text{ implies } \boldsymbol{v}_k = \boldsymbol{v}_{k-1} + y_t \boldsymbol{x}_t) \\
&= \boldsymbol{v}_{k-1}^\top A_k^{-1} \boldsymbol{v}_{k-1} + 2 y_t \left(A_k^{-1} \boldsymbol{v}_{k-1}\right)^\top \boldsymbol{x}_t + \boldsymbol{x}_t^\top A_k^{-1} \boldsymbol{x}_t \\
&= \boldsymbol{v}_{k-1}^\top A_k^{-1} \boldsymbol{v}_{k-1} + 2 y_t \, \boldsymbol{w}_t^\top \boldsymbol{x}_t + \boldsymbol{x}_t^\top A_k^{-1} \boldsymbol{x}_t \\
&\quad (\text{since } \widehat{y}_t \neq y_t \text{ implies } X_k = S_t \text{ and } A_k^{-1} \boldsymbol{v}_{k-1} = \boldsymbol{w}_t) \\
&\leq \boldsymbol{v}_{k-1}^\top A_k^{-1} \boldsymbol{v}_{k-1} + \boldsymbol{x}_t^\top A_k^{-1} \boldsymbol{x}_t \\
&\quad (\text{since } \widehat{y}_t \neq y_t \text{ implies } y_t \, \boldsymbol{w}_t^\top \boldsymbol{x}_t \leq 0).
\end{aligned}$$

Now, note that A_k can be recursively defined using $A_k = A_{k-1} + \boldsymbol{x}_t \boldsymbol{x}_t^\top$. Hence, applying the Sherman–Morrison formula (3.2), we get

$$\boldsymbol{v}_{k-1}^\top A_k^{-1} \boldsymbol{v}_{k-1} = \boldsymbol{v}_{k-1}^\top A_{k-1}^{-1} \boldsymbol{v}_{k-1} - \frac{\left(\boldsymbol{v}_{k-1}^\top A_{k-1}^{-1} \boldsymbol{x}_t\right)^2}{1 + \boldsymbol{x}_t^\top A_{k-1}^{-1} \boldsymbol{x}_t} \leq \boldsymbol{v}_{k-1}^\top A_{k-1}^{-1} \boldsymbol{v}_{k-1},$$

where the inequality holds since A_{k-1}^{-1} is the inverse of a positive definite matrix (and so A_{k-1}^{-1} is positive definite). Thus we get

$$\boldsymbol{v}_k^\top A_k^{-1} \boldsymbol{v}_k \leq \boldsymbol{v}_{k-1}^\top A_{k-1}^{-1} \boldsymbol{v}_{k-1} + \boldsymbol{x}_t^\top A_k^{-1} \boldsymbol{x}_t,$$

holding for all $k = 1, \dots, m = |\mathcal{M}|$. Summing the last inequality over k (or, equivalently, over $t \in \mathcal{M}$) and using $\boldsymbol{v}_0 = \boldsymbol{0}$ yields

$$\begin{aligned}
\boldsymbol{v}_m^\top A_m^{-1} \boldsymbol{v}_m &\leq \sum_{k=1}^{m} \boldsymbol{x}_t^\top A_k^{-1} \boldsymbol{x}_t \\
&= \sum_{k=1}^{m} \left(1 - \frac{\det(A_{k-1})}{\det(A_k)}\right) \\
&\quad (\text{using Lemma A.1 from [5] or Lemma D.1 in Appendix D}) \\
&\leq \sum_{k=1}^{m} \ln \frac{\det(A_k)}{\det(A_{k-1})} \qquad (\text{since } 1 - x \leq -\ln x \text{ for all } x > 0)
\end{aligned}$$

$$
\begin{aligned}
&= \ln \frac{\det(A_m)}{\det(A_0)} \\
&= \ln \frac{\det(aI_n + X_m X_m^\top)}{\det(aI_n)} \\
&= \sum_{i=1}^{n} \ln\left(1 + \frac{\lambda_i}{a}\right), \qquad \text{(A.1)}
\end{aligned}
$$

where $\lambda_1, \ldots, \lambda_n$ are the eigenvalues of $X_m X_m^\top$.

To conclude the proof, note that $A_m^{1/2}$ does exist since A_m is positive definite (see, e.g., [31, Chap. 3]). Pick any unit norm vector $\boldsymbol{u} \in \mathbb{R}^n$ and let $\boldsymbol{z} = A_m^{1/2}\boldsymbol{u}$. Then, using the Cauchy–Schwarz inequality, we have

$$
\begin{aligned}
\sqrt{\boldsymbol{v}_m^\top A_m^{-1} \boldsymbol{v}_m} &= \left\| A_m^{-1/2} \boldsymbol{v}_m \right\| \\
&\geq \frac{\left(A_m^{-1/2}\boldsymbol{v}_m\right)^\top \boldsymbol{z}}{\|\boldsymbol{z}\|} \\
&= \frac{\boldsymbol{v}_m^\top \boldsymbol{u}}{\sqrt{\boldsymbol{u}^\top A_m \boldsymbol{u}}} \\
&= \frac{\boldsymbol{v}_m^\top \boldsymbol{u}}{\sqrt{a + \boldsymbol{u}^\top X_m X_m^\top \boldsymbol{u}}} \\
&\geq \frac{\gamma\, m - D_\gamma(\boldsymbol{u}; \mathcal{S})}{\sqrt{a + \boldsymbol{u}^\top X_m X_m^\top \boldsymbol{u}}}, \qquad \text{(A.2)}
\end{aligned}
$$

where the last inequality follows from the very definition of $D_\gamma(\boldsymbol{u}; \mathcal{S})$ and holds for any $\gamma > 0$. Putting together (A.1) and (A.2) and solving for m gives the statement of the theorem.

Appendix B. Proof of Lemma 3.2. We first check the limiting behavior of f as a function of a: we have $f(a, \lambda, r) \to \infty$ as $a \to 0$ and $f(a, \lambda, r) \to r$ as $a \to \infty$ for any fixed $\lambda, r > 0$. To prove the remainder of part 1, note that since $\partial^2 f / \partial a^2$ has the same sign as

$$
2\,a\,\lambda + r\,\lambda - a\,r, \qquad \text{(B.1)}
$$

it follows that when $\lambda \geq r/2$, no choice of $a > 0$ exists, which makes (B.1) negative. In this case, $f(a, \lambda, r)$, viewed as a function of a, is convex and decreasing. Hence its minimal value r is achieved only asymptotically ($a \to \infty$).

When $\lambda < r/2$ a finite value of a exists which minimizes f. However, computing the minimizing a analytically does not seem to be easy. Therefore we resort to the approximation $a = \frac{r\lambda}{r - 2\lambda}$, which is actually the value of a where (B.1) vanishes, i.e., where f changes concavity. To prove part 2 we set $\beta = r/(2\lambda) > 1$ and $g(\beta) = \frac{2\beta - 1}{2\beta(\beta - 1)} \ln(2\beta - 1) = f(\frac{r\lambda}{r - 2\lambda}, \lambda, r)/r$. It thus suffices to show that $g(\beta) < 1$ for all $\beta > 1$ and $\lim_{\beta \to \infty} g(\beta) = 0$. The first statement can be proved by showing that $\lim_{\beta \to 1} g(\beta) = 1$ and that for any $\beta > 1$ the first derivative of $\ln(2\beta - 1)$ is always smaller than the first derivative of $\frac{2\beta(\beta - 1)}{2\beta - 1}$. We omit the details of these easy calculations. The second statement trivially follows from $g(\beta) = O(\ln \beta / \beta)$, as $\beta \to \infty$.

Appendix C. Proof of Theorem 4.1. The proof is a more refined version of the proof of Theorem 3.1 and proceeds along the same lines.

Again, we assume that the kth mistake occurs when processing example $(\boldsymbol{x}_t, y_t)$, $t = t_k$, and we let $\mathcal{M} \subseteq \{1, 2, \dots\}$ be the set of trials where a mistake occurred. Let $A_k = a_k I_n + X_k X_k^\top$, $k = 1, 2, \dots$, and $A_0 = a_1 I_n$. We have

$$A_k = A_{k-1} + (a_k - a_{k-1}) I_n + \boldsymbol{x}_t \boldsymbol{x}_t^\top, \quad k = 1, 2, \dots, \tag{C.1}$$

where $a_0 = a_1$. We study the evolution of the quantity $\boldsymbol{v}_k^\top A_k^{-1} \boldsymbol{v}_k$ over mistaken trials. As in the proof of Theorem 3.1, we have

$$\boldsymbol{v}_k^\top A_k^{-1} \boldsymbol{v}_k \le \boldsymbol{v}_{k-1}^\top A_k^{-1} \boldsymbol{v}_{k-1} + \boldsymbol{x}_t^\top A_k^{-1} \boldsymbol{x}_t. \tag{C.2}$$

We need to bound $\boldsymbol{v}_{k-1}^\top A_k^{-1} \boldsymbol{v}_{k-1}$ from above. From (C.1) we see that when $k \ge 2$ the matrix A_k is the sum of the nonsingular matrices A_{k-1} and $(a_k - a_{k-1}) I_n + \boldsymbol{x}_t \boldsymbol{x}_t^\top$ (the latter is nonsingular since $a_k > a_{k-1}$). Thus, when $k \ge 2$, computing A_k^{-1} can be done through the general inversion formula

$$(B + C)^{-1} = B^{-1} - B^{-1}(B^{-1} + C^{-1})^{-1} B^{-1}$$

with $B = A_{k-1}$ and $C = (a_k - a_{k-1}) I_n + \boldsymbol{x}_t \boldsymbol{x}_t^\top$. Now, since both B and C are positive definite, so is the matrix $B^{-1}(B^{-1} + C^{-1})^{-1} B^{-1}$ (this easily follows from the fact that the inverse of a positive definite matrix is again positive definite and that both the sum and the product of two positive definite matrices give rise to positive definite matrices). Hence if $k \ge 2$, we can write

$$\begin{aligned}
\boldsymbol{v}_{k-1}^\top A_k^{-1} \boldsymbol{v}_{k-1} &= \boldsymbol{v}_{k-1}^\top (B + C)^{-1} \boldsymbol{v}_{k-1} \\
&= \boldsymbol{v}_{k-1}^\top \left(B^{-1} - B^{-1}(B^{-1} + C^{-1})^{-1} B^{-1} \right) \boldsymbol{v}_{k-1} \\
&\le \boldsymbol{v}_{k-1}^\top B^{-1} \boldsymbol{v}_{k-1} \\
&= \boldsymbol{v}_{k-1}^\top A_{k-1}^{-1} \boldsymbol{v}_{k-1}. \qquad \text{(C.3)}
\end{aligned}$$

On the other hand, when $k = 1$, inequality (C.3) is trivially true since $\boldsymbol{v}_0 = \boldsymbol{0}$. Thus the inequality holds for all $k \ge 1$. We plug (C.3) back into (C.2), sum over $k = 1, \dots, m = |\mathcal{M}|$, and take into account that $\boldsymbol{v}_0 = \boldsymbol{0}$. We obtain

$$\begin{aligned}
\boldsymbol{v}_m^\top A_m^{-1} \boldsymbol{v}_m &\le \sum_{k=1}^{m} \boldsymbol{x}_t^\top A_k^{-1} \boldsymbol{x}_t \\
&= \sum_{k=1}^{m} \left(1 - \frac{\det(A_{k-1} + (a_k - a_{k-1}) I_n)}{\det(A_k)} \right) \\
&\quad \text{(applying Lemma A.1 in [5] or Lemma D.1 in Appendix D to (C.1))} \\
&\le \sum_{k=1}^{m} \ln \frac{\det(A_k)}{\det(A_{k-1} + (a_k - a_{k-1}) I_n)}. \qquad \text{(C.4)}
\end{aligned}$$

The presence of matrix term $(a_k - a_{k-1}) I_n$ makes the rest of the proof a bit more involved than the proof of Theorem 3.1. Since we want to obtain bounds in terms of eigenvalues of matrices, we rewrite the rightmost side of (C.4) as a function of the eigenvalues of matrices $X_k X_k^\top$. Let $\lambda_{k,i}$ be the ith eigenvalue of matrix $X_k X_k^\top$, with $\lambda_{0,i} = 0$. We can write

$$\ln \frac{\det(A_k)}{\det(A_{k-1} + (a_k - a_{k-1}) I_n)} = \sum_{i=1}^{n} \ln \frac{a_k + \lambda_{k,i}}{a_k + \lambda_{k-1,i}}.$$

Now, simple manipulations yield

$$\begin{aligned}\sum_{k=1}^{m}\sum_{i=1}^{n}\ln\frac{a_k+\lambda_{k,i}}{a_k+\lambda_{k-1,i}} &= \sum_{k=1}^{m}\sum_{i=1}^{n}\ln\left(\frac{a_k}{a_{k-1}}\,\frac{a_{k-1}+\lambda_{k-1,i}}{a_k+\lambda_{k,i}}\right) \quad \text{(recall that } a_0=a_1\text{)}\\ &\quad+\sum_{k=1}^{m}\sum_{i=1}^{n}\ln\left(\frac{a_k+\lambda_{k,i}}{a_k+\lambda_{k-1,i}}\right)\\ &\quad+\sum_{i=1}^{n}\ln\left(\frac{a_m+\lambda_{m,i}}{a_m}\right)\\ &= \sum_{k=1}^{m}\sum_{i=1}^{n}\ln\left(\frac{a_k}{a_{k-1}}\,\frac{a_{k-1}+\lambda_{k-1,i}}{a_k+\lambda_{k-1,i}}\right)+\sum_{i=1}^{n}\ln\left(1+\frac{\lambda_{m,i}}{a_m}\right).\end{aligned}$$

Hence from (C.4) we have obtained

$$\boldsymbol{v}_m^\top A_m^{-1}\boldsymbol{v}_m \le \sum_{k=1}^{m}\sum_{i=1}^{n}\ln\left(\frac{a_k}{a_{k-1}}\,\frac{a_{k-1}+\lambda_{k-1,i}}{a_k+\lambda_{k-1,i}}\right)+\sum_{i=1}^{n}\ln\left(1+\frac{\lambda_{m,i}}{a_m}\right). \tag{C.5}$$

The reader can see that (C.5) is analogous to (A.1) but for the presence of a spurious double sum term.

We turn to upper bounding the double sum in (C.5) as a function of m. We proceed as follows. We first note that for $k=1$ the inner sum is zero. Therefore we continue by assuming $k\ge 2$ in the inner sum. Since $X_{k-1}\,X_{k-1}^\top$ has size $n\times n$ and has rank at most $k-1$, only $\min\{k-1,n\}$ among the eigenvalues $\lambda_{k-1,1},\ldots,\lambda_{k-1,n}$ can be nonzero. Also, as we have observed elsewhere, $X_{k-1}\,X_{k-1}^\top$ has the same nonzero eigenvalues as the (Gram) matrix $X_{k-1}^\top X_{k-1}$. Since the trace of a matrix equals the sum of its eigenvalues and the trace of $X_{k-1}^\top X_{k-1}$ is at most $(k-1)\,R^2$, we have, for $k=2,\ldots,m$,

$$\begin{aligned}&\sum_{i=1}^{n}\ln\left(\frac{a_k}{a_{k-1}}\,\frac{a_{k-1}+\lambda_{k-1,i}}{a_k+\lambda_{k-1,i}}\right)\\ &\le \max\left\{\sum_{j=1}^{k'}\ln\left(\frac{a_k}{a_{k-1}}\,\frac{a_{k-1}+\mu_j}{a_k+\mu_j}\right)\,:\,\mu_1,\ldots,\mu_{k'}\ge 0,\ \sum_{j=1}^{k'}\mu_j\le(k-1)\,R^2\right\},\end{aligned}$$

where $k'=\min\{k-1,n\}$. The maximum is achieved when all μ_j equal $(k-1)\,R^2/k'$. Thus

$$\begin{aligned}\sum_{i=1}^{n}\ln\left(\frac{a_k}{a_{k-1}}\,\frac{a_{k-1}+\lambda_{k-1,i}}{a_k+\lambda_{k-1,i}}\right) &\le k'\,\ln\left(\frac{a_k}{a_{k-1}}\,\frac{a_{k-1}+(k-1)\,R^2/k'}{a_k+(k-1)\,R^2/k'}\right)\\ &= k'\,\ln\left(\frac{k'\,a_k\,a_{k-1}+a_k\,(k-1)\,R^2}{k'\,a_k\,a_{k-1}+a_{k-1}\,(k-1)\,R^2}\right)\end{aligned} \tag{C.6}$$

for $k=2,3,\ldots,m$. Now, a derivative argument shows that the function $f(x;\alpha,\beta)=$

$x \ln\left(\frac{x+\alpha}{x+\beta}\right)$ is increasing in $x > 0$ whenever $\alpha > \beta \geq 0$. Therefore we see that

$$\begin{aligned}
\text{(C.6)} &= \frac{1}{a_k\, a_{k-1}} f\big(a_k\, a_{k-1} k';\, a_k(k-1)\, R^2,\; a_{k-1}(k-1)\, R^2\big) \\
&\leq \frac{1}{a_k\, a_{k-1}} f\big(a_k\, a_{k-1}(k-1);\, a_k(k-1)\, R^2,\; a_{k-1}(k-1)\, R^2\big) \\
&= (k-1)\, \ln\left(\frac{a_k}{a_{k-1}}\, \frac{a_{k-1}+R^2}{a_k+R^2}\right).
\end{aligned}$$

Hence we can write

$$\begin{aligned}
\sum_{k=1}^{m}\sum_{i=1}^{n} \ln\left(\frac{a_k}{a_{k-1}}\, \frac{a_{k-1}+\lambda_{k-1,i}}{a_k+\lambda_{k-1,i}}\right) &\leq \sum_{k=2}^{m} (k-1)\, \ln\left(\frac{a_k}{a_{k-1}}\, \frac{a_{k-1}+R^2}{a_k+R^2}\right) \\
&= \sum_{k=1}^{m-1} k\, \ln\left(\frac{k+1}{k}\, \frac{c\,R^2 k + R^2}{c\,R^2(k+1)+R^2}\right) \\
&\quad \text{(using } a_k = c\,R^2 k\text{)} \\
&= \sum_{k=1}^{m-1} k\, \ln\left(1 + \frac{1}{c\,k^2 + c\,k + k}\right) \\
&\leq \sum_{k=1}^{m-1} \frac{1}{c\,k+c+1} \qquad (\text{using } \ln(1+x) \leq x) \\
&\leq \frac{1}{c} \int_{1+1/c}^{m+1/c} \frac{dx}{x} \\
&= \frac{1}{c}\, \ln \frac{m+1/c}{1+1/c} \\
&= B(c,m).
\end{aligned}$$

To bound $\sqrt{\boldsymbol{v}_m^\top A_m^{-1} \boldsymbol{v}_m}$ from below, one can proceed as in the proof of Theorem 3.1, yielding

$$\sqrt{\boldsymbol{v}_m^\top A_m^{-1} \boldsymbol{v}_m} \geq \frac{\gamma\, m - D_\gamma(\boldsymbol{u}; \mathcal{S})}{\sqrt{a_m + \boldsymbol{u}^\top X_m\, X_m^\top \boldsymbol{u}}},$$

where $a_m = c\,R^2 m$. We plug this lower bound back into (C.5) together with the previous upper bound. After rearranging we obtain

$$\left(\frac{\gamma\, m - D_\gamma(\boldsymbol{u}; \mathcal{S})}{\sqrt{a_m + \boldsymbol{u}^\top X_m\, X_m^\top \boldsymbol{u}}}\right)^2 \leq B(c,m) + \sum_{i=1}^{n} \ln\left(1 + \frac{\lambda_{m,i}}{a_m}\right).$$

Solving for m occurring in the numerator of the left-hand side gives the desired bound.

Appendix D. Pseudoinverse and SVD of a matrix. In this section we recall basic facts about the pseudoinverse and the SVD of a matrix. This background material can be found, e.g., in [6]. We then exploit these facts to prove two technical lemmas which will be used in the subsequent sections. These lemmas build on ancillary results proven, e.g., in [5, 15, 16].

The pseudoinverse of an $n \times n$ real matrix A is the (unique) $n \times n$ matrix A^+ satisfying the following four conditions:

$$\begin{aligned} A\,A^+A &= A, \\ A^+A\,A^+ &= A^+, \\ (A^+A)^\top &= (A^+\,A), \\ (A\,A^+)^\top &= (A\,A^+). \end{aligned}$$

If the matrix is nonsingular, then the pseudoinverse coincides with the usual inverse, i.e., $A^+ = A^{-1}$.

In order to state the properties of the pseudoinverse that we need, we exploit the well-known connection between the pseudoinverse of a matrix and its SVD. In what follows we will focus only on (symmetric) positive semidefinite matrices. This allows us to simplify the exposition and replace the notion of singular value with the more familiar notion of eigenvalue.

Let $r \leq n$ be the rank of A. An SVD of a matrix A takes the form $A = U\,D\,U^\top$, where U is an orthogonal matrix (i.e., such that $U\,U^\top = U^\top U = I_n$), and D is a diagonal matrix whose first r diagonal entries are the positive eigenvalues $\lambda_1, \ldots, \lambda_r$ of A, and the remaining entries are zero. The first r columns of U form an orthonormal basis for $\mathrm{span}(A)$, the space spanned by the columns of A, whereas the remaining $n-r$ columns of U form an orthonormal basis for the orthogonal complement $\mathrm{span}^\perp(A)$. Recall that in the case under consideration, $\mathrm{span}^\perp(A)$ coincides with $\mathrm{null}(A)$, the null space of A.

A more convenient form of an SVD is one in which only the eigenvectors corresponding to positive eigenvalues are shown. Let U_r be the $n \times r$ matrix made up of the first r columns of U and let D_r be the $r \times r$ diagonal matrix whose diagonal entries are the positive eigenvalues of A. Then one immediately sees that the finer SVD holds:

$$A = U_r\,D_r\,U_r^\top. \tag{D.1}$$

Every positive semidefinite matrix admits an SVD like (D.1).

Given an SVD of a matrix, computing its pseudoinverse is a rather simple matter. In particular, given (D.1), one can easily show that

$$A^+ = U_r\,D_r^{-1}\,U_r^\top, \tag{D.2}$$

where D_r^{-1} is the inverse of the (nonsingular) matrix D_r.

Many properties of pseudoinverses which are relevant to this paper can be derived from (D.1) and (D.2). For instance, one can immediately see that, viewed as linear transformations from $\mathbb{R}^n$ to $\mathbb{R}^n$, both A^+A and $A\,A^+$ are the identical transformation onto $\mathrm{span}(A)$ and the null transformation on $\mathrm{span}^\perp(A)$, i.e.,

$$A^+A\,\boldsymbol{x} = A\,A^+\boldsymbol{x} = \begin{cases} \boldsymbol{x} & \text{if } \boldsymbol{x} \in \mathrm{span}(A), \\ 0 & \text{if } \boldsymbol{x} \in \mathrm{span}^\perp(A). \end{cases} \tag{D.3}$$

This property easily follows from the matrix identity

$$A^+A = A\,A^+ = U_r\,U_r^\top. \tag{D.4}$$

We are now ready to prove the following lemma, which generalizes the valuable Lemma A.1 in [5] to positive *semi*definite matrices.

LEMMA D.1. *Let A be an arbitrary $n \times n$ positive semidefinite matrix, let $\boldsymbol{x}$ be an arbitrary vector in $\mathbb{R}^n$, and let $B = A - \boldsymbol{x}\,\boldsymbol{x}^\top$. Then*

$$\boldsymbol{x}\,A^+\,\boldsymbol{x} = \begin{cases} 1 & \text{if } \boldsymbol{x} \notin \operatorname{span}(B), \\ 1 - \frac{\det_{\neq 0}(B)}{\det_{\neq 0}(A)} < 1 & \text{if } \boldsymbol{x} \in \operatorname{span}(B), \end{cases} \tag{D.5}$$

where $\det_{\neq 0}(M)$ denotes the product of the nonzero eigenvalues of matrix M. Note that $\det_{\neq 0}(M) = \det(M)$ when M is nonsingular.

Proof. If $\boldsymbol{x} = 0$, the lemma is trivially verified. Hence we continue by assuming $\boldsymbol{x} \neq 0$. We first prove that $\boldsymbol{x}^\top A^+ \boldsymbol{x} = 1$ if and only if $\boldsymbol{x} \notin \operatorname{span}(B)$. From Lemma A.3 in [16] it follows that $B\,A^+\boldsymbol{x} = 0$ when $\boldsymbol{x} \notin \operatorname{span}(B)$. Hence, using (D.3) we can write

$$0 = B\,A^+\boldsymbol{x} = (A - \boldsymbol{x}\boldsymbol{x}^\top)A^+\boldsymbol{x} = \boldsymbol{x} - \boldsymbol{x}\,\boldsymbol{x}^\top A^+\boldsymbol{x},$$

which implies $\boldsymbol{x}^\top A^+ \boldsymbol{x} = 1$. On the other hand, if $\boldsymbol{x} \in \operatorname{span}(B)$, we can always apply the Sherman–Morrison formula (3.2) and conclude that

$$\boldsymbol{x}^\top A^+ \boldsymbol{x} = \boldsymbol{x}^\top (B + \boldsymbol{x}\,\boldsymbol{x}^\top)^+ \boldsymbol{x} = \frac{\boldsymbol{x}^\top B^+ \boldsymbol{x}}{1 + \boldsymbol{x}^\top B^+ \boldsymbol{x}} < 1.$$

Let us now continue by assuming $\boldsymbol{x} \in \operatorname{span}(B)$. Let $A = U_r D_r U_r^\top$ be an SVD for A, in the sense of (D.1). Since $U_r\,U_r^\top \boldsymbol{x} = \boldsymbol{x}$ we can write

$$B = A - \boldsymbol{x}\boldsymbol{x}^\top = U_r M U_r^\top,$$

where M is the $r \times r$ symmetric matrix $M = D_r - U_r^\top \boldsymbol{x}\boldsymbol{x}^\top U_r$. We now claim that B and M have the same nonzero eigenvalues, so that $\det_{\neq 0}(B) = \det_{\neq 0}(M)$. Let $\mu_1, \mu_2, \ldots, \mu_k$, $k \leq r$, be the nonzero eigenvalues of M and let $\boldsymbol{e}_1, \boldsymbol{e}_2, \ldots, \boldsymbol{e}_k$ be the corresponding orthonormal eigenvectors. It is then easy to verify that $\mu_1, \mu_2, \ldots, \mu_k$ are also eigenvalues of B with corresponding orthonormal eigenvectors $U_r\boldsymbol{e}_1, U_r\boldsymbol{e}_2$, $\ldots, U_r\boldsymbol{e}_k$. Let E_k be the orthonormal matrix whose columns are $\boldsymbol{e}_1, \boldsymbol{e}_2, \ldots, \boldsymbol{e}_k$, let $M = E_k S_k E_k^\top$ be an SVD of M, and let $\boldsymbol{v} \in \operatorname{span}^\perp(U_r E_k)$. We have

$$B\boldsymbol{v} = U_r M U_r^\top \boldsymbol{v} = U_r E_k S_k E_k^\top U_r^\top \boldsymbol{v} = 0,$$

where the last equality follows from the orthogonality $\boldsymbol{v}^\top U_r E_k = 0$. Thus $\boldsymbol{v} \in \operatorname{null}(B)$, $\operatorname{rank}(B) = \operatorname{rank}(M)$, and the only nonzero eigenvalues of B are $\mu_1, \mu_2, \ldots, \mu_k$.

Computing the nonzero eigenvalues of M is actually fairly straightforward. We have

$$M = D_r - U_r^\top \boldsymbol{x}\boldsymbol{x}^\top U_r = D_r(I_r - D_r^{-1} U_r^\top \boldsymbol{x}\boldsymbol{x}^\top U_r).$$

Now, $\det_{\neq 0}(D_r) = \det(D_r) = \det_{\neq 0}(A)$, while it is easy to verify by direct inspection that the matrix $I_r - D_r^{-1} U_r^\top \boldsymbol{x}\boldsymbol{x}^\top U_r$ has eigenvalue 1 with multiplicity $r-1$ corresponding to $r-1$ eigenvectors forming an orthogonal basis for $\operatorname{span}^\perp(D_r^{-1} U_r^\top \boldsymbol{x})$, and eigenvalue $1 - \boldsymbol{x}^\top U_r D_r^{-1} U_r^\top \boldsymbol{x} = 1 - \boldsymbol{x}^\top A^+ \boldsymbol{x}$ corresponding to eigenvector $D_r^{-1} U_r^\top \boldsymbol{x}$. Since $\boldsymbol{x} \in \operatorname{span}(B)$ we already know that $1 - \boldsymbol{x}^\top A^+ \boldsymbol{x} > 0$. Therefore we have obtained

$$\det_{\neq 0}(B) = \det_{\neq 0}(M) = \det(D_r) \det_{\neq 0}(I_r - D_r^{-1} U_r^\top \boldsymbol{x}\boldsymbol{x}^\top U_r) = \det_{\neq 0}(A)(1 - \boldsymbol{x}^\top A^+ \boldsymbol{x}).$$

Rearranging gives the desired result. ∎

LEMMA D.2. *Let A be an arbitrary $n \times n$ positive semidefinite matrix. Set $B = A - \boldsymbol{x}\,\boldsymbol{x}^\top$, where $\boldsymbol{x} \notin \operatorname{span}(B)$. Then for any $\boldsymbol{v}, \boldsymbol{w} \in \operatorname{span}(B)$ we have $\boldsymbol{v}^\top A^+ \boldsymbol{w} = \boldsymbol{v}^\top B^+ \boldsymbol{w}$.*

Proof. We have

$$\begin{aligned}
\boldsymbol{v}^\top A^+ \boldsymbol{w} &= \boldsymbol{v}^\top B^+ B A^+ \boldsymbol{w} \\
&\quad \text{(by (D.3) and the fact that } \boldsymbol{v} \in \operatorname{span}(B)) \\
&= \boldsymbol{v}^\top B^+ (A - \boldsymbol{x}\boldsymbol{x}^\top) A^+ \boldsymbol{w} \\
&= \boldsymbol{v}^\top B^+ A A^+ \boldsymbol{w} - \boldsymbol{v}^\top B^+ \boldsymbol{x}\,\boldsymbol{x}^\top A^+ \boldsymbol{w} \\
&= \boldsymbol{v}^\top B^+ \boldsymbol{w} - \boldsymbol{v}^\top B^+ \boldsymbol{x}\,\boldsymbol{x}^\top A^+ \boldsymbol{w} \\
&\quad \text{(by (D.3) and the fact that } \boldsymbol{w} \in \operatorname{span}(B) \subseteq \operatorname{span}(A)) \\
&= \boldsymbol{v}^\top B^+ \boldsymbol{w} \\
&\quad \text{(since } \boldsymbol{x}^\top A^+ \boldsymbol{w} = 0 \text{ by Lemma A.3 in [16]).} \qquad \square
\end{aligned}$$

Appendix E. Proof of Theorem 4.2. The proof is again similar to the proof of Theorem 3.1.

Let $\mathcal{M} \subseteq \{1, 2, \dots\}$ be the set of trials where the algorithm in Figure 4.2 made a mistake, and let $A_k = X_k X_k^\top$, $k = 1, \dots, m = |\mathcal{M}|$. We investigate how the quadratic form $\boldsymbol{v}_k^\top A_k^+ \boldsymbol{v}_k$ changes over mistaken trials. Suppose the kth mistake occurred in trial $t = t_k$. Proceeding as in the proof of Theorem 3.1, one can show that

$$\boldsymbol{v}_k^\top A_k^+ \boldsymbol{v}_k \le \boldsymbol{v}_{k-1}^\top A_k^+ \boldsymbol{v}_{k-1} + \boldsymbol{x}_t^\top A_k^+ \boldsymbol{x}_t. \tag{E.1}$$

Now, as in the proof of Theorem 3.1, if $\boldsymbol{x}_t \in \operatorname{span}(A_k)$, we can apply the Sherman–Morrison formula (3.2) and obtain $\boldsymbol{v}_{k-1}^\top A_k^+ \boldsymbol{v}_{k-1} \le \boldsymbol{v}_{k-1}^\top A_{k-1}^+ \boldsymbol{v}_{k-1}$. On the other hand, in the case when $\boldsymbol{x}_t \notin \operatorname{span}(A_k)$ we can apply Lemma D.2 and conclude that $\boldsymbol{v}_{k-1}^\top A_k^+ \boldsymbol{v}_{k-1} = \boldsymbol{v}_{k-1}^\top A_{k-1}^+ \boldsymbol{v}_{k-1}$. Thus in both cases $\boldsymbol{v}_{k-1}^\top A_k^+ \boldsymbol{v}_{k-1} \le \boldsymbol{v}_{k-1}^\top A_{k-1}^+ \boldsymbol{v}_{k-1}$. Combining with (E.1) we obtain

$$\boldsymbol{v}_k^\top A_k^+ \boldsymbol{v}_k \le \boldsymbol{v}_{k-1}^\top A_{k-1}^+ \boldsymbol{v}_{k-1} + \boldsymbol{x}_t^\top A_k^+ \boldsymbol{x}_t,$$

holding for all $k = 1, \dots, m$. We now sum over $k = 1, \dots, m$ using $\boldsymbol{v}_0 = \boldsymbol{0}$. We get

$$\boldsymbol{v}_m^\top A_m^+ \boldsymbol{v}_m \le \sum_{k=1}^{m} \boldsymbol{x}_t^\top A_k^+ \boldsymbol{x}_t. \tag{E.2}$$

In order to upper bound the right-hand side of (E.2), we proceed as follows. We separate the mistaken trials k such that $\boldsymbol{x}_t \in \operatorname{span}(A_{k-1})$ (i.e., the trials such that $\operatorname{rank}(A_k) = \operatorname{rank}(A_{k-1})$) from the mistaken trials k such that $\boldsymbol{x}_t \notin \operatorname{span}(A_{k-1})$ (i.e., the trials such that $\operatorname{rank}(A_k) = \operatorname{rank}(A_{k-1}) + 1$). Then, in applying Lemma D.1, we count $1 - \frac{\det_{\neq 0}(A_{k-1})}{\det_{\neq 0}(A_k)}$ for the first kind of trials and 1 for the second kind. Finally, we upper bound taking the worst possible case of arranging the two kinds of trials within the sequence $\mathcal{M}$, taking into account that the number of trials of the second kind is equal to $r = \operatorname{rank}(A_m)$.

We assume the following general scenario:

$$\operatorname{rank}(A_k) = \begin{cases} 1, & k = 1, 2, \dots, 1 + i_1, \\ 2, & k = 2 + i_1, 3 + i_1, \dots, 2 + i_2, \\ 3, & k = 3 + i_2, 4 + i_2, \dots, 3 + i_3, \\ \vdots & \\ r, & k = r + i_{r-1}, r + i_{r-1} + 1, \dots, r + i_r, \end{cases}$$

where $0 \le i_1 \le i_2 \le \cdots \le i_r = m - r$. With this notation the right-hand side of (E.2) can be rewritten as follows (here and in what follows we assume $i_0 = 0$):

$$\begin{aligned}
\sum_{k=1}^{m} \boldsymbol{x}_t^\top A_k^+ \boldsymbol{x}_t &= \sum_{\ell=1}^{r} \left(\boldsymbol{x}_t^\top A_{\ell+i_{\ell-1}}^+ \boldsymbol{x}_t + \sum_{k=\ell+i_{\ell-1}+1}^{\ell+i_\ell} \boldsymbol{x}_t^\top A_k^+ \boldsymbol{x}_t \right) \\
&= \sum_{\ell=1}^{r} \left(1 + \sum_{k=\ell+i_{\ell-1}+1}^{\ell+i_\ell} \left(1 - \frac{\det_{\neq 0}(A_{k-1})}{\det_{\neq 0}(A_k)} \right) \right) \qquad \text{(using Lemma D.1)} \\
&= r + \sum_{\ell=1}^{r} \sum_{k=\ell+i_{\ell-1}+1}^{\ell+i_\ell} \left(1 - \frac{\det_{\neq 0}(A_{k-1})}{\det_{\neq 0}(A_k)} \right) \\
&\le r + \sum_{\ell=1}^{r} \sum_{k=\ell+i_{\ell-1}+1}^{\ell+i_\ell} \ln \frac{\det_{\neq 0}(A_k)}{\det_{\neq 0}(A_{k-1})} \\
&= r + \sum_{\ell=1}^{r} \ln \frac{\det_{\neq 0}(A_{\ell+i_\ell})}{\det_{\neq 0}(A_{\ell+i_{\ell-1}})}. \qquad \text{(E.3)}
\end{aligned}$$

We now proceed by bounding from above the logarithmic terms in (E.3). By construction we have $\operatorname{rank}(A_{\ell+i_\ell}) = \operatorname{rank}(A_{\ell+i_{\ell-1}}) = \ell$, which is also equal to the number of nonzero eigenvalues of the two matrices. Let λ_i, $i = 1, \dots, \ell$, denote the positive eigenvalues of $A_{\ell+i_{\ell-1}}$. Since $A_{\ell+i_\ell} - A_{\ell+i_{\ell-1}}$ is positive semidefinite, the positive eigenvalues of $A_{\ell+i_\ell}$ can be expressed as $\lambda_i + \mu_i$, $i = 1, \dots, \ell$, for some nonnegative values μ_i, such that $\sum_{i=1}^{\ell} \mu_i \le d_\ell R^2$, where $d_\ell = i_\ell - i_{\ell-1}$ is the number of rank-one matrices of the form $\boldsymbol{x}_t \boldsymbol{x}_t^\top$ which have been added to $A_{\ell+i_{\ell-1}}$ in order to obtain $A_{\ell+i_\ell}$. We have

$$\begin{aligned}
\ln \frac{\det_{\neq 0}(A_{\ell+i_\ell})}{\det_{\neq 0}(A_{\ell+i_{\ell-1}})} &= \ln \prod_{i=1}^{\ell} \frac{\lambda_i + \mu_i}{\lambda_i} \\
&\le \sum_{i=1}^{\ell} \ln \left(1 + \frac{\mu_i}{\lambda^*} \right) \quad \text{(recall that } \lambda^* \le \lambda_i, \, i = 1, \dots, \ell) \\
&\le \max_{\mu_1, \dots, \mu_\ell \,:\, \sum_{i=1}^{\ell} \mu_i \le R^2 d_\ell} \sum_{i=1}^{\ell} \ln \left(1 + \frac{\mu_i}{\lambda^*} \right) \\
&= \ell \ln \left(1 + \frac{R^2 d_\ell}{\lambda^* \ell} \right)
\end{aligned}$$

since the maximum is achieved when $\mu_i = R^2 d_\ell / \ell$, $i = 1, \dots, \ell$. Now, since $\sum_{\ell=1}^{r} d_\ell = m - r \le m$, we can plug back into (E.3) and maximize over $d_1, \dots, d_r$ such that $\sum_{\ell=1}^{r} d_\ell \le m$. We have

$$\begin{aligned}
\sum_{t \in \mathcal{M}} \boldsymbol{x}_t^\top A_k^+ \boldsymbol{x}_t &\le r + \sum_{\ell=1}^{r} \ell \ln \left(1 + \frac{R^2 d_\ell}{\lambda^* \ell} \right) \\
&\le r + \max_{d_1, \dots, d_r \,:\, \sum_{\ell=1}^{r} d_\ell \le m} \sum_{\ell=1}^{r} \ell \ln \left(1 + \frac{R^2 d_\ell}{\lambda^* \ell} \right)
\end{aligned}$$

$$= r + \frac{r(r+1)}{2} \ln\left(1 + \frac{2\,R^2}{\lambda^*}\frac{m}{r(r+1)}\right) \tag{E.4}$$

(since the maximum is achieved when $d_\ell = \frac{2\,\ell\,m}{r(r+1)}$, $\ell = 1, \ldots, r$),

which is the required upper bound on the right-hand side of (E.2).

When $\boldsymbol{u} \notin \mathrm{span}^{\perp}(A_m)$ we can bound $\sqrt{\boldsymbol{v}_m^\top A_m^+ \boldsymbol{v}_m}$ from below as in the proof of Theorem 3.1. This yields

$$\sqrt{\boldsymbol{v}_m^\top A_m^+ \boldsymbol{v}_m} \geq \frac{\gamma\, m - D_\gamma(\boldsymbol{u}; \mathcal{S})}{\sqrt{\boldsymbol{u}^\top X_m\, X_m^\top \boldsymbol{u}}}.$$

Plugging back into (E.2) and combining with (E.4) gives

$$\left(\frac{\gamma\, m - D_\gamma(\boldsymbol{u}; \mathcal{S})}{\sqrt{\boldsymbol{u}^\top X_m\, X_m^\top \boldsymbol{u}}}\right)^2 \leq r + \frac{r(r+1)}{2} \ln\left(1 + \frac{2\,R^2}{\lambda^*}\frac{m}{r(r+1)}\right).$$

Solving for m occurring in the numerator of the left-hand side gives (4.4). On the other hand, when $\boldsymbol{u} \in \mathrm{span}^{\perp}(A_m)$ we have $\frac{D_\gamma(\boldsymbol{u};\mathcal{S})}{\gamma} = m$ and $\boldsymbol{u}^\top X_m\, X_m^\top \boldsymbol{u} = 0$, and thus (4.4) is vacuously verified as an equality.

Acknowledgments. Thanks to Bob Williamson for fruitful conversations and to the anonymous reviewers, whose comments turned out to be very useful in improving the presentation of this paper.

REFERENCES

[1] M. Aizerman, E. Braverman, and L. Rozonoer, *Theoretical foundations of the potential function method in pattern recognition learning*, Autom. Remote Control, 25 (1964), pp. 821–837.
[2] D. Angluin, *Queries and concept learning*, Machine Learning, 2 (1988), pp. 319–342.
[3] P. Auer, N. Cesa-Bianchi, and C. Gentile, *Adaptive and self-confident on-line learning algorithms*, J. Comput. System Sci., 64 (2002), pp. 48–75.
[4] P. Auer and M. Warmuth, *Tracking the best disjunction*, Machine Learning, 32 (1998), pp. 127–150.
[5] K. Azoury and M. Warmuth, *Relative loss bounds for on-line density estimation with the exponential family of distributions*, Machine Learning, 43 (2001), pp. 211–246.
[6] A. Ben-Israel and T. Greville, *Generalized Inverses: Theory and Applications*, John Wiley and Sons, New York, 1974.
[7] H. Block, *The Perceptron: A model for brain functioning*, Rev. Modern Phys., 34 (1962), pp. 123–135.
[8] A. Borodin and R. El-Yaniv, *Online Computation and Competitive Analysis*, Cambridge University Press, Cambridge, UK, 1998.
[9] N. Cesa-Bianchi, A. Conconi, and C. Gentile, *On the generalization ability of on-line learning algorithms*, IEEE Trans. Inform. Theory, 50 (2004), pp. 2050–2057.
[10] N. Cesa-Bianchi, Y. Freund, D. Helmbold, D. Haussler, R. Schapire, and M. Warmuth, *How to use expert advice*, J. ACM, 44 (1997), pp. 427–485.
[11] N. Cesa-Bianchi, Y. Freund, D. Helmbold, and M. Warmuth, *On-line prediction and conversion strategies*, Machine Learning, 25 (1996), pp. 71–110.
[12] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines*, Cambridge University Press, Cambridge, UK, 2001.
[13] L. Devroye, L. Győrfi, and G. Lugosi, *A Probabilistic Theory of Pattern Recognition*, Springer-Verlag, New York, 1996.
[14] R. Duda, P. Hart, and D. G. Stork, *Pattern Classification*, John Wiley and Sons, New York, 2000.
[15] J. Forster and M. Warmuth, *Relative expected instantaneous loss bounds*, J. Comput. System Sci., 64 (2002), pp. 76–102.

[16] J. Forster and M. Warmuth, *Relative loss bounds for temporal-difference learning*, Machine Learning, 51 (2003), pp. 23–50.
[17] C. Gentile, *A new approximate maximal margin classification algorithm*, J. Machine Learning Res., 2 (2001), pp. 213–242.
[18] C. Gentile, *The robustness of the p-norm algorithms*, Machine Learning, 53 (2003), pp. 265–299.
[19] C. Gentile and M. Warmuth, *Linear hinge loss and average margin*, in Proceedings of the 1998 Conference on Advances in Neural Information Processing Systems 10, MIT Press, Cambridge, MA, 1999, pp. 225–231.
[20] A. Grove, N. Littlestone, and D. Schuurmans, *General convergence results for linear discriminant updates*, Machine Learning, 43 (2001), pp. 173–210.
[21] M. Herbster and M. Warmuth, *Tracking the best expert*, Machine Learning, 32 (1998), pp. 151–178.
[22] M. Herbster and M. Warmuth, *Tracking the best linear predictor*, J. Machine Learning Res., 1 (2001), pp. 281–309.
[23] A. Hoerl and R. Kennard, *Ridge regression: Biased estimation for nonorthogonal problems*, Technometrics, 12 (1970), pp. 55–67.
[24] R. Horn and C. Johnson, *Matrix Analysis*, Cambridge University Press, Cambridge, UK, 1985.
[25] J. Kivinen and M. Warmuth, *Relative loss bounds for multidimensional regression problems*, Machine Learning, 45 (2001), pp. 301–329.
[26] J. Kivinen, M. Warmuth, and P. Auer, *The Perceptron algorithm vs. Winnow: Linear vs. logarithmic mistake bounds when few input variables are relevant*, Artificial Intelligence, 97 (1997), pp. 325–343.
[27] Y. Li and P. Long, *The relaxed online maximum margin algorithm*, Machine Learning, 46 (2002), pp. 361–387.
[28] N. Littlestone, *Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm*, Machine Learning, 2 (1988), pp. 285–318.
[29] N. Littlestone, *Redundant noisy attributes, attribute errors, and linear threshold learning using Winnow*, in Proceedings of the 4th Annual Workshop on Computational Learning Theory, Morgan-Kaufmann, San Mateo, CA, 1991, pp. 147–156.
[30] N. Littlestone and M. Warmuth, *The weighted majority algorithm*, Inform. and Comput., 108 (1994), pp. 212–261.
[31] M. Marcus and H. Minc, *Introduction to Linear Algebra*, Dover, New York, 1965.
[32] A. Novikoff, *On convergence proofs of Perceptrons*, in Proceedings of the Symposium on the Mathematical Theory of Automata, Vol. XII, Polytechnic Institute of Brooklyn, 1962, pp. 615–622.
[33] R. Rifkin, G. Yeo, and T. Poggio, *Regularized least squares classification*, in Advances in Learning Theory: Methods, Model and Applications, NATO Sci. Ser. III Comput. Systems Sci. 190, IOS Press, Amsterdam, 2003, pp. 131–153.
[34] F. Rosenblatt, *The Perceptron: A probabilistic model for information storage and organization in the brain*, Psych. Rev., 65 (1958), pp. 386–408.
[35] G. Saunders, A. Gammerman, and V. Vovk, *Ridge regression learning algorithm in dual variables*, in Proceedings of the 15th International Conference on Machine Learning, Morgan-Kaufmann, San Francisco, CA, 1998, pp. 515–521.
[36] B. Schölkopf and A. Smola, *Learning with Kernels*, MIT Press, Cambridge, MA, 2002.
[37] J. A. K. Suykens, T. Van Gestel, J. De Brabanter, B. De Moor, and J. Vandewalle, *Least Squares Support Vector Machines*, World Scientific, Singapore, 2002.
[38] V. Vapnik, *Statistical Learning Theory*, John Wiley and Sons, New York, 1998.
[39] V. Vovk, *Aggregating strategies*, in Proceedings of the 3rd Annual Workshop on Computational Learning Theory, Morgan-Kaufmann, San Mateo, CA, 1990, pp. 372–383.
[40] V. Vovk, *Competitive on-line statistics*, Internat. Statist. Rev., 69 (2001), pp. 213–248.
[41] R. Williamson, A. Smola, and B. Schölkopf, *Generalization bounds for regularization networks and support vector machines via entropy numbers of compact operators*, IEEE Trans. Inform. Theory, 47 (2001), pp. 2516–2532.