The Margin Perceptron with Unlearning

Constantinos Panagiotakopoulos COSTAPAN@ENG.AUTH.GR
Petroula Tsampouka PETROULA@GEN.AUTH.GR
Physics Division, School of Technology, Aristotle University of Thessaloniki, 54124 Thessaloniki, Greece

Abstract

We introduce into the classical Perceptron algorithm with margin a mechanism of unlearning which in the course of the regular update allows for a reduction of possible contributions from "very well classified" patterns to the weight vector. The resulting incremental classification algorithm, called Margin Perceptron with Unlearning (MPU), provably converges in a finite number of updates to any desirable chosen before running approximation of either the maximal margin or the optimal 1-norm soft margin solution. Moreover, an experimental comparative evaluation involving representative linear Support Vector Machines reveals that the MPU algorithm is very competitive.

1. Introduction

Support Vector Machines (SVMs) (Boser et al., 1992; Vapnik, 1998; Cristianini & Shawe-Taylor, 2000) have been extensively used as linear classifiers either in the space where the patterns originally reside or in high dimensional feature spaces induced by kernels. SVMs appear to be very successful at addressing the problem of minimising an objective function involving the empirical risk while at the same time keeping low the complexity of the classifier. As measures of the empirical risk various quantities have been proposed with the 1- and 2-norm loss functions being the most widely accepted ones giving rise to the optimisation problems known as L1- and L2-SVMs (Cortes & Vapnik, 1995). In the case that the 2-norm loss function takes the place of the empirical risk an equivalent formulation exists which renders the dataset linearly separable in a high dimensional feature space. Therefore, one can either solve the general optimisation problem consisting of two terms, namely the capacity term and the 2-norm loss or just attempt to minimise the capacity term in the high dimensional space as long as it ensures zero loss for the patterns mapped in that space. In the second treatment minimisation of the capacity term coincides with maximisation of the margin. SVMs traditionally follow the second approach.

On the other hand solving the maximum margin problem has also been the goal of various Perceptron-like algorithms (Li & Long, 2002; Gentile, 2001; Tsampouka & Shawe-Taylor, 2007). The key characteristic of such algorithms is that they work in the primal space in an online manner, i.e. processing one example at a time. Cycling repeatedly through the patterns they update their internal state stored in the weight vector each time a misclassification condition is satisfied by a pattern. More specifically, the Perceptron-like algorithms mentioned above are able to attain any approximation of the maximum margin, thereby producing infinitely close approximations of the L2-SVM solution. On the contrary, the classical Perceptron algorithm with margin (Duda & Hart, 1973) provably achieves only up to 1/2 of the maximum margin (Krauth & Mézard, 1987). The weight vector of the Perceptron algorithm (Rosenblatt, 1958) is built progressively by adding to it the patterns found misclassified multiplied by the corresponding labels. This procedure is characterised as learning. However, by introducing in the present work what we call unlearning, a mechanism allowing for a reduction or even elimination of the contribution that some patterns have to the solution weight vector when they are "very well classified", we find that the Perceptron's inability to provably achieve maximum margin is miraculously remedied. The possibility that the contribution of a given pattern to the current solution may increase or decrease with running time is very common to SVM algorithms although the procedure by which this is achieved does not tempt one to pay particular attention to a clear-cut distinction between learning and unlearning.

Since the L1-SVM problem is not known to admit an equivalent maximum margin interpretation via a mapping to an appropriate space Perceptron-like algorithms appeared so far unable to deal with such a task. In early attempts (Guyon & Stork, 1999) at ad-

Appearing in *Proceedings of the 27th International Conference on Machine Learning*, Haifa, Israel, 2010. Copyright 2010 by the author(s)/owner(s).

dressing the L1-SVM problem with a Perceptron an upper bound was set on the number of updates associated with any pattern before that pattern ended up being ignored in the process, thereby establishing artificially linear separability. The defect of the method was the impossibility for the patterns ignored to reenter the process rendering the effect of any early updates performed towards wrong directions totally irreversible. As a consequence, the outcome lacked the uniqueness dictated by optimisation theory since it depended heavily on the sequence according to which the patterns were presented to the algorithm. Here, we follow a procedure which does not suffer from the previous restrictions and through which the Perceptron is able in a finite number of steps to achieve any desirable fixed before running approximation of the optimal solution. Again, the mechanism of unlearning may be identified as the decisive factor in phasing out the misleading effect that some patterns have, mostly during early stages of the algorithm.

The decision of whether the contribution of a pattern to the weight vector should be reduced concerns only the pattern that is currently presented to the algorithm. Consequently, the mechanism of unlearning becomes vacuous if our algorithm is implemented in a strictly online setting in which all examples are considered to be distinct. This reveals an important difference from the Budget Perceptron (Crammer et al., 2003), an online algorithm featuring a removal of past examples from the expression of the weight vector which, however, only takes place whenever the pattern currently presented to the algorithm is inserted. Furthermore, in a batch setting we aim at convergence to the known SVM solution.

The paper is organised as follows. Section 2 describes the algorithm. Section 3 is devoted to a theoretical analysis. In Section 4 we give details concerning the implementation of our algorithm while in Section 5 we deliver our experimental results. Finally, Section 6 contains our conclusions.

2. The Margin Perceptron Algorithm with Unlearning

Given a training set which may be already mapped into a feature space of higher dimensionality we place all patterns in the same position at a distance ρ in an additional dimension, thereby constructing an embedding of our data into the so-called augmented space (Duda & Hart, 1973). The advantage of this embedding is that the linear hypothesis in the augmented space becomes homogeneous. Throughout our discussion we assume that the augmented patterns are multiplied by the corresponding labels in order to allow for a uniform treatment of both categories of patterns. Also, $R \equiv \max_k \|\boldsymbol{y}_k\|$ with $\boldsymbol{y}_k$ the k-th augmented pattern multiplied by its label.

First we assume that the training set is linearly separable in the appropriate feature space. The relation characterising optimally correct classification of the training patterns by a weight vector $\boldsymbol{u}$ of unit norm in the augmented space is

$$\boldsymbol{u} \cdot \boldsymbol{y}_k \geq \gamma_{\mathrm{d}} \equiv \max_{\boldsymbol{u}' \,:\, \left\|\boldsymbol{u}'\right\| = 1} \min_i \{\boldsymbol{u}' \cdot \boldsymbol{y}_i\} \qquad \forall k. \tag{1}$$

Here γ_{d} is the maximum margin in the augmented space with respect to hyperplanes passing through the origin which should be distinguished from the maximum geometric margin γ in the non-augmented feature space.

The Margin Perceptron with Unlearning (MPU) at its t-th step maintains two vectors. The first is the usual augmented weight vector $\boldsymbol{a}_t$ in the primal representation having the dimensionality M of the feature space. The second is a vector of non-negative integer-valued components I_k^t and dimensionality equal to the number N of patterns which keeps track of the update history of the algorithm. Both are initially set to zero, i.e. $\boldsymbol{a}_0 = \boldsymbol{0}$ and $I_k^0 = 0 \ \ \forall k$, and are updated according to the rule

$$\begin{aligned} \boldsymbol{a}_{t+1} &= \boldsymbol{a}_t + \eta_t \boldsymbol{y}_k \\ I_k^{t+1} &= \ I_k^t + \eta_t \end{aligned} \tag{2}$$

each time the training pattern $\boldsymbol{y}_k$ presented to the algorithm satisfies either one of the conditions appearing in the following definition of η_t

$$\eta_t = \begin{cases} +1, \text{ if } \ \boldsymbol{a}_t \cdot \boldsymbol{y}_k \leq b \\ -1, \text{ if } \ \boldsymbol{a}_t \cdot \boldsymbol{y}_k \geq b + \delta b \ \text{ and } \ I_k^t > 0. \end{cases} \tag{3}$$

Here b and δb are positive constants. We shall refer to an update with $\eta_t = +1$ as a learning update (step) whereas to an update with $\eta_t = -1$ as an unlearning update (step). Thus, the variable I_k^t is equal to the difference between the number of learning and unlearning steps associated with the k-th pattern. The total number of learning or unlearning steps will be denoted as t^+ or t^-, respectively. Obviously, the total number of updates t is

$$t = t^+ + t^- \tag{4}$$

and

$$\sum_k I_k^t = \sum_{\tau=0}^{t-1} \eta_\tau = t^+ - t^-. \tag{5}$$

Algorithm 1 The Margin Perceptron with Unlearning

Input: A dataset $S = (\boldsymbol{y}_1, \ldots, \boldsymbol{y}_k, \ldots, \boldsymbol{y}_N)$ with $\boldsymbol{y}_k$ the k-th augmented pattern multiplied by its label
Define: $R \equiv \max_k \|\boldsymbol{y}_k\|$
Require: $I, b > 0,\ \ \delta b > R^2$
Initialise: $t = 0,\ \boldsymbol{a}_0 = \boldsymbol{0},\ I_k^0 = 0\ \forall k$
  repeat
    for $k = 1$ **to** N **do**
      $p_{tk} = \boldsymbol{a}_t \cdot \boldsymbol{y}_k$
      if $(p_{tk} \leq b$ **and** $I_k^t < I)$ **then**
        $\boldsymbol{a}_{t+1} = \boldsymbol{a}_t + \boldsymbol{y}_k$
        $I_k^{t+1} = I_k^t + 1$
        $t \leftarrow t + 1$
      else if $I_k^t = 0$ **then**
        continue
      else if $p_{tk} \geq b + \delta b$ **then**
        $\boldsymbol{a}_{t+1} = \boldsymbol{a}_t - \boldsymbol{y}_k$
        $I_k^{t+1} = I_k^t - 1$
        $t \leftarrow t + 1$
      end if
    end for
  until no update made within the **for** loop

It should be clear that with the variables I_k^t produced by the algorithm as described above the weight vector admits the so-called dual representation

$$\boldsymbol{a}_t = \sum_k I_k^t \boldsymbol{y}_k \tag{6}$$

as a linear combination with positive coefficients of the training patterns. Consequently, the I_k^t's play the role of the dual variables. A learning step takes place whenever the pattern $\boldsymbol{y}_k$ satisfies the misclassification condition $\boldsymbol{a}_t \cdot \boldsymbol{y}_k \leq b$ meaning that $\boldsymbol{y}_k$ is either a classification or a margin error. We shall refer to such a pattern as misclassified. Instead, an unlearning step takes place whenever the pattern $\boldsymbol{y}_k$ is "very well classified", i.e. $\boldsymbol{a}_t \cdot \boldsymbol{y}_k \geq b + \delta b$, and provided this pattern does have a non-vanishing contribution to the dual representation (6) of the current weight vector. Thus, a prerequisite for unlearning is that learning has previously taken place involving the pattern in question. Moreover, unlearning may be regarded as a mechanism preventing the dual representation (6) from expanding unnecessarily. Notice, that due to the "discrete" nature of the update rule the dual variables I_k^t remain naturally non-negative without having to be clipped as in the case of SVMs.

An important feature of the algorithm is the existence of the gap δb between the threshold b of the misclassification condition $\boldsymbol{a}_t \cdot \boldsymbol{y}_k \leq b$ and the threshold $b + \delta b$ of the condition $\boldsymbol{a}_t \cdot \boldsymbol{y}_k \geq b + \delta b$ for exceedingly good classification. Actually, it is not very difficult to see that such a gap should necessarily satisfy the constraint

$$\delta b > R^2 \tag{7}$$

in order for the algorithm to have a chance of converging in a finite number of steps. Indeed, let us assume that the pattern $\boldsymbol{y}_k$ presented to the algorithm satisfies $\boldsymbol{a}_t \cdot \boldsymbol{y}_k = b$. Then, a learning update takes place as a result of which the weight vector becomes $\boldsymbol{a}_{t+1} = \boldsymbol{a}_t + \boldsymbol{y}_k$. Now, consider a situation in which the same pattern is presented once again to the algorithm without another update having intervened. We have $\boldsymbol{a}_{t+1} \cdot \boldsymbol{y}_k = b + \|\boldsymbol{y}_k\|^2$ and $I_k^{t+1} > 0$. Therefore, if $\delta b \leq \|\boldsymbol{y}_k\|^2$ an unlearning step will take place which will exactly cancel the effect of the learning step. This procedure may be repeated again and again leading to a closed loop. If, instead, $\delta b > \|\boldsymbol{y}_k\|^2\ \ \forall k$, i.e. $\delta b > R^2$ such unwanted situations are naturally avoided. The same condition also ensures that

$$t^+ > t^-, \text{ if } t > 0.$$

Obviously, one step before we arrive at the situation where the number of learning steps equals the number of unlearning steps we should have $t^+ = t^- + 1$ meaning that all dual variables vanish except for one which takes the value 1. So, let us assume that this happens at the t-th step of the algorithm and that the only non-vanishing dual variable is associated with the k-th pattern, i.e. $I_k^t = 1$ holds. Then, $\boldsymbol{a}_t = \boldsymbol{y}_k$ from where $\boldsymbol{a}_t \cdot \boldsymbol{y}_k = \|\boldsymbol{y}_k\|^2 < \delta b < b + \delta b$ forbidding the unlearning step that would lead to equality between learnings and unlearnings.

In the case that the assumption of linear separability in the feature space is relaxed we impose the further condition that a learning update is allowed provided that the dual variable I_k^t associated with the misclassified pattern $\boldsymbol{y}_k$ satisfies the constraint $I_k^t < I$, where I is a positive integer. In such a case the update rule (2) remains the same but η_t is now defined by

$$\eta_t = \begin{cases} +1, \text{ if } \ \boldsymbol{a}_t \cdot \boldsymbol{y}_k \leq b & \text{and } \ I_k^t < I \\ -1, \text{ if } \ \boldsymbol{a}_t \cdot \boldsymbol{y}_k \geq b + \delta b & \text{and } \ I_k^t > 0. \end{cases} \tag{8}$$

Notice that again as a result of the "discrete" nature of the update rule the dual variables satisfy naturally the condition

$$0 \leq I_k^t \leq I \ \ \forall k \tag{9}$$

without any need for clipping. Setting $I = \infty$ in (8) we recover (3).

Finally, in the case that the mapping into the feature space is induced by a kernel and a primal representation of $\boldsymbol{a}_t$ is unavailable it suffices to update the dual variables I_k^t according to (2) and perform the inner products using the dual representation (6) of $\boldsymbol{a}_t$.

3. Theoretical Analysis

We first consider a linearly separable training set possessing maximum margin $\gamma_{\rm d}$ in the augmented space with respect to hyperplanes passing through the origin.

Theorem 1 *The Margin Perceptron with Unlearning and $I=\infty$ converges in a finite number $t_{\rm c}$ of updates satisfying the bound*

$$t_{\rm c} \leq \frac{R^2+2b}{\gamma_{\rm d}^2}\left(1+\frac{R^2+2b}{4(\delta b-R^2)}\right). \tag{10}$$

Moreover, the zero-threshold solution hyperplane possesses margin $\gamma'_{\rm d}$ which is a fraction f of the maximum margin $\gamma_{\rm d}$ obeying the inequality

$$f \equiv \gamma'_{\rm d}/\gamma_{\rm d} > \left(1+\min\{\delta b, b+R^2\}/b\right)^{-1}. \tag{11}$$

Finally, an after-running lower bound on f involving the margin $\gamma'_{\rm d}$ achieved, the length $\|\boldsymbol{a}_{t_{\rm c}}\|$ of the solution weight vector $\boldsymbol{a}_{t_{\rm c}}$ and the final difference between the number of learning and unlearning steps $(t_{\rm c}^+ - t_{\rm c}^-)$ is given by

$$f = \gamma'_{\rm d}/\gamma_{\rm d} \geq \gamma'_{\rm d}(t_{\rm c}^+ - t_{\rm c}^-)/\|\boldsymbol{a}_{t_{\rm c}}\|. \tag{12}$$

Proof Taking the inner product with the optimal direction $\boldsymbol{u}$ in (6) and using (1) and (5) we obtain

$$\|\boldsymbol{a}_t\| \geq \boldsymbol{a}_t\cdot\boldsymbol{u} = \sum_k I_k^t(\boldsymbol{y}_k\cdot\boldsymbol{u}) \geq \gamma_{\rm d}\sum_k I_k^t = \gamma_{\rm d}(t^+-t^-). \tag{13}$$

From the update rule (2) taking into account the conditions under which the update takes place and the definition of η_t we get

$$\begin{aligned}\|\boldsymbol{a}_{t+1}\|^2 - \|\boldsymbol{a}_t\|^2 &= \|\boldsymbol{y}_k\|^2 + 2\eta_t\boldsymbol{a}_t\cdot\boldsymbol{y}_k \\ &\leq R^2 + 2\eta_t b - \delta b(1-\eta_t)\end{aligned}$$

a repeated application of which, using also (4) and (5) and the initial condition $\boldsymbol{a}_0=\boldsymbol{0}$, gives

$$\begin{aligned}\|\boldsymbol{a}_t\|^2 &\leq R^2 t + 2b\sum_{\tau=0}^{t-1}\eta_\tau - \delta b\left(t-\sum_{\tau=0}^{t-1}\eta_\tau\right) \\ &= R^2 t + 2b(t^+-t^-) - \delta b(t-(t^+-t^-)) \\ &= (R^2+2b)(t^+-t^-) - 2(\delta b - R^2)t^-. \end{aligned} \tag{14}$$

Combining (13) and (14) we obtain a Novikoff-like (Novikoff, 1962) squeezing relation

$$\gamma_{\rm d}^2(t^+-t^-)^2 \leq \|\boldsymbol{a}_t\|^2 \leq (R^2+2b)(t^+-t^-) - 2(\delta b - R^2)t^-. \tag{15}$$

From (15) we get

$$\begin{aligned}2(\delta b-R^2)t^- &\leq (R^2+2b)(t^+-t^-) - \gamma_{\rm d}^2(t^+-t^-)^2 \\ &\leq (R^2+2b)^2/4\gamma_{\rm d}^2\end{aligned}$$

by making use of the fact that the quantity $(R^2+2b)(t^+-t^-) - \gamma_{\rm d}^2(t^+-t^-)^2$ acquires a maximum with respect to (t^+-t^-) for $(t^+-t^-) = (R^2+2b)/2\gamma_{\rm d}^2$. Then, taking into account (7) we immediately obtain

$$t^- \leq \frac{(R^2+2b)^2}{8\gamma_{\rm d}^2(\delta b - R^2)}. \tag{16}$$

Again from (15) by dropping the last term in the upper bound on $\|\boldsymbol{a}_t\|^2$ we have $\gamma_{\rm d}^2(t^+-t^-)^2 \leq (R^2+2b)(t^+-t^-)$ or

$$t^+ - t^- \leq (R^2+2b)/\gamma_{\rm d}^2. \tag{17}$$

Combining (16) and (17) we obtain the upper bound on the number t of updates given by the r.h.s. of (10).

Upon convergence of the algorithm in $t_{\rm c}$ updates it is obvious that all the patterns $\boldsymbol{y}_k$ which have a non-vanishing contribution to the expansion (6) of the solution weight vector $\boldsymbol{a}_{t_{\rm c}}$ necessarily have their inner product with $\boldsymbol{a}_{t_{\rm c}}$ in the interval $(b, b+\delta b)$, i.e.

$$b < \boldsymbol{a}_{t_{\rm c}}\cdot\boldsymbol{y}_k < b+\delta b\ , \ \text{ if } \ I_k^{t_{\rm c}} > 0. \tag{18}$$

From (6) with $t=t_{\rm c}$ we get

$$\|\boldsymbol{a}_{t_{\rm c}}\|^2 = \boldsymbol{a}_{t_{\rm c}}\cdot\left(\sum_k I_k^{t_{\rm c}}\boldsymbol{y}_k\right) = \sum_k I_k^{t_{\rm c}}\boldsymbol{a}_{t_{\rm c}}\cdot\boldsymbol{y}_k$$

or, taking into account (5) and (18),

$$b(t_{\rm c}^+ - t_{\rm c}^-) < \|\boldsymbol{a}_{t_{\rm c}}\|^2 < (b+\delta b)(t_{\rm c}^+ - t_{\rm c}^-). \tag{19}$$

Since upon convergence all patterns violate the misclassification condition we have $\boldsymbol{a}_{t_{\rm c}}\cdot\boldsymbol{y}_k > b \quad \forall k$ or equivalently $\boldsymbol{u}_{t_{\rm c}}\cdot\boldsymbol{y}_k > b/\|\boldsymbol{a}_{t_{\rm c}}\| \quad \forall k$ with $\boldsymbol{u}_{t_{\rm c}} \equiv \boldsymbol{a}_{t_{\rm c}}/\|\boldsymbol{a}_{t_{\rm c}}\|$. Stated differently, the margin achieved satisfies $\gamma'_{\rm d} = \min_k\{\boldsymbol{u}_{\boldsymbol{t_{\rm c}}}\cdot\boldsymbol{y}_k\} > b/\|\boldsymbol{a}_{t_{\rm c}}\|$. Additionally, from (13)

$$1/\gamma_{\rm d} \geq (t_{\rm c}^+ - t_{\rm c}^-)/\|\boldsymbol{a}_{t_{\rm c}}\|. \tag{20}$$

Thus, we obtain

$$f = \gamma'_{\rm d}/\gamma_{\rm d} > (b/\|\boldsymbol{a}_{t_{\rm c}}\|)/\gamma_{\rm d} \geq b(t_{\rm c}^+ - t_{\rm c}^-)/\|\boldsymbol{a}_{t_{\rm c}}\|^2. \tag{21}$$

If we now make use in (21) of the upper bound on $\|\boldsymbol{a}_{t_{\rm c}}\|^2$ from (19) we get $f > (1+\delta b/b)^{-1}$. If, instead, we employ (15) to obtain the bound $\|\boldsymbol{a}_{t_{\rm c}}\|^2 \leq (R^2+2b)(t_{\rm c}^+ - t_{\rm c}^-)$ we get $f > \left(2+R^2/b\right)^{-1}$. The above lower bounds on f written compactly lead to (11).

Finally, (12) follows readily from (20). □

Remark 1 From (11) it becomes apparent that as $b\to\infty$ with δb kept fixed the MPU algorithm with $I=\infty$ is guaranteed before running to converge to the maximum margin solution, i.e. $\gamma'_{\rm d}\to\gamma_{\rm d}$. If, instead, we first take the limit $\delta b\to\infty$ (thereby recovering the classical Perceptron with margin) and then the limit $b\to\infty$ we get $\gamma'_{\rm d} > \gamma_{\rm d}/2$.

Remark 2 The lower bound on $\|\boldsymbol{a}_{t_c}\|^2$ from (15) with the upper bound from (19) lead to $t_c^+ - t_c^- < (b + \delta b)/\gamma_d^2$. Also, from $\gamma_d \geq \gamma_d' > b/\|\boldsymbol{a}_{t_c}\|$ we get $\|\boldsymbol{a}_{t_c}\|^2 > b^2/\gamma_d^2$ which combined with the upper bound on $\|\boldsymbol{a}_{t_c}\|^2$ from (19) leads to $t_c^+ - t_c^- > b^2(b+\delta b)^{-1}\gamma_d^{-2}$. Thus, upon convergence of the MPU algorithm it holds that

$$(1+\delta b/b)^{-1} b\gamma_d^{-2} < t_c^+ - t_c^- < (1+\delta b/b) b\gamma_d^{-2}$$

meaning that $t_c^+ - t_c^-$ is strongly constrained. Moreover, $(t_c^+ - t_c^-)/b \to \gamma_d^{-2}$ as $b \to \infty$. This is the analog of the well-known result $\sum_k \alpha_k^\star = \gamma^{-2}$ for the hard margin SVM with $\alpha_k^\star$ being the optimal dual variables.

Remark 3 As we already pointed out the MPU algorithm in a strictly online setting reduces to the classical Perceptron with margin. Nevertheless, our derivation of the upper bound on the number of updates given by the r.h.s. of (10) does hold in the context of any related strictly online scenario, e.g. a variation of the variable-size cache Budget Perceptron, in the t-th step of which the insertion ($\eta_t = +1$) or the removal ($\eta_t = -1$) of the pattern $\boldsymbol{y}_k$ is performed according to the rule (2) under the conditions stated in (3) (with $I_k^t \in \{0,1\}$). Crucial in this respect is the validity of $\delta b > R^2$. The r.h.s. of (17) was shown in (Crammer et al., 2003) to be an upper bound on the variable cache size of the Budget Perceptron but no mistake bound was obtained.

Now we relax the assumption of linear separability of the training set in the feature space and we move on to the analysis of the MPU algorithm with $I < \infty$.

Theorem 2 *The Margin Perceptron with Unlearning converges in a finite number t_c of updates satisfying the bound*

$$t_c \leq NI\frac{2b+\delta b}{\delta b - R^2}. \tag{22}$$

Let us consider the "objective" function

$$\mathcal{J}(\boldsymbol{w}, C_p) \equiv \frac{1}{2}\|\boldsymbol{w}\|^2 + C_p \sum_k \max\{1 - \boldsymbol{w}\cdot\boldsymbol{y}_k, 0\},$$

where $C_p > 0$ is a "penalty" parameter and

$$\mathcal{J}_{\rm opt}(C_p) \equiv \min_{\boldsymbol{w}} \mathcal{J}(\boldsymbol{w}, C_p)$$

its minimal value with C_p fixed. Then, with $\boldsymbol{a}_{t_c}$ being the solution weight vector and provided $b > \delta b$

$$\frac{\delta\mathcal{J}}{\mathcal{J}_{\rm opt}} \equiv \frac{\mathcal{J}(\boldsymbol{a}_{t_c}/b, I/b) - \mathcal{J}_{\rm opt}(I/b)}{\mathcal{J}_{\rm opt}(I/b)} < \delta \equiv \frac{2\delta b/b}{1-\delta b/b}. \tag{23}$$

Moreover, $\frac{\delta\mathcal{J}}{\mathcal{J}_{\rm opt}}$ satisfies the after-running bound

$$\frac{\delta\mathcal{J}}{\mathcal{J}_{\rm opt}} \leq \delta_{\rm aft} \equiv \frac{\frac{1}{2}\|\boldsymbol{a}_{t_c}\|^2 + I\sum_k \max\{b - \boldsymbol{a}_{t_c}\cdot\boldsymbol{y}_k, 0\}}{b(t_c^+ - t_c^-) - \frac{1}{2}\|\boldsymbol{a}_{t_c}\|^2} - 1. \tag{24}$$

Proof From (5) and the restriction $I_k^t \leq I$ we have

$$t^+ - t^- = \sum_k I_k^t \leq NI. \tag{25}$$

Also, from (14) and taking into account (25) we get

$$2(\delta b - R^2)t^- \leq (R^2+2b)(t^+ - t^-) \leq NI(R^2+2b)$$

or

$$t^- \leq NI\frac{R^2+2b}{2(\delta b - R^2)}$$

which combined with (25) leads to the upper bound on the number t of updates given by the r.h.s. of (22).

Let us assume that the algorithm has converged in t_c updates to the solution weight vector $\boldsymbol{a}_{t_c}$. We define

$$\boldsymbol{w}_c = \frac{\boldsymbol{a}_{t_c}}{b},\ \ \alpha_k^c = \frac{I_k^{t_c}}{b}\ \ (0 \leq \alpha_k^c \leq \frac{I}{b}),\ \ \epsilon = \frac{\delta b}{b} < 1$$

and notice that only the α_k^c's with k from the sets

$$\begin{aligned}\mathcal{K}_1 &= \{k:\ \alpha_k^c > 0,\ \ 1 < \boldsymbol{w}_c\cdot\boldsymbol{y}_k < 1+\epsilon\}\\ \mathcal{K}_2 &= \{k:\ \alpha_k^c = I/b,\ \ \boldsymbol{w}_c\cdot\boldsymbol{y}_k \leq 1\}\end{aligned} \tag{26}$$

are non-zero and consequently

$$\boldsymbol{w}_c = \sum_k \alpha_k^c \boldsymbol{y}_k = \sum_{k\in\mathcal{K}_1} \alpha_k^c \boldsymbol{y}_k + \sum_{k\in\mathcal{K}_2} \alpha_k^c \boldsymbol{y}_k.$$

Then,

$$\begin{aligned}\|\boldsymbol{w}_c\|^2 &= \boldsymbol{w}_c\cdot\left(\sum_k \alpha_k^c \boldsymbol{y}_k\right) = \sum_k \alpha_k^c \boldsymbol{w}_c\cdot\boldsymbol{y}_k\\ &= \sum_{k\in\mathcal{K}_1} \alpha_k^c \boldsymbol{w}_c\cdot\boldsymbol{y}_k + \sum_{k\in\mathcal{K}_2} \alpha_k^c \boldsymbol{w}_c\cdot\boldsymbol{y}_k\\ &< (1+\epsilon)\sum_{k\in\mathcal{K}_1}\alpha_k^c + \sum_{k\in\mathcal{K}_2}\alpha_k^c - \sum_{k\in\mathcal{K}_2}\alpha_k^c + \sum_{k\in\mathcal{K}_2}\alpha_k^c \boldsymbol{w}_c\cdot\boldsymbol{y}_k\\ &< (1+\epsilon)\left(\sum_{k\in\mathcal{K}_1}\alpha_k^c + \sum_{k\in\mathcal{K}_2}\alpha_k^c\right) - \sum_{k\in\mathcal{K}_2}\alpha_k^c\,(1 - \boldsymbol{w}_c\cdot\boldsymbol{y}_k)\\ &= (1+\epsilon)\sum_k \alpha_k^c - \frac{I}{b}\sum_k \max\{1 - \boldsymbol{w}_c\cdot\boldsymbol{y}_k, 0\}.\end{aligned} \tag{27}$$

Now, taking into account (27) we have

$$\begin{aligned}\mathcal{J}(\boldsymbol{w}_c, I/b) &= \|\boldsymbol{w}_c\|^2 - \frac{1}{2}\|\boldsymbol{w}_c\|^2\\ &\quad + \frac{I}{b}\sum_k \max\{1 - \boldsymbol{w}_c\cdot\boldsymbol{y}_k, 0\}\\ &= \|\boldsymbol{w}_c\|^2 - \frac{1}{2}\sum_{i,j}\alpha_i^c\alpha_j^c \boldsymbol{y}_i\cdot\boldsymbol{y}_j\\ &\quad + \frac{I}{b}\sum_k \max\{1 - \boldsymbol{w}_c\cdot\boldsymbol{y}_k, 0\}\\ &< (1+\epsilon)\sum_k \alpha_k^c - \frac{1}{2}\sum_{i,j}\alpha_i^c\alpha_j^c \boldsymbol{y}_i\cdot\boldsymbol{y}_j.\end{aligned} \tag{28}$$

From the weak duality theorem it follows that

$$\mathcal{J}(\boldsymbol{w}, C_{\rm p}) \geq \mathcal{L}(\boldsymbol{\alpha}) = \sum_k \alpha_k - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j \boldsymbol{y}_i \cdot \boldsymbol{y}_j,$$

where $\mathcal{L}(\boldsymbol{\alpha})$ is the dual Lagrangian and $0 \leq \alpha_k \leq C_{\rm p}$. Thus, setting $C_{\rm p} = I/b$ and $\alpha_k = \alpha_k^{\rm c}$ we get

$$\mathcal{J}(\boldsymbol{w}_{\rm c}, I/b) \geq \mathcal{J}_{\rm opt}(I/b) \geq \sum_k \alpha_k^{\rm c} - \frac{1}{2}\sum_{i,j} \alpha_i^{\rm c} \alpha_j^{\rm c} \boldsymbol{y}_i \cdot \boldsymbol{y}_j. \tag{29}$$

Combining (28) with (29) we obtain

$$\mathcal{J}(\boldsymbol{w}_{\rm c}, I/b) - \mathcal{J}_{\rm opt}(I/b) < \epsilon \sum_k \alpha_k^{\rm c}. \tag{30}$$

Moreover, taking into account (27) we get

$$\sum_k \alpha_k^{\rm c} - \frac{1}{2}\sum_{i,j} \alpha_i^{\rm c} \alpha_j^{\rm c} \boldsymbol{y}_i \cdot \boldsymbol{y}_j = \sum_k \alpha_k^{\rm c} - \frac{1}{2}\left\|\boldsymbol{w}_{\rm c}\right\|^2$$
$$> \sum_k \alpha_k^{\rm c} - \frac{1}{2}(1+\epsilon)\sum_k \alpha_k^{\rm c} = \frac{1}{2}(1-\epsilon)\sum_k \alpha_k^{\rm c}$$

which combined with (29) leads to

$$\mathcal{J}_{\rm opt}(I/b) > \frac{1}{2}(1-\epsilon)\sum_k \alpha_k^{\rm c}. \tag{31}$$

From (30) and (31) we immediately obtain (23).

Finally, using $\sum_k \alpha_k^{\rm c} = \sum_k I_k^{t_{\rm c}}/b = (t_{\rm c}^+ - t_{\rm c}^-)/b$, the definition of $\mathcal{J}$ and the lower bound on $\mathcal{J}_{\rm opt}$ from (29) we easily derive (24). □

Remark 4 From the before-running "accuracy" δ of (23) it follows that as b grows with δb kept fixed the MPU algorithm is guaranteed to converge to solutions for which the objective function $\mathcal{J}(\boldsymbol{a}_{t_{\rm c}}/b, I/b)$ approaches as close as one wishes the optimal one $\mathcal{J}_{\rm opt}(C_{\rm p})$ with $C_{\rm p} = I/b$. Thus, if we allow only b to grow while keeping δb and I fixed we will asymptotically approach an optimal objective with vanishing penalty parameter $C_{\rm p}$. If, instead, our goal is to approach the optimal objective $\mathcal{J}_{\rm opt}(C_{\rm p})$ with a given value of $C_{\rm p}$ we should follow a limiting process in which both I and b increase with their ratio kept all the way equal to the desirable value of $C_{\rm p}$. A simple way of doing so is to choose first a sufficiently large value for the integer I and then determine b from the relation $b = I/C_{\rm p}$ with the value of δb kept fixed. To select the appropriate value of I which ensures before running the desired accuracy δ we should, of course, express δ in terms of I and $C_{\rm p}$ as $\delta = 2C_{\rm p}\delta b I^{-1}\left(1 - C_{\rm p}\delta b I^{-1}\right)^{-1}$ from where $I = [C_{\rm p}\delta b(2+\delta)\delta^{-1}] + 1$ (with $[x]$ the integer part of $x \geq 0$). We conclude that the size of the upper bound I on the dual variables in the MPU algorithm does not depend on the penalty parameter $C_{\rm p}$ alone but on the required accuracy δ as well.

4. Implementation

To reduce the computational cost we construct a two-member sequence of reduced "active sets" of data points with the second being a subset of the first. As we cycle once through the full dataset the largest active set (first-level active set) is formed from the points of the full dataset either satisfying the condition $\boldsymbol{a}_t \cdot \boldsymbol{y}_k \leq \bar{c}b$ with $\bar{c} \geq 1$ or having $I_k^t > 0$. The second-level active set is built as we cycle once through the first-level active set and comprises the points $\boldsymbol{y}_k$ for which I_k^t undergoes a change and remains positive. Of course, as we cycle through the full dataset or the first-level active set along with the selection of points which form the first-level or the second-level active set we do perform updates whenever a point satisfies the conditions either for learning or unlearning. The second-level active set is presented repetitively to the algorithm for $N_{\rm ep_2}$ mini-epochs. Then, the first-level active set becomes the set under consideration which is presented once again to the algorithm. During each round involving the first-level set, a new second-level set is constructed and a new cycle of $N_{\rm ep_2}$ passes begins. By invoking the first-level set for the $(N_{\rm ep_1}+1)^{\rm th}$ time we trigger the loading of the full dataset and the procedure starts all over again until no update occurs as we run over the full dataset. It is understood, of course, that the maximum number of rounds $N_{\rm ep_1}$ and $N_{\rm ep_2}$ for each active set is not exhausted in the case that no update occurs during a round. Finally, following a common practice every time we make use of the full dataset we actually employ a permuted instance of it since presenting the points to the algorithm in a different order helps avoiding the disorientation caused by spurious patterns in the sequence of examples. Evidently, the whole procedure amounts to a different way of sequentially presenting the patterns to the algorithm and does not affect the validity of our analysis.

An additional mechanism providing a very substantial improvement of the computational efficiency is the one of performing multiple updates once a data point is presented to the algorithm[1]. It is understood, of course, that in order for a multiple update to be compatible with our theoretical analysis it should be equivalent to a certain number of updates occuring as a result of repeatedly presenting to the algorithm the point in question. In the case of learning we are allowed to perform $\min\{\lambda, I - I_k^t\}$ learning updates at once with $\lambda = [(b - \boldsymbol{a}_t \cdot \boldsymbol{y}_k)\left\|\boldsymbol{y}_k\right\|^{-2}] + 1$. Indeed, for any positive integer $n < \min\{\lambda, I - I_k^t\} \leq \lambda$ and $\boldsymbol{a}_{t+n} = \boldsymbol{a}_t + n\boldsymbol{y}_k$ it holds that $\boldsymbol{a}_{t+n} \cdot \boldsymbol{y}_k = \boldsymbol{a}_t \cdot \boldsymbol{y}_k + n\left\|\boldsymbol{y}_k\right\|^2 \leq \boldsymbol{a}_t \cdot \boldsymbol{y}_k +$

[1]The MPU with multiple updates bears some resemblance to the algorithm of (Hsieh et al., 2008).

Table 1. Results of a high accuracy comparative study of the MPU, DCD, SVM$^{\text{perf}}$ and Pegasos algorithms.

data set	MPU		DCD			SVM$^{\text{perf}}$			Pegasos		
	$\mathcal{J}$	Secs	ϵ	$\mathcal{J}$	Secs	ϵ	$\mathcal{J}$	Secs	iter	$\mathcal{J}$	Secs
Adult	11434.4	0.4	0.06	11434.4	0.3	0.004	11434.4	9.3	3×10^7	11434.9	397.0
Web	6605.17	0.1	0.07	6605.15	0.1	0.001	6605.22	11.1	2×10^7	6605.31	199.7
Physics	49643.6	0.4	0.05	49642.6	2.0	0.02	49644.1	2.2	5×10^5	49644.2	13.3
Real-sim	5402.25	0.5	0.01	5402.22	0.4	0.001	5402.58	6.1	2×10^7	5402.53	395.3
News20	2562.57	4.9	0.003	2562.57	4.2	0.001	2562.71	659.6	1×10^7	2562.69	1673.0
Webspam	69592.1	3.1	0.07	69591.4	2.8	0.001	69592.9	10.3	9×10^6	69599.6	297.7
Covertype	340490	22.2	0.08	340491	46.6	0.008	340493	505.4	1×10^8	340505	2447.9
C11	44725.4	5.7	0.02	44725.3	5.1	0.001	44730.5	29.2	1×10^8	44730.3	2743.6
CCAT	92990.2	8.1	0.02	92989.3	7.9	0.001	92992.8	54.3	1×10^8	93000.5	3103.6

$(\lambda-1)\left\|\boldsymbol{y}_k\right\|^2 \leq \boldsymbol{a}_t \cdot \boldsymbol{y}_k + (b - \boldsymbol{a}_t \cdot \boldsymbol{y}_k)\left\|\boldsymbol{y}_k\right\|^{-2}\left\|\boldsymbol{y}_k\right\|^2 = b$ meaning that after n consecutive updates the pattern $\boldsymbol{y}_k$ is still misclassified with $I_k^{t+n} = I_k^t + n < I$ and consequently one more learning update involving that pattern could legitimately take place. Similarly, in the case of unlearning we can show that we are allowed to perform $\min\{\lambda, I_k^t\}$ unlearning updates at once with $\lambda = [(\boldsymbol{a}_t \cdot \boldsymbol{y}_k - (b + \delta b))\left\|\boldsymbol{y}_k\right\|^{-2}] + 1$. Obviously, the gain from the mechanism of multiple updates lies in the fact that at the expense of a single inner product calculation we effectively perform numerous updates.

Finally, we point out that we may terminate the MPU algorithm with $I < \infty$ before it converges when the after-running accuracy δ_{aft} entering (24) (with $\boldsymbol{a}_{t_{\text{c}}}$ and $(t_{\text{c}}^+ - t_{\text{c}}^+)$ now being the final weight vector and the final value of $(t^+ - t^-)$, respectively) falls below a certain desirable level δ_{stop} at the end of a complete cycle through the full dataset. Notice, that the derivation of (24) relies only on the validity of (5) and (9) which are always respected by the algorithm. Moreover, if the guaranteed before-running accuracy δ of (23) is equal or smaller than δ_{stop} it is obvious that, despite the early stopping, the algorithm will necessarily achieve the required accuracy δ_{stop} in at most the number of updates given by the r.h.s. of (22). Similarly, if $I = \infty$ we may terminate the MPU algorithm when the lower bound of (12) on f falls below the r.h.s. of (11).

5. Experiments

We concentrate on the more popular L1-SVM problem and compare the MPU algorithm with SVMs on the basis of their ability to arrive fast at a certain approximation of the optimal 1-norm soft margin solution. Moreover, in order to be able to train on rather large datasets we choose as feature space the initial instance space. Recently, a lot of effort has been devoted to developing algorithms which are able to train fast on large datasets provided linear kernels are employed. Among such algorithms we choose for our experiments the Dual Coordinate Descent (DCD) algorithm (Hsieh et al., 2008), SVM$^{\text{perf}}$ (Joachims, 2006), the first cutting-plane algorithm for training linear SVMs and Pegasos (Shalev-Schwartz et al., 2007), a stochastic gradient descent algorithm. The source for DCD (version 1.5) is available at `http://www.csie.ntu.edu.tw/~cjlin/liblinear`, for SVM$^{\text{perf}}$ (version 2.50) at `http://smvlight.joachims.org` and for Pegasos at `http://ttic.uchicago.edu/~shai/code`.

The datasets we used for training are the binary Adult (N=32561, M=123) and Web (N=49749, M=300) UCI datasets as compiled by Platt, the training set of the KDD04 Physics dataset (N=50000, M=70 after removing the 8 columns containing missing features) obtainable from `http://kodiak.cs.cornell.edu/kddcup/datasets.html`, the Real-sim (N=72309, M=20958), News20 (N=19996, M=1355191) and Webspam (unigram treatment with N=350000 and M=254) datasets all available at `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets`, the multiclass Covertype (N=581012, M=54) UCI dataset and the full Reuters RCV1 dataset (N=804414, M=47236) obtainable from `http://www.jmlr.org/papers/volume5/lewis04a/lyrl2004_rcv1v2_README.htm`. In the case of the Covertype dataset we study the binary classification problem of the first class versus rest while for the RCV1 we consider both the binary text classification tasks of the C11 and CCAT classes versus rest. The Physics and Covertype datasets were rescaled by multiplying all the features with 0.001. The experiments were conducted on a 2.5 GHz Intel Core 2 Duo processor with 3 GB RAM running Windows Vista. Our codes written in C++ were compiled using the g++ compiler under Cygwin.

In our experiments we chose $C_{\text{p}} = 1$. Also, we did

Table 2. Results of a low (1%) accuracy comparative study of the MPU, DCD, SVM$^{\text{perf}}$ and Pegasos algorithms. We report training times in secs keeping the dataset ordering of Table 1.

MPU	0.1	0.1	0.1	0.3	1.6	1.2	6.2	3.1	3.7
DCD	0.2	0.1	0.1	0.2	0.6	1.4	9.1	3.5	3.6
SVM$^{\text{perf}}$	1.5	0.3	0.3	0.8	12.8	7.4	76.5	12.9	19.3
Pegasos	4.0	1.0	0.3	3.2	10.6	13.7	141.9	27.9	31.6

not include any bias term (i.e. we did not perform any augmentation). The implementation of the MPU algorithm followed closely the description of Section 4 with active set parameters $\bar{c} = 1.01$, $N_{\text{ep}_1} = 3$ and $N_{\text{ep}_2} = 10$. Moreover, we did make use of the mechanism of early stopping. We also assumed $\delta b = 3R^2$ throughout. For Pegasos we chose $k = 30$ and $\lambda = N^{-1}$ (since $(\lambda N)^{-1}$ plays the role of C_{p}). In the first experiment our procedure was to obtain a very good approximation of the optimal objective $\mathcal{J}_{\text{opt}}$ (i.e. $\frac{\delta\mathcal{J}}{\mathcal{J}_{\text{opt}}} \leq 10^{-4}$) using MPU with $\delta = 10^{-5}$ and $\delta_{\text{stop}} = 10^{-4}$ and then try to achieve comparable values of $\mathcal{J}$ with the other linear SVM solvers. For DCD and SVM$^{\text{perf}}$ we employed the "accuracy" ϵ (≤ 0.1) whereas for Pegasos the number of iterations iter in order to obtain the required value of $\mathcal{J}$. Table 1 contains our experimental results (objective $\mathcal{J}$ and training runtime) for this high accuracy comparative study. SVM$^{\text{perf}}$ is clearly considerably slower than DCD and MPU but much faster than Pegasos. DCD, instead, appears to be statistically the fastest but still its runtimes are very close to the ones of MPU. We also carried out a second experiment aiming at obtaining a value of $\mathcal{J}$ which deviates from $\mathcal{J}_{\text{opt}}$ by 1%. The results of this low accuracy study are reported in Table 2. We see that DCD and MPU are still the fastest with close runtimes followed by SVM$^{\text{perf}}$. The remarkable finding is, however, the spectacular improvement that Pegasos presents. In the light of our results we may conclude that the MPU algorithm is very competitive.

6. Conclusions

By enriching the classical Margin Perceptron with a mechanism of unlearning which allows the algorithm to recover from any wrong decisions taken on the way we succeded in tackling both the hard margin and the 1-norm soft margin problems. We derived upper bounds on the number of steps required in order for the algorithm to achieve any desirable fixed before running approximation of the optimal solution for both tasks and described strategies for its efficient implementation. Finally, the experiments provided evidence that the new algorithm is a very competitive linear SVM.

References

Boser, B., Guyon, I., and Vapnik, V. A training algorithm for optimal margin classifiers. In *COLT*, pp. 144–152, 1992.

Cortes, C. and Vapnik, V. Support vector networks. *Machine Learning*, 20:273–297, 1995.

Crammer, K., Kandola, J., and Singer, Y. Online classification on a budget. In *NIPS 16*, 2003.

Cristianini, N. and Shawe-Taylor, J. *An Introduction to Support Vector Machines*. Cambridge University Press, 2000.

Duda, R. O. and Hart, P. E. *Pattern Classsification and Scene Analysis*. Wiley, 1973.

Gentile, C. A new approximate maximal margin classification algorithm. *JMLR*, 2:213–242, 2001.

Guyon, I. and Stork, D. Linear discriminant and support vector classifiers. In *Advances in Large Margin Classifiers*. MIT Press, 1999.

Hsieh, C.-J., Chang, K.-W., Lin, C.-J., Keerthi, S. S., and Sundararajan, S. A dual coordinate descent method for large-scale linear SVM. In *ICML*, pp. 408–415, 2008.

Joachims, T. Training linear SVMs in linear time. In *KDD'06*, pp. 217–226. ACM Press, 2006.

Krauth, W. and Mézard, M. Learning algorithms with optimal stability in neural networks. *Journal of Physics A*, 20:L745–L752, 1987.

Li, Y. and Long, P. The relaxed online maximum margin algorithm. *Machine Learning*, 46:361–387, 2002.

Novikoff, A. B. J. On convergence proofs on perceptrons. In *Proc. Symp. Math. Theory Automata, Vol. 12*, pp. 615–622, 1962.

Rosenblatt, F. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65 (6):386–408, 1958.

Shalev-Schwartz, S., Singer, Y., and Srebro, N. Pegasos: Primal estimated sub-gradient solver for SVM. In *ICML*, pp. 807–814, 2007.

Tsampouka, P. and Shawe-Taylor, J. Approximate maximum margin algorithms with rules controlled by the number of mistakes. In *ICML*, pp. 903–910, 2007.

Vapnik, V. *Statistical Learning Theory*. Wiley, 1998.