

Step Size Adaptation in Reproducing Kernel Hilbert Space

S.V. N. Vishwanathan
Nicol N. Schraudolph
Alex J. Smola

SVN.VISHWANATHAN@NICTA.COM.AU
NIC.SCHRAUDOLPH@NICTA.COM.AU
ALEX.SMOLA@NICTA.COM.AU

*Statistical Machine Learning Program,
National ICT Australia, Locked Bag 8001,
Canberra ACT 2601, Australia*

*Research School of Information Sciences and Engineering,
Australian National University,
Canberra ACT 0200, Australia*

Editor: Thorsten Joachims

Abstract

This paper presents an online Support Vector Machine (SVM) that uses the Stochastic Meta-Descent (SMD) algorithm to adapt its step size automatically. We formulate the online learning problem as a stochastic gradient descent in Reproducing Kernel Hilbert Space (RKHS) and translate SMD to the nonparametric setting, where its gradient trace parameter is no longer a coefficient vector but an element of the RKHS. We derive efficient updates that allow us to perform the step size adaptation in linear time. We apply the online SVM framework to a variety of loss functions, and in particular show how to handle structured output spaces and achieve efficient online multi-class classification. Experiments show that our algorithm outperforms more primitive methods for setting the gradient step size.

Keywords: Online SVM, Stochastic Meta-Descent, Structured Output Spaces.

1. Introduction

Stochastic (“online”) gradient methods incrementally update their hypothesis by descending a stochastic approximation of the gradient computed from just the current observation. Although they require more iterations to converge than traditional deterministic (“batch”) techniques, each iteration is faster as there is no need to go through the entire training set to measure the current gradient. For large, redundant data sets, or continuing (potentially non-stationary) streams of data, stochastic gradient thus outperforms classical optimization methods. Much work in this area centers on the key issue of choosing an appropriate time-dependent gradient step size η_t .

Though support vector machines (SVMs) were originally conceived as batch techniques with time complexity quadratic to cubic in the training set size, recent years have seen the development of online variants (Herbster, 2001; Kivinen et al., 2004; Crammer et al., 2004; Weston et al., 2005; Kim et al., 2005) which overcome this limitation. To date, online kernel methods based on stochastic gradient descent (Kivinen et al., 2004; Kim et al., 2005) have either held η_t constant, or let it decay according to some fixed schedule. Here we adopt the more sophisticated approach of *stochastic*

meta-descent (SMD): performing a simultaneous stochastic gradient descent on the step size itself. Translating this into the kernel framework yields a fast online optimization method for SVMs.

Outline. In Section 2 we review gradient-based step size adaptation algorithms so as to motivate our subsequent derivation of SMD. We briefly survey kernel-based online methods in Section 3, then present the online SVM algorithm with a systematic, unified view of various loss functions (including losses on structured label domains) in Section 4. Section 5 then introduces online SVMD, our novel application of SMD to the online SVM. Here we also derive linear-time incremental updates and standard SVM extensions for SVMD, and discuss issues of buffer management and time complexity. Experiments comparing SVMD to the online SVM are then presented in Section 6, followed by a discussion.

2. Stochastic Meta-Descent

The SMD algorithm (Schraudolph, 1999, 2002) for gradient step size adaptation can considerably accelerate the convergence of stochastic gradient descent; its applications to date include independent component analysis (Schraudolph and Giannakopoulos, 2000), nonlinear principal component analysis in computational fluid dynamics (Milano, 2002), visual tracking of articulated objects (Bray et al., 2005, 2007), policy gradient reinforcement learning (Schraudolph et al., 2006), and training of conditional random fields (Vishwanathan et al., 2006).

2.1 Gradient-Based Step Size Adaptation

Let V be a vector space, $\theta \in V$ a parameter vector, and $J : V \rightarrow \mathbb{R}$ the objective function which we would like to optimize. We assume that J is twice differentiable almost¹ everywhere. Denote by $J_t : V \rightarrow \mathbb{R}$ the stochastic approximation of the objective function at time t . Our goal is to find θ such that $\mathbb{E}_t[J_t(\theta)]$ is minimized. An adaptive version of stochastic gradient descent works by setting

$$\theta_{t+1} = \theta_t - \eta_t \cdot g_t, \quad \text{where } g_t = \partial_{\theta_t} J_t(\theta_t), \quad (1)$$

using ∂_{θ_t} as a shorthand for $\frac{\partial}{\partial \theta} \Big|_{\theta=\theta_t}$. Unlike conventional gradient descent algorithms where η_t is scalar, here $\eta_t \in \mathbb{R}_+^n$, and \cdot denotes component-wise (Hadamard) multiplication. In other words, each coordinate of θ has its own positive step size that serves as a diagonal conditioner. Since we need to choose suitable values we shall adapt η by a simultaneous meta-level gradient descent.

A straightforward implementation of this idea is the *delta-delta* algorithm (Sutton, 1981; Jacobs, 1988), which updates η via

$$\begin{aligned} \eta_{t+1} &= \eta_t - \mu \partial_{\eta_t} J_{t+1}(\theta_{t+1}) \\ &= \eta_t - \mu \partial_{\theta_{t+1}} J_{t+1}(\theta_{t+1}) \cdot \partial_{\eta_t} \theta_{t+1} \\ &= \eta_t + \mu g_{t+1} \cdot g_t, \end{aligned} \quad (2)$$

where $\mu \in \mathbb{R}$ is a scalar meta-step size. In a nutshell, step sizes are decreased where a negative auto-correlation of the gradient indicates oscillation about a local minimum, and increased otherwise. Unfortunately such a simplistic approach has several problems:

1. Since gradient descent implements a discrete approximation to an infinitesimal (differential) process in any case, we can in practice ignore non-differentiability of J on a set of measure zero, as long as our implementation of the gradient function returns a subgradient at those points.

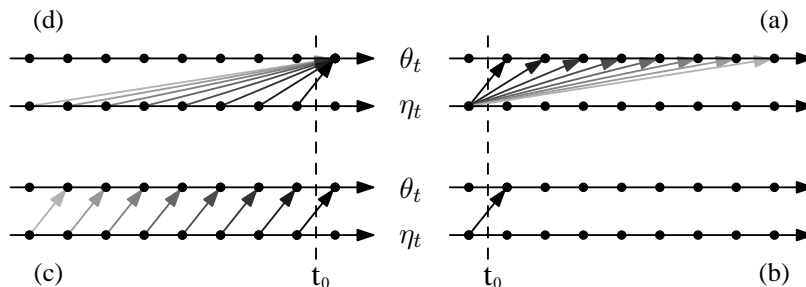


Figure 1: Dependence of a parameter θ on its step size η at time t_0 . (a) Future parameter values depend on the current step size; the dependence diminishes over time due to the ongoing adaptation of η . (b) Standard step size adaptation methods capture only the immediate effect, even when (c) past gradients are exponentially smoothed. (d) SMD, by contrast, iteratively models the dependence of the current parameter on an exponentially weighted past history of step sizes, thereby capturing long-range effects. Figure adapted from Bray et al. (2005).

Firstly, (2) allows step sizes to become negative. This can be avoided by updating η multiplicatively, e.g. via *exponentiated gradient descent* (Kivinen and Warmuth, 1997).

Secondly, delta-delta’s cure is worse than the disease: individual step sizes are meant to address ill-conditioning, but (2) actually squares the condition number. The auto-correlation of the gradient must therefore be normalized before it can be used. A popular (if extreme) form of normalization is to consider only the sign of the auto-correlation. Such sign-based methods (Jacobs, 1988; Tollenaere, 1990; Silva and Almeida, 1990; Riedmiller and Braun, 1993), however, do not cope well with stochastic approximation of the gradient since the non-linear sign function does not commute with the expectation operator (Almeida et al., 1999). More recent algorithms (Harmon and Baird, 1996; Almeida et al., 1999; Schraudolph, 1999, 2002) therefore use multiplicative (hence linear) normalization factors to condition the step size update.

Finally, (2) fails to take into account that changes in step size not only affect the current, but also future parameter updates (see Figure 1). In recognition of this shortcoming, g_t in (2) is usually replaced with an exponential running average of past gradients (Jacobs, 1988; Tollenaere, 1990; Silva and Almeida, 1990; Riedmiller and Braun, 1993; Almeida et al., 1999). Although such ad-hoc smoothing does improve performance, it does not properly capture long-term dependencies, the average still being one of immediate, single-step effects (Figure 1c).

By contrast, Sutton (1992) modeled the long-term effect of step sizes on future parameter values in a linear system by carrying the relevant partials forward in time, and found that the resulting step size adaptation can outperform a less than perfectly matched Kalman filter. Stochastic meta-descent (SMD) extends this approach to arbitrary twice-differentiable nonlinear systems, takes into account the full Hessian instead of just the diagonal, and applies an exponential decay to the partials being carried forward (Figure 1d).

2.2 SMD Algorithm

SMD employs two modifications to address the problems described above: it adjusts step sizes in log-space, and optimizes over an exponentially decaying trace of gradients. Thus $\log \eta$ is updated

as follows:

$$\begin{aligned}
 \log \boldsymbol{\eta}_{t+1} &= \log \boldsymbol{\eta}_t - \mu \sum_{i=0}^t \lambda^i \partial_{\log \boldsymbol{\eta}_{t-i}} J(\boldsymbol{\theta}_{t+1}) \\
 &= \log \boldsymbol{\eta}_t - \mu \partial_{\boldsymbol{\theta}_{t+1}} J(\boldsymbol{\theta}_{t+1}) \cdot \sum_{i=0}^t \lambda^i \partial_{\log \boldsymbol{\eta}_{t-i}} \boldsymbol{\theta}_{t+1} \\
 &= \log \boldsymbol{\eta}_t - \mu \mathbf{g}_{t+1} \cdot \mathbf{v}_{t+1},
 \end{aligned} \tag{3}$$

where the vector $\mathbf{v} \in V$ characterizes the long-term dependence of the system parameters on their past step sizes over a time scale governed by the decay factor $0 \leq \lambda \leq 1$.

Note that virtually the same derivation holds if—as will be the case in Section 5—we wish to adapt only a single, scalar step size η_t for all system parameters; the only change necessary is to replace the Hadamard product in (3) with an inner product.

Element-wise exponentiation of (3) yields the desired multiplicative update

$$\begin{aligned}
 \boldsymbol{\eta}_{t+1} &= \boldsymbol{\eta}_t \cdot \exp(-\mu \mathbf{g}_{t+1} \cdot \mathbf{v}_{t+1}) \\
 &\approx \boldsymbol{\eta}_t \cdot \max\left(\frac{1}{2}, 1 - \mu \mathbf{g}_{t+1} \cdot \mathbf{v}_{t+1}\right),
 \end{aligned} \tag{4}$$

where the approximation eliminates an expensive exponentiation operation for each step size update. The particular bi-linearization we use, $e^u \approx \max(\frac{1}{2}, 1 + u)$,

- matches the exponential in value and first derivative at $u = 0$, and thus becomes accurate in the limit of small μ ;
- ensures that all elements of $\boldsymbol{\eta}$ remain strictly positive; and
- improves robustness by reducing the effect of outliers: $u \gg 0$ leads to linear² rather than exponential growth in step sizes, while for $u \ll 0$ they are at most cut in half.

The choice of $\frac{1}{2}$ as the lower bound stems from the fact that a gradient descent converging on a minimum of a differentiable function can overshoot that minimum by at most a factor of two, since otherwise it would by definition be divergent. A reduction by at most $\frac{1}{2}$ in step size thus suffices to maintain stability from one iteration to the next.

To compute the gradient trace \mathbf{v} efficiently, we expand $\boldsymbol{\theta}_{t+1}$ in terms of its recursive definition (1):

$$\begin{aligned}
 \mathbf{v}_{t+1} &:= \sum_{i=0}^t \lambda^i \partial_{\log \boldsymbol{\eta}_{t-i}} \boldsymbol{\theta}_{t+1} \\
 &= \sum_{i=0}^t \lambda^i \partial_{\log \boldsymbol{\eta}_{t-i}} \boldsymbol{\theta}_t - \sum_{i=0}^t \lambda^i \partial_{\log \boldsymbol{\eta}_{t-i}} (\boldsymbol{\eta}_t \cdot \mathbf{g}_t) \\
 &\approx \lambda \mathbf{v}_t - \boldsymbol{\eta}_t \cdot \mathbf{g}_t - \boldsymbol{\eta}_t \cdot \left[\partial_{\boldsymbol{\theta}_t} \mathbf{g}_t \sum_{i=0}^t \lambda^i \partial_{\log \boldsymbol{\eta}_{t-i}} \boldsymbol{\theta}_t \right]
 \end{aligned} \tag{5}$$

Here we have used $\partial_{\log \boldsymbol{\eta}_t} \boldsymbol{\theta}_t = 0$, and approximated

$$\sum_{i=1}^t \lambda^i \partial_{\log \boldsymbol{\eta}_{t-i}} \log \boldsymbol{\eta}_t \approx 0 \tag{6}$$

2. A quadratic approximation with similar properties would be $e^u \approx \begin{cases} \frac{1}{2} u^2 + u + 1 & \text{if } u > -1; \\ \frac{1}{2} & \text{otherwise.} \end{cases}$

which amounts to stating that the step size adaptation (in log space) must be in equilibrium at the time scale determined by λ . Noting that $\partial_{\theta_t} \mathbf{g}_t$ is the Hessian \mathbf{H}_t of $J_t(\boldsymbol{\theta}_t)$, we arrive at the simple iterative update

$$\mathbf{v}_{t+1} = \lambda \mathbf{v}_t - \boldsymbol{\eta}_t \cdot (\mathbf{g}_t + \lambda \mathbf{H}_t \mathbf{v}_t). \quad (7)$$

Since the initial parameters $\boldsymbol{\theta}_0$ do not depend on any step sizes, $\mathbf{v}_0 = \mathbf{0}$.

2.3 Efficiency and Conditioning

Although the Hessian \mathbf{H} of a system with n parameters has $O(n^2)$ entries, efficient indirect methods from algorithmic differentiation are available to compute its product with an arbitrary vector within the same time as 2–3 gradient evaluations (Pearlmutter, 1994; Griewank, 2000). For non-convex systems (where positive semi-definiteness of the Hessian cannot be guaranteed) SMD uses an extended Gauss-Newton approximation of \mathbf{H} for which a similar but even faster technique exists (Schraudolph, 2002). An iteration of SMD—comprising (1), (4), and (7)—thus consumes less than 3 times as many floating-point operations as simple gradient descent.

Iterating (7) while holding $\boldsymbol{\theta}$ and $\boldsymbol{\eta}$ constant would drive \mathbf{v} towards the fixpoint

$$\mathbf{v} \rightarrow -[\lambda \mathbf{H} + (1 - \lambda) \text{diag}(1/\boldsymbol{\eta})]^{-1} \mathbf{g}, \quad (8)$$

which is a Levenberg-Marquardt gradient step with a trust region conditioned by $\boldsymbol{\eta}$ and scaled by $1/(1 - \lambda)$. For $\lambda = 1$ this reduces to a Newton (resp. Gauss-Newton) step, which converges rapidly but may become unstable in regions of low curvature. In practice, we find that SMD performs best when λ is pushed as close to 1 as possible without losing stability.

Note that in this regime, the $\mathbf{g} \cdot \mathbf{v}$ term in (4) is approximately affine invariant, with the inverse curvature matrix in (8) compensating for the scale of the gradient auto-correlation. This means that the meta-step size μ is relatively problem-independent; in experiments we typically use values within an order of magnitude of $\mu = 0.1$. Likewise, well-adapted step sizes ($\boldsymbol{\eta} \cdot \mathbf{g} \approx \mathbf{H}^{-1} \mathbf{g}$) will condition the update not only of $\boldsymbol{\theta}$ (1) but also of \mathbf{v} (7). Thus SMD maintains an adaptive conditioning of all its updates, provided it is given reasonable initial step sizes $\boldsymbol{\eta}_0$ to begin with.

3. Survey of Online Kernel Methods

The Perceptron algorithm (Rosenblatt, 1958) is arguably one of the simplest online learning algorithms. Given a set of labeled instances $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_m, y_m)\} \subset \mathcal{X} \times \mathcal{Y}$ where $\mathcal{X} \subseteq \mathbb{R}^d$ and $y_i \in \{\pm 1\}$ the algorithm starts with an initial weight vector $\boldsymbol{\theta} = \mathbf{0}$. It then predicts the label of a new instance \mathbf{x} to be $\hat{y} = \text{sign}(\langle \boldsymbol{\theta}, \mathbf{x} \rangle)$. If \hat{y} differs from the true label y then the vector $\boldsymbol{\theta}$ is updated as $\boldsymbol{\theta} = \boldsymbol{\theta} + y\mathbf{x}$. This is repeated until all points are well classified. The following result bounds the number of mistakes made by the Perceptron algorithm (Freund and Schapire, 1999, Theorem 1):

Theorem 1 *Let $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_m, y_m)\}$ be a sequence of labeled examples with $\|\mathbf{x}_i\| \leq R$. Let $\boldsymbol{\theta}$ be any vector with $\|\boldsymbol{\theta}\| = 1$ and let $\gamma > 0$. Define the deviation of each example as*

$$d_i = \max(0, \gamma - y_i \langle \boldsymbol{\theta}, \mathbf{x}_i \rangle),$$

and let $D = \sqrt{\sum_i d_i^2}$. Then the number of mistakes of the Perceptron algorithm on this sequence is bounded by $(\frac{R+D}{\gamma})^2$.

This generalizes the original result (Block, 1962; Novikoff, 1962; Minsky and Papert, 1969) for the case when the points are strictly separable, *i.e.*, when there exists a $\boldsymbol{\theta}$ such that $\|\boldsymbol{\theta}\| = 1$ and $y_i \langle \boldsymbol{\theta}, \mathbf{x}_i \rangle \geq \gamma$ for all (\mathbf{x}_i, y_i) .

The so-called *kernel trick* has recently gained popularity in machine learning (Schölkopf and Smola, 2002). As long as all operations of an algorithm can be expressed with inner products, the kernel trick can be used to *lift* the algorithm to a higher-dimensional *feature space*: The inner product in the feature space produced by the mapping $\phi : \mathcal{X} \rightarrow \mathcal{H}$ is represented by a *kernel* $k(\mathbf{x}, \mathbf{x}') = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle_{\mathcal{H}}$. We can now drop the condition $\mathcal{X} \subseteq \mathbb{R}^d$ but instead require that \mathcal{H} be a Reproducing Kernel Hilbert Space (RKHS).

To *kernelize* the Perceptron algorithm, we first use ϕ to map the data into feature space, and observe that the weight vector can be expressed as $\boldsymbol{\theta} = \sum_{j \in \mathcal{J}} y_j \phi(\mathbf{x}_j)$, where \mathcal{J} is the set of indices where mistakes occurred. We can now compute $\langle \boldsymbol{\theta}, \mathbf{x}_i \rangle = \sum_{j \in \mathcal{J}} y_j \langle \phi(\mathbf{x}_j), \phi(\mathbf{x}_i) \rangle = \sum_{j \in \mathcal{J}} y_j k(\mathbf{x}_j, \mathbf{x}_i)$, replacing explicit computation of ϕ with kernel evaluations.

The main drawback of the Perceptron algorithm is that it does not maximize the margin of separation between the members of different classes. Frieß et al. (1998) address this issue with their closely related Kernel-Adatron (KA). The KA algorithm uses a weight vector $\boldsymbol{\theta} = \sum_i \alpha_i y_i \phi(\mathbf{x}_i)$. Initially all α_i are set to 1. For a new instance (\mathbf{x}, y) we compute $z = 1 - y \sum_i \alpha_i y_i K(\mathbf{x}_i, \mathbf{x})$, and update the corresponding α as $\alpha := \alpha + \eta z$ if $\alpha + \eta z > 0$; otherwise we set $\alpha = 0$.³ Frieß et al. (1998) show that if the data is separable, this algorithm converges to the maximum margin solution in a finite number of iterations, and that the error rate decreases exponentially with the number of iterations.

To address the case where the data is not separable in feature space, Freund and Schapire (1999) work with a kernelized Perceptron but use the online-to-batch conversion procedure of Helmbold and Warmuth (1995) to derive their Voted-Perceptron algorithm. Essentially, every weight vector generated by the kernel Perceptron is retained, and the decision rule is a majority vote amongst the predictions generated by these weight vectors. They prove the following mistake bound:

Theorem 2 (Freund and Schapire, 1999, Corollary 1) *Let $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_m, y_m)\}$ be a sequence of training samples and $(\mathbf{x}_{m+1}, y_{m+1})$ a test sample, all taken i.i.d. at random. Let $R = \max_{1 \leq i \leq m+1} \|\mathbf{x}_i\|$. For $\|\boldsymbol{\theta}\| = 1$ and $\gamma > 0$, let*

$$D_{\boldsymbol{\theta}, \gamma} = \sqrt{\sum_{i=1}^{m+1} (\max(0, \gamma - y_i \langle \boldsymbol{\theta}, \mathbf{x}_i \rangle)^2)}.$$

Then the probability (under resampling) that the Voted-Perceptron algorithm does not predict y_{m+1} on test sample \mathbf{x}_{m+1} after one pass through the sequence of training samples is at most

$$\frac{2}{m+1} \mathbb{E} \left[\inf_{\|\boldsymbol{\theta}\|=1; \gamma>0} \left(\frac{R + D_{\boldsymbol{\theta}, \gamma}}{\gamma} \right)^2 \right],$$

where \mathbb{E} denotes the expectation under resampling.

Another online algorithm which aims to maximize the margin of separation between classes is LASVM (Bordes et al., 2005). This follows a line of *budget* (kernel Perceptron) algorithms which

3. In the interest of a clear exposition we ignore the *offset* b here.

sport a removal step (Crammer et al., 2004; Weston et al., 2005). Briefly, LASVM tries to solve the SVM Quadratic Programming (QP) problem in an online manner. If the new instance violates the KKT conditions then it is added to the so-called *active set* during the PROCESS step. A REPROCESS step is run to identify points in the active set whose coefficients are constrained at either their upper or lower bound; such points are then discarded from the active set. Bordes et al. (2005) have shown that in the limit LASVM solves the SVM QP problem, although no rates of convergence or mistake bounds have been proven.

The Ballseptron is another variant of the Perceptron algorithm which takes the margin of separation between classes into account (Shalev-Shwartz and Singer, 2005). In contrast to the classic Perceptron, the Ballseptron updates its weight vector even for well-classified instances if they are close to the decision boundary. More precisely, if a ball $B(\mathbf{x}, r)$ of radius r around the instance \mathbf{x} intersects the decision boundary, the worst-violating point in B is used as a pseudo-instance for updating the weight vector. Shalev-Shwartz and Singer (2005) show that appropriate choice of r yields essentially the same bound as Theorem 1 above; this bound can be tightened further when the number of margin errors is strictly positive.

Another notable effort to derive a margin-based online learning algorithm is ALMA_p , the Approximate Large Margin Algorithm w.r.t. norm p (Gentile, 2001). Following Gentile and Littlestone (1999), the notion of a margin is extended to p -norms: Let $\mathbf{x}' = \mathbf{x}/\|\mathbf{x}\|_p$, and $\|\boldsymbol{\theta}\|_q \leq 1$, where $\frac{1}{p} + \frac{1}{q} = 1$. Then the p -margin of (\mathbf{x}, y) w.r.t. $\boldsymbol{\theta}$ is defined as $y_i \langle \boldsymbol{\theta}, \mathbf{x}' \rangle$. Like other Perceptron-inspired algorithms, ALMA_p does not perform an update if the current weight vector classifies the current instance with a large p -margin. If a margin violation occurs, however, the algorithm performs a p -norm Perceptron update, then projects the obtained $\boldsymbol{\theta}$ to the q -norm unit ball to maintain the constraint $\|\boldsymbol{\theta}\|_q \leq 1$. ALMA_p is one of the few Perceptron-derived online algorithms we know of which modify their learning rate: Its p -norm Perceptron update step scales with the number of corrections which have occurred so far. ALMA_p can be kernelized only for $p = 2$.

Many large-margin algorithms (Li and Long, 2002; Crammer and Singer, 2003; Herbster, 2001) are based on the same general principle: They explicitly maximize the margin and update their weights only when a margin violation occurs. These violating instances are inserted into the kernel expansion with a suitable coefficient. To avoid potential over-fitting and reduce computational complexity, these algorithms either implement a removal step or work with a fixed-size buffer. The online SVM (Kivinen et al., 2004) is one such algorithm.

4. Online SVM

We now present the online SVM (*aka* NORMA) algorithm (Kivinen et al., 2004) from a loss function and regularization point of view, with additions and modifications for logistic regression, novelty detection, multiclass classification, and graph-structured label domains. This sets the scene for our application of SMD to the online SVM in Section 5. While many of the loss functions discussed below have been proposed before, we present them here in a common, unifying framework that cleanly separates the roles of loss function and optimization algorithm.

4.1 Optimization Problem

Let \mathcal{X} be the space of observations, and \mathcal{Y} the space of labels. We use $|\mathcal{Y}|$ to denote the size of \mathcal{Y} . Given a sequence $\{(\mathbf{x}_i, y_i) | \mathbf{x}_i \in \mathcal{X}, y_i \in \mathcal{Y}\}$ of examples and a loss function $l : \mathcal{X} \times \mathcal{Y} \times \mathcal{H} \rightarrow \mathbb{R}$,

our goal is to minimize the regularized risk

$$J(f) = \frac{1}{m} \sum_{i=1}^m l(\mathbf{x}_i, y_i, f) + \frac{c}{2} \|f\|_{\mathcal{H}}^2, \tag{9}$$

where \mathcal{H} is a Reproducing Kernel Hilbert Space (RKHS) of functions on $\mathcal{X} \times \mathcal{Y}$. Its defining kernel is denoted by $k : (\mathcal{X} \times \mathcal{Y})^2 \rightarrow \mathbb{R}$, which satisfies $\langle f, k((\mathbf{x}, y), \cdot) \rangle_{\mathcal{H}} = f(\mathbf{x}, y)$ for all $f \in \mathcal{H}$. In a departure from tradition, but keeping in line with [Altun et al. \(2004\)](#); [Tsochantaridis et al. \(2004\)](#); [Cai and Hofmann \(2004\)](#), we let our kernel depend on the labels as well as the observations. Finally, we make the assumption that l only depends on f via its evaluations at $f(\mathbf{x}_i, y_i)$ and that l is piecewise differentiable.

By the reproducing property of \mathcal{H} we can compute derivatives of the evaluation functional. That is,

$$\partial_f f(\mathbf{x}, y) = \partial_f \langle f, k((\mathbf{x}, y), \cdot) \rangle_{\mathcal{H}} = k((\mathbf{x}, y), \cdot). \tag{10}$$

Since l depends on f only via its evaluations we can see that $\partial_f l(\mathbf{x}, y, f) \in \mathcal{H}$, and more specifically

$$\partial_f l(\mathbf{x}, y, f) \in \text{span}\{k((\mathbf{x}, \tilde{y}), \cdot) \text{ where } \tilde{y} \in \mathcal{Y}\}. \tag{11}$$

Let (\mathbf{x}_t, y_t) denote the example presented to the online algorithm at time instance t . Using the stochastic approximation of $J(f)$ at time t :

$$J_t(f) := l(\mathbf{x}_t, y_t, f) + \frac{c}{2} \|f\|_{\mathcal{H}}^2 \tag{12}$$

and setting

$$g_t := \partial_f J_t(f_t) = \partial_f l(\mathbf{x}_t, y_t, f_t) + c f_t, \tag{13}$$

we obtain the following online learning algorithm:

Algorithm 1 Online learning (adaptive step size)

1. Initialize $f_0 = 0$
 2. **Repeat**
 - (a) Draw data sample (\mathbf{x}_t, y_t)
 - (b) Adapt step size η_t
 - (c) Update $f_{t+1} \leftarrow f_t - \eta_t g_t$
-

Practical considerations are how to implement steps 2(b) and 2(c) efficiently. We will discuss 2(c) below. Step 2(b), which primarily distinguishes the present paper from the previous work of [Kivinen et al. \(2004\)](#), is discussed in Section 5.

Observe that, so far, our discussion of the online update algorithm is independent of the particular loss function used. In other words, to apply our method to a new setting we simply need to compute the corresponding loss function and its gradient. We discuss particular examples of loss functions and their gradients in the next section.

4.2 Loss Functions

A multitude of loss functions are commonly used to derive seemingly different kernel methods. This often blurs the similarities as well as subtle differences between these methods. In this section, we discuss some commonly used loss functions and put them in perspective. We begin with loss functions on unstructured output domains, then proceed to cases where the label space \mathcal{Y} is structured. Since our online update depends on it, we will state the gradient of all loss functions we present below, and give its kernel expansion coefficients. For piecewise linear loss functions, we employ one-sided derivatives at the points where they are not differentiable — *cf.* Footnote 1.

4.2.1 LOSS FUNCTIONS ON UNSTRUCTURED OUTPUT DOMAINS

Binary Classification uses the hinge or soft margin loss (Bennett and Mangasarian, 1992; Cortes and Vapnik, 1995)

$$l(\mathbf{x}, y, f) = \max(0, 1 - yf(\mathbf{x})) \quad (14)$$

where \mathcal{H} is defined on \mathcal{X} alone. We have

$$\partial_f l(\mathbf{x}, y, f) = \begin{cases} 0 & \text{if } yf(\mathbf{x}) \geq 1 \\ -yk(\mathbf{x}, \cdot) & \text{otherwise} \end{cases} \quad (15)$$

Multiclass Classification employs a definition of the margin arising from log-likelihood ratios (Crammer and Singer, 2000). This leads to

$$l(\mathbf{x}, y, f) = \max(0, 1 + \max_{\tilde{y} \neq y} f(\mathbf{x}, \tilde{y}) - f(\mathbf{x}, y)) \quad (16)$$

$$\partial_f l(\mathbf{x}, y, f) = \begin{cases} 0 & \text{if } f(\mathbf{x}, y) \geq 1 + f(\mathbf{x}, y^*) \\ k((\mathbf{x}, y^*), \cdot) - k((\mathbf{x}, y), \cdot) & \text{otherwise} \end{cases} \quad (17)$$

Here we defined y^* to be the maximizer of the $\max_{\tilde{y} \neq y}$ operation. If several y^* exist we pick one of them arbitrarily, *e.g.* by dictionary order.

Logistic Regression works by minimizing the negative log-likelihood. This loss function is used in Gaussian Process classification (MacKay, 1998). For binary classification this yields

$$l(\mathbf{x}, y, f) = \log(1 + \exp(-yf(\mathbf{x}))) \quad (18)$$

$$\partial_f l(\mathbf{x}, y, f) = -yk(\mathbf{x}, \cdot) \frac{1}{1 + \exp(yf(\mathbf{x}))} \quad (19)$$

Again the RKHS \mathcal{H} is defined on \mathcal{X} only.

Multiclass Logistic Regression works similarly to the example above. The only difference is that the log-likelihood arises from a conditionally multinomial model (MacKay, 1998). This means that

$$l(\mathbf{x}, y, f) = -f(\mathbf{x}, y) + \log \sum_{\tilde{y} \in \mathcal{Y}} \exp f(\mathbf{x}, \tilde{y}) \quad (20)$$

$$\partial_f l(\mathbf{x}, y, f) = \sum_{\tilde{y} \in \mathcal{Y}} k((\mathbf{x}, \tilde{y}), \cdot) [p(\tilde{y}|\mathbf{x}, f) - \delta_{y, \tilde{y}}], \quad (21)$$

$$\text{where we used } p(y|\mathbf{x}, f) = \frac{e^{f(\mathbf{x}, y)}}{\sum_{\tilde{y} \in \mathcal{Y}} e^{f(\mathbf{x}, \tilde{y})}}. \quad (22)$$

Novelty Detection uses a trimmed version of the log-likelihood as a loss function. In practice this means that labels are ignored and the one-class margin needs to exceed 1 (Schölkopf et al., 2001). This leads to

$$l(\mathbf{x}, y, f) = \max(0, 1 - f(\mathbf{x})) \quad (23)$$

$$\partial_f l(\mathbf{x}, y, f) = \begin{cases} 0 & \text{if } f(\mathbf{x}) \geq 1 \\ -k(\mathbf{x}, \cdot) & \text{otherwise} \end{cases} \quad (24)$$

4.2.2 LOSS FUNCTIONS ON STRUCTURED LABEL DOMAINS

In many applications the output domain has an inherent structure. For example, document categorization deals with the problem of assigning a set of documents to a set of pre-defined topic hierarchies or taxonomies. Consider a typical taxonomy shown in Figure 2 which is based on a subset of the open directory project (<http://www.dmoz.org/>). If a document describing CDROMs is classified under hard disk drives ('HDD'), intuitively the loss should be smaller than when the same document is classified under 'Cables'. Roughly speaking, the value of the loss function should depend on the length of the shortest path connecting the actual label to the predicted label *i.e.*, the loss function should respect the structure of the output space (Tsochantaridis et al., 2004).

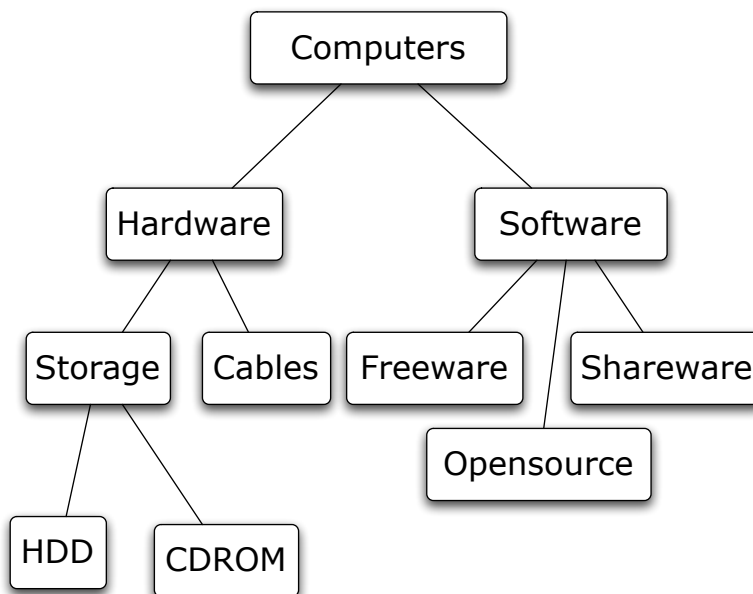


Figure 2: A taxonomy based on the open directory project.

To formalize our intuition, we need to introduce some notation. A weighted graph $G = (V, E)$ is defined by a set of nodes V and edges $E \subseteq V \times V$, such that, each edge $(v_i, v_j) \in E$ is assigned a non-negative weight $w(v_i, v_j) \in \mathbb{R}^+$. A path from $v_1 \in V$ to $v_n \in V$ is a sequence of nodes $v_1 v_2 \dots v_n$ such that $(v_i, v_{i+1}) \in E$. The weight of a path is the sum of the weights on the edges. For an undirected graph, $(v_i, v_j) \in E \implies (v_j, v_i) \in E \wedge w(v_i, v_j) = w(v_j, v_i)$.

A graph is said to be connected if every pair of nodes in the graph are connected by a path. In the sequel we will deal exclusively with connected graphs, and let $\Delta_G(v_i, v_j)$ denote the weight of the shortest (*i.e.*, minimum weight) path from v_i to v_j . If the output labels are nodes in a graph G , the following loss function takes the structure of G into account:

$$l(\mathbf{x}, y, f) = \max\{0, \max_{\tilde{y} \neq y} [\Delta_G(\tilde{y}, y) + f(\mathbf{x}, \tilde{y})] - f(\mathbf{x}, y)\}. \quad (25)$$

This loss requires that the output labels \tilde{y} which are “far away” from the actual label y (on the graph) must be classified with a larger margin while nearby labels are allowed to be classified with a smaller margin. More general notions of distance, including kernels on the nodes of the graph, can also be used here instead of the shortest path $\Delta_G(\tilde{y}, y)$.

Analogous to (21), by defining y^* to be the maximizer of the $\max_{\tilde{y} \neq y}$ operation we can write the gradient of the loss as:

$$\partial_f l(\mathbf{x}, y, f) = \begin{cases} 0 & \text{if } f(\mathbf{x}, y) \geq \Delta(y, y^*) + f(\mathbf{x}, y^*) \\ k((\mathbf{x}, y^*), \cdot) - k((\mathbf{x}, y), \cdot) & \text{otherwise} \end{cases} \quad (26)$$

The multiclass loss (16) is a special case of graph-based loss (25): consider a simple two-level tree in which each label is a child of the root node, and every edge has a weight of $\frac{1}{2}$. In this graph, any two labels $y \neq \tilde{y}$ will have $\Delta(y, \tilde{y}) = 1$, and thus (25) reduces to (16). We will employ a similar but multi-level tree-structured loss in our experiments on hierarchical document categorization (Section 6.4).

4.2.3 LOSS FUNCTION SUMMARY AND EXPANSION COEFFICIENTS

Note that the gradient always has the form

$$\partial_f l(\mathbf{x}_t, y_t, f_t) =: \langle \boldsymbol{\xi}_t, k((\mathbf{x}_t, \cdot), \cdot) \rangle \quad (27)$$

where $\boldsymbol{\xi}$ denotes the *expansion coefficient(s)*—more than one in the multiclass and structured label domain cases—arising from the derivative of the loss at time t .

Table 1 summarizes the tasks, loss functions, and expansion coefficients we have considered above. Similar derivations can be found for ε -insensitive regression, Huber’s robust regression, or LMS problems.

4.3 Coefficient Updates

Since the online update in step 2(c) of Algorithm 1 is not particularly useful in Hilbert space, we now rephrase it in terms of kernel function expansions. This extends and complements the reasoning of Kivinen et al. (2004) as applied to the various loss functions of the previous section. From (12) it follows that $g_t = \partial_f l(\mathbf{x}_t, y_t, f_t) + cf_t$ and consequently

$$\begin{aligned} f_{t+1} &= f_t - \eta_t [\partial_f l(\mathbf{x}_t, y_t, f_t) + cf_t] \\ &= (1 - \eta_t c) f_t - \eta_t \partial_f l(\mathbf{x}_t, y_t, f_t). \end{aligned} \quad (28)$$

Using the initialization $f_1 = 0$ this implies that

$$f_{t+1}(\cdot) = \sum_{i=1}^t \sum_y \alpha_{tiy} k((\mathbf{x}_i, y), \cdot). \quad (29)$$

Table 1: Loss functions and gradient expansion coefficients.

task	loss function $l(\mathbf{x}_t, y_t, f_t)$	expansion coefficient(s) ξ_t
Novelty Detection	$\max(0, 1 - f_t(\mathbf{x}_t))$	$\xi_t = \begin{cases} 0 & \text{if } f_t(\mathbf{x}_t) \geq 1 \\ -1 & \text{otherwise} \end{cases}$
Binary Classification	$\max(0, 1 - y_t f_t(\mathbf{x}_t))$	$\xi_t = \begin{cases} 0 & \text{if } y_t f_t(\mathbf{x}_t) \geq 1 \\ -y_t & \text{otherwise} \end{cases}$
Multiclass Classification	$\max[0, 1 - f_t(\mathbf{x}_t, y_t) + \max_{\tilde{y} \neq y_t} f_t(\mathbf{x}_t, \tilde{y})]$	$\xi_t = \mathbf{0}$ if $f_t(\mathbf{x}_t, y_t) \geq 1 + f_t(\mathbf{x}_t, y^*)$ $\xi_{t,y_t} = -1, \xi_{t,y^*} = 1$ otherwise
Graph-Struct. Label Domains	$\max\{0, -f(\mathbf{x}_t, y_t) + \max_{\tilde{y} \neq y_t} [\Delta(y_t, \tilde{y}) + f(\mathbf{x}_t, \tilde{y})]\}$	$\xi_t = \mathbf{0}$ if $f_t(\mathbf{x}_t, y_t) \geq \Delta(y_t, y^*) + f_t(\mathbf{x}_t, y^*)$ $\xi_{t,y_t} = -1, \xi_{t,y^*} = 1$ otherwise
Binary Logistic Regression	$\log(1 + e^{-y_t f_t(\mathbf{x}_t)})$	$\xi_t = \frac{-y_t}{1 + e^{y_t f_t(\mathbf{x}_t)}}$
Multiclass Logistic Regression	$\log \sum_{\tilde{y} \in \mathcal{Y}} e^{f_t(\mathbf{x}_t, \tilde{y})} - f_t(\mathbf{x}_t, y_t)$	$\xi_{t,y} = p(y \mathbf{x}_t, f_t) - \delta_{y,y_t}$

With some abuse of notation we will use the same expression for the cases where \mathcal{H} is defined on \mathcal{X} rather than $\mathcal{X} \times \mathcal{Y}$. In this setting we replace (29) by the sum over i only (with corresponding coefficients α_{ti}). Whenever necessary we will use α_t to refer to the entire coefficient vector (or matrix) and α_{ti} (or α_{tiy}) will refer to the specific coefficients. Observe that we can write

$$g_t(\cdot) = \sum_{i=1}^t \sum_y \gamma_{tiy} k((\mathbf{x}_i, y), \cdot), \quad (30)$$

$$\text{where } \gamma_t := \begin{bmatrix} c \alpha_{t-1} \\ \xi_t^\top \end{bmatrix}. \quad (31)$$

We can now rewrite the update equation (28) using only the expansion coefficients as

$$\alpha_t = \begin{bmatrix} (1 - \eta_t c) \alpha_{t-1} \\ -\eta_t \xi_t^\top \end{bmatrix} = \begin{bmatrix} \alpha_{t-1} \\ 0 \end{bmatrix} - \eta_t \gamma_t. \quad (32)$$

Note that conceptually α grows indefinitely as it acquires an additional row with each new data sample. Practical implementations will of course retain only a buffer of past examples with nonzero coefficients (see Section 5.5).

5. Online SVM

We now show how the SMD framework described in Section 2 can be used to adapt the step size for online SVMs. The updates given in Section 4 remain as before, the only difference being that the step size η_t is adapted before its value is used to update α .

5.1 Scalar Representation

Since we are dealing with an optimization in a RKHS only scalar variants are possible.⁴ The scalar equivalent of (4) is

$$\eta_{t+1} = \eta_t \max\left(\frac{1}{2}, 1 - \mu \langle g_{t+1}, v_{t+1} \rangle\right), \quad (33)$$

where μ is the meta-step size described in Section 2. The update for v is now given by

$$v_{t+1} = \lambda v_t - \eta_t (g_t + \lambda \mathbf{H}_t v_t), \quad (34)$$

where \mathbf{H}_t is the Hessian of the objective function. Note that now \mathbf{H}_t is an operator in Hilbert space. For $J_t(f)$ as defined in (12), this operator has a form that permits efficient computation of $\mathbf{H}_t v_t$:

For piecewise linear loss functions, such as (15), (17), and (24), we have $\mathbf{H}_t = c\mathbf{I}$, where \mathbf{I} is the identity operator, and obtain the simple update

$$v_{t+1} = (1 - \eta_t c) \lambda v_t - \eta_t g_t. \quad (35)$$

For other losses, note that l only depends on f via its evaluations at (x, y) . This means that \mathbf{H}_t differs from $c\mathbf{I}$ only by a low-rank object. In particular, for logistic regression (19) we have

$$\mathbf{H}_t - c\mathbf{I} = \rho(\mathbf{x}_t) k(\mathbf{x}_t, \cdot) \otimes k(\mathbf{x}_t, \cdot), \quad (36)$$

where $\rho(\mathbf{x}_t) := e^{y_t f_t(\mathbf{x}_t)} / (1 + e^{y_t f_t(\mathbf{x}_t)})^2$, and \otimes denotes the outer product between functions in \mathcal{H} , obeying $(u \otimes v)w = u \langle v, w \rangle$ for $u, v, w \in \mathcal{H}$. Likewise, for multiclass logistic regression (21) we have

$$\mathbf{H}_t - c\mathbf{I} = \sum_{y, \tilde{y} \in \mathcal{Y}} \rho(\mathbf{x}_t, y, \tilde{y}) k((\mathbf{x}_t, y), \cdot) \otimes k((\mathbf{x}_t, \tilde{y}), \cdot), \quad (37)$$

$$\text{where } \rho(\mathbf{x}_t, y, \tilde{y}) := \delta_{y, \tilde{y}} p(\tilde{y} | \mathbf{x}_t, f_t) - p(\tilde{y} | \mathbf{x}_t, f_t) p(y | \mathbf{x}_t, f_t). \quad (38)$$

5.2 Expansion in Hilbert Space

The above discussion implies that v can be expressed as a linear combination of kernel functions, and consequently is also a member of the RKHS defined by $k(\cdot, \cdot)$. Thus v cannot be updated explicitly, as is done in the normal SMD algorithm (Section 2). Instead we write

$$v_{t+1}(\cdot) = \sum_{i=1}^t \sum_y \beta_{tiy} k((\mathbf{x}_i, y), \cdot) \quad (39)$$

and update the coefficients β . This is sufficient for our purpose because we only need to be able to compute the inner products $\langle g, v \rangle_{\mathcal{H}}$ in order to update η . Below, we first discuss the case when $H = c\mathbf{I}$ and then extend the discussion to handle the off diagonal entries.

Diagonal Hessian. Analogous to the update on α we can determine the updates on β via

$$\beta_t = \begin{bmatrix} (1 - \eta_t c) \lambda \beta_{t-1} \\ 0 \end{bmatrix} - \eta_t \gamma_t. \quad (40)$$

Although (40) suffices in principle to implement the overall algorithm, a naive implementation of the inner product $\langle g_t, v_t \rangle$ in (33) takes $O(t^2)$ time, rendering it impractical. We show in Section 5.3 how we can exploit the incremental nature of the updates to compute this inner product in linear time.

4. The situation is different for reduced-rank expansions which are parametrized by the reduced set vectors.

Non-diagonal Hessian. The only modification to (40) that we need to take into account is the contribution of the off-diagonal entries of \mathbf{H}_t to β_t . A close look at (36) and (37) reveals that the low-rank modification to $c\mathbf{I}$ only happens in the subspace spanned by $k((\mathbf{x}_t, y), \cdot)$ for $y \in \mathcal{Y}$. This means that we can express

$$(\mathbf{H}_t - c\mathbf{I})v_t = \sum_{\tilde{y} \in \mathcal{Y}} \chi_{t\tilde{y}} k((\mathbf{x}_t, \tilde{y}), \cdot), \quad (41)$$

allowing us to derive the analog of (40):

$$\beta_t = \begin{bmatrix} (1 - \eta_t c)\lambda\beta_{t-1} \\ 0 \end{bmatrix} - \eta_t(\gamma_t + \lambda\chi_t). \quad (42)$$

In the case of binary logistic regression we have

$$(\mathbf{H}_t - c\mathbf{I})v_t = \rho(\mathbf{x}_t)v_t(\mathbf{x}_t)k(\mathbf{x}_t, \cdot), \quad (43)$$

and for multiclass logistic regression

$$(\mathbf{H}_t - c\mathbf{I})v_t = \sum_{y, \tilde{y} \in \mathcal{Y}} \rho(\mathbf{x}_t, y, \tilde{y})v_t(\mathbf{x}_t, y)k((\mathbf{x}_t, \tilde{y}), \cdot). \quad (44)$$

5.3 Linear-Time Incremental Updates

We now turn to computing $\langle g_{t+1}, v_{t+1} \rangle$ in linear time by bringing it into an incremental form. For ease of exposition we will consider the case where $\mathbf{H}_t = c\mathbf{I}$; an extension to non-diagonal Hessians is straightforward but tedious. We use the notation $f(\mathbf{x}_t, \cdot)$ to denote the vector of $f(\mathbf{x}_t, \tilde{y})$ for $\tilde{y} \in \mathcal{Y}$. Expanding g_{t+1} into $cf_{t+1} + \xi_{t+1}$ we can write

$$\langle g_{t+1}, v_{t+1} \rangle = c\pi_{t+1} + \xi_{t+1}^\top v_{t+1}(\mathbf{x}_{t+1}, \cdot), \quad (45)$$

where $\pi_t := \langle f_t, v_t \rangle$. The function update (28) yields

$$\pi_{t+1} = (1 - \eta_t c) \langle f_t, v_{t+1} \rangle - \eta_t \xi_t^\top v_{t+1}(\mathbf{x}_t, \cdot). \quad (46)$$

The v update (34) then gives us

$$\langle f_t, v_{t+1} \rangle = (1 - \eta_t c)\lambda\pi_t - \eta_t \langle f_t, g_t \rangle, \quad (47)$$

and using $g_t = cf_t + \xi_t$ again we have

$$\langle f_t, g_t \rangle = c\|f_t\|^2 + \xi_t^\top f_t(\mathbf{x}_t, \cdot). \quad (48)$$

Finally, the squared norm of f can be maintained via:

$$\|f_{t+1}\|^2 = (1 - \eta_t c)^2 \|f_t\|^2 - 2\eta_t(1 - \eta_t c)\xi_t^\top f_t(\mathbf{x}_t, \cdot) + \eta_t^2 \xi_t^\top k((\mathbf{x}_t, \cdot), (\mathbf{x}_t, \cdot))\xi_t. \quad (49)$$

The above sequence (45)–(49) of equations, including the evaluation of the associated functionals, can be performed in $O(t)$ time, and thus allows us to efficiently compute $\langle g_{t+1}, v_{t+1} \rangle$.

5.4 Extensions

We will now implement in the context of SVM two important standard extensions to the SVM framework: offsets, and the specification of the fraction of points which violate the margin via the so-called ν -trick. Both of these extensions create new parameters, which we will also tune by stochastic gradient descent, again using SMD for step size adaptation.

5.4.1 HANDLING OFFSETS

In many situations, for instance in binary classification, it is advantageous to add an offset $\mathbf{b} \in \mathbb{R}^{|\mathcal{Y}|}$ to the prediction $f \in \mathcal{H}$. While the update equations described above remain unchanged, the offset parameter \mathbf{b} is now adapted as well:

$$\mathbf{b}_{t+1} = \mathbf{b}_t - \eta_{\mathbf{b},t} \cdot \partial_{\mathbf{b}} J_t(f_t + \mathbf{b}_t) = \mathbf{b}_t - \eta_{\mathbf{b},t} \cdot \boldsymbol{\xi}_t. \quad (50)$$

Applying the standard SMD equations (4) and (7) to the case at hand, we update the offset step sizes $\eta_{\mathbf{b}}$ via

$$\eta_{\mathbf{b},t+1} = \eta_{\mathbf{b},t} \cdot \max\left(\frac{1}{2}, 1 - \mu_{\mathbf{b}} \boldsymbol{\xi}_{t+1} \cdot \mathbf{v}_{\mathbf{b},t+1}\right), \quad (51)$$

where $\mu_{\mathbf{b}}$ is the meta-step size for adjusting $\eta_{\mathbf{b}}$, and $\mathbf{v}_{\mathbf{b}}$ is adapted as

$$\mathbf{v}_{\mathbf{b},t+1} = \lambda_{\mathbf{b}} \mathbf{v}_{\mathbf{b},t} - \eta_{\mathbf{b},t} \cdot \boldsymbol{\xi}_t. \quad (52)$$

Note that we can adjust the offset for each class in \mathcal{Y} individually.

5.4.2 THE ν -TRICK

The so called ν -trick allows one to pre-specify the fraction, $0 < \nu < 1$, of points which violate the margin. For instance, when performing novelty detection using the ν -trick, the loss function that is commonly used is

$$l(\mathbf{x}, y, f) = \max(0, \epsilon - f(\mathbf{x})) - \nu\epsilon. \quad (53)$$

Here, the regularization parameter is fixed at $c = 1$, but the margin is adapted with the additional constraint $\epsilon > 0$. This can easily be taken into account by adapting ϵ in log-space. Observe that

$$\partial_{\log \epsilon} J_t(f_t) = \epsilon \partial_{\epsilon} J_t(f_t) = -\epsilon (\boldsymbol{\xi}_t + \nu), \quad (54)$$

and therefore the updates for ϵ can now be written as

$$\epsilon_{t+1} = \epsilon_t \exp(-\eta_{\epsilon,t} \partial_{\log \epsilon} J_t(f_t)) \quad (55)$$

$$= \epsilon_t \exp(\eta_{\epsilon,t} \epsilon_t (\boldsymbol{\xi}_t + \nu)). \quad (56)$$

We now use SMD to adapt the margin step size $\eta_{\epsilon,t}$:

$$\eta_{\epsilon,t+1} = \eta_{\epsilon,t} \max\left(\frac{1}{2}, 1 + \mu_{\epsilon} v_{\epsilon,t} \epsilon_t (\boldsymbol{\xi}_t + \nu)\right), \quad (57)$$

where $v_{\epsilon,t}$ is updated as

$$v_{\epsilon,t+1} = \lambda_{\epsilon} v_{\epsilon,t} + \eta_{\epsilon,t} \epsilon_t (\boldsymbol{\xi}_t + \nu) (1 + \lambda_{\epsilon} v_{\epsilon,t}). \quad (58)$$

This is a straightforward application of the SMD update equations (4) and (7), taking into account that ϵ is adapted in log-space.

This completes our description of the online SVM algorithm. Since it comprises a rather large number of update equations, it is non-trivial to arrange them in an appropriate order. Algorithm 2 shows the ordering which we have found most advantageous.

Algorithm 2 Online $[\nu]$ -SVMD

1. Initialize
 2. **Repeat**
 - (a) data, prediction and loss:
 - i. draw data sample (\mathbf{x}_t, y_t)
 - ii. calculate prediction $f_t(\mathbf{x}_t)$
 - iii. calculate loss $l(\mathbf{x}_t, y_t, f_t)$
 - (b) obtain gradients:
 - i. calculate $\xi_t = \partial_f l(\mathbf{x}_t, y_t, f_t)$
 - ii. (39) calculate $v_t(\mathbf{x}_t)$
 - iii. (45) calculate $\langle g_t, v_t \rangle$
 - iv. (31) calculate g resp. γ_t
 - (c) perform SMD:
 - i. (33) update step size(s) $[\eta_\epsilon, \eta_b,] \eta$
 - ii. (43)/(44) if \mathbf{H} non-diag.: compute χ
 - iii. (40)/(42) update $[\epsilon, v_\epsilon, \mathbf{v}_b,] v$ resp. β
 - (d) maintain incrementals:
 - i. (48) calculate $\langle f_t, g_t \rangle$
 - ii. (47) calculate $\langle f_t, v_{t+1} \rangle$
 - iii. (39) calculate $v_{t+1}(\mathbf{x}_t)$
 - iv. (46) update π
 - v. (49) update $\|f\|^2$
 - (e) (32) update function f resp. α
-

5.5 Time Complexity and Buffer Management

The time complexity of online SVMD is dominated by the cost of the kernel expansions in steps 2(a)ii, 2(b)ii, and 2(d)iii of Algorithm 2, which grows linearly with the size of the expansion. Since unlimited growth would be undesirable, we maintain a Least Recently Used (LRU) circular buffer which stores only the last ω non-zero expansion coefficients; each kernel expansion then takes $O(\omega|\mathcal{Y}|)$ time.

The online SVM (NORMA) algorithm does not require steps 2(b)ii or 2(d)iii, but still has to employ step 2(a)ii to make a prediction, so its asymptotic time complexity is $O(\omega|\mathcal{Y}|)$ as well. The two algorithms thus differ in time complexity only by a constant factor; in practice we observe online SVMD to be 3–4 times slower per iteration than NORMA.

Limiting the kernel expansion to the most *recent* ω non-zero terms makes sense because at each iteration t the coefficients α_i with $i < t$ are multiplied by a factor $(1 - \eta_t c) < 1$. After sufficiently many iterations α_i will thus have shrunk to a small value whose contribution to $f(\mathbf{x}_t)$ becomes negligible — and likewise for β_i 's contribution to $v(\mathbf{x}_t)$. If the loss function has a bounded gradient (as in all cases of Table 1), then it can be shown that the truncation error thus introduced

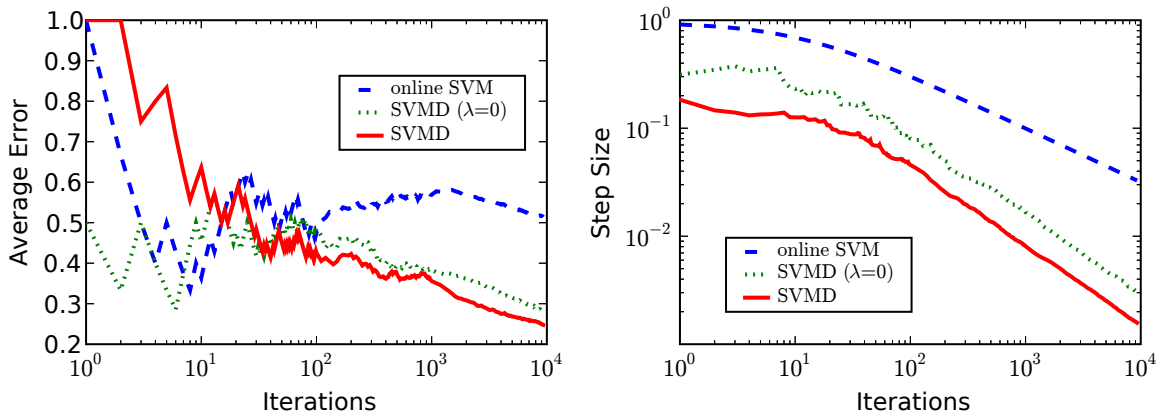


Figure 3: ν -SVM binary classification over a single run through the USPS dataset. Current average error (left) and step size (right) for SVMD with $\lambda = 0.95$ (solid), $\lambda = 0$ (dotted), and online SVM with step size decay (59), using $\tau = 10$ (dashed).

decreases exponentially with the number of terms retained (Kivinen et al., 2004, Proposition 1), so good solutions can be obtained with limited buffer size ω .

A good buffer management scheme has to deal with two conflicting demands: To the extent that the dataset is non-stationary, it is desirable to remove old items from the buffer in order to reduce the effect of obsolete data. The truncation error, on the other hand, is reduced by using as large a buffer as possible. Although we employ a simple LRU circular buffer to good effect here, smarter buffer management strategies which explicitly remove the least important point based on some well-defined criterion (Crammer et al., 2004; Weston et al., 2005; Dekel et al., 2006) could also be adapted to work with our algorithm.

6. Experiments

We now evaluate the performance of SMD’s step size adaptation in RKHS by comparing the online SVMD algorithm described in Section 5 above to step size adaptation based only on immediate, single-step effects — obtained by setting $\lambda = 0$ in SVMD — and to the conventional online SVM (aka NORMA) algorithm (Kivinen et al., 2004) with a scheduled step size decay of

$$\eta_t = \sqrt{\tau/(\tau + t)}, \quad (59)$$

where τ is hand-tuned to obtain good performance. We do not use offsets here; where we employ the ν -trick (cf. Section 5.4.2), we always set $\eta_{\epsilon,0} = 1$, $\mu_\epsilon = \mu$, and $\lambda_\epsilon = \lambda$.

6.1 USPS Dataset

For our first set of experiments we use the well-known USPS dataset (LeCun et al., 1989) with the RBF kernel

$$k(\mathbf{x}, \mathbf{x}') = \exp\left(\frac{-\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma^2}\right), \quad (60)$$

setting $\sigma = 8$ via cross-validation. We extend this kernel to the multiclass case via the delta kernel:

$$k((\mathbf{x}, y), (\mathbf{x}', y')) := k(\mathbf{x}, \mathbf{x}') \delta_{yy'}. \quad (61)$$

In the spirit of online learning, we train for just a single run through the data, so that no digit is seen more than once by any algorithm. For a fair comparison, all algorithms started with the same initial step size η_0 , and had the data presented in the same order. For implementation efficiency, we only store the last 512 support vectors in a circular buffer.

6.1.1 BINARY CLASSIFICATION

Figure 3 shows our results for binary classification. Here, the data was split into two classes comprising the digits 0–4 and 5–9, respectively. We use ν -SVM with $\nu = 0.05$, $\eta_0 = 1$, and $\mu = 1$ and plot current average error rate — that is, the total number of errors on the training samples seen so far divided by the iteration number — and step size. Observe that online SVMD (solid) is initially slower to learn, but after about 20 iterations it overtakes the online SVM (dashed), and overall makes only about half as many classification errors. The single-step version of SVMD with $\lambda = 0$ (dotted) has the fastest early convergence but is asymptotically inferior to SVMD proper, though still far better than the online SVM with scheduled step size decay.

6.1.2 MULTICLASS CLASSIFICATION

Figure 4 shows our results for 10-way multiclass classification using soft margin loss with $\eta_0 = 0.1$, $\mu = 0.1$, and $c = 1/(500n)$, where n is the number of training samples. Again online SVMD (solid) makes only about half as many classification errors overall as the online SVM (dashed), with the single-step ($\lambda = 0$) variant of SVMD (dotted) falling in between.

We found (by cross-validation) the online SVM with fixed decay schedule to perform best here for $\eta_0 = 0.1$. SVMD, on the other hand, is less dependent on a particular value of η_0 since it can adaptively adjust its step size. In this experiment, for instance, SVMD raised η significantly above its initial value of 0.1 — something that a predetermined decay schedule cannot do. We generally find the performance of online SVMD to be fairly independent of the initial step size.

6.2 Non-stationary USPS Counting Sequence

For our next set of experiments we rearrange the USPS data to create a highly non-stationary problem: we take 600 samples of each digit, then present them in the order corresponding to a 3-digit decimal counter running twice from 000 through 999 (Figure 5). This creates pronounced non-stationarities in the distribution of labels: '0' for instance is very frequent early in the sequence but rare towards the end.

6.2.1 MULTICLASS CLASSIFICATION

Here we perform 10-way multiclass classification on the USPS counting sequence, using ν -SVMD with soft margin loss, $\nu = 0.05$, $\eta_0 = 1$, and $\mu = 1$. As Figure 6 shows, ν -SVMD (solid) makes significantly fewer classification errors than the controls: The *average* error rate for ν -SVMD over its entire first (and only) pass through this sequence is less than 19% here. Online ν -SVM with scheduled step size decay, on the other hand, has serious difficulty with the non-stationary nature of our data and performs barely above change level (90% error rate); even the simple step size

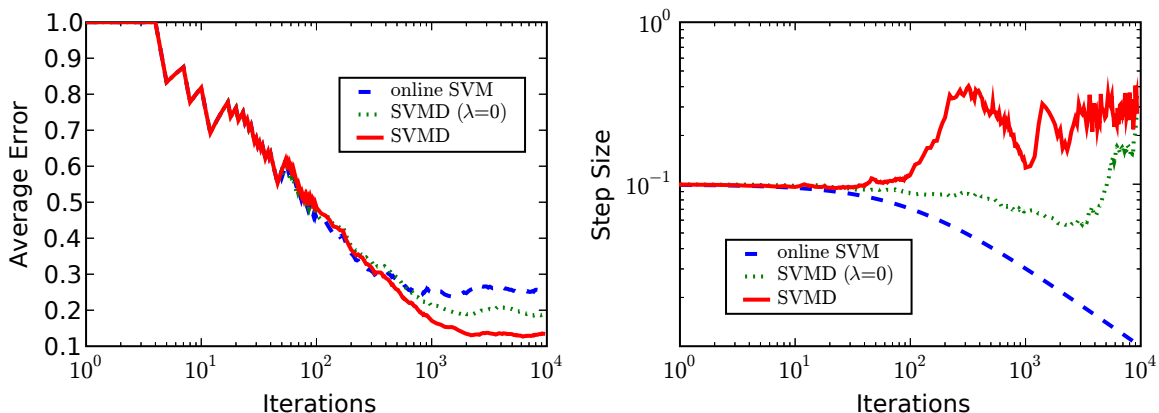


Figure 4: Online 10-way multiclass classification over a single run through the USPS dataset. Current average error (left) and step size (right) for SVMD with $\lambda = 0.99$ (solid), $\lambda = 0$ (dotted), and online SVM with step size decay (59), using $\tau = 100$ (dashed).

adaptation obtained for $\lambda = 0$ clearly outperforms it. This is not surprising since a step size decay schedule typically assumes stationarity. By contrast, the decay factor of SVMD can be adjusted to match the time scale of non-stationarity; here $\lambda = 0.95$ yielded the best performance. In other experiments, we found (as one would expect) $\lambda = 1$ to work well for stationary data.

6.2.2 NOVELTY DETECTION

We also perform novelty detection with SVMD ($\mu = \lambda = 1$) on the USPS counting sequence. Figure 7 (left) shows that though SVMD markedly reduces the initial step size, it does not anneal it down to zero. Its ongoing reactivity is evidenced by the occurrence of spikes in η_t that correspond to identifiable events in our counting sequence. Specifically, major increases in η_t can be observed after seeing the first non-zero digit at $t = 6$, as the counter wraps from 19 to 20 at $t = 60$ (and likewise at $t = 120, 150, 180$), then at $t = 300, 1200, 1500$ as the hundreds wrap from 099 to 100, 399 to 400, and 499 to 500, respectively, and finally at $t = 3000$ as the entire sequence wraps around from 999 to 000. Many more minor peaks and troughs occur in between, closely tracking the fractal structure of our counting sequence.



Figure 5: To create a highly non-stationary problem, we rearranged 6000 USPS digits into a 3-digit counting sequence, running twice from 000 to 999.

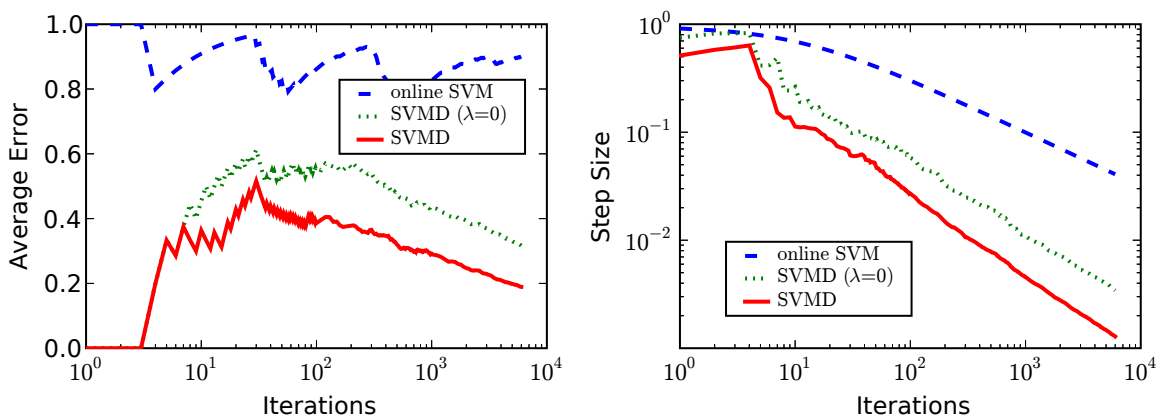


Figure 6: Online 10-way multiclass classification over a single run through our USPS counting sequence (Figure 5). Current average error (left) and step size (right) for ν -SVMD with $\lambda = 0.95$ (solid), $\lambda = 0$ (dotted), and online ν -SVM with step size decay (59), using $\tau = 100$ (dashed).

6.3 MNIST Dataset: Effect of Buffer Size

We now scale up our 10-way multiclass classification experiments to a much larger dataset: 60000 digits from MNIST. We illustrate the effect of the circular buffer size on classification accuracy, using $\lambda = 1$, $\mu = 0.01$, and a polynomial kernel of degree 9. On a single run through the data, a buffer size of 256 yields around 20% average error (Figure 7, right). This reduces to 3.9% average error when the buffer size is increased to 4096; averaged over the last 4500 digits, the error rate is as low as 2.9%. For comparison, batch SVM algorithms achieve (at far greater computational cost) a generalization error rate of 1.4% on this data (Burges and Schölkopf, 1997; Schölkopf and Smola, 2002, p. 341).

6.4 WIPO-alpha Dataset: Tree-Structured Loss

For our experiments on document categorization we use the WIPO-alpha dataset published in 2002 by the World Intellectual Property Organization (WIPO).⁵ The dataset consists of 75250 patents which have been classified into 4656 categories according to a standard taxonomy known as *international patent classification* (IPC, <http://www.wipo.int/classifications/en/>). Each document is assigned labels from a four-level hierarchy comprising sections, classes, sub-classes and groups. A typical label might be ‘D05C 1/00’ which would be read as Section D (Textiles; Paper), class 05 (Sewing; Embroidering; Tufting), sub-class C (Embroidering; Tufting) and group 1/00 (Apparatus, devices, or tools for hand embroidering).

The IPC defines an undirected taxonomy tree. A tree is a graph with no cycles — *i.e.*, no paths whose start and end vertices coincide — and one node designated as the root. We use $\tilde{y} \preceq y$ to denote that \tilde{y} is an ancestor of y , *i.e.*, the path from y to the root contains \tilde{y} .⁶

5. This data is now available on request from WIPO (<http://www.wipo.int/>).

6. Note that according to this definition, every node is an ancestor of itself; this is deliberate.

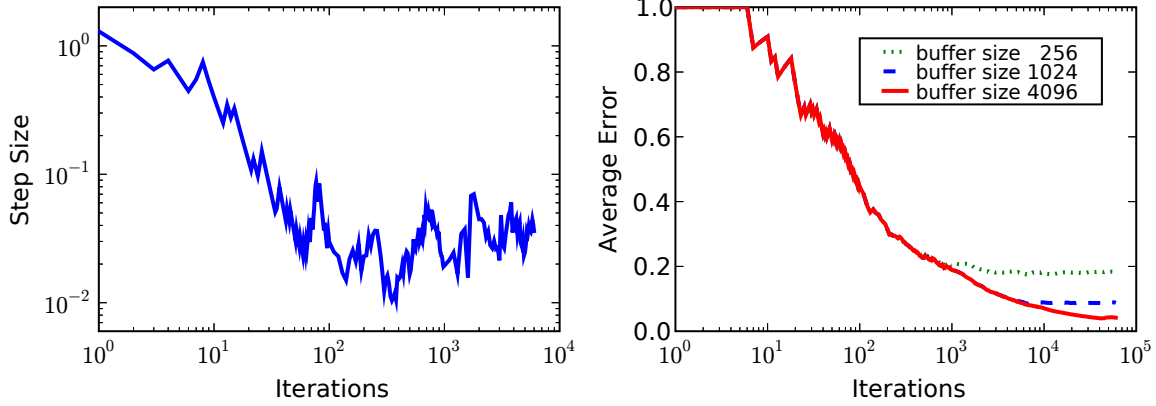


Figure 7: Left: The step size for novelty detection with SVM on the USPS counting sequence (Figure 5) closely follows the non-stationarity of the data. Right: Average error of SVM on the MNIST data set, for three different circular buffer sizes.

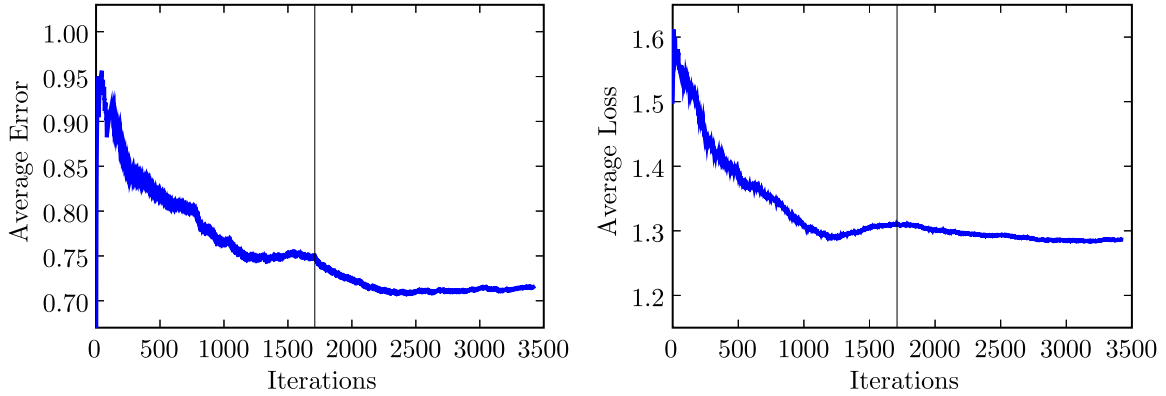


Figure 8: Average error (left) and loss (right) for SVM over two passes (separated by vertical line) through section D of the WIPO-alpha dataset.

Following Cai and Hofmann (2004), we perform document categorization experiments on the WIPO-alpha dataset, using a loss function that is a special case of our graph-structured loss (25). Here the graph G is the taxonomy tree with a weight of $\frac{1}{2}$ on every edge, and the weighted distance between nodes is defined as (Cai and Hofmann, 2004):

$$\Delta_G(y, \tilde{y}) := \left(\sum_{\substack{z: z \leq y \\ \wedge z \not\leq \tilde{y}}} \frac{1}{2} \right) + \left(\sum_{\substack{z: z \leq \tilde{y} \\ \wedge z \not\leq y}} \frac{1}{2} \right). \quad (62)$$

A patent could have potentially many categories, but it has exactly one primary category. Following Cai and Hofmann (2004) we concentrate on predicting the primary category using the title

and claim contents. Furthermore, for the sake of reporting results, we restrict ourselves to Section D from the dataset. This section contains 1710 documents categorized into 160 categories. Pre-processing of the data consisted of document parsing, tokenization, and term normalization in order to produce a bag-of-words representation. The bag-of-words vector is normalized to unit length. We use a product kernel which is defined as

$$k((\mathbf{x}, y), (\mathbf{x}', y')) := k(\mathbf{x}, \mathbf{x}') \kappa(y, y'). \quad (63)$$

For the document vectors we use a linear dot-product kernel

$$k(\mathbf{x}, \mathbf{x}') := \mathbf{x}^\top \mathbf{x}', \quad (64)$$

while for the labels we use

$$\kappa(y, y') := \sum_{\substack{z: z \preceq y \\ \wedge z \preceq y'}} 1, \quad (65)$$

which counts the number of common ancestors of y and y' . Like [Cai and Hofmann \(2004\)](#), we set the regularizer c to the reciprocal of the number of data points. We use a buffer of size 1024, initial step size $\eta_0 = 1$, meta-step size $\mu = 0.1$, and decay parameter $\lambda = 0.999$. In [Figure 8](#) we plot the current average error rate and graph-structured loss [\(25\)](#) over two passes through the data.

The WIPO-alpha dataset is known to be difficult to learn ([Cai and Hofmann, 2004](#); [Tsochantaridis et al., 2004](#)). Only 94 out of the 160 possible categories contain four or more examples while as many as 34 categories have exactly one sample, which makes it extremely hard for an online learning algorithm to predict well. Even the best offline algorithms have a reported error rate of 57.2% ([Cai and Hofmann, 2004](#)). SVM performs competitively on this challenging dataset, achieving an average error rate of around 75% over its first pass through the data, and 68% over its second pass. It reduces the average loss to around 1.32 over the first pass, and 1.28 over both passes ([Figure 8](#)). We found that further runs through the dataset did not yield a significant increase in accuracy or reduction of the loss.

7. Discussion

We presented online SVMD, an extension of the SMD step size adaptation method to the kernel framework. Using an incremental approach to reduce a naive $O(t^2)$ computation to $O(t)$, we showed how the SMD parameters can be updated efficiently even though they now reside in an RKHS. We addressed the difficult cases of multiclass classification and logistic regression, where the Hessian operator in RKHS includes non-diagonal terms. We also showed how SVMD can be adapted to deal with structured output spaces. In experiments online SVMD outperformed the conventional online SVM (*aka* NORMA) algorithm with scheduled step size decay for binary and multiclass classification, especially on a non-stationary problem. In particular, it accelerated convergence to a good solution, which is one of the main aims of performing step size adaptation. In novelty detection experiments we observe that SVMD is able to closely track the non-stationarity in the data and adapt the step sizes correspondingly. With a reasonable buffer size SVMD attains competitive performance in a single pass through the MNIST dataset. On a difficult document categorization task using the WIPO-alpha dataset, SVMD performed well compared to the best offline algorithm.

Empirically we observe that in all our experiments the SVMD algorithm significantly speeds up the convergence of the conventional online SVM algorithm. It would be interesting to obtain worst case loss bounds for SVMD akin to those of [Kivinen et al. \(2004\)](#). The main technical challenge here is that the SMD update consists of three interleaved updates, and applying a straightforward analysis using Bregman divergences ([Gentile and Littlestone, 1999](#); [Azoury and Warmuth, 2001](#)) is infeasible. Established methods for proving worst case loss bounds rely on the cancellation of telescoped terms, which works only when the step size η is held constant. In the case of SVMD, however, step sizes change from iteration to iteration. Even worse, their update is governed by two other feedback equations. A more sophisticated analysis, possibly involving second-order information, will have to be developed to establish similar loss bounds for SVMD.

SMD is a generic method to hasten the convergence of stochastic gradient descent methods. In combination with the kernel trick this provides a powerful learning tool. Other kernel algorithms which rely on stochastic gradient descent — *e.g.*, that of [Kim et al. \(2005\)](#) — could also be accelerated with SMD; this is a focus of our ongoing work in this area.

Acknowledgments

We would like to thank Alexandros Karatzoglou and Chang Chui for their help with early implementations, Lijuan Cai and Thomas Hofmann for making a pre-preprocessed version of the WIPO-Alpha dataset available to us, and the anonymous reviewers for ICML, NIPS, and JMLR for their many helpful comments. National ICT Australia is funded by the Australian Government’s Department of Communications, Information Technology and the Arts and the Australian Research Council through Backing Australia’s Ability and the ICT Center of Excellence program. This work is supported by the IST Program of the European Community, under the Pascal Network of Excellence, IST-2002-506778.

References

- L. B. Almeida, T. Langlois, J. D. Amaral, and A. Plakhov. Parameter adaptation in stochastic optimization. In D. Saad, editor, *On-Line Learning in Neural Networks*, Publications of the Newton Institute, chapter 6, pages 111–134. Cambridge University Press, 1999.
- Y. Altun, A. J. Smola, and T. Hofmann. Exponential families for conditional random fields. In *Uncertainty in Artificial Intelligence (UAI)*, pages 2–9, Arlington, Virginia, 2004. AUAI Press.
- K. Azoury and M. K. Warmuth. Relative loss bounds for on-line density estimation with the exponential family of distributions. *Machine Learning*, 43(3):211–246, 2001. Special issue on Theoretical Advances in On-line Learning, Game Theory and Boosting.
- A. G. Barto and R. S. Sutton. Goal seeking components for adaptive intelligence: An initial assessment. Technical Report AFWAL-TR-81-1070, Air Force Wright Aeronautical Laboratories, Wright-Patterson AFB, Ohio 45433, USA, 1981.
- K. P. Bennett and O. L. Mangasarian. Robust linear programming discrimination of two linearly inseparable sets. *Optim. Methods Softw.*, 1:23–34, 1992.

- H. D. Block. The perceptron: A model for brain functioning. *Reviews of Modern Physics*, 34: 123–135, 1962. Reprinted in *Neurocomputing* by Anderson and Rosenfeld.
- A. Bordes, S. Ertekin, J. Weston, and L. Bottou. Fast kernel classifiers with online and active learning. *Journal of Machine Learning Research*, 6:1579–1619, September 2005.
- M. Bray, E. Koller-Meier, P. Müller, N. N. Schraudolph, and L. Van Gool. Stochastic optimization for high-dimensional tracking in dense range maps. *IEE Proceedings Vision, Image & Signal Processing*, 152(4):501–512, 2005.
- M. Bray, E. Koller-Meier, N. N. Schraudolph, and L. V. Gool. Fast stochastic optimization for articulated structure tracking. *Image and Vision Computing*, 25(3):352–364, March 2007.
- C. J. C. Burges and B. Schölkopf. Improving the accuracy and speed of support vector learning machines. In M. C. Mozer, M. I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems 9*, pages 375–381, Cambridge, MA, 1997. MIT Press.
- L. Cai and T. Hofmann. Hierarchical document categorization with support vector machines. In *Proceedings of the Thirteenth ACM conference on Information and knowledge management*, pages 78–87, New York, NY, USA, 2004. ACM Press.
- C. Cortes and V. Vapnik. Support vector networks. *Machine Learning*, 20(3):273–297, 1995.
- K. Crammer and Y. Singer. On the learnability and design of output codes for multiclass problems. In N. Cesa-Bianchi and S. Goldman, editors, *Proc. Annual Conf. Computational Learning Theory*, pages 35–46, San Francisco, CA, 2000. Morgan Kaufmann Publishers.
- K. Crammer and Y. Singer. Ultraconservative online algorithms for multiclass problems. *Journal of Machine Learning Research*, 3:951–991, January 2003.
- K. Crammer, J. Kandola, and Y. Singer. Online classification on a budget. In S. Thrun, L. Saul, and B. Schölkopf, editors, *Advances in Neural Information Processing Systems 16*, pages 225–232, Cambridge, MA, 2004. MIT Press.
- O. Dekel, S. Shalev-Shwartz, and Y. Singer. The Forgetron: A kernel-based perceptron on a fixed budget. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems 18*, Cambridge, MA, 2006. MIT Press.
- Y. Freund and R. E. Schapire. Large margin classification using the perceptron algorithm. *Machine Learning*, 37(3):277–296, 1999.
- T.-T. Frieß, N. Cristianini, and C. Campbell. The kernel adatron algorithm: A fast and simple learning procedure for support vector machines. In J. Shavlik, editor, *Proc. Intl. Conf. Machine Learning*, pages 188–196. Morgan Kaufmann Publishers, 1998.
- C. Gentile. A new approximate maximal margin classification algorithm. *Journal of Machine Learning Research*, 2:213–242, December 2001.
- C. Gentile and N. Littlestone. The robustness of the p-norm algorithms. In *Proc. Annual Conf. Computational Learning Theory*, pages 1–11, Santa Cruz, California, United States, 1999. ACM Press, New York, NY.

- A. Griewank. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. Frontiers in Applied Mathematics. SIAM, Philadelphia, 2000.
- M. E. Harmon and L. C. Baird, III. Multi-player residual advantage learning with general function approximation. Technical Report WL-TR-1065, Wright Laboratory, WL/AACF, Wright-Patterson Air Force Base, OH 45433-7308, 1996. http://www.leemon.com/papers/sim_tech/sim_tech.pdf.
- D. Helmbold and M. K. Warmuth. On weak learning. *Journal of Computer and System Sciences*, 50(3):551–573, June 1995.
- M. Herbster. Learning additive models online with fast evaluating kernels. In D. P. Helmbold and R. C. Williamson, editors, *Proc. Annual Conf. Computational Learning Theory*, volume 2111 of *Lecture Notes in Computer Science*, pages 444–460. Springer, 2001.
- R. Jacobs. Increased rates of convergence through learning rate adaptation. *Neural Networks*, 1: 295–307, 1988.
- K. Kim, M. O. Franz, and B. Schölkopf. Iterative kernel principal component analysis for image modeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(9):1351–1366, 2005.
- J. Kivinen and M. K. Warmuth. Exponentiated gradient versus gradient descent for linear predictors. *Information and Computation*, 132(1):1–64, January 1997.
- J. Kivinen, A. J. Smola, and R. C. Williamson. Online learning with kernels. *IEEE Transactions on Signal Processing*, 52(8), Aug 2004.
- Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1:541–551, 1989.
- Y. Li and P. M. Long. The relaxed online maximum margin algorithm. *Machine Learning*, 46(1–3): 361–387, 2002.
- D. J. C. MacKay. Introduction to Gaussian processes. In C. M. Bishop, editor, *Neural Networks and Machine Learning*, pages 133–165. Springer, Berlin, 1998.
- M. Milano. *Machine Learning Techniques for Flow Modeling and Control*. PhD thesis, Eidgenössische Technische Hochschule (ETH), Zürich, Switzerland, 2002.
- M. Minsky and S. Papert. *Perceptrons: An Introduction To Computational Geometry*. MIT Press, Cambridge, MA, 1969.
- A. B. J. Novikoff. On convergence proofs on perceptrons. In *Proceedings of the Symposium on the Mathematical Theory of Automata*, volume 12, pages 615–622. Polytechnic Institute of Brooklyn, 1962.
- B. A. Pearlmutter. Fast exact multiplication by the Hessian. *Neural Computation*, 6(1):147–160, 1994.

- M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *Proc. International Conference on Neural Networks*, pages 586–591, San Francisco, CA, 1993. IEEE, New York.
- F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.
- B. Schölkopf and A. Smola. *Learning with Kernels*. MIT Press, Cambridge, MA, 2002.
- B. Schölkopf, J. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson. Estimating the support of a high-dimensional distribution. *Neural Comput.*, 13(7):1443–1471, 2001.
- N. N. Schraudolph. Fast curvature matrix-vector products for second-order gradient descent. *Neural Computation*, 14(7):1723–1738, 2002.
- N. N. Schraudolph. Local gain adaptation in stochastic gradient descent. In *Proc. Intl. Conf. Artificial Neural Networks*, pages 569–574, Edinburgh, Scotland, 1999. IEE, London.
- N. N. Schraudolph and X. Giannakopoulos. Online independent component analysis with local learning rate adaptation. In S. A. Solla, T. K. Leen, and K.-R. Müller, editors, *Neural Information Processing Systems*, volume 12, pages 789–795, Vancouver, Canada, 2000. MIT Press.
- N. N. Schraudolph, J. Yu, and D. Aberdeen. Fast online policy gradient learning with SMD gain vector adaptation. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems 18*, Cambridge, MA, 2006. MIT Press.
- S. Shalev-Shwartz and Y. Singer. A new perspective on an old perceptron algorithm. In P. Auer and R. Meir, editors, *Proc. Annual Conf. Computational Learning Theory*, number 3559 in Lecture Notes in Artificial Intelligence, pages 264–279, Bertinoro, Italy, June 2005. Springer-Verlag.
- F. M. Silva and L. B. Almeida. Acceleration techniques for the backpropagation algorithm. In L. B. Almeida and C. J. Wellekens, editors, *Neural Networks: Proc. EURASIP Workshop*, volume 412 of *Lecture Notes in Computer Science*, pages 110–119. Springer Verlag, 1990.
- R. S. Sutton. Adaptation of learning rate parameters, 1981. URL <http://www.cs.ualberta.ca/~sutton/papers/sutton-81.pdf>. Appendix C of (Barto and Sutton, 1981).
- R. S. Sutton. Gain adaptation beats least squares? In *Proceedings of the 7th Yale Workshop on Adaptive and Learning Systems*, pages 161–166, 1992. URL <http://www.cs.ualberta.ca/~sutton/papers/sutton-92b.pdf>.
- T. Tollenaere. SuperSAB: Fast adaptive back propagation with good scaling properties. *Neural Networks*, 3:561–573, 1990.
- I. Tsochantaridis, T. Hofmann, T. Joachims, and Y. Altun. Support vector machine learning for interdependent and structured output spaces. In *Proc. Intl. Conf. Machine Learning*, New York, NY, USA, 2004. ACM Press. ISBN 1-58113-828-5.
- S. V. N. Vishwanathan, N. N. Schraudolph, M. Schmidt, and K. Murphy. Accelerated training conditional random fields with stochastic gradient methods. In *Proc. Intl. Conf. Machine Learning*, pages 969–976, New York, NY, USA, 2006. ACM Press. ISBN 1-59593-383-2.

J. Weston, A. Bordes, and L. Bottou. Online (and offline) on an even tighter budget. In *Proceedings of International Workshop on Artificial Intelligence and Statistics*, 2005.