Math. Program., Ser. B (2011) 127:3–30
DOI 10.1007/s10107-010-0420-4

FULL LENGTH PAPER

Pegasos: primal estimated sub-gradient solver for SVM

Shai Shalev-Shwartz · Yoram Singer · Nathan Srebro · Andrew Cotter

Received: 30 August 2009 / Accepted: 17 November 2009 / Published online: 16 October 2010

Abstract We describe and analyze a simple and effective stochastic sub-gradient descent algorithm for solving the optimization problem cast by Support Vector Machines (SVM). We prove that the number of iterations required to obtain a solution of accuracy ϵ is $\tilde{O}(1/\epsilon)$, where each iteration operates on a single training example. In contrast, previous analyses of stochastic gradient descent methods for SVMs require $\Omega(1/\epsilon^2)$ iterations. As in previously devised SVM solvers, the number of iterations also scales linearly with $1/\lambda$, where λ is the regularization parameter of SVM. For a linear kernel, the total run-time of our method is $\tilde{O}(d/(\lambda\epsilon))$, where d is a bound on the number of non-zero features in each example. Since the run-time does *not* depend directly on the size of the training set, the resulting algorithm is especially suited for learning from large datasets. Our approach also extends to non-linear kernels while working solely on the primal objective function, though in this case the runtime does depend linearly on the training set size. Our algorithm is particularly well suited for large text classification problems, where we demonstrate an order-of-magnitude speedup over previous SVM learning methods.

S. Shalev-Shwartz
School of Computer Science and Engineering,
The Hebrew University of Jerusalem, Jerusalem, Israel
e-mail: shais@cs.huji.ac.il

Y. Singer
Google, Mountain View, CA, USA
e-mail: singer@google.com

N. Srebro (✉) · A. Cotter
Toyota Technological Institute at Chicago, Chicago, IL, USA
e-mail: nati@ttic.edu

A. Cotter
e-mail: cotter@ttic.edu

Keywords SVM · Stochastic gradient descent

Mathematics Subject Classification (2000) First · Second · More

1 Introduction

Support Vector Machines (SVMs) are effective and popular classification learning tool [36,12]. The task of learning a support vector machine is typically cast as a constrained quadratic programming problem. However, in its native form, it is in fact an unconstrained empirical loss minimization with a penalty term for the norm of the classifier that is being learned. Formally, given a training set $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$, where $\mathbf{x}_i \in \mathbb{R}^n$ and $y_i \in \{+1, -1\}$, we would like to find the minimizer of the problem

$$\min_{\mathbf{w}} \frac{\lambda}{2}\|\mathbf{w}\|^2 + \frac{1}{m}\sum_{(\mathbf{x},y)\in S} \ell(\mathbf{w}; (\mathbf{x}, y)), \tag{1}$$

where

$$\ell(\mathbf{w}; (\mathbf{x}, y)) \;=\; \max\{0, 1 - y\,\langle \mathbf{w}, \mathbf{x}\rangle\}, \tag{2}$$

and $\langle \mathbf{u}, \mathbf{v}\rangle$ denotes the standard inner product between the vectors $\mathbf{u}$ and $\mathbf{v}$. We denote the objective function of Eq. 1 by $f(\mathbf{w})$. We say that an optimization method finds an ϵ-accurate solution $\hat{\mathbf{w}}$ if $f(\hat{\mathbf{w}}) \leq \min_{\mathbf{w}} f(\mathbf{w}) + \epsilon$. The standard SVM problem also includes an unregularized bias term. We omit the bias throughout the coming sections and revisit the incorporation of a bias term in Sect. 6.

We describe and analyze in this paper a simple stochastic sub-gradient descent algorithm, which we call Pegasos, for solving Eq. 1. At each iteration, a single training example is chosen at random and used to estimate a sub-gradient of the objective, and a step with pre-determined step-size is taken in the opposite direction. We show that with high probability over the choice of the random examples, our algorithm finds an ϵ-accurate solution using only $\tilde{O}(1/(\lambda\epsilon))$ iterations, while each iteration involves a single inner product between $\mathbf{w}$ and $\mathbf{x}$. Put differently, the overall runtime required to obtain an ϵ accurate solution is $\tilde{O}(n/(\lambda\epsilon))$, where n is the dimensionality of $\mathbf{w}$ and $\mathbf{x}$. Moreover, this runtime can be reduced to $\tilde{O}(d/(\lambda\epsilon))$ where d is the number of non-zero features in each example $\mathbf{x}$. Pegasos can also be used with non-linear kernels, as we describe in Sect. 4. We would like to emphasize that a solution is found in probability solely due to the randomization steps employed by the algorithm and *not* due to the data set. The data set is not assumed to be random, and the analysis holds for any data set S. Furthermore, the runtime does *not* depend on the number of training examples and thus our algorithm is especially suited for large datasets.

Before indulging into the detailed description and analysis of Pegasos, we would like to draw connections to and put our work in context of some of the more recent work on SVM. For a more comprehensive and up-to-date overview of relevant work see the references in the papers cited below as well as the web site

dedicated to kernel methods at http://www.kernel-machines.org. Due to the centrality of the SVM optimization problem, quite a few methods were devised and analyzed. The different approaches can be roughly divided into the following categories.

Interior Point (IP) methods IP methods (see for instance [7] and the references therein) cast the SVM learning task as a quadratic optimization problem subject to linear constraints. The constraints are replaced with a barrier function. The result is a sequence of unconstrained problems which can be optimized very efficiently using Newton or Quasi-Newton methods. The advantage of IP methods is that the dependence on the accuracy ϵ is double logarithmic, namely, $\log(\log(1/\epsilon))$. Alas, IP methods typically require run time which is cubic in the number of examples m. Moreover, the memory requirements of IP methods are $O(m^2)$ which renders a direct use of IP methods very difficult when the training set consists of many examples. It should be noted that there have been several attempts to reduce the complexity based on additional assumptions (see e.g. [15]). However, the dependence on m remains super linear. In addition, while the focus of the paper is the optimization problem cast by SVM, one needs to bear in mind that the optimization problem is a proxy method for obtaining good classification error on unseen examples. Achieving a very high accuracy in the optimization process is usually unnecessary and does not translate to a significant increase in the generalization accuracy. The time spent by IP methods for finding a single accurate solution may, for instance, be better utilized for trying different regularization values.

Decomposition methods To overcome the quadratic memory requirement of IP methods, decomposition methods such as SMO [29] and SVM-Light [20] tackle the dual representation of the SVM optimization problem, and employ an active set of constraints thus working on a subset of dual variables. In the extreme case, called row-action methods [8], the active set consists of a single constraint. While algorithms in this family are fairly simple to implement and entertain general asymptotic convergence properties [8], the time complexity of most of the algorithms in this family is typically super linear in the training set size m. Moreover, since decomposition methods find a feasible dual solution and their goal is to maximize the dual objective function, they often result in a rather slow convergence rate to the optimum of the primal objective function. (See also the discussion in [19].)

Primal optimization Most existing approaches, including the methods discussed above, focus on the dual of Eq. 1, especially when used in conjunction with non-linear kernels. However, even when non-linear kernels are used, the Representer theorem [23] allows us to re-parametrize $\mathbf{w}$ as $\mathbf{w} = \sum \alpha_i y_i \mathbf{x}_i$ and cast the primal objective Eq. 1 as an unconstrained optimization problem with the variables $\alpha_1, \ldots, \alpha_m$ (see Sect. 4). Tackling the primal objective directly was studied, for example, by Chapelle [10], who considered using smooth loss functions instead of the hinge loss, in which case the optimization problem becomes a smooth unconstrained optimization problem. Chapelle then suggested using various optimization approaches such as conjugate gradient descent and Newton's method. We take a similar approach here, however we cope with the non-differentiability of the hinge-loss directly by using sub-gradients instead of gradients. Another important distinction is that Chapelle views the optimization

problem as a function of the variables α_i. In contrast, though Pegasos maintains the same set of variables, the optimization process is performed with respect to $\mathbf{w}$, see Sect. 4 for details.

Stochastic gradient descent The Pegasos algorithm is an application of a stochastic sub-gradient method (see for example [25,34]). In the context of machine learning problems, the efficiency of the stochastic gradient approach has been studied in [1,3,5,6,26,27]. In particular, it has been claimed and experimentally observed that, "Stochastic algorithms yield the best generalization performance despite being the worst optimization algorithms". This claim has recently received formal treatment in [4,32].

Two concrete algorithms that are closely related to the Pegasos algorithm and are also variants of stochastic sub-gradient methods are the NORMA algorithm [24] and a stochastic gradient algorithm due to Zhang [37]. The main difference between Pegasos and these variants is in the procedure for setting the step size. We elaborate on this issue in Sect. 7. The convergence rate given in [24] implies that the number of iterations required to achieve ϵ-accurate solution is $O(1/(\lambda\,\epsilon)^2)$. This bound is inferior to the corresponding bound of Pegasos. The analysis in [37] for the case of regularized loss shows that the squared Euclidean distance to the optimal solution converges to zero but the rate of convergence depends on the step size parameter. As we show in Sect. 7, tuning this parameter is crucial to the success of the method. In contrast, Pegasos is virtually parameter free. Another related recent work is Nesterov's general primal-dual subgradient method for the minimization of non-smooth functions [28]. Intuitively, the ideas presented in [28] can be combined with the stochastic regime of Pegasos. We leave this direction and other potential extensions of Pegasos for future research.

Online methods Online learning methods are very closely related to stochastic gradient methods, as they operate on only a single example at each iteration. Moreover, many online learning rules, including the Perceptron rule, can be seen as implementing a stochastic gradient step. Many such methods, including the Perceptron and the Passive Aggressive method [11] also have strong connections to the "margin" or norm of the predictor, though they do not directly minimize the SVM objective. Nevertheless, online learning algorithms were proposed as fast alternatives to SVMs (e.g. [16]). Such algorithms can be used to obtain a predictor with low generalization error using an online-to-batch conversion scheme [9]. However, the conversion schemes do not necessarily yield an ϵ-accurate solutions to the original SVM problem and their performance is typically inferior to direct batch optimizers. As noted above, Pegasos shares the simplicity and speed of online learning algorithms, yet it is guaranteed to converge to the optimal SVM solution.

Cutting planes approach Recently, Joachims [21] proposed SVM-Perf, which uses a cutting planes method to find a solution with accuracy ϵ in time $O(md/(\lambda\epsilon^2))$. This bound was later improved by Smola et al. [33] to $O(md/(\lambda\epsilon))$. The complexity guarantee for Pegasos avoids the dependence on the data set size m. In addition, while SVM-Perf yields very significant improvements over decomposition methods for large data sets, our experiments (see Sect. 7) indicate that Pegasos is substantially faster than SVM-Perf.

Fig. 1 The Pegasos algorithm

INPUT: S, λ, T
INITIALIZE: Set $\mathbf{w}_1 = 0$
FOR $t = 1, 2, \ldots, T$
  Choose $i_t \in \{1, \ldots, |S|\}$ uniformly at random.
  Set $\eta_t = \frac{1}{\lambda t}$
  If $y_{i_t} \langle \mathbf{w}_t, \mathbf{x}_{i_t} \rangle < 1$, then:
    Set $\mathbf{w}_{t+1} \leftarrow (1 - \eta_t \lambda)\mathbf{w}_t + \eta_t y_{i_t} \mathbf{x}_{i_t}$
  Else (if $y_{i_t} \langle \mathbf{w}_t, \mathbf{x}_{i_t} \rangle \geq 1$):
    Set $\mathbf{w}_{t+1} \leftarrow (1 - \eta_t \lambda)\mathbf{w}_t$
  [Optional: $\mathbf{w}_{t+1} \leftarrow \min\left\{1, \frac{1/\sqrt{\lambda}}{\|\mathbf{w}_{t+1}\|}\right\} \mathbf{w}_{t+1}$]
OUTPUT: $\mathbf{w}_{T+1}$

2 The Pegasos algorithm

As mentioned above, Pegasos performs stochastic gradient descent on the primal objective Eq. 1 with a carefully chosen stepsize. We describe in this section the core of the Pegasos procedure in detail and provide pseudo-code. We also present a few variants of the basic algorithm and discuss few implementation issues.

2.1 The basic Pegasos algorithms

On each iteration Pegasos operates as follow. Initially, we set $\mathbf{w}_1$ to the zero vector. On iteration t of the algorithm, we first choose a random training example $(\mathbf{x}_{i_t}, y_{i_t})$ by picking an index $i_t \in \{1, \ldots, m\}$ uniformly at random. We then replace the objective in Eq. 1 with an approximation based on the training example $(\mathbf{x}_{i_t}, y_{i_t})$, yielding:

$$f(\mathbf{w}; i_t) \;=\; \frac{\lambda}{2}\|\mathbf{w}\|^2 + \ell(\mathbf{w}; (\mathbf{x}_{i_t}, y_{i_t})). \tag{3}$$

We consider the sub-gradient of the above approximate objective, given by:

$$\nabla_t = \lambda\, \mathbf{w}_t - \mathbb{1}\left[y_{i_t} \left\langle \mathbf{w}_t, \mathbf{x}_{i_t} \right\rangle < 1\right] y_{i_t} \mathbf{x}_{i_t}, \tag{4}$$

where $\mathbb{1}[y \langle \mathbf{w}, \mathbf{x} \rangle < 1]$ is the indicator function which takes a value of one if its argument is true ($\mathbf{w}$ yields non-zero loss on the example $(\mathbf{x}, y)$), and zero otherwise. We then update $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta_t \nabla_t$ using a step size of $\eta_t = 1/(\lambda t)$. Note that this update can be written as:

$$\mathbf{w}_{t+1} \leftarrow \left(1 - \frac{1}{t}\right) \mathbf{w}_t + \eta_t\, \mathbb{1}\left[y_{i_t} \left\langle \mathbf{w}_t, \mathbf{x}_{i_t} \right\rangle < 1\right] y_{i_t} \mathbf{x}_{i_t}. \tag{5}$$

After a predetermined number T of iterations, we output the last iterate $\mathbf{w}_{T+1}$. The pseudo-code of Pegasos is given in Fig. 1.

2.2 Incorporating a projection step

The above description of Pegasos is a verbatim application of the stochastic gradient-descent method. A potential variation is the gradient-projection approach where we limit the set of admissible solutions to the ball of radius $1/\sqrt{\lambda}$. To enforce this property, we project $\mathbf{w}_t$ after each iteration onto this sphere by performing the update:

$$\mathbf{w}_{t+1} \leftarrow \min\left\{1, \frac{1/\sqrt{\lambda}}{\|\mathbf{w}_{t+1}\|}\right\} \mathbf{w}_{t+1}. \tag{6}$$

Our initial presentation and implementation of Pegasos [31] included a projection step, while here we include it as an optional step. However, the newly revised analysis presented in this paper does not require such a projection and establishes almost the same guarantees for the basic (without projection) Pegasos algorithm. We did not notice major differences between the projected and unprojected variants in our experiments (see Sect. 7).

2.3 Mini-batch iterations

In our analysis, we actually consider a more general algorithm that utilizes k examples at each iteration, where $1 \leq k \leq m$ is a parameter that needs to be provided to the algorithm. That is, at each iteration, we choose a subset $A_t \subset [m] = \{1, \ldots, m\}$, $|A_t| = k$, of k examples uniformly at random among all such subsets. When $k = m$ each iteration handles the original objective function. This case is often referred to as batch or deterministic iterations. To underscore the difference between the fully deterministic case and the stochastic case, we refer to the subsamples in the latter case as mini-batches and call the process mini-batch iterates. We thus consider the approximate objective function:

$$f(\mathbf{w}; A_t) = \frac{\lambda}{2}\|\mathbf{w}\|^2 + \frac{1}{k}\sum_{i \in A_t} \ell(\mathbf{w}; (\mathbf{x}_i, y_i)). \tag{7}$$

Note that we overloaded our original definition of f and that the original objective can be denoted as $f(\mathbf{w}) = f(\mathbf{w}; [m])$. As before, we consider the sub-gradient of the approximate objective given by:

$$\nabla_t = \lambda\, \mathbf{w}_t - \frac{1}{k}\sum_{i \in A_t} \mathbb{1}\left[y_i \left\langle \mathbf{w}_t, \mathbf{x}_i \right\rangle < 1\right] y_i \mathbf{x}_i. \tag{8}$$

We update $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta_t \nabla_t$ using the same predetermined step size $\eta_t = 1/(\lambda t)$. Pseudo-code of this more general algorithm is given in Fig. 2. As before, we include an optional projection step.

When $k = m$ we choose $A_t = S$ on each round t and we obtain the deterministic sub-gradient descent method. In the other extreme case, when $k = 1$, we recover the stochastic sub-gradient algorithm of Fig. 1.

Fig. 2 The mini-batch Pegasos algorithm

INPUT: S, λ, T, k
INITIALIZE: Set $\mathbf{w}_1 = 0$
FOR $t = 1, 2, \ldots, T$
Choose $A_t \subseteq [m]$, where $|A_t| = k$, uniformly at random
Set $A_t^+ = \{i \in A_t : y_i \langle \mathbf{w}_t, \mathbf{x}_i \rangle < 1\}$
Set $\eta_t = \frac{1}{\lambda t}$
Set $\mathbf{w}_{t+1} \leftarrow (1 - \eta_t \lambda)\mathbf{w}_t + \frac{\eta_t}{k} \sum_{i \in A_t^+} y_i \mathbf{x}_i$
[Optional: $\mathbf{w}_{t+1} \leftarrow \min\left\{1, \frac{1/\sqrt{\lambda}}{\|\mathbf{w}_{t+1}\|}\right\} \mathbf{w}_{t+1}$]
OUTPUT: $\mathbf{w}_{T+1}$

In the above description we refer to A_t as chosen uniformly at random among the subsets of $[m]$ of size k, i.e. chosen without repetitions. Our analysis still holds when A_t is a multi-set chosen i.i.d. with repetitions.

2.4 Sparse feature vectors

We conclude this section with a short discussion of implementation details when the instances are sparse, namely, when each instance has very few non-zero elements. In this case, we can represent $\mathbf{w}$ as a pair $(\mathbf{v}, a)$ where $\mathbf{v} \in \mathbb{R}^n$ is a vector and a is a scalar. The vector $\mathbf{w}$ is defined as $\mathbf{w} = a\,\mathbf{v}$. We do not require the vector $\mathbf{v}$ to be normalized and hence we over-represent $\mathbf{w}$. However, using this representation, it is easily verified that the total number of operations required for performing one iteration of the basic Pegasos algorithm (with $k = 1$) is $O(d)$, where d is the number of non-zero elements in $\mathbf{x}$.

When projection steps are included, we represent $\mathbf{w}$ as a triplet $(\mathbf{v}, a, \nu)$ with the following variables $\nu = \|\mathbf{w}\| = a\,\|\mathbf{v}\|$. Storing the norm of $\mathbf{w}$ allows us to perform the projection step using a constant number of operations involving only a and ν. After $\mathbf{w}$ is updated, the stored norm ν needs to be updated, which can again be done in time $O(d)$ as before.

3 Analysis

In this section we analyze the convergence properties of Pegasos. Although our main interest is in the special case where $k = 1$ given in Fig. 1, we actually analyze here the more general mini-batch variant of Fig. 2. Throughout this section we denote

$$\mathbf{w}^\star = \operatorname*{argmin}_{\mathbf{w}} f(\mathbf{w}). \tag{9}$$

Recall that on each iteration of the algorithm, we focus on an instantaneous objective function $f(\mathbf{w}; A_t)$. We start by bounding the average instantaneous objective of the algorithm relatively to the average instantaneous objective of the optimal solution. We first need the following lemma which is based on a result from [17], though we provide the proof here for completeness. The lemma relies on the notion of strongly convex functions (see for example [30]). A function f is called λ-strongly convex if $f(\mathbf{w}) - \frac{\lambda}{2}\|\mathbf{w}\|^2$ is a convex function.

Lemma 1 *Let $f_1, \ldots, f_T$ be a sequence of λ-strongly convex functions. Let B be a closed convex set and define $\Pi_B(\mathbf{w}) = \arg\min_{\mathbf{w}' \in B} \|\mathbf{w} - \mathbf{w}'\|$. Let $\mathbf{w}_1, \ldots, \mathbf{w}_{T+1}$ be a sequence of vectors such that $\mathbf{w}_1 \in B$ and for $t \geq 1$, $\mathbf{w}_{t+1} = \Pi_B(\mathbf{w}_t - \eta_t \nabla_t)$, where ∇_t belongs to the sub-gradient set of f_t at $\mathbf{w}_t$ and $\eta_t = 1/(\lambda t)$. Assume that for all t, $\|\nabla_t\| \leq G$. Then, for all $\mathbf{u} \in B$ we have*

$$\frac{1}{T}\sum_{t=1}^{T} f_t(\mathbf{w}_t) \leq \frac{1}{T}\sum_{t=1}^{T} f_t(\mathbf{u}) + \frac{G^2(1+\ln(T))}{2\,\lambda\,T}.$$

Proof Since f_t is strongly convex and ∇_t is in the sub-gradient set of f_t at $\mathbf{w}_t$ we have that (see [30])

$$\langle \mathbf{w}_t - \mathbf{u}, \nabla_t \rangle \;\geq\; f_t(\mathbf{w}_t) - f_t(\mathbf{u}) + \tfrac{\lambda}{2}\|\mathbf{w}_t - \mathbf{u}\|^2. \tag{10}$$

Next, we show that

$$\langle \mathbf{w}_t - \mathbf{u}, \nabla_t \rangle \;\leq\; \frac{\|\mathbf{w}_t - \mathbf{u}\|^2 - \|\mathbf{w}_{t+1} - \mathbf{u}\|^2}{2\,\eta_t} + \tfrac{\eta_t}{2} G^2. \tag{11}$$

Let $\mathbf{w}'_t$ denote $\mathbf{w}_t - \eta_t \nabla_t$. Since $\mathbf{w}_{t+1}$ is the projection of $\mathbf{w}'_t$ onto B, and $\mathbf{u} \in B$ we have that $\|\mathbf{w}'_t - \mathbf{u}\|^2 \geq \|\mathbf{w}_{t+1} - \mathbf{u}\|^2$. Therefore,

$$\begin{aligned}\|\mathbf{w}_t - \mathbf{u}\|^2 - \|\mathbf{w}_{t+1} - \mathbf{u}\|^2 &\geq \|\mathbf{w}_t - \mathbf{u}\|^2 - \|\mathbf{w}'_t - \mathbf{u}\|^2 \\ &= 2\eta_t \langle \mathbf{w}_t - \mathbf{u}, \nabla_t \rangle - \eta_t^2 \|\nabla_t\|^2.\end{aligned}$$

Rearranging the above and using the assumption $\|\nabla_t\| \leq G$ yields Eq. 11. Comparing Eqs. 10 and 11 and summing over t we obtain

$$\begin{aligned}&\sum_{t=1}^{T} \left(f_t(\mathbf{w}_t) - f_t(\mathbf{u})\right) \\ &\quad\leq \sum_{t=1}^{T} \left(\frac{\|\mathbf{w}_t - \mathbf{u}\|^2 - \|\mathbf{w}_{t+1} - \mathbf{u}\|^2}{2\,\eta_t} + \tfrac{\lambda}{2}\|\mathbf{w}_t - \mathbf{u}\|^2 \right) + \frac{G^2}{2}\sum_{t=1}^{T} \eta_t\,.\end{aligned}$$

Next, we use the definition $\eta_t = 1/(\lambda\, t)$ and note that the first sum on the right-hand side of the above equation collapses to $-\lambda\,(T+1)\,\|\mathbf{w}_{T+1} - \mathbf{u}\|^2$. Thus,

$$\begin{aligned}\sum_{t=1}^{T} (f_t(\mathbf{w}_t) - f_t(\mathbf{u})) &\leq -\lambda\,(T+1)\,\|\mathbf{w}_{T+1} - \mathbf{u}\|^2 + \frac{G^2}{2\,\lambda}\sum_{t=1}^{T}\frac{1}{t} \\ &\leq \frac{G^2}{2\,\lambda}(1+\ln(T))\,.\end{aligned}$$

□

Based on the above lemma, we are now ready to bound the average instantaneous objective of Pegasos.

Theorem 1 *Assume that for all* $(\mathbf{x}, y) \in S$ *the norm of* $\mathbf{x}$ *is at most* R. *Let* $\mathbf{w}^\star$ *be as defined in Eq. 9 and let* $c = (\sqrt{\lambda} + R)^2$ *whenever we perform the projection step and* $c = 4\,R^2$ *whenever we do not perform the projection step. Then, for* $T \geq 3$,

$$\frac{1}{T}\sum_{t=1}^{T} f(\mathbf{w}_t; A_t) \;\leq\; \frac{1}{T}\sum_{t=1}^{T} f(\mathbf{w}^\star; A_t) + \frac{c(1+\ln(T))}{2\lambda T}.$$

Proof To simplify our notation we use the shorthand $f_t(\mathbf{w}) = f(\mathbf{w}; A_t)$. The update of the algorithm can be rewritten as $\mathbf{w}_{t+1} = \Pi_B(\mathbf{w}_t - \eta_t \nabla_t)$, where ∇_t is defined in Eq. 8 and B is the Euclidean ball of radius $1/\sqrt{\lambda}$ if we perform a projection step and otherwise $B = \mathbb{R}^n$. Thus, to prove the theorem it suffices to show that the conditions stated in Lemma 1 hold. Since f_t is a sum of a λ-strongly convex function ($\frac{\lambda}{2}\|\mathbf{w}\|^2$) and a convex function (the average hinge-loss over A_t), it is clearly λ-strongly convex. Next, we derive a bound on $\|\nabla_t\|$. If we perform a projection step then using the fact that $\|\mathbf{w}_t\| \leq 1/\sqrt{\lambda}$ and that $\|\mathbf{x}\| \leq R$ combined with the triangle inequality we obtain $\|\nabla_t\| \leq \sqrt{\lambda} + R$. If we do not perform a projection step then we can first rewrite the update step as

$$\mathbf{w}_{t+1} \;=\; \left(1 - \tfrac{1}{t}\right)\mathbf{w}_t - \tfrac{1}{t\lambda}\mathbf{v}_t, \tag{12}$$

where $\mathbf{v}_t = \frac{1}{|A_t|}\sum_{i \in A_t} \mathbb{1}_{y_i \langle \mathbf{w}_t, \mathbf{x}_t\rangle < 1} y_i \mathbf{x}_i$. Therefore, the initial weight of each $\mathbf{v}_i$ is $\frac{1}{\lambda i}$ and then on rounds $j = i+1, \ldots, t$ it will be multiplied by $1 - \frac{1}{j} = \frac{j-1}{j}$. Thus, the overall weight of $\mathbf{v}_i$ in $\mathbf{w}_{t+1}$ is

$$\tfrac{1}{\lambda i}\prod_{j=i+1}^{t}\tfrac{j-1}{j} \;=\; \tfrac{1}{\lambda t},$$

which implies that we can rewrite $\mathbf{w}_{t+1}$ as

$$\mathbf{w}_{t+1} = \frac{1}{\lambda t}\sum_{i=1}^{t}\mathbf{v}_i. \tag{13}$$

From the above we immediately get that $\|\mathbf{w}_{t+1}\| \leq R/\lambda$ and therefore $\|\nabla_t\| \leq 2R$. Finally, we need to prove that $\mathbf{w}^\star \in B$. If we do not perform projections then we have $\mathbf{w}^\star \in \mathbb{R}^n = B$. Otherwise, we need to show that $\|\mathbf{w}^\star\| \leq 1/\sqrt{\lambda}$. To do so, we examine the dual form of the SVM problem and use the strong duality theorem. In its more traditional form, the SVM learning problem was described as the following constrained optimization problem,

$$\frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{m}\xi_i \;\text{ s.t. }\; \forall i \in [m] : \xi_i \geq 0,\; \xi_i \geq 1 - y_i \langle \mathbf{w}, \mathbf{x}_i\rangle. \tag{14}$$

Setting $C = 1/(\lambda m)$ this problem becomes equivalent to our formulation given in Eqs. 1 and 2. The dual problem of Eq. 14 is,

$$\sum_{i=1}^{m} \alpha_i - \frac{1}{2} \left\| \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i \right\|^2 \quad \text{s.t. } \forall i \in [m] : 0 \leq \alpha_i \leq C. \tag{15}$$

Let us denote the optimal primal and dual solutions by $(\mathbf{w}^\star, \boldsymbol{\xi}^\star)$ and $\boldsymbol{\alpha}^\star$, respectively. The primal solution can be written in terms of its dual counterpart as $\mathbf{w}^\star = \sum_{i=1}^{m} \alpha_i^\star y_i \mathbf{x}_i$. At the optimum value $\boldsymbol{\alpha}^\star$, Eq. 15 can be rewritten as,

$$\|\boldsymbol{\alpha}^\star\|_1 - \frac{1}{2}\|\mathbf{w}^\star\|^2.$$

Moreover, from strong duality we know that the primal objective value is equal to the dual objective value at the optimum, thus

$$\frac{1}{2}\|\mathbf{w}^\star\|^2 + C\,\|\boldsymbol{\xi}^\star\|_1 = \|\boldsymbol{\alpha}^\star\|_1 - \frac{1}{2}\|\mathbf{w}^\star\|^2.$$

note that $\|\boldsymbol{\alpha}^\star\|_\infty \leq C = \frac{1}{\lambda m}$. Therefore, $\|\boldsymbol{\alpha}^\star\|_1 \leq 1/\lambda$ and we get that

$$\frac{1}{2}\|\mathbf{w}^\star\|^2 \leq \frac{1}{2}\|\mathbf{w}^\star\|^2 + C\,\|\boldsymbol{\xi}^\star\|_1 = \|\boldsymbol{\alpha}^\star\|_1 - \frac{1}{2}\|\mathbf{w}^\star\|^2 \leq \frac{1}{\lambda} - \frac{1}{2}\|\mathbf{w}^\star\|^2 .$$

Rearranging the terms yields $\|\mathbf{w}^\star\| \leq 1/\sqrt{\lambda}$. The bound in the theorem now follows from Lemma 1. □

We now turn to obtaining a bound on the overall objective $f(\mathbf{w}_t)$ evaluated at a single predictor $\mathbf{w}_t$. The convexity of f implies that:

$$f\left(\frac{1}{T}\sum_{t=1}^{T} \mathbf{w}_t\right) \;\leq\; \frac{1}{T}\sum_{t=1}^{T} f(\mathbf{w}_t). \tag{16}$$

Using the above inequality and Theorem 1, we immediately obtain Corollary 1 which provides a convergence analysis for the deterministic case when $k = m$ where $f(\mathbf{w}, A_t) = f(\mathbf{w})$.

Corollary 1 *Assume that the conditions stated in Theorem 1 and that $A_t = S$ for all t. Let $\bar{\mathbf{w}} = \frac{1}{T}\sum_{t=1}^{T} \mathbf{w}_t$. Then,*

$$f\,(\bar{\mathbf{w}}) \;\leq\; f(\mathbf{w}^\star) + \frac{c(1+\ln(T))}{2\,\lambda\,T}.$$

When $A_t \subset S$, Corollary 1 no longer holds. However, Kakade and Tewari [22] have shown that a similar bound holds with high probability as long as A_t is sampled from S.

Lemma 2 (Corollary 7 in [22]) *Assume that the conditions stated in Theorem 1 hold and that for all t, each element in A_t is sampled uniformly at random from S (with or without repetitions). Assume also that $R \geq 1$ and $\lambda \leq 1/4$. Then, with a probability of at least $1 - \delta$ we have*

$$\frac{1}{T}\sum_{t=1}^{T} f(\mathbf{w}_t) - f(\mathbf{w}^\star) \leq \frac{21\, c\, \ln(T/\delta)}{\lambda\, T}.$$

Combining the above with Eq. 16 we immediately obtain the following corrolary.

Corollary 2 *Assume that the conditions stated in Lemma 2 hold and let $\bar{\mathbf{w}} = \frac{1}{T}\sum_{t=1}^{T} \mathbf{w}_t$. Then, with probability of at least $1 - \delta$ we have*

$$f(\bar{\mathbf{w}}) \leq f(\mathbf{w}^\star) + \frac{21\, c\, \ln(T/\delta)}{\lambda\, T}.$$

The previous corollaries hold for the average hypothesis $\bar{\mathbf{w}}$. In practice, the final hypothesis, $\mathbf{w}_{T+1}$, often provides better results. We next bridge this gap by providing a similar convergence rate for a different mechanism of choosing the output vector. To do so, we first show that at least half of the hypotheses are good.

Lemma 3 *Assume that the conditions stated in Lemma 2 hold. Then, if t is selected at random from $[T]$, we have with a probability of at least $\frac{1}{2}$ that*

$$f(\mathbf{w}_t) \leq f(\mathbf{w}^\star) + \frac{42\, c\, \ln(T/\delta)}{\lambda\, T}.$$

Proof Define a random variable $Z = f(\mathbf{w}_t) - f(\mathbf{w}^\star)$ where the randomness is over the choice of the index t. From the definition of $\mathbf{w}^\star$ as the minimizer of $f(\mathbf{w})$ we clearly have that Z is a non-negative random variable. Thus, from Markov inequality $\mathbb{P}[Z \geq 2\,\mathbb{E}[Z]] \leq \frac{1}{2}$. The claim now follows by combining the fact that $\mathbb{E}[Z] = \frac{1}{T}\sum_{t=1}^{T} f(\mathbf{w}_t) - f(\mathbf{w}^\star)$ with the bound given in Lemma 2. □

Based on the above lemma we conclude that if we terminate the procedure at a random iteration, in at least half of the cases the last hypothesis is an accurate solution. Therefore, we can simply try a random stopping time and evaluate the error of the last hypothesis.[1] The above lemma tells us that on average after two attempts we are likely to find a good solution.

4 Using Mercer kernels

One of the main benefits of SVMs is that they can be used with *kernels* rather then with direct access to the feature vectors $\mathbf{x}$. The crux of this property stems from the

[1] To do so, we can simply calculate the objective on the entire data set or estimate it according to a sample of size $O(1/(\lambda\,\epsilon))$, where ϵ is the desired accuracy (see [35]).

Representer Theorem [23], which implies that the optimal solution of Eq. 1 can be expressed as a linear combination of the training instances. It is therefore possible to train and use a SVM without direct access to the training instances, and instead only access their inner products as specified through a kernel operator. That is, instead of considering predictors which are linear functions of the training instances $\mathbf{x}$ themselves, we consider predictors which are linear functions of some implicit mapping $\phi(\mathbf{x})$ of the instances. Training then involves solving the minimization problem:

$$\min_{\mathbf{w}} \frac{\lambda}{2}\|\mathbf{w}\|^2 + \frac{1}{m}\sum_{(\mathbf{x},y)\in S} \ell(\mathbf{w};(\phi(\mathbf{x}),y)), \tag{17}$$

where

$$\ell(\mathbf{w};(\phi(\mathbf{x}),y)) \;=\; \max\{0, 1 - y\,\langle \mathbf{w}, \phi(\mathbf{x})\rangle\}. \tag{18}$$

However, the mapping $\phi(\cdot)$ is never specified explicitly but rather through a kernel operator $K(\mathbf{x},\mathbf{x}') = \langle \phi(\mathbf{x}), \phi(\mathbf{x}')\rangle$ yielding the inner products after the mapping $\phi(\cdot)$.

One possible and rather common approach for solving the optimization problem 17 is to switch to the dual problem, which can be written in terms of inner products of vectors $\phi(\cdot)$. Therefore, the dual problem can be solely expressed using kernel operators. However, solving the dual problem is not necessary. Following [16,24,10], the approach we take here is to directly minimize the primal problem while still using kernels.

We now show that the Pegasos algorithm can be implemented using only kernel evaluations, without direct access to the feature vectors $\phi(\mathbf{x})$ or explicit access to the weight vector $\mathbf{w}$. For simplicity, we focus on adapting the basic Pegasos algorithm given in Fig. 1 without the optional projection step. As we have shown in the proof of Theorem 1 (in particular, Eq. 13), for all t we can rewrite $\mathbf{w}_{t+1}$ as

$$\mathbf{w}_{t+1} = \frac{1}{\lambda t}\sum_{i=1}^{t} \mathbb{1}_{y_{i_t}\langle \mathbf{w}_t, \phi(\mathbf{x}_{i_t})\rangle < 1} y_{i_t}\phi(\mathbf{x}_{i_t}).$$

For each t, let $\boldsymbol{\alpha}_{t+1} \in \mathbb{R}^m$ be the vector such that $\alpha_{t+1}[j]$ counts how many times example j has been selected so far and we had a non-zero loss on it, namely,

$$\alpha_{t+1}[j] \;=\; |\{t' \le t : i_{t'} = j \wedge y_j \langle \mathbf{w}_{t'}, \phi(\mathbf{x}_j)\rangle < 1\}|.$$

Instead of keeping in memory the weight vector $\mathbf{w}_{t+1}$, we will represent $\mathbf{w}_{t+1}$, using $\boldsymbol{\alpha}_{t+1}$ according to

$$\mathbf{w}_{t+1} = \frac{1}{\lambda t}\sum_{j=1}^{m} \alpha_{t+1}[j]\, y_j \phi(\mathbf{x}_j).$$

It is now easy to implement the Pegasos algorithm by maintaining the vector $\boldsymbol{\alpha}$. The pseudo-code of this kernelized implementation of Pegasos is given in Fig. 3. Note that

Fig. 3 The kernelized Pegasos algorithm

INPUT: S, λ, T

INITIALIZE: Set $\boldsymbol{\alpha}_1 = 0$

FOR $t = 1, 2, \ldots, T$

Choose $i_t \in \{0, \ldots, |S|\}$ uniformly at random.

For all $j \neq i_t$, set $\alpha_{t+1}[j] = \alpha_t[j]$

If $y_{i_t} \frac{1}{\lambda t} \sum_j \alpha_t[j] y_{i_t} K(\mathbf{x}_{i_t}, \mathbf{x}_j) < 1$, then:

Set $\alpha_{t+1}[i_t] = \alpha_t[i_t] + 1$

Else:

Set $\alpha_{t+1}[i_t] = \alpha_t[i_t]$

OUTPUT: $\boldsymbol{\alpha}_{T+1}$

only one element of $\boldsymbol{\alpha}$ is changed at each iteration. It is also important to emphasize that although the feature mapping $\phi(\cdot)$ was used in the above mathematical derivations, the pseudo-code of the algorithm itself makes use *only* of kernel evaluations and obviously does *not* refer to the implicit mapping $\phi(\cdot)$.

Since the iterates $\mathbf{w}_t$ remain as before (just their representation changes), the guarantees on the accuracy after a number of iterations are still valid. We are thus guaranteed to find an ϵ-accurate solution after $\tilde{O}(1/(\lambda\epsilon))$ iterations. However, checking for non-zero loss at iteration t might now require as many as $\min(t, m)$ kernel evaluations, bringing the overall runtime to $\tilde{O}(m/(\lambda\epsilon))$. Therefore, although the number of iterations required does not depend on the number of training examples, the runtime does.

It is worthwhile pointing out that even though the solution is represented in terms of the variables $\boldsymbol{\alpha}$, we are still calculating the sub-gradient with respect to the weight vector $\mathbf{w}$. A different approach, that was taken, e.g., by Chapelle [10], is to rewrite the primal problem as a function of $\boldsymbol{\alpha}$ and then taking gradients with respect to $\boldsymbol{\alpha}$. Concretely, the Representer theorem guarantees that the optimal solution of Eq. 17 is spanned by the training instances, i.e. it is of the form, $\mathbf{w} = \sum_{i=1}^{m} \alpha[i]\phi(\mathbf{x}_i)$. In optimizing Eq. 17 we can therefore focus only on predictors of this form, parametrized through $\boldsymbol{\alpha} \in \mathbb{R}^m$. The training objective can then be written in terms of the α variables and kernel evaluations:

$$\min_{\alpha} \frac{\lambda}{2} \sum_{i,j=1}^{m} \alpha[i]\alpha[j]K(\mathbf{x}_i, \mathbf{x}_j) + \frac{1}{m} \sum_{i=1}^{m} \max\{0, 1 - y_i \sum_{j=1}^{m} \alpha[j]K(\mathbf{x}_i, \mathbf{x}_j)\}. \quad (19)$$

Now, one can use stochastic gradient updates for solving Eq. 19, where gradients should be taken w.r.t. $\boldsymbol{\alpha}$. We emphasize again that our approach is different as we compute sub-gradients w.r.t. $\mathbf{w}$. Setting the step direction according to the sub-gradient w.r.t $\mathbf{w}$ has two important advantages. First, only at most one new non-zero $\alpha[i]$ is introduced at each iteration, as opposed to a sub-gradient step w.r.t. $\boldsymbol{\alpha}$ which will involve all m coefficients. More importantly, the objective given in Eq. 19 is not necessarily strongly-convex w.r.t. $\boldsymbol{\alpha}$, even though it is strongly convex w.r.t. $\mathbf{w}$. Thus, a gradient descent approach using gradients w.r.t. $\boldsymbol{\alpha}$ might require $\Omega(1/\epsilon^2)$ iterations

to achieve accuracy ϵ. Interestingly, Chapelle also proposes preconditioning the gradients w.r.t. $\boldsymbol{\alpha}$ by the kernel matrix, which effectively amounts to taking gradients w.r.t. $\mathbf{w}$, as we do here. Unsurprisingly given the above discussion, Chapelle observes much better results with this preconditioning.

5 Other prediction problems and loss functions

So far, we focused on the SVM formulation for binary classification using the hinge-loss. In this section we show how Pegasos can seamlessly be adapted to other prediction problems in which we use other loss functions.

The basic observation is that the only place in which we use the fact that $\ell(\mathbf{w}; (\mathbf{x}, y))$ is the hinge-loss (Eq. 2) is when we calculated a sub-gradient of $\ell(\mathbf{w}; (\mathbf{x}, y))$ with respect to $\mathbf{w}$. The assumptions we made are that ℓ is convex and that the norm of the sub-gradient is at most R. The generality of these assumptions implies that we can apply Pegasos with any loss function which satisfies these requirements.

5.1 Examples

Example 1 (Binary classification with the log-loss) Instead of the hinge-loss, other loss functions can also be used with binary labels $y \in \{+1, -1\}$. A popular choice is the log-loss defined as: $\ell(\mathbf{w}, (\mathbf{x}, y)) = \log(1 + \exp(-y \langle \mathbf{w}, \mathbf{x} \rangle))$. It is easy to verify that the log loss is a convex function whose gradient w.r.t. $\mathbf{w}$ satisfies $\|\nabla\| \leq \|\mathbf{x}\|$.

Example 2 (Regression with the ϵ-insensitive loss) We now turn to regression problems over the reals, that is $y \in \mathbb{R}$. The standard Support Vector Regression formulation uses the loss function defined as $\ell(\mathbf{w}; (\mathbf{x}, y)) = \max\{0, |\langle \mathbf{w}, \mathbf{x} \rangle - y| - \epsilon\}$. This loss is also convex with a sub-gradient bounded by $\|\mathbf{x}\|$.

Example 3 (Cost-sensitive multiclass categorization) In multi-class categorization problems, the goal is to predict a label $y \in \mathcal{Y}$ where $\mathcal{Y}$ is a finite discrete set of classes. A possible loss function is the so-called cost-sensitive loss defined as:

$$\ell(\mathbf{w}; (\mathbf{x}, y)) = \max_{y' \in \mathcal{Y}} \delta(y', y) - \langle \mathbf{w}, \phi(\mathbf{x}, y) \rangle + \langle \mathbf{w}, \phi(\mathbf{x}, y') \rangle, \tag{20}$$

where $\delta(y', y)$ is the cost of predicting y' instead of y and $\phi(\mathbf{x}, y)$ is a mapping from input-label pair $(\mathbf{x}, y)$ into a vector space. See for example [11]. The multiclass loss is again a convex loss function whose sub-gradient is bounded by $2 \max_{y'} \|\phi(\mathbf{x}, y')\|$.

Example 4 (Multiclass categorization with the log-loss) Given the same setting of the above multiclass example, we can also generalize the log loss to handle multiclass problems. Omitting the cost term, the multiclass loss amounts to:

$$\ell(\mathbf{w}; (\mathbf{x}, y)) = \log \left(1 + \sum_{r \neq y} e^{\langle \mathbf{w}, \phi(\mathbf{x}, r) \rangle - \langle \mathbf{w}, \phi(\mathbf{x}, y) \rangle} \right), \tag{21}$$

where $\phi(\mathbf{x}, y)$ is defined above. The log-loss version of the multiclass loss is convex as well with a bounded sub-gradient whose value is at most, $2 \max_{y'} \|\phi(\mathbf{x}, y')\|$.

Example 5 (Sequence prediction) Sequence prediction is similar to cost-sensitive multi-class categorization, but the set of targets, $\mathcal{Y}$, can be very large. For example, in phoneme recognition tasks, $\mathcal{X}$ is the set of all speech utterances and $\mathcal{Y}$ is the set of all phoneme sequences. Therefore, $|\mathcal{Y}|$ is exponential in the length of the sequence. Nonetheless, if the functions ϕ and δ adheres to a specific structure then we can still calculate sub-gradients efficiently and therefore solve the resulting optimization problem efficiently using Pegasos.

To recap the examples provided above we give a table of the sub-gradients of some popular loss functions. To remind the reader, given a convex function $f(\mathbf{w})$, a sub-gradient of f at $\mathbf{w}_0$ is a vector $\mathbf{v}$ which satisfies:

$$\forall \mathbf{w}, \quad f(\mathbf{w}) - f(\mathbf{w}_0) \geq \langle \mathbf{v}, \mathbf{w} - \mathbf{w}_0 \rangle .$$

The following two properties of sub-gradients are used for calculating the sub-gradients in the table below.

1. If $f(\mathbf{w})$ is differentiable at $\mathbf{w}_0$, then the gradient of f at $\mathbf{w}_0$ is the unique sub-gradient of f at $\mathbf{w}_0$.
2. If $f(\mathbf{w}) = \max_i f_i(\mathbf{w})$ for r differentiable functions $f_1, \ldots, f_r$, and $j = \arg\max_i f_i(\mathbf{w}_0)$, then the gradient of f_j at $\mathbf{w}_0$ is a sub-gradient of f at $\mathbf{w}_0$.

Based on the above two properties, we now show explicitly how to calculate a sub-gradient for several loss functions. In the following table, we use the notation $z = \langle \mathbf{w}_t, \mathbf{x}_i \rangle$.

Loss function	Subgradient
$\ell(z, y_i) = \max\{0, 1 - y_i z\}$	$\mathbf{v}_t = \begin{cases} -y_i \mathbf{x}_i & \text{if } y_i z < 1 \\ \mathbf{0} & \text{otherwise} \end{cases}$
$\ell(z, y_i) = \log(1 + e^{-y_i z})$	$\mathbf{v}_t = -\frac{y_i}{1+e^{y_i z}} \mathbf{x}_i$
$\ell(z, y_i) = \max\{0, \lvert y_i - z \rvert - \epsilon\}$	$\mathbf{v}_t = \begin{cases} \mathbf{x}_i & \text{if } z - y_i > \epsilon \\ -\mathbf{x}_i & \text{if } y_i - z > \epsilon \\ \mathbf{0} & \text{otherwise} \end{cases}$
$\ell(z, y_i) = \max_{y \in \mathcal{Y}} \delta(y, y_i) - z_{y_i} + z_y$	$\mathbf{v}_t = \phi(\mathbf{x}_i, \hat{y}) - \phi(\mathbf{x}_i, y_i)$ where $\hat{y} = \arg\max_y \delta(y, y_i) - z_{y_i} + z_y$
$\ell(z, y_i) = \log\left(1 + \sum_{r \neq y_i} e^{z_r - z_{y_i}}\right)$	$\mathbf{v}_t = \sum_r p_r \phi(\mathbf{x}_i, r) - \phi(\mathbf{x}_i, y_i)$ where $p_r = e^{z_r} / \sum_j e^{z_j}$

6 Incorporating a bias term

In many applications, the weight vector $\mathbf{w}$ is augmented with a bias term which is a scalar, typically denoted as b. The prediction for an instance $\mathbf{x}$ becomes $\langle \mathbf{w}, \mathbf{x} \rangle + b$ and the loss is accordingly defined as,

$$\ell\left((\mathbf{w}, b); (\mathbf{x}, y)\right) = \max\{0, 1 - y(\langle \mathbf{w}, \mathbf{x}\rangle + b)\}. \tag{22}$$

The bias term often plays a crucial role when the distribution of the labels is uneven as is typically the case in text processing applications where the negative examples vastly outnumber the positive ones. We now review several approaches for learning the bias term while underscoring the advantages and disadvantages of each approach.

The first approach is rather well known and its roots go back to early work on pattern recognition [14]. This approach simply amounts to adding one more feature to each instance $\mathbf{x}$ thus increasing the dimension to $n+1$. The artificially added feature always takes the same value. We assume without loss of generality that the value of the constant feature is 1. Once the constant feature is added the rest of the algorithm remains intact, thus the bias term is not explicitly introduced. The analysis can be repeated verbatim and we therefore obtain the same convergence rate for this modification. Note however that by equating the $n+1$ component of $\mathbf{w}$ with b, the norm-penalty counterpart of f becomes $\|\mathbf{w}\|^2 + b^2$. The disadvantage of this approach is thus that we solve a relatively different optimization problem. On the other hand, an obvious advantage of this approach is that it requires no modifications to the algorithm itself rather than a modest increase in the dimension and it can thus be used without any restriction on A_t.

An alternate approach incorporates b explicitly by defining the loss as given in Eq. 22 while *not* penalizing for b. Formally, the task is to find an approximate solution to the following problem:

$$\min_{\mathbf{w},b} \frac{\lambda}{2}\|\mathbf{w}\|^2 + \frac{1}{m}\sum_{(\mathbf{x},y)\in S} [1 - y(\langle \mathbf{w}, \mathbf{x}\rangle + b)]_+ . \tag{23}$$

Note that all the sub-gradients calculations with respect to $\mathbf{w}$ remain intact. The sub-gradient with respect to b is also simple to compute. This approach is also very simple to implement and can be used with any choice of A_t, in particular, sets consisting of a single instance. The caveat of this approach is that the function f ceases to be strongly convex due to the incorporation of b. Precisely, the objective function f becomes piece-wise linear in the direction of b and is thus no longer strongly convex. Therefore, the analysis presented in the previous section no longer holds. An alternative proof technique yields a slower convergence rate of $O(1/\sqrt{T})$.

A third method entertains the advantages of the two methods above at the price of a more complex algorithm that is applicable only for large batch sizes (large values of k), but not for the basic Pegasos algorithm (with $k = 1$). The main idea is to rewrite the optimization problem given in Eq. 23 as $\min_{\mathbf{w}} \frac{\lambda}{2}\|\mathbf{w}\|^2 + g(\mathbf{w}; S)$ where

$$g(\mathbf{w}; S) = \min_{b} \frac{1}{m}\sum_{(\mathbf{x},y)\in S} [1 - y(\langle \mathbf{w}, \mathbf{x}\rangle + b)]_+ . \tag{24}$$

Based on the above, we redefine $f(\mathbf{w}; A_t)$ to be $\frac{\lambda}{2}\|\mathbf{w}\|^2 + g(\mathbf{w}; A_t)$. On each iteration of the algorithm, we find a sub-gradient of $f(\mathbf{w}; A_t)$ and subtract it (multiplied by η_t) from $\mathbf{w}_t$. The problem however is how to find a sub-gradient of $g(\mathbf{w}; A_t)$, as $g(\mathbf{w}; A_t)$

is defined through a minimization problem over b. This essentially amounts to solving the minimization problem in Eq. 24. The latter problem is a generalized weighted median problem that can be solved efficiently in time $O(k)$. The above adaptation indeed work for the case $k = m$ where we have $A_t = S$ and we obtain the same rate of convergence as in the no-bias case. However, when $A_t \neq S$ we cannot apply the analysis from the previous section to our case since the expectation of $f(\mathbf{w}; A_t)$ over the choice of A_t is no longer equal to $f(\mathbf{w}; S)$. When A_t is large enough, it might be possible to use more involved measure concentration tools to show that the expectation of $f(\mathbf{w}; A_t)$ is close enough to $f(\mathbf{w}; S)$ so as to still obtain fast convergence properties.

A final possibility is to search over the bias term b in an external loop, optimizing the weight vector $\mathbf{w}$ using Pegasos for different possible values of b. That is, consider the objective:

$$J(b; S) = \min_{\mathbf{w}} \frac{1}{m} \sum_{(\mathbf{x}, y) \in S} [1 - y(\langle \mathbf{w}, \mathbf{x} \rangle + b)]_+ . \tag{25}$$

For a fixed b, the minimization problem in Eq. 25 is very similar to SVM training without a bias term, and can be optimized using Pegasos. The objective $J(b; S)$ is convex in the single scalar variable b, and so $J(b; S)$ can be optimized to within accuracy ϵ by binary search using $O(\log 1/\epsilon)$ evaluations of $J(b; S)$, i.e. $O(\log 1/\epsilon)$ applications of the Pegasos algorithm. Since this modification introduced only an additional logarithmic factor, the overall runtime for training an SVM with a bias term remains $\tilde{O}(d/(\lambda\epsilon))$. Although incorporating a regularized or unregularized bias term might be better in practice, the latter "outer loop" approach is the only method that we are aware of which guarantees an overall runtime of $\tilde{O}(d/(\lambda\epsilon))$.

7 Experiments

In this section we present experimental results that demonstrate the merits of our algorithm. We start by demonstrating the practicality of Pegasos for solving large scale linear problems, especially when the feature vectors are sparse. In particular, we compare its runtime on three large datasets to the runtimes of the state-of-the-art solver SVM-Perf [21], a cutting plane algorithm designed specifically for use with sparse feature vectors, as well as of two more conventional SVM solvers: LASVM [2] and SVM-Light [20]. We next demonstrate that Pegasos can also be a reasonable approach for large scale problems involving non-linear kernels by comparing it to LASVM and SVM-Light on four large data sets using Gaussian kernels. We then investigate the effect of various parameters and implementation choices on Pegasos: we demonstrate the runtime dependence on the regularization parameter λ; we explore the empirical behavior of the mini-batch variant and the dependence on the mini-batch size k; and we compare the effect of sampling training examples both with and without replacement.

Finally, we compare Pegasos to two previously proposed methods that are based on stochastic gradient descent: Norma [24] by Kivinen et al. and to the method by Zhang [37].

We also include in our experiments a comparison with stochastic Dual Coordinate Ascent (DCA). Following Pegasos's initial presentation [31], stochastic DCA was suggested as an alternative optimization method for SVMs [18]. DCA shares numerous similarities with Pegasos. Like Pegasos, at each iteration only a single (random) training example $(y_i, \mathbf{x}_i)$ is considered, and if $y_i \langle \mathbf{w}, \mathbf{x}_i \rangle < 1$, an update of the form $\mathbf{w} \leftarrow \mathbf{w} + \eta y_i \mathbf{x}_i$ is performed. However, the DCA step size η is not predetermined, but rather chosen so as to maximize the dual objective. DCA's convergence properties and the differences between DCA and Pegasos behavior are not yet well understood. For informational purposes, we include a comparison to DCA in our empirical evaluations.

Our implementation of Pegasos is based on the algorithm from Fig. 1, outputting the last weight vector rather than the average weight vector, as we found that in practice it performs better. We did not incorporate a bias term in any of our experiments. We found that including an unregularized bias term does not significantly change the predictive performance for any of the data sets used. Furthermore, most methods we compare to, including [18,21,24,37], do not incorporate a bias term either. Nonetheless, there are clearly learning problems where the incorporation of the bias term could be beneficial.

We used our own implementation of Pegasos, as well as stochastic DCA, and both were instrumented to periodically output the weight vector $\mathbf{w}$ or, in the kernel case, the vector of coefficients α. The source code for SVM-Perf, LASVM and SVM-Light were downloaded from their respective authors' web pages, and were similarly modified. These modifications allowed us to generate traces of each algorithm's progress over time, which were then used to generate all plots and tables. Whenever a runtime is reported, the time spent inside the instrumentation code, as well as the time spent loading the data file, is *not* included. All implementations are in C/C++, and all experiments were performed on a single core of a load-free machine with an Intel Core i7 920 CPU and 12G of RAM.

7.1 Linear kernel

Our first set of experiments, which evaluate the performance of Pegasos in constructing linear SVMs, were performed on three large datasets with very different feature counts and sparsity, which were kindly provided by Thorsten Joachims. The astro-ph dataset classifies abstracts of papers from the physics ArXiv according to whether they belong in the astro-physics section; CCAT is a classification task taken from the Reuters RCV1 collection; and cov1 is class 1 of the covertype dataset of Blackard, Jock & Dean. The following table provides details of the dataset characteristics, as well as the value of the regularization parameter λ used in the experiments (all of which are taken from [21]):

Dataset	Training size	Testing size	Features	Sparsity (%)	λ
astro-ph	29,882	32,487	99,757	0.08	5×10^{-5}
CCAT	781,265	23,149	47,236	0.16	10^{-4}
cov1	522,911	58,101	54	22.22	10^{-6}

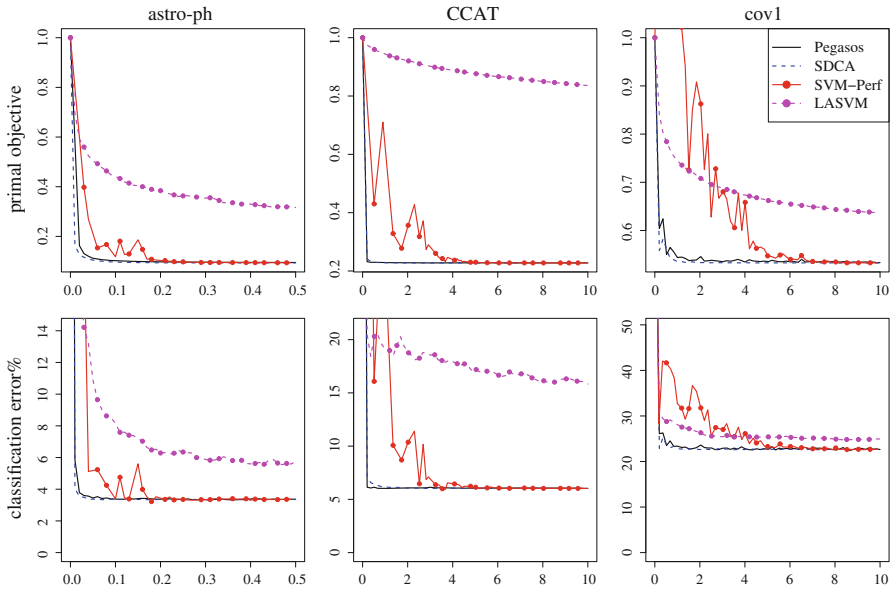

Fig. 4 Comparison of linear SVM optimizers. Primal suboptimality (*top row*) and testing classification error (*bottom row*), for one run each of Pegasos, stochastic DCA, SVM-Perf, and LASVM, on the astro-ph (*left*), CCAT (*center*) and cov1 (*right*) datasets. In all plots the horizontal axis measures runtime in seconds

Figure 4 contains traces of the primal suboptimality, and testing classification error, achieved by Pegasos, stochastic DCA, SVM-Perf, and LASVM. The latter of these is not an algorithm specialized for linear SVMs, and therefore should not be expected to perform as well as the others. Neither Pegasos nor stochastic DCA have a natural stopping criterion. Hence, in order to uniformly summarize the performance of the various algorithms, we found the first time at which the primal suboptimality was less than some predetermined termination threshold ϵ. We chose this threshold for each dataset such that a primal suboptimality less than ϵ guarantees a classification error on test data which is at most 1.1 times the test data classification error at the optimum. (For instance, if full optimization of SVM yields a test classification error of 1%, then we chose ϵ such that a ϵ-accurate optimization would guarantee test classification error of at most 1.1%.) The time taken to satisfy the termination criterion, on each dataset, for each algorithm, along with classification errors on test data achieved at termination, are reported in Table 1.

Based both on the plots of Fig. 4, and on Table 1, we can see that, SVM-Perf is a very fast method on it own. Indeed, SVM-Perf was shown in [21] to achieve a speedup over SVM-Light of several orders of magnitude on most datasets. Nonetheless, Pegasos and stochastic DCA achieve a significant improvement in run-time over SVM-Perf. It is interesting to note that the performance of Pegasos does not depend on the number of examples but rather on the value of λ. Indeed, the runtime of Pegasos for the Covertype dataset is longer than its runtime for CCAT, although the latter dataset is larger. This issue is explored further in Sect. 7.3 given in the sequel.

Table 1 Training runtime and test error achieved (in parentheses) using various optimization methods on linear SVM problems

Dataset	Pegasos	SDCA	SVM-Perf	LASVM
astro-ph	0.04*s* (3.56%)	0.03*s* (3.49%)	0.1*s* (3.39%)	54*s* (3.65%)
CCAT	0.16*s* (6.16%)	0.36*s* (6.57%)	3.6*s* (5.93%)	> 18000*s*
cov1	0.32*s* (23.2%)	0.20*s* (22.9%)	4.2*s* (23.9%)	210*s* (23.8%)

The suboptimality thresholds used for termination are $\epsilon = 0.0275, 0.00589$ and 0.0449 on the astro-ph, CCAT and cov1 datasets (respectively). The testing classification errors at the optima of the SVM objectives are 3.36, 6.03 and 22.6%

7.2 Experiments with Gaussian kernels

Pegasos is particularly well suited for optimization of linear SVMs, in which case the runtime does not depend on the data set size. However, as we show in the sequel, the kernelized Pegasos variant described in Sect. 4 gives good performance on a range of kernel SVM problems, provided that these problems have sufficient regularization. Although Pegasos does not outperform state-of-the-art methods in our experiments, it should be noted that Pegasos is a very simple method to implement, requiring only a few lines of code.

The experiments in this section were performed on four datasets downloaded from Léon Bottou's LASVM web page.[2] The USPS and MNIST datasets were used for the task of classifying the digit 8 versus the rest of the classes. In the following table, γ is the parameter controlling the width of the Gaussian kernel $K(x, y) = e^{-\gamma \|x-y\|_2^2}$, and λ is the Pegasos regularization parameter.

Dataset	Training size	Testing size	γ	λ
Reuters	7,770	3,299	1	1.29×10^{-4}
Adult	32,562	16,282	0.05	3.07×10^{-5}
USPS	7,329	1,969	2	1.36×10^{-4}
MNIST	60,000	10,000	0.02	1.67×10^{-5}

The parameters for the Reuters dataset are taken from [2], while those for the Adult dataset are from [29]. The parameters for the USPS and MNIST datasets are based on those in [2], but we increased the regularization parameters by a factor of 1,000. This change resulted in no difference in the testing set classification error at the optimum on the USPS dataset, and increased it from 0.46 to 0.57% on MNIST. We discuss the performance of Pegasos with smaller values of the regularization parameter λ in the next section.

Figure 5 contains traces of the primal suboptimalities in both linear and log scales, and the testing classification error, achieved by Pegasos, stochastic DCA, SVM-Light, and LASVM. As in the linear experiments, we chose a primal suboptimality threshold for each dataset which guarantees a testing classification error within 10% of that at the optimum. The runtime required to achieve these targets, along with the test classification errors, are reported in Table 2.

[2] http://leon.bottou.org/projects/lasvm.

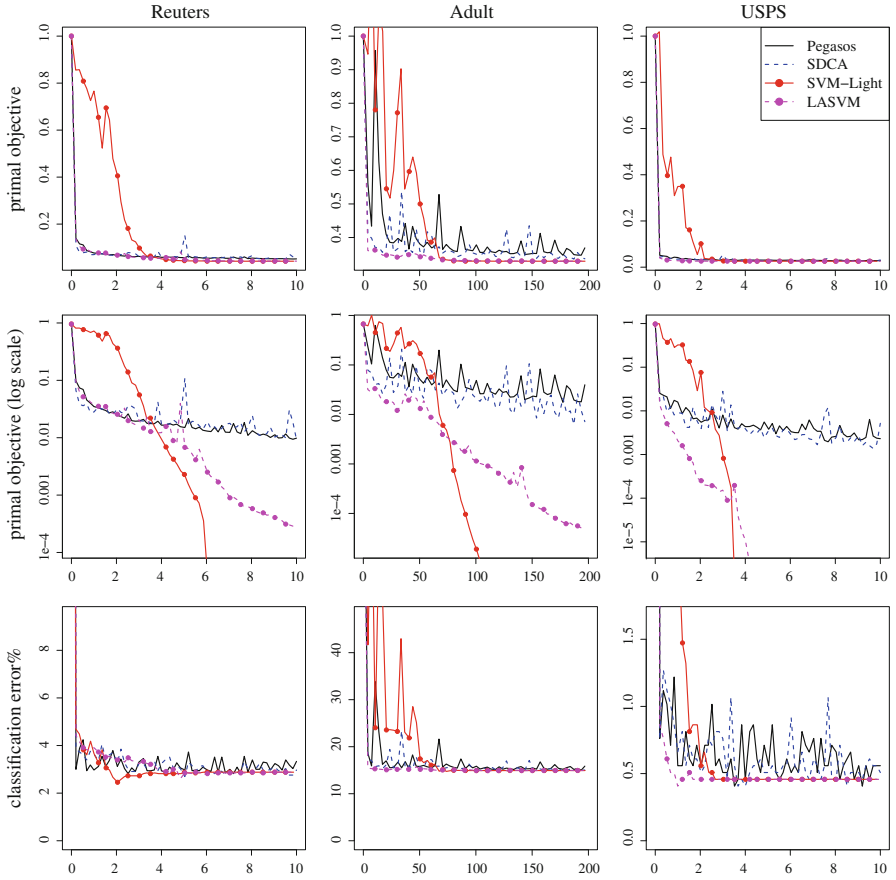

Fig. 5 Comparison of kernel SVM optimizers. Primal suboptimality (*top row*), primal suboptimality in log scale (*middle row*) and testing classification error (*bottom row*), for one run each of Pegasos, stochastic DCA, SVM-Light, and LASVM, on the Reuters (*left column*), Adult (*center column*) and USPS (*right column*) datasets. Plots of traces generated on the MNIST dataset (not shown) appear broadly similar to those for the USPS dataset. The horizontal axis is runtime in seconds

Table 2 Training runtime and test error achieved (in parentheses) using various optimization methods on linear SVM problems

Dataset	Pegasos	SDCA	SVM-Light	LASVM
Reuters	15*s* (2.91%)	13*s* (3.15%)	4.1*s* (2.82%)	4.7*s* (3.03%)
Adult	30*s* (15.5%)	4.8*s* (15.5%)	59*s* (15.1%)	1.5*s* (15.6%)
USPS	120*s* (0.457%)	21*s* (0.508%)	3.3*s* (0.457%)	1.8*s* (0.457%)
MNIST	4200*s* (0.6%)	1800*s* (0.56%)	290*s* (0.58%)	280*s* (0.56%)

$\epsilon = 0.00719, 0.0445, 0.000381$ and 0.00144 on the Reuters, Adult, USPS and MNIST datasets (respectively). The testing classification errors at the optima of the SVM objectives are 2.88, 14.9, 0.457 and 0.57%

As in the linear case, Pegasos (and stochastic DCA) achieve a reasonably low value of the primal objective very quickly, much faster than SVM-Light. However, on the USPS and MNIST datasets, very high optimization accuracy is required in order to

Table 3 Relative kernel evaluation throughputs: the number of kernel evaluations per second of runtime divided by Pegasos's number of kernel evaluations per second of runtime on the same dataset

Dataset	Pegasos	SDCA	SVM-Light	LASVM
Reuters	1	1.03	1.14	0.88
Adult	1	0.90	0.94	0.60
USPS	1	0.97	0.69	0.81
MNIST	1	0.94	0.99	0.61

achieve near-optimal predictive performance, and such accuracy is much harder to achieve using the stochastic methods. Note that the test error on these data sets is very small (roughly 0.5%).

Furthermore, when using kernels, LASVM essentially dominates Pegasos and stochastic DCA, even when relatively low accuracy is required. On all four datasets, LASVM appears to enjoy the best properties of the other algorithms: it both makes significant progress during early iterations, and converges rapidly in later iterations. Nevertheless, the very simple method Pegasos still often yields very good predictive performance, with a competitive runtime.

It can also be interesting to compare the different methods also in terms of the number of kernel evaluations performed. All of the implementations use the same sparse representation for vectors, so the amount of time which it takes to perform a single kernel evaluation should, for each dataset, be roughly the same across all four algorithms. However, various other factors, such overhead resulting from the complexity of the implementation, or caching of portions of the kernel matrix, do affect the number of kernel evaluations which are performed in a given unit of time. Nevertheless, the discrepancy between the relative performance in terms of runtime and the relative performance in terms of number of kernel evaluations is fairly minor. To summarize this discrepancy, we calculated for each method and each data set the kernel evaluation throughput: the number of kernel evaluations performed per second of execution in the above runs. For each data set, we then normalized these throughputs by dividing each method's throughput by the Pegasos throughput, thus obtaining a relative measure indicating whether some methods are using much more, or much fewer, kernel evaluations, relative to their runtime. The resulting relative kernel evaluation throughputs are summarized in Table 3. It is unsurprising that Pegasos and stochastic DCA, as the simplest algorithms, tend to have performance most dominated by kernel evaluations. If we were to compare the algorithms in terms of the number of kernel evaluations, rather than elapsed time, then LASVMs performance would generally improve slightly relative to the others. But in any case, the change to the relative performance would not be dramatic.

7.3 Effect of the regularization parameter λ

We return now to the influence of the values of the regularization parameter λ on the runtime of Pegasos and stochastic DCA. Recall that in the previous section, we choose to use a much larger value of λ in our experiments with the MNIST and USPS datasets.

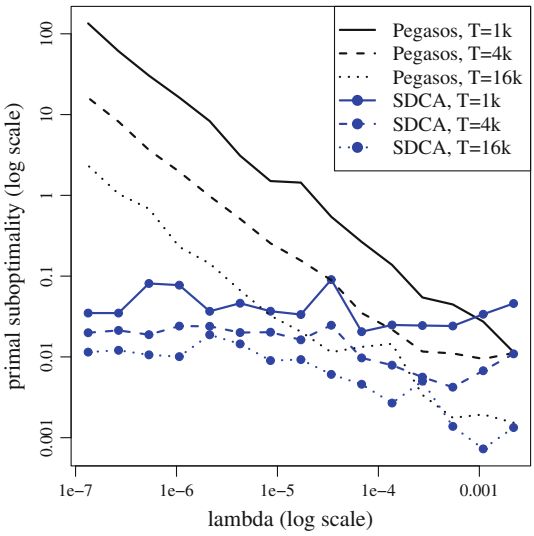

Fig. 6 Demonstration of dependence of Pegasos' performance on regularization, on the USPS dataset. This plot shows (on a log-log scale) the primal suboptimalities of Pegasos and stochastic DCA after certain fixed numbers of iterations, for various values of λ

Figure 6 shows the suboptimalities of the primal objective achieved by Pegasos and stochastic DCA after certain fixed numbers of iterations, on the USPS dataset, as a function of the regularization parameter λ. As predicted by the formal analysis, the primal suboptimality after a fixed number of Pegasos iterations is inversely proportional to λ. Hence, the runtime to achieve a predetermined suboptimality threshold would increase in proportion to λ. Very small values of λ (small amounts of regularization) result in rather long runtimes. This phenomenon has been observed before and there have been rather successful attempts to improve Pegasos when λ is small (see for example [13]).

It is interesting to note that stochastic DCA does not seem to suffer from this problem. Although Pegasos and stochastic DCA have comparable runtimes for moderate values of λ, stochastic DCA is much better behaved when λ is very small (i.e. when the problem is barely infused with regularization).

7.4 Experiments with the mini-batch variant

In this section, we explore the influence of the mini-batch size, k, OF the mini-batch variant of Fig. 2. Increasing k does not reduce our theoretical guarantees on the number of iterations T that are required to attain a primal suboptimality goal. Since the runtime of each iteration scales linearly with k, the convergence analysis suggests that increasing k would only cause a linear increase in the overall runtime kT required to achieve a predetermined accuracy goal. We show that in practice, for moderate sizes of k, a roughly linear (in k) improvement in the number of required iterations T can be achieved, leaving the overall runtime kT almost fixed. For a serial implementation of Pegasos, this result would be uninteresting. However, using large samples for computing the subgradients can be useful in a parallel implementation, where the $O(k)$

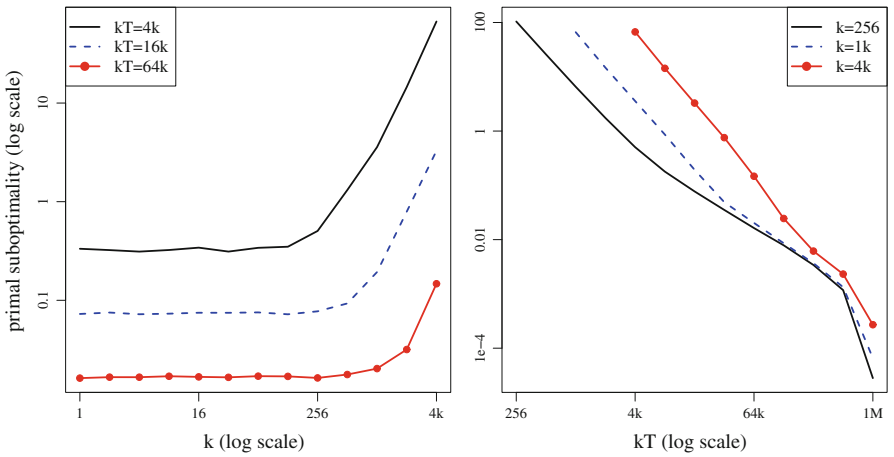

Fig. 7 The effect of the mini-batch size on the runtime of Pegasos for the astro-ph dataset. The first plot shows the primal suboptimality achieved for certain fixed values of overall runtime kT, for various values of the mini-batch size k. The second plot shows the primal suboptimality achieved for certain fixed values of k, for various values of kT. Very similar results were achieved for the CCAT dataset

work of each iteration could be done in parallel, thus reducing the overall required elapsed time.

Figure 7 includes two plots which illustrate the impact of k on the performance of Pegasos. The first plot shows that, on the astro-ph dataset, for sufficiently small values of k, the primal suboptimality achieved after T iterations is roughly proportional to the product kT. This property holds in the region for which the curves are roughly horizontal, which in this experiment, corresponds to mini-batch sizes of up to a few hundred training points.

Note also that the three curves on the left hand side plot of Fig. 7 start increasing at different values of k. It appears that, when k is overly large, there is initially indeed a loss of performance. However, as the number of iterations increases, the slower behavior due to the mini-batch size is alleviated.

The second plot further underscores this phenomenon. We can see that, for three values of k, all significantly greater than 100, the experiments with the largest mini-batch size made the least progress while performing the same amount of computation. However, as the number of iterations grows, the suboptimalities become similar. The end result is that the overall runtime does not seem to be strongly dependent on the mini-batch size. We do not yet have a good quantitative theoretical understanding of the mini-batch results observed here.

7.5 Comparison of sampling procedures

The analysis of Pegasos requires sampling with replacement at each iteration. Based on private communication with Léon Bottou we experimented with sampling *without* replacement. Specifically, we chose a random permutation over the training examples and performed updates in accordance to the selected order. Once we traversed all the

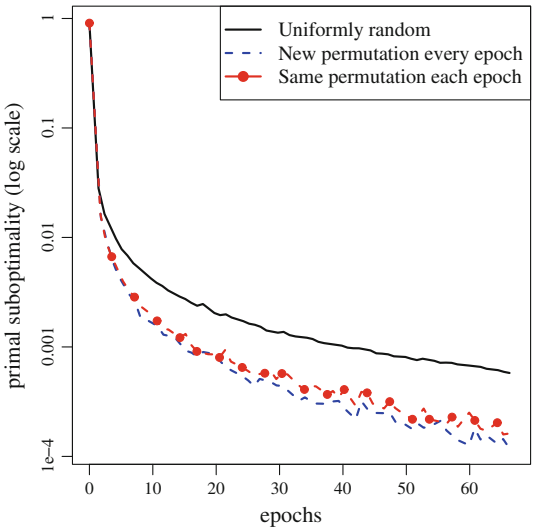

Fig. 8 The effect of different sampling methods on the performance of Pegasos for the astro-ph dataset. The curves show the primal suboptimality achieved by uniform i.i.d. sampling, sampling from a fixed permutation, and sampling from a different permutation for every epoch

permuted examples, we chose a new permutation and iteratively repeated the process. We also experimented with a further simplified approach in which a single random permutation is drawn and then used repeatedly.

Figure 8 indicates that, on the astro-ph dataset, the sampling without replacement procedures outperform significantly the uniform i.i.d. sampling procedure. Further, it seems that choosing a new permutation every epoch, rather than keeping the permutation intact, provides some slight further improvement. We would like to note though that while the last experiment underscores the potential for additional improvement in the convergence rate, the rest of the experiments reported in the paper were conducted in accordance with the formal analysis using uniform sampling with replacements.

7.6 Comparison to other stochastic gradient methods

In our last set of experiments, we compared Pegasos to Norma [24] and to a variant of stochastic gradient descent due to Zhang [37]. Both methods share similarities with Pegasos when $k = 1$, and differ in their schemes for setting the learning rate η_t. Theorem 4 from [24], suggests to set $\eta_t = p/(\lambda\sqrt{t})$, where $p \in (0, 1)$. Based on the bound given in the theorem, the optimal choice of p is $0.5(2 + 0.5T^{-1/2})^{1/2}$, which for $t \geq 100$ is in the range $[0.7, 0.716]$. Plugging the optimal value of p into Theorem 4 in [24] yields the bound $O(1/(\lambda\sqrt{T}))$. We therefore conjectured that Pegasos would converge significantly faster than Norma. On the left hand side of Fig. 9 we compare Pegasos (with the optional projection step) to Norma on the Astro-Physics dataset. We divided the dataset to a training set with 29,882 examples and a test set with 32,487 examples. We report the final objective value and the average hinge-loss on the test set. As in [21], we set $\lambda = 2 \times 10^{-4}$. It is clear from the figure that Pegasos outperforms Norma. Moreover, Norma fails to converge even after 10^6 iterations. The

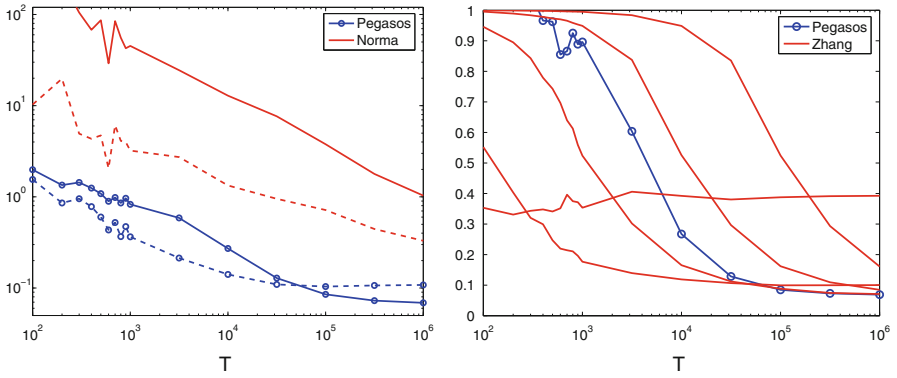

Fig. 9 Comparisons of Pegasos to Norma (*left*), and Pegasos to stochastic gradient descent with a fixed learning rate (*right*) on the Astro-Physics datset. In the left hand side plot, the *solid curves* designate the objective value while the *dashed curves* show the test loss

poor performance of Norma can be attributed to the fact that the value of λ here is rather small.

We now turn to comparing Pegasos to the algorithm of Zhang [37] which simply sets $\eta_t = \eta$, where η is a (fixed) small number. A major disadvantage of this approach is that finding an adequate value for η is a difficult task on its own. Based on the analysis given in [37] we started by setting η to be 10^{-5}. Surprisingly, this value turned out to be a poor choice and the optimal choice of η was substantially larger. On the right hand side of Fig. 9 we illustrate the convergence of stochastic gradient descent with η_t set to be a fixed value from the set $\{0.001, 0.01, 0.1, 1, 10\}$. It is apparent that for some choices of η the method converges at about the same rate of Pegasos while for other choices of η the method fails to converge. For large datasets, the time required for evaluating the objective is often much longer than the time required for training a model. Therefore, searching for η is significantly more expensive than running the algorithm a single time. The apparent advantage of Pegasos is due to the fact that we do not need to search for a good value for η but rather have a predefined schedule for η_t.

8 Conclusions

We described and analyzed a simple and effective algorithm for approximately minimizing the objective function of SVM. We derived fast rate of convergence results and experimented with the algorithm. Our empirical results indicate that for linear kernels, Pegasos achieves state-of-the-art results, despite of, or possibly due to, its simplicity. When used with more complex kernels, Pegasos may still be a simple competitive alternative to more complicated methods, especially when fairly lax optimization can be tolerated.

Recently, Bottou and Bousquet [4] proposed to analyse optimization algorithms from the perspective of the underlying machine learning task. In a subsequent paper [32], we analyze Pegasos and other SVM training methods from a machine

learning perspective, and showed that Pegasos is more efficient than other methods when measuring the runtime required to guarantee good predictive performance (test error).

Acknowledgments We would like to thank Léon Bottou for useful discussions and suggestions, and Thorsten Joachims and Léon Bottou for help with the experiments. We would also like to thank the anonymous reviewers for their helpful comments. Part of this work was done while SS and NS were visiting IBM Research Labs, Haifa, Israel. This work was partially supported by grant I-773-8.6/2003 from the German Israeli Foundation (GIF) and by the Israel Science Foundation under grant number 522/04.

References

1. Amari, S.: Natural gradient works efficiently in learning. Neural Comput. **10**, 251–276 (1998)
2. Bordes, A., Ertekin, S., Weston, J., Bottou, L.: Fast kernel classifiers with online and active learning. J. Mach. Learn. Res. **6**, 1579–1619 (2005)
3. Bottou, L.: Online Algorithms and Stochastic Approximations. In: Saad, D. (ed.) Online learning and neural networks, Cambridge University Press, Cambridge (1998)
4. Bottou, L., Bousquet, O.: The tradeoffs of large scale learning. In: Advances in Neural Information Processing Systems 20, pp. 161–168 (2008)
5. Bottou, L., LeCun, Y.: Large scale online learning. In: Thrun, S., Saul, L., Schölkopf, B. (eds.) Advances in Neural Information Processing Systems 16, MIT Press, Cambridge (2004)
6. Bottou, L., Murata, N. : Stochastic approximations and efficient learning. In: Arbib, M.A. (ed.) The Handbook of Brain Theory and Neural Networks, The MIT Press, Cambridge (2002)
7. Boyd, S., Vandenberghe, L.: Convex Optimization. 2nd edn. Cambridge University Press, Cambridge (2004)
8. Censor, Y., Zenios, S.: Parallel Optimization: Theory, Algorithms, and Applications. Oxford University Press, New York (1997)
9. Cesa-Bianchi, N., Conconi, A., Gentile, C.: On the generalization ability of on-line learning algorithms. IEEE Trans. Inf, Theory **50**(9), 2050–2057 (2004)
10. Chapelle, O.: Training a support vector machine in the primal. Neural Comput. **19**(5), 1155–1178 (2007). doi:10.1162/neco.2007.19.5.1155. http://www.mitpressjournals.org/doi/abs/10.1162/neco.2007.19.5.1155
11. Crammer, K., Dekel, O., Keshet, J., Shalev-Shwartz, S., Singer, Y.: Online passive aggressive algorithms. J. Mach. Learn. Res. **7**, 551–585 (2006)
12. Cristianini, N., Shawe-Taylor, J.: An Introduction to Support Vector Machines. Cambridge University Press, Cambridge (2000)
13. Do, C., Le, Q., Foo, C.: Proximal regularization for online and batch learning. In: Proceedings of the 26th International Conference on Machine Learning (2009)
14. Duda, R.O., Hart, P.E.: Pattern Classification and Scene Analysis. Wiley, New York (1973)
15. Fine, S., Scheinberg, K.: Efficient SVM training using low-rank kernel representations. J. Mach. Lear. Res. **2**, 242–264 (2001)
16. Freund, Y., Schapire, R.E.: Large margin classification using the perceptron algorithm. Mach. Learn. **37**(3), 277–296 (1999)
17. Hazan, E., Kalai, A., Kale, S., Agarwal, A.: Logarithmic regret algorithms for online convex optimization. In: Proceedings of the Nineteenth Annual Conference on Computational Learning Theory (2006)
18. Hsieh, C., Chang, K., Lin, C., Keerthi, S., Sundararajan, S.: A dual coordinate descent method for large-scale linear SVM. In: ICML, pp. 408–415 (2008)
19. Hush, D., Kelly, P., Scovel, C., Steinwart, I.: Qp algorithms with guaranteed accuracy and run time for support vector machines. J. Mach. Learn. Res. (2006)
20. Joachims, T. : Making large-scale support vector machine learning practical. In: Schölkopf, B., Burges, C., Smola, A. (eds.) Advances in Kernel Methods—Support Vector Learning., MIT Press, Cambridge (1998)
21. Joachims, T.: Training linear SVMs in linear time. In: Proceedings of the ACM Conference on Knowledge Discovery and Data Mining (KDD), pp. 216–226 (2006)

22. Kakade, S., Tewari, A.: On the generalization ability of online strongly convex programming algorithms. In: Advances in Neural Information Processing Systems 22 (2009)
23. Kimeldorf, G., Wahba, G.: Some results on tchebycheffian spline functions. J. Math. Anal. Appl. **33**, 82–95 (1971)
24. Kivinen, J., Smola, A.J., Williamson, R.C.: Online learning with kernels. IEEE Trans. Signal Process. **52**(8), 2165–2176 (2002)
25. Kushner, H., Yin, G.: Stochastic Approximation Algorithms and Applications. Springer, New York (1997)
26. Murata, N.: A statistical study of on-line learning. In: Saad, D. (ed.) Online Learning and Neural Networks, Cambridge University Press, Cambridge (1998)
27. Murata, N., Amari, S.: Statistical analysis of learning dynamics. Signal Process. **74**(1), 3–28 (1999)
28. Nesterov, Y.: Primal-dual subgradient methods for convex problems. Tech. rep., Center for Operations Research and Econometrics (CORE), Catholic University of Louvain (UCL) (2005)
29. Platt, J.C.: Fast training of Support Vector Machines using sequential minimal optimization. In: Schölkopf, B., Burges, C., Smola, A. (eds.) Advances in Kernel Methods—Support Vector Learning, MIT Press, Cambridge (1998)
30. Rockafellar, R.: Convex Analysis. Princeton University Press, Princeton (1970)
31. Shalev-Shwartz, S., Singer, Y., Srebro, N.: Pegasos: Primal Estimated sub-GrAdient SOlver for SVM. In: Proceedings of the 24th International Conference on Machine Learning, pp. 807–814 (2007)
32. Shalev-Shwartz, S., Srebro, N.: SVM optimization: inverse dependence on training set size. In: Proceedings of the 25th International Conference on Machine Learning, pp. 928–935 (2008)
33. Smola, A., Vishwanathan, S., Le, Q.: Bundle methods for machine learning. In: Advances in Neural Information Processing Systems 21 (2007)
34. Spall, J.C.: Introduction to Stochastic Search and Optimization. Wiley, New York (2003)
35. Sridharan, K., Srebro, N., Shalev-Shwartz, S.: Fast rates for regularized objectives. In: Advances in Neural Information Processing Systems 22 (2009)
36. Vapnik, V.N.: Statistical Learning Theory. Wiley, New York (1998)
37. Zhang, T.: Solving large scale linear prediction problems using stochastic gradient descent algorithms. In: Proceedings of the Twenty-First International Conference on Machine Learning (2004)