Pegasos: Primal Estimated sub-GrAdient SOlver for SVM

Shai Shalev-Shwartz SHAIS@CS.HUJI.AC.IL
School of Computer Science and Engineering, The Hebrew University, Jerusalem, Israel

Yoram Singer SINGER@GOOGLE.COM
Google inc., Mountain View, USA and The Hebrew University, Jerusalem, Israel

Nathan Srebro NATI@UCHICAGO.EDU
Toyota Technological Institute, Chicago, USA

Abstract

We describe and analyze a simple and effective iterative algorithm for solving the optimization problem cast by Support Vector Machines (SVM). Our method alternates between stochastic gradient descent steps and projection steps. We prove that the number of iterations required to obtain a solution of accuracy ϵ is $\tilde{O}(1/\epsilon)$. In contrast, previous analyses of stochastic gradient descent methods 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 can seamlessly be adapted to employ non-linear kernels while working solely on the primal objective function. We demonstrate the efficiency and applicability of our approach by conducting experiments on large text classification problems, comparing our solver to existing state-of-the-art SVM solvers. For example, it takes less than 5 seconds for our solver to converge when solving a text classification problem from Reuters Corpus Volume 1 (RCV1) with $800,000$ training examples.

Appearing in *Proceedings of the 24^{th} International Conference on Machine Learning*, Corvallis, OR, 2007.

1. Introduction

Support Vector Machines (SVMs) are effective and popular classification learning tool (Vapnik, 1998; Cristianini & Shawe-Taylor, 2000). The task of learning a support vector machine is 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

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

where

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

We denote the objective function of Eq. (1) by $f(\mathbf{w})$. 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 original SVM problem also includes a bias term, b. We omit the bias throughout the first sections and defer the description of an extension which employs a bias term to Sec. 4.

We describe and analyze in this paper a simple iterative algorithm, called Pegasos, for solving Eq. (1). The algorithm performs T iterations and also requires an additional parameter k, whose role is explained in the sequel. Pegasos alternates between stochastic subgradient descent steps and projection steps. The parameter k determines the number of examples from S the algorithm uses on each iteration for estimating the subgradient. When $k = m$, Pegasos reduces to a variant of the subgradient projection method. We show that in this case the number of iterations that is required in order to achieve an ϵ- accurate solution is $\tilde{O}(1/(\lambda\epsilon))$. At the other extreme, when $k = 1$, we recover a variant of the stochastic (sub)gradient method. In the stochastic case,

we analyze the probability of obtaining a good approximate solution. Specifically, we show that with probability of at least $1-\delta$ our algorithm finds an ϵ-accurate solution using only $\tilde{O}(1/(\delta\lambda\epsilon))$ iterations, while each iteration involves a single inner product between $\mathbf{w}$ and $\mathbf{x}$. This rate of convergence does *not* depend on the size of the training set and thus our algorithm is especially suited for large datasets.

Before indulging in the 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 (Boyd & Vandenberghe, 2004) and the references therein) cast SVM learning 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 are $O(m^2)$ which renders a direct use of IP methods very difficult when the training set has many examples. It should be noted that there have been several attempts to reduce the complexity based on additional assumptions (see e.g. (Fine & Scheinberg, 2001)). 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 bare in mind that the optimization problem is a proxy method for obtaining good classification error on unseen examples. Achieving very high accuracy in the optimization process is usually unnecessary and does not translate to a significant increase in generalization accuracy. The time spent by IP methods for finding a single accurate solution may, for instance, be better utilized for finding numerous approximate solutions for multiple choices of λ.

Decomposition methods: To overcome the quadratic memory requirement of IP methods, decomposition methods such as SMO (Platt, 1998) and SVM-Light (Joachims, 1998) switch to 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 (Censor & Zenios, 1997), the active set consists of a single constraint. While algorithms in this family are fairly simple to implement and entertain general asymptotic convergence properties (Censor & Zenios, 1997), 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 (Hush et al., 2006)).

Some of the decomposition methods do yield though a regret bound in the online learning setting. For instance, the Passive Aggressive (Crammer et al., 2006) applies the objective function of SVM to each example. Online learning algorithms were also suggested as fast alternatives to SVM (see (Freund & Schapire, 1999)). Such algorithms can be used to obtain a predictor with low generalization error using an online-to-batch conversion scheme (Cesa-Bianchi et al., 2004). However, the conversion schemes do not necessarily yield 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 but is guaranteed to converge to the actual SVM solution.

Gradient based methods: Unconstrained gradient methods were very common in optimization until the emergence of the ultra-fast IP methods. While gradient based methods are known to exhibit slow convergence rates, the computational demands imposed by large scale classification and regression problems of high dimension feature space, revived the theoretical and applied interest in gradient methods. The Pegasos algorithm is an improved stochastic sub-gradient method. Two concrete algorithms that are closely related to the Pegasos algorithm that are based on gradient methods are the NORMA algorithm (Kivinen et al., 2002) and a stochastic gradient algorithm by Zhang (2004). The Pegasos algorithm uses a sub-sample of k training examples to compute an approximate sub-gradient. When $k=1$ the Pegasos algorithm becomes very similar to the aforementioned methods of Kivienen et. al and Zhang with a few notable and crucial differences. First, after each gradient-based update, Pegasos employs a projection step of $\mathbf{w}$ onto the L_2 ball of radius $1/\sqrt{\lambda}$. This modification enables the usage of a very aggressive decrease in the learning rate and yields an improved $\tilde{O}(1/\epsilon)$ rate of convergence rather than $O(1/\epsilon^2)$ rate. This theoretical improvement is also reflected in our experiments. When $k=m$ Pegasos results in a modified gradient-descent algorithm with an improved convergence rate. In addition to the superior rate of convergence, Pegasos can incorporate a bias term (discussed in Sec. 4) and can utilize parallel computation power for appropriate choices of k.

Last, we would like to point to the SVM-Perf algorithm recently proposed by Joachims (2006) for linear SVMs.

SVM-Perf uses cutting planes to find a solution with accuracy ϵ in time $O(md/(\lambda\epsilon^2))$. The complexity guarantee for Pegasos avoids the dependence on the data set size m and reduces the dependence on the accuracy to only $\tilde{O}(1/\epsilon)$. In practice, while SVM-Perf yields very significant improvements over decomposition methods for large data sets, our experiments (see Sec. 5) demonstrate that Pegasos is substantially faster than SVM-Perf.

2. The Pegasos Algorithm

In this section we describe the Pegasos algorithm for solving the optimization problem given in Eq. (1). The algorithm receives as input two parameters: T - the number of iterations to perform; k - the number of examples to use for calculating sub-gradients. Initially, we set $\mathbf{w}_1$ to any vector whose norm is at most $1/\sqrt{\lambda}$. On iteration t of the algorithm, we first choose a set $A_t \subseteq S$ of size k. Then, we replace the objective in Eq. (1) with an approximate objective function,

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

Note that we overloaded our original definition of f as the original objective can be denoted either as $f(\mathbf{w})$ or as $f(\mathbf{w}; S)$. We interchangeably use both notations depending on the context . Next, we set the learning rate $\eta_t = 1/(\lambda t)$ and define A_t^+ to be the set of examples for which $\mathbf{w}$ suffers a non-zero loss. We now perform a two-step update as follows. We scale $\mathbf{w}_t$ by $(1 - \eta_t\,\lambda)$ and for all examples $(\mathbf{x}, y) \in A_t^+$ we add to $\mathbf{w}$ the vector $\frac{y\eta_t}{k}\,\mathbf{x}$. We denote the resulting vector by $\mathbf{w}_{t+\frac{1}{2}}$. This step can be also written as $\mathbf{w}_{t+\frac{1}{2}} = \mathbf{w}_t - \eta_t\nabla_t$, where

$$\nabla_t = \lambda\,\mathbf{w}_t - \tfrac{1}{|A_t|}\textstyle\sum_{(\mathbf{x},y)\in A_t^+} y\,\mathbf{x}\;. \qquad (3)$$

The definition of the hinge-loss implies that ∇_t is a sub-gradient of $f(\mathbf{w}; A_t)$ at $\mathbf{w}_t$. Last, we set $\mathbf{w}_{t+1}$ to be the projection of $\mathbf{w}_{t+\frac{1}{2}}$ onto the set

$$B \;=\; \{\mathbf{w} : \|\mathbf{w}\| \le 1/\sqrt{\lambda}\}\;. \qquad (4)$$

That is, $\mathbf{w}_{t+1}$ is obtained by scaling $\mathbf{w}_{t+\frac{1}{2}}$ by $\min\left\{1, 1/(\sqrt{\lambda}\|\mathbf{w}_{t+\frac{1}{2}}\|)\right\}$. As we show in our analysis below, the optimal solution of SVM is in the set B. Informally speaking, we can always project back onto the set B as we only get closer to the optimum. The output of Pegasos is the last vector $\mathbf{w}_{T+1}$. The pseudo-code of Pegasos is given in Fig. 1.

Note that if we choose $A_t = S$ on each round t then we obtain the subgradient projection method. On the other extreme, if we choose A_t to contain a single randomly selected example, then we recover a variant of the stochastic gradient method. In general, we allow A_t to be a set of k examples sampled i.i.d. from S.

INPUT: S, λ, T, k
INITIALIZE: Choose $\mathbf{w}_1$ s.t. $\|\mathbf{w}_1\| \le 1/\sqrt{\lambda}$
FOR $t = 1, 2, \ldots, T$
  Choose $A_t \subseteq S$, where $|A_t| = k$
  Set $A_t^+ = \{(\mathbf{x}, y) \in A_t : y\,\langle \mathbf{w}_t, \mathbf{x}\rangle < 1\}$
  Set $\eta_t = \frac{1}{\lambda t}$
  Set $\mathbf{w}_{t+\frac{1}{2}} = (1 - \eta_t\,\lambda)\mathbf{w}_t + \frac{\eta_t}{k}\sum_{(\mathbf{x},y)\in A_t^+} y\,\mathbf{x}$
  Set $\mathbf{w}_{t+1} = \min\left\{1, \frac{1/\sqrt{\lambda}}{\|\mathbf{w}_{t+\frac{1}{2}}\|}\right\}\mathbf{w}_{t+\frac{1}{2}}$
OUTPUT: $\mathbf{w}_{T+1}$

Figure 1. The Pegasos Algorithm.

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 triplet $(\mathbf{v}, a, \nu)$ where $\mathbf{v}$ is a dense vector and a, ν are scalars. The vector $\mathbf{w}$ is defined through the triplet as follows: $\mathbf{w} = a\,\mathbf{v}$ and ν stores the squared norm of $\mathbf{w}$, $\nu = \|\mathbf{w}\|^2$. Using this representation, it is easily verified that the total number of operations required for performing one iteration of Pegasos with $k = 1$ is $O(d)$, where d is the number of non-zero elements in $\mathbf{x}$.

3. Analysis

In this section we analyze the convergence properties of Pegasos. Throughout this section we denote

$$\mathbf{w}^\star = \operatorname*{argmin}_{\mathbf{w}} f(\mathbf{w}) \;\;. \qquad (5)$$

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 generalizes a result from (Hazan et al., 2006). The lemma relies on the notion of strongly convex functions. A detailed proof and further explanations can be found in (Shalev-Shwartz & Singer, 2007).

Lemma 1. *Let $f_1, \ldots, f_T$ be a sequence of λ-strongly convex functions w.r.t. the function $\frac{1}{2}\|\cdot\|^2$. 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 $t \ge 1$, $\mathbf{w}_{t+1} = \Pi_B(\mathbf{w}_t - \eta_t\nabla_t)$, where ∇_t is a subgradient of f_t at $\mathbf{w}_t$ and $\eta_t = 1/(\lambda t)$. Assume that for all t, $\|\nabla_t\| \le G$. Then, for all $\mathbf{u} \in B$ we have*

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

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. (5) and let $c = (\sqrt{\lambda} + R)^2$. Then, for $T \geq 3$,*

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

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. (3) and B is defined in Eq. (4). Thus, for proving 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 also λ-strongly convex (see Lemma 1 in (Shalev-Shwartz & Singer, 2007)). Next, we derive a bound on $\|\nabla_t\|$. Using the facts that $\|\mathbf{w}_t\| \leq 1/\sqrt{\lambda}$ and that $\|\mathbf{x}\| \leq R$ combined with the triangle inequality we obtain $\|\nabla_t\| \leq \sqrt{\lambda} + R$. Finally, we need to prove that $\mathbf{w}^\star \in B$. To do so, we use the fact that there exists a vector $\boldsymbol{\alpha}^\star \in [0,1]^m$ such that

$$\frac{\lambda}{2}\|\mathbf{w}^\star\|^2 + \frac{1}{m}\sum_{(\mathbf{x},y)\in S} \ell(\mathbf{w}^\star; (\mathbf{x}, y)) = -\frac{\lambda}{2}\|\mathbf{w}^\star\|^2 + \frac{\|\boldsymbol{\alpha}^\star\|_1}{m} \;.$$

(The above equality is derived by applying the strong duality theorem to the SVM optimization problem.) Rearranging the above and using the non-negativity of the hinge-loss gives that $\|\mathbf{w}^\star\| \leq 1/\sqrt{\lambda}$. The bound in the theorem follows now using simple algebraic manipulations. □

Note that the convexity of f implies that

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

Using the above inequality and Thm. 1, we immediately obtain the following corollary which gives a convergence analysis for the case $A_t = S$.

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

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

Based on the above corollary, the number of iterations required for achieving a solution of accuracy ϵ is $\tilde{O}(R^2/(\lambda\epsilon))$. Joachims (2006) recently suggested a method, called SVM-Perf, which requires $O(R^2/(\lambda\epsilon^2))$ iterations. The cost of each iteration of SVM-Perf is $O(m\,d)$, where m is the number of examples and d is the effective dimension of the examples. The complexity of a single iteration of Pegasos when $A_t = S$ is also $O(m\,d)$, with smaller constants.[1] We therefore obtain a significant improvement with a simpler algorithm.

When $A_t \neq S$, Corollary 1 no longer holds. In addition, the final hypothesis we use in Fig. 1 is $\mathbf{w}_{T+1}$ rather than the average hypothesis. The next theorem bridges this gap as it implies that the same convergence rate still holds in expectation if we randomly choose a stopping time.

Theorem 2. *Assume that the conditions stated in Thm. 1 hold and for all t, A_t is chosen i.i.d. from S. Let r be an integer picked uniformly at random from $[T]$. Then,*

$$\mathbb{E}_{A_1,\ldots,A_T}\mathbb{E}_r[f(\mathbf{w}_r)] \;\leq\; f(\mathbf{w}^\star) + \frac{c\ln(T)}{\lambda T} \;.$$

Proof. To simplify our notation, denote by A_i^j the sequence of sets $(A_i, \ldots, A_j)$. Taking expectation of the inequality given in Thm. 1 we obtain

$$\begin{aligned} E_{A_1^T}[\frac{1}{T}\sum_{t=1}^{T} f(\mathbf{w}_t; A_t)] \;\leq\; \mathbb{E}_{A_1^T}[\frac{1}{T}\sum_{t=1}^{T} f(\mathbf{w}^\star; A_t)] \\ + \frac{c\ln(T)}{\lambda T} \;. \end{aligned} \tag{6}$$

We now analyze the two expectations given in Eq. (6). Since $\mathbf{w}^\star$ does not depend on the choice of A_1^T, we have,

$$\begin{aligned} \mathbb{E}_{A_1^T}[\frac{1}{T}\sum_{t=1}^{T} f(\mathbf{w}^\star; A_t)] \;&=\; \frac{1}{T}\sum_{t=1}^{T} \mathbb{E}_{A_1^T}[f(\mathbf{w}^\star; A_t)] \\ &=\; \frac{1}{T}\sum_{t=1}^{T} \mathbb{E}_{A_t}[f(\mathbf{w}^\star; A_t)] \\ &=\; f(\mathbf{w}^\star) \;. \end{aligned} \tag{7}$$

Next, we analyze the expectation at the left-hand side of Eq. (6). Note that $\mathbf{w}_t$ only depends on A_1^{t-1}. Thus,

$$\mathbb{E}_{A_1^T}[\frac{1}{T}\sum_{t=1}^{T} f(\mathbf{w}_t; A_t)] \;=\; \frac{1}{T}\sum_{t=1}^{T} \mathbb{E}_{A_1^t}[f(\mathbf{w}_t; A_t)] \;. \tag{8}$$

Recall that the law of total expectation implies that for any two random variables X, Y, $\mathbb{E}_X[f(X)] = \mathbb{E}_Y\mathbb{E}_X[f(X)|Y]$. Setting $X = A_1^t$ and $Y = A_1^{t-1}$ we get that

$$\begin{aligned} \mathbb{E}_{A_1^t}[f(\mathbf{w}_t; A_t)] \;&=\; \mathbb{E}_{A_1^{t-1}}[\mathbb{E}_{A_1^t}[f(\mathbf{w}_t; A_t)|A_1^{t-1}]] \\ &=\; \mathbb{E}_{A_1^{t-1}}[f(\mathbf{w}_t)] \;=\; \mathbb{E}_{A_1^T}[f(\mathbf{w}_t)] \;. \end{aligned}$$

Combining the above with Eq. (8) we obtain

$$\mathbb{E}_{A_1^T}[\frac{1}{T}\sum_{t=1}^{T} f(\mathbf{w}_t; A_t)] \;=\; \mathbb{E}_{A_1^T}[\frac{1}{T}\sum_{t=1}^{T} f(\mathbf{w}_t)] \;.$$

[1] Seemingly, to calculate $\bar{\mathbf{w}}$ we need additional $O(n)$ operations at each iteration, where n is the dimension of $\mathbf{w}$. However, if $n > m\,d$ we can also save $\mathbf{w}_t$ as a linear combination of the m instances in the training set and update only the coefficients. Thus, the cost of each iteration never exceeds $O(m\,d)$.

Combining the above with Eq. (7) and Eq. (6), and noting that $\mathbb{E}_r[f(\mathbf{w}_r)] = \frac{1}{T}\sum_{t=1}^{T} f(\mathbf{w}_t)$ we conclude our proof. □

The above theorem states that, in expectation, the stochastic version of the algorithm will converge as fast as the deterministic version. The next theorem provides a very simple concentration bound.

Theorem 3. *Assume that the conditions stated in Thm. 2 hold. Let $\delta \in (0,1)$. Then, with probability of at least $1-\delta$ over the choices of $(A_1, \ldots, A_T)$ and the index r we have that*

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

Proof. Let Z be the random variable $f(\mathbf{w}_r) - f(\mathbf{w}^\star)$. 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 $\mathbb{P}[Z \geq a] \leq \mathbb{E}[Z]/a$. Setting $\mathbb{E}[Z]/a = \delta$ and using Thm. 2 we obtain that $a = \frac{\mathbb{E}[Z]}{\delta} \leq \frac{c\,\ln(T)}{\delta\,\lambda\,T}$. □

Let us now discuss the implications of Thm. 3. First, by taking $T = \infty$ we immediately obtain from Thm. 3 convergence in the limit. In addition, we can use Thm. 3 for analyzing the convergence of the last weight vector. We do so by viewing T as a random index drawn from $\{1, \ldots, \hat{T}\}$, where $\hat{T} > T$. Since $\mathbf{w}_T$ does not depend on $A_{T+1}, \ldots, A_{\hat{T}}$, we can terminate the algorithm after T iterations and return $\mathbf{w}_T$. Using Thm. 3 we know that

$$f(\mathbf{w}_T) - f(\mathbf{w}^\star) \;\leq\; \frac{c\ln(\hat{T})}{\delta\,\lambda\,\hat{T}} \;\leq\; \frac{c\ln(T)}{\delta\,\lambda\,T} \;\;.$$

We conclude this section with a discussion on the dependence of the convergence rate on the confidence parameter δ and on the accuracy parameter ϵ. From Thm. 3 we obtain that to achieve accuracy ϵ with confidence $1-\delta$ we need $\tilde{O}(\frac{1}{\lambda\,\delta\,\epsilon})$ iterations. In contrast, by applying previously studied conversions of online algorithms in the PAC setting (e.g. (Cesa-Bianchi et al., 2004; Cesa-Bianchi & Gentile, 2006)) one can obtain accuracy of ϵ with confidence $1-\delta$ using $\tilde{O}(\frac{\ln(1/\delta)}{\lambda\,\epsilon^2})$ iterations. Thus, as long as the desired confidence is not too high, our convergence rate is significantly better. If we would like to have a very high confidence, we can use a simple amplification technique (a.k.a. boosting the confidence), to construct a few candidate vectors such that with confidence $1-\delta$ at least one of the vectors has accuracy of $\tilde{O}(\frac{\ln(1/\delta)}{\lambda\,T})$. Let s denote the smallest integer larger than $\ln(1/\delta)$. We run the algorithm s times while setting the number of iterations for each run to T/s. We then randomly choose one vector from the vectors constructed by each run. Denote by $\hat{\mathbf{w}}_1, \ldots, \hat{\mathbf{w}}_s$ the resulting weight vectors. Using Thm. 3 with a confidence value of $1/e$, we know that with probability of at least $1 - 1/e$ we have that

$$f(\hat{\mathbf{w}}_i) - f(\mathbf{w}^\star) \leq \frac{c\,e\,\ln(T)}{\lambda\,T/s} \leq \frac{c\,e\,\ln(T)\,\left\lceil \ln\left(\frac{1}{\delta}\right)\right\rceil}{\lambda\,T} \;\;. \quad (9)$$

Therefore, the probability that for *all* runs the above inequality does not hold is at most $e^{-s} \leq \delta$. In other words, with probability of at least $1-\delta$, at least one of the vectors $\hat{\mathbf{w}}_i$ satisfies Eq. (9). We have therefore shown a method that uses $\tilde{O}(\frac{\ln(1/\delta)}{\lambda\,\epsilon})$ iterations for constructing $\lceil \ln(1/\delta) \rceil$ weight vectors, where at least one of them is ϵ-accurate. Finally, we need to pick an ϵ-accurate vector from the set of s vectors. This requires $O(\frac{1}{\lambda\epsilon^2})$ examples in the general case, since estimating the objective of each $\hat{\mathbf{w}}_i$ up to accuracy of ϵ requires $O(\frac{1}{\lambda\epsilon^2})$ examples. Note however that if we would like to simply choose the best performing vector with respect to the zero-one error over a validation set (rather than based on the objective value of SVM), we only need $O(\frac{1}{\epsilon^2})$ examples. That is, we gain a factor of $1/\lambda$. We leave further exploration of this issue to future work.

4. Extensions

In this section we discuss a few extensions to our basic classification learning algorithm. These extensions broaden the set of applications that can be tackled by our approach. Due to the lack of space we confine ourselves to a rather high level overview of two extensions and defer the complete details to a long version of this paper. We would like to note though that we have devised generalizations of Pegasos to complex decision problems, from multiclass categorization to learning with structured data.

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,

$$\ell\left((\mathbf{w}, b); (\mathbf{x}, y)\right) = \max\{0, 1 - y(\langle \mathbf{w}, \mathbf{x} \rangle + b)\} \;. \quad (10)$$

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 briefly describe three different approaches to learn the bias term and underscore 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 (Duda & Hart, 1973). 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 take the same value. We assume w.l.o.g 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 slightly 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 thus can be used without any restriction on A_t.

The second approach incorporates b explicitly by defining the loss as given in Eq. (10) while *not* penalizing for b. Formally, the task is to find an approximate solution to the following problem,

$$\min_{\mathbf{w},b} \frac{\lambda}{2}\|\mathbf{w}\|^2 + \frac{1}{m}\sum_{(\mathbf{x},y)\in S} [1 - y(\langle \mathbf{w}, \mathbf{x}\rangle + b)]_+ \quad . \quad (11)$$

Note that all the sub-gradients calculations w.r.t $\mathbf{w}$ remain intact. The sub-gradient with respect to b is also simple to compute. For a sample A_t it amounts to, $\frac{-1}{|A_t|}\sum_{(\mathbf{x},y)\in A_t^+} y$ and thus requires only k additions and subtractions and a single devision. 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. This is due to the fact that with the incorporation of b, 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 $O(1/\sqrt{T})$.

The last method entertains the advantages of the two methods above at the price of a more complex algorithm that is applicable only for large values of k. The main idea is to rewrite the optimization problem given in Eq. (11) as $\min_{\mathbf{w}} \frac{\lambda}{2}\|\mathbf{w}\|^2 + g(\mathbf{w}; S)$ where

$$g(\mathbf{w}; S) = \min_b \tfrac{1}{m}\textstyle\sum_{(\mathbf{x},y)\in S} [1 - y(\langle \mathbf{w}, \mathbf{x}\rangle + b)]_+ . \quad (12)$$

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 subgradient of $f(\mathbf{w}; A_t)$ and subtract it (multiplied by η_t) from $\mathbf{w}_t$. Finally, we project the resulting vector so that its norm will not exceed $1/\sqrt{\lambda}$. The problem however is how to find a subgradient of $g(\mathbf{w}; A_t)$, as $g(\mathbf{w}; A_t)$ is defined through a minimization problem over b. It can be shown that the complexity of finding a subgradient of $g(\mathbf{w}; A_t)$ is equivalent to the complexity of solving the minimization problem in Eq. (12). The latter problem is a generalized weighted median problem that can be solved efficiently in time $O(k)$. We omit the details due to the lack of space.

Table 1. Training time in CPU-seconds

	Pegasos	SVM-Perf	SVM-Light
CCAT	2	77	20,075
Covertype	6	85	25,514
astro-ph	2	5	80

The above adaptation indeed work for the case $A_t = S$ 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, we can use more involved measure concentration tools to show that the expectation of $f(\mathbf{w}; A_t)$ is close enough to $f(\mathbf{w}; S)$. We again omit the details due to the lack of space.

Using Mercer kernels: One of the main benefits of support vector machines is their ability to incorporate and construct non-linear predictors using kernels which satisfy Mercer's conditions. The crux of this property stems from the representer theorem (Kimeldorf & Wahba, 1971), which implies that the optimal solution of SVM can be expressed as a linear combination of its constraints. In the classification problem, the representer theorem implies that $\mathbf{w}$ is a linear combination of the instances $\mathbf{x}_i$. The common approach for solving the optimization problem for SVM when kernels are employed is to switch to the dual problem and find the optimal set of dual variables. Following (Freund & Schapire, 1999; Kivinen et al., 2002), we outline a different approach and directly minimize the primal problem while still using kernels. The main observation is that if $\mathbf{w}_1$ is initialized to be the zero vector, then at each iteration of the algorithm $\mathbf{w}_t$ can be written as $\mathbf{w}_t = \sum_{i\in I_t} \alpha_i \mathbf{x}_i$, where I_t is a subset of $\{1, \ldots, m\}$. The above claim can be easily proved using an inductive argument. Therefore, we can store the set I_t and the coefficients α_i instead of storing $\mathbf{w}_t$. It is now easy to verify that the algorithm in Fig. 1 can be used in conjunction with kernels, by representing $\mathbf{w}_t$ using I_t and α_i, calculating inner product operations using $\langle \mathbf{w}_t, \mathbf{x}_t\rangle = \sum_{i\in I_t} \alpha_i \langle \mathbf{x}_i, \mathbf{x}_t\rangle$, and evaluating the norm of $\mathbf{w}_t$ using $\|\mathbf{w}_t\|^2 = \sum_{i,j\in I_t} \alpha_i\alpha_j \langle \mathbf{x}_i, \mathbf{x}_j\rangle$. Based on the analysis in previous sections, Pegasos finds an ϵ-accurate solution using $\tilde{O}(1/(\delta\lambda\epsilon))$ iterations, while each iteration involves a single inner product between $\mathbf{w}$ and $\mathbf{x}$. Note however that each inner product operation between $\mathbf{w}$ and $\mathbf{x}$ may require $\min\{m, \tilde{O}(1/(\delta\lambda\epsilon))\}$ evaluations of the kernel function.

5. Experiments

In this section we present experimental results that demonstrate different merits of our algorithm and its accompanying analysis. We start by showing that Pegasos is in-

deed a practical tool for solving large scale problems. In particular, we compare its runtime to a new state-of-the-art solver (Joachims, 2006) on three large datasets. Next, we compare Pegasos to two previously proposed methods that are based on stochastic gradient descent, namely to Norma (Kivinen et al., 2002) and to the method given in (Zhang, 2004). Finally, we explore the empirical behavior of the algorithm with respect to the parameter k. In all of the experiments we did not incoprorate a bias term since (Joachims, 2006; Kivinen et al., 2002; Zhang, 2004) do not incorporate that term either. Additionally, we used the algorithm as in Fig. 1, omitting the stage of boosting the confidence, as we found empirically that in practice it was not necessary.

In our first experiment we compared Pegasos to the SVM-Perf algorithm (Joachims, 2006). We used the following datasets, which were provided to us by T. Joachims.
(1) The binary text classification task CCAT from the Reuters RCV1 collection. There are 804,414 examples and there are 47,236 features with sparsity 0.16% in this dataset.
(2) Classification of abstracts of scientific papers from the Physics ArXiv according to whether they are in the Astro-physics section. There are 99,757 features of high sparsity (0.08%). There are 62,369 examples in this dataset.
(3) Class 1 in the Covertype dataset of Blackard, Jock & Dean, which is comparably low-dimensional with 54 features and a sparsity of 22.22%. There are 581,012 examples in this dataset.

Table 4 lists the cpu-time of Pegasos and SVM-Perf on the datasets described above. SVM-Perf (Joachims, 2006) is a cutting plane algorithm for solving SVM that is based on a reformulation of the SVM problem. It was shown in (Joachims, 2006) that SVM-Perf is substantially faster than SVM-Light, achieving a speedup of several orders of magnitude on most datasets. We run both Pegasos and SVM-Perf on the three datasets with values of λ as given in (Joachims, 2006), namely, $\lambda = 10^{-4}$ for CCAT, $\lambda = 2 \cdot 10^{-4}$ for Astro-physics, and $\lambda = 10^{-6}$ for Covertype. We used the latest version of SVM-perf, implemented in C, as provided by T. Joachims. We implemented Pegasos in C++ and run all the experiments on a 2.8GHz Intel Xeon processor with 4GB of main memory under Linux. For completeness, we added to the table the runtime of SVM-Light as reported in (Joachims, 2006). As can be seen in the table, although SVM-Perf is by itself very fast, Pegasos still achieves a significant improvement in run-time. We calculated the objective value of the solutions obtained by Pegasos and SVM-Perf. For all three datasets, the objective value of Pegasos never exceeded that of SVM-Perf by more than 0.001. In addition, the generalization error of both methods was virtually identical. 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 larger than its runtime for CCAT, although the latter dataset is larger.

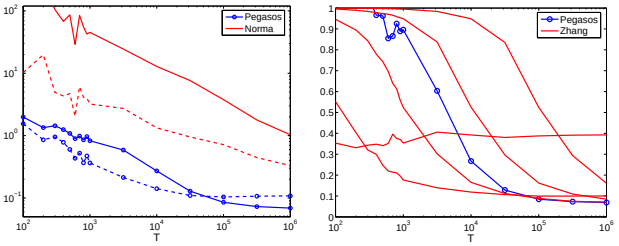

Figure 2. Comparisons of Pegasos to Norma (left) and Pegasos to stochastic gradient descent with a fixed learning rate (right) on the Astro-Physics datset. In the left plot, the solid lines designate the objective value and the dashed lines depict the loss on the test set.

In our next experiment, we compared Pegasos to Norma (Kivinen et al., 2002) and to a variant of stochastic gradient descent described in (Zhang, 2004). Both methods are similar to Pegasos when setting $k = 1$ with two differences. First, there is no projection step. Second, the scheduling of the learning rate, η_t, is different. In Norma (Thm. 4), it is suggested to set $\eta_t = p/(\lambda\sqrt{t})$, where $p \in (0, 1)$. Based on the bound given in Thm. 4 of (Kivinen et al., 2002), 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 Thm. 4 in (Kivinen et al., 2002) yields the bound $O(1/(\lambda\sqrt{T}))$. We therefore hypothesized that Pegasos would converge much faster than Norma. In Fig. 5 (left) we compare Pegasos to Norma on the Astro-Physics dataset. We split the dataset into a training set with 29,882 examples and a test set with 32,487 examples and report the final objective value and the average hinge-loss over the test set. As in (Joachims, 2006), we set $\lambda = 2 \cdot 10^{-4}$. As can be seen, Pegasos clearly outperforms Norma. In fact, Norma fails to converge even after 10^6 iterations. This can be attributed to the fact that the value of λ here is rather small. As mentioned before, the differences between Pegasos and Norma are both the different learning rate and the projection step which is absent in Norma. We also experimented with a version of Pegasos without the projection step and with a version of Norma that includes a projection step. We found that the projection step is important for the convergence of Pegasos, especially when T is small, and that a projection step also improves the performance of Norma. However, Pegasos still outperforms the version of Norma that includes an additional projection step. We omit the graphs due to the lack of space. We now turn to comparing Pegasos to the algorithm from (Zhang, 2004) which simply sets $\eta_t = \eta$, where η is a (fixed) small number. A major disadvantage of this

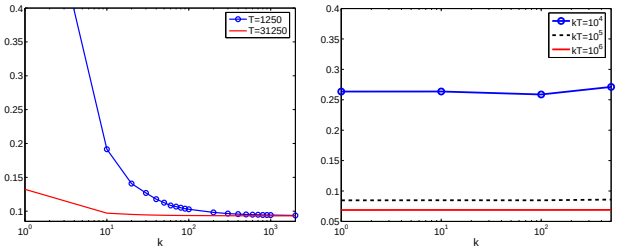

Figure 3. The effect of k on the objective value of Pegasos on the Astro-Physics dataset. Left: T is fixed. Right: kT is fixed.

approach is that finding an adequate value for η is a difficult task on its own. Based on the analysis given in (Zhang, 2004) we started by setting η to be 10^{-5}. Surprisingly, this turned out to be a poor choice and the optimal choice of η was substantially larger. In Fig. 5 (right) 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. We would like to emphasize that for large datasets, the time required for evaluating the objective is 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 of η_t.

In our last experiment, we examined the effect of the parameter k on the convergence of the algorithm. Our analysis implies that the convergence of Pegasos does not depend on k. Based on this fact, the optimal choice of k in terms of run time should be $k = 1$. In Fig. 3 (left) we depict the objective value obtained by Pegasos as a function of k when T is fixed. It is clear from the figure that, in contrast to our bounds, the convergence of Pegasos improves as k gets larger. This fact may be important in a distributed computing environment. As long as k is smaller than the number of CPUs, the complexity of Pegasos still depends solely on T (and on $log(k)$), while the throughput greatly improves. An interesting question is how to set k for a single CPU. In this case, the runtime of Pegasos is of the order of kT. In Fig. 3 (right) we show that the convergence rate of Pegasos as a function of k is approximately constant, for a wide range of values of k, so long as kT is kept fixed. We leave further research of both of the theoretical and practical aspects of the choice of k to future work.

6. Conclusions

We described and analyzed a simple and effective algorithm for approximately minimizing the objective function of SVM. The algorithm, called Pegasos, is a modified stochastic gradient method in which every gradient descent step is accompanied with a projection step. 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 or because of its simplicity. We plan to investigate all the questions we surfaced in this paper as well as to conduct thorough experiments with non-linear kernels. In addition, we have started investigating the usage of similar paradigms in other learning problems such as L_1-SVM and other loss functions.

Acknowledgements Part of this work was done while SS and NS were visiting IBM research labs, Haifa, Israel. This work was supported by grant I-773-8.6/2003 from the German Israeli Foundation (GIF).

References

Boyd, S., & Vandenberghe, L. (2004). *Convex optimization*. Cambridge University Press.

Censor, Y., & Zenios, S. (1997). *Parallel optimization: Theory, algorithms, and applications*. Oxford University Press, NY.

Cesa-Bianchi, N., Conconi, A., & Gentile, C. (2004). On the generalization ability of on-line learning algorithms. *IEEE Transactions on Information Theory*, *50*, 2050–2057.

Cesa-Bianchi, N., & Gentile, C. (2006). Improved risk tail bounds for on-line algorithms. *NIPS*.

Crammer, K., Dekel, O., Keshet, J., Shalev-Shwartz, S., & Singer, Y. (2006). Online passive aggressive algorithms. *JMLR*, *7*.

Cristianini, N., & Shawe-Taylor, J. (2000). *An introduction to support vector machines*. Cambridge University Press.

Duda, R. O., & Hart, P. E. (1973). *Pattern classification and scene analysis*. Wiley.

Fine, S., & Scheinberg, K. (2001). Efficient svm training using low-rank kernel representations. *JMLR*, *2*, 242–264.

Freund, Y., & Schapire, R. E. (1999). Large margin classification using the perceptron algorithm. *Mach. Learning*, *37*, 277–296.

Hazan, E., Kalai, A., Kale, S., & Agarwal, A. (2006). Logarithmic regret algorithms for online convex optimization. *COLT*.

Hush, D., Kelly, P., Scovel, C., & Steinwart, I. (2006). Qp algorithms with guaranteed accuracy and run time for support vector machines. *JMLR*.

Joachims, T. (1998). Making large-scale support vector machine learning practical. In B. Schölkopf, C. Burges and A. Smola (Eds.), *Advances in kernel methods - support vector learning*. MIT Press.

Joachims, T. (2006). Training linear svms in linear time. *KDD*.

Kimeldorf, G., & Wahba, G. (1971). Some results on tchebycheffian spline functions. *J. Math. Anal. Applic.*, *33*, 82–95.

Kivinen, J., Smola, A. J., & Williamson, R. C. (2002). Online learning with kernels. *IEEE' TSP*, *52*, 2165–2176.

Platt, J. C. (1998). Fast training of Support Vector Machines using sequential minimal optimization. In B. Schölkopf, C. Burges and A. Smola (Eds.), *Advances in kernel methods - support vector learning*. MIT Press.

Shalev-Shwartz, S., & Singer, Y. (2007). *Logarithmic regret algorithms for strongly convex repeated games* (Technical Report). The Hebrew University.

Vapnik, V. N. (1998). *Statistical learning theory*. Wiley.

Zhang, T. (2004). Solving large scale linear prediction problems using stochastic gradient descent algorithms. *ICML*.