On Pegasos and first order optimizers for online convex optimization

Based on: "Pegasos: primal estimated sub-gradient solver for SVM" (S. Shalev-Shwartz, Y. Singer et al.)

Romain Chor

March 2020

Introduction

Online convex optimization (OCO), as seen by Elad Hazan, can be linked to game theory. Indeed, its framework can be seen as a structured repeated game. The protocol of this learning framework is as follows:
At time t, the player makes a decision $x_t \in \mathcal{K}$ where $\mathcal{K}$ is a convex set, let us say $\mathbb{R}^d$. Afterwards, a convex cost function $f_t : \mathcal{K} \to \mathbb{R}$ is revealed. The cost incurred by the decision x_t is $f_t(x_t)$ and the player can update its next decision with it in order to minimize its cost. Think of a poker player trying to find the best strategy to stop losing his money.

The objective of this paper is to introduce Pegasos, an optimizer for Support Vector Machines (SVM) proposed by Shalev-Schwartz and Singer in the OCO framework. Some specific recalls on SVM and convex optimization will be made. For more general notions, the lector is invited to have a look at the references.

Pegasos will be numerically compared to other first-order solvers on MNIST handwritten digits dataset and convergence results will be given.

Contents

Chapter 1

The mathematical framework

1.1 Support Vector Machines (SVMs)

Let $\mathcal{D}_n = \{(X_i,y_i)\}_{i=1,\dots,n}$ be some data with $(X_i)_i$ being features and $(y_i)_i$ class labels.

For binary classification tasks, a SVM model construct a hyperplane that separates the dataset into two groups, defining a class $\{-1\}$ and a class $\{+1\}$. From this arises a possible definition for SVM:

Definition 1 (Support Vector Machine)**.** *A SVM is a linear classifier* $g_n := sign(<\omega^*,.>+b^*)$ *where* (ω^*,b^*) *are solutions to the following minimization problem:*

$$\underset{\omega\in\mathbb{R}^d,\ b\in\mathbb{R}}{\text{minimize}} \quad \frac{\lambda}{2}\|\omega\|_2^2 + \frac{1}{n}\sum_{i=1}^{n} max\ (0,\ 1-y_i(\langle\omega,X_i\rangle+b)) \tag{1.1}$$

where $\lambda>0$ *is a regularization parameter and* $\|.\|_2$ *is the standard Euclidean norm.*
The solutions define the separating hyperplane: $\{x\in R^d : \langle\omega^*,x\rangle+b^*=0\}$.

Intuitively, a good separation is achieved by the hyperplane that has the largest distance to the nearest training-data point of any class. SVMs are said to maximize the margin. Two possible situations can happen:
(1) The training dataset is said to be linearly separable i.e. $\exists(\omega,b)\in\mathbb{R}^d\times\mathbb{R}:\ \forall i,\ y_i(\omega^t X_i+b)\geq 0$: the SVM is meant to make no classification error. In this case, it is called a hard-margin classifier.
(2) The training dataset is not linearly separable: the SVM still maximizes the margin but accepts classification errors.
The function $(y,y')\in\mathbb{R}^2\mapsto\max(0,1-yy')$, called hinge loss function takes into account this idea.

From now, we will consider the intercept b **as being zero for simplicity and denote** $l_{X_i,y_i}(w) := max(0,\ 1-y_i(\langle w,X_i\rangle)$ **for the hinge loss associated to observation** (X_i,y_i)**.**

1.2 Basics of convex optimization and subgradient methods

Remark. The objective function in (1.1) is not differentiable at $\omega=0$. It is however convex.

To solve (1.1) we will need the following notions:

Definition 2 (Subdifferential)**.** *Let* $f:\mathbb{R}^d\to(-\infty,\infty]$ *be a convex function. The subdifferential of* f *at* $x\in\mathbb{R}^d$ *is defined by*

$$\partial f(x) = \{v\in\mathbb{R}^d : \forall y\in\mathbb{R}^d,\ f(y)\geq f(x)+v^t(y-x)\}$$

The elements of $\partial f(x)$ *are called subgradients of* f *at* x.

Theorem 1 (Fermat's rule)**.** *Let $f : \mathbb{R}^d \to (-\infty, \infty]$ be a convex function. Then,*

$$x \in \mathbb{R}^d \text{ is a global minimizer of } f \quad \Leftrightarrow \quad 0 \in \partial f(x^*)$$

To minimize (1.1), one way is to use (sub)gradient descent (GD), which is a special case of online algorithms, to iteratively approach the solution. However, despite having good convergence guarantees, gradient descent can be slow to run. The necessity to evaluate the gradient of the objective function at each iteration slows down the algorithm. Indeed, at iteration t the gradient of the objective function in (1.1) is given by:

$$\nabla_t := \frac{1}{n}\sum_{i=1}^{n} \nabla l_{X_i,y_i}(w_t) + \lambda w_t \quad \text{where } \nabla l_{X_i,y_i}(w_t) = (-y_i.X_i)\mathbf{1}_{y_i\langle w_t, X_i\rangle \leq 1}$$

Therefore, it takes n operations at each iteration to evaluate the gradient. To overcome this computational difficulty, we usually use stochastic subgradient descent (SGD):

Algorithm 1 Stochastic subgradient descent for SVM training

Initialization: $w_1 = 0$
for $t = 1$ to T **do**
  (1) Pick $i_t \in [n]$ uniformly at random
  (2) Let $\tilde{\nabla}_t := \nabla l_{X_{i_t},y_{i_t}}(w_t) + \lambda w_t$ where $\nabla l_{X_{i_t},y_{i_t}}(w_t) = (-y_{i_t}.X_{i_t})\mathbf{1}_{y_{i_t}\langle w_t, X_{i_t}\rangle \leq 1}$
  Update: $w_{t+1} \leftarrow w_t - \eta_t \tilde{\nabla}_t$
end for
Return: $\bar{w}_T := \frac{1}{T}\sum_{t=1}^{T} w_t$

Stochastic subgradient descent picks one observation, evaluates the gradient of a surrogate objective function based on this observation and makes the update. Roughly speaking, one iteration of GD corresponds to n iterations of SGD. The reduction is huge if n is large.

Note that SGD comes with many variants, depending on the choice of the learning rate or the update. A few ones will be implemented for numerical comparison.

Chapter 2

Pegasos: a stochastic subgradient solver for SVM

Pegasos performs a stochastic subgradient descent on the primal objective (1.1) with a carefully chosen stepsize. We will introduce the original algorithm and a variant.

2.1 Basic Pegasos algorithm

Pegasos algorithm basically performs the classical stochastic subgradient descent update with a learning rate $\eta_t = 1/(\lambda t)$ and projection step on the Euclidean ball of radius $1/\sqrt{\lambda}$. More formally, the parameter update at each iteration can be rewritten as:

$$w_{t+1} \leftarrow (1 - 1/t)w_t \; + \; \eta_t y_{i_t} X_{i_t} \mathbf{1}_{y_{i_t}\langle w_t, X_{i_t}\rangle \leq 1}$$
$$w_{t+1} \leftarrow min\left\{1, \frac{1}{\sqrt{\lambda}\|w_{t+1}\|_2}\right\} w_{t+1}$$

Remark. Note that unlike Algorithm 1, Pegasos outputs the last iterate w_{T+1}.
This is an empirical choice from the authors based on numerical results.
However, for comparison we will output $\mathbf{\bar{w}_T = \frac{1}{T}\sum_{t=1}^{T} w_t}$ in our implementation.

2.2 Mini-batch Pegasos

Stochastic (sub)gradient descent algorithms come with variants called mini-batch algorithms. Instead of selecting a single training example at each iteration, those algorithms choose a subset $A_t \subset [n]$ s.t. $|A_t| = k$ of indices taken uniformly at random. According to the authors, these indices can be sampled with or without replacement. The latter is equivalent to pick a random permutation of $[n]$ of size k.
We will **not** use mini-batches for numerical experiments, they are only introduced here for following analysis.

Algorithm 2 Mini-batch Pegasos

Initialization: $w_1 = 0$
for $t = 1$ to T **do**
(1) Sample A_t ($|A_t| = k$) of $[m]$
(2) Set $A_t^+ = \{i \in A_t : y_i \langle w_t, X_i \rangle \leq 1\}$ and $\eta_t = 1/\lambda t$
Update: $w_{t+1} \leftarrow (1 - \eta_t \lambda) w_t + \frac{\eta_t}{k} \sum_{i \in A_t^+} y_i X_i$
Project: $w_{t+1} \leftarrow min\left\{1, \frac{1}{\sqrt{\lambda}\|w_{t+1}\|_2}\right\} w_{t+1}$
end for
Return: $\bar{w}_T := \frac{1}{T}\sum_{t=1}^{T} w_t$

2.3 Convergence results and regret bounds

Although the objective is to minimize the objective function f defined by $f(w) = \frac{\lambda}{2}\|\omega\|_2^2 + \frac{1}{n}\sum_{i=1}^{n}\max(0, 1 - y_i(\langle\omega, X_i\rangle + b))$ over a convex set $\mathcal{K}$, we want to analyse the algorithms in an online convex optimization (OCO) framework. Therefore, we will focus on regret defined by

$$\text{regret}_T := \sum_{t=1}^{T} f_t(w_t) - \min_{w\in\mathcal{K}} \sum_{t=1}^{T} f_t(w)$$

Recall that the main interest is to analyze Pegasos in the case $k = 1$ to compare with stochastic gradient descent. However the following analysis focusses on the mini-batch variant as it is more general.
On each iteration of the algorithm, we focus on an instantaneous objective function $f_t(w) := f(w; A_t) = \frac{\lambda}{2}\|w\|_2^2 + \frac{1}{k}\sum_{i\in A_t} l_{X_i,y_i}(w)$. Note that as the hinge loss is a convex function and $\frac{\lambda}{2}\|.\|^2$ a λ-strongly convex function, f_t is λ-strongly convex.

In the following we denote $w^* = \text{argmin}_{w\in\mathcal{K}} f(w)$.

Lemma 1. *Let $f_t,...,f_T$ be a sequence of λ-strongly convex functions and let $\mathcal{K}$ be a closed convex set. We define $\Pi_{\mathcal{K}}(w) := \text{argmin}_{w'\in\mathcal{K}}\|w - w'\|_2$. Let $w_1,...,w_{T+1}$ be a sequence of vectors such that $w_1 \in \mathcal{K}$ and $\forall t \geq 1$, $w_{t+1} = \Pi_{\mathcal{K}}(w_t - \eta_t\nabla_t)$ with ∇_t is the subgradient set of f_t at w_t and $\eta_t = 1/\lambda t$. Assume that for all t, $\|\nabla_t\|_2 \leq G$. Then, for all $u \in \mathcal{K}$,*

$$\frac{1}{T}\sum_{t=1}^{T} f_t(w_t) \;\leq\; \frac{1}{T}\sum_{t=1}^{T} f_t(w^*) + \frac{c(1 + ln(T))}{2\lambda T}$$

Theorem 2. *For S being any bounded set, assume that for all $(x,y) \in S$, $\|x\|_2 \leq R$.*
Let $w^* := \text{argmin}_{w\in K}\ f(w)$ and $c = (\sqrt{\lambda} + R)^2$. Then, for $T \geq 3$

$$\frac{1}{T}\sum_{t=1}^{T} f_t(w_t) \;\leq\; \frac{1}{T}\sum_{t=1}^{T} f_t(w^*) + \frac{c(1 + ln(T))}{2\lambda T}$$

Proof. The proof directly comes from lemma 1 hence we need to prove the following :
- $\forall t \in [T]$, f_t is λ-strongly convex
- $\|\nabla_t\|_2 \leq G$, with $G \in \mathbb{R}$
- $w^* \in \mathcal{K}$

The update can be rewritten $w_{t+1} = \Pi_{\mathcal{K}}(w_t - \eta_t\nabla_t)$ with ∇_t being the sub-gradient of the objective function f_t and $\mathcal{K}$ the Euclidean ball of radius $1/\sqrt{\lambda}$ if the projection step is done and $\mathcal{K} = \mathbb{R}^d$ otherwise.
As precised above, f_t is λ-strongly convex as a sum of a convex function and a λ-strongly convex function. Then,
- If the projection step is performed, knowing that $\|w_t\|_2 \leq 1/\sqrt{\lambda}$ and $\|x\|_2 \leq R$, we have $\|\nabla_t\|_2 \leq \sqrt{\lambda} + R$ by the triangle inequality.
- Otherwise, we admit that $\|w_{t+1}\|_2 \leq R/\lambda$. Therefore, $\|\nabla_t\|_2 \leq 2R$.

Finally, without the projection we have trivially $w^* \in \mathbb{R}^d = B$. With the projection, we need to examine the dual form of the SVM problem (1.1). Note that (1.1) can be rewritten in a constrained form:

$$\underset{\omega\in\mathbb{R}^d,\ \xi\in\mathbb{R}^n}{\text{minimize}} \quad \frac{\lambda}{2}\|\omega\|_2^2 + C\sum_{i=1}^{n}\xi_i \quad \text{s.t.} \quad \forall i \in [n]:\ \xi_i \geq 0,\ \xi_i \geq 1 - y_i\langle w, X_i\rangle \tag{2.1}$$

The associated dual problem is:

$$\underset{\alpha\in\mathbb{R}^n}{\text{minimize}} \quad \sum_{i=1}^{n}\alpha_i+\frac{1}{2}\left\|\sum_{i=1}^{n}\alpha_i y_i X_i\right\|_2^2 \quad \text{s.t.} \quad \forall i\in[n]: 0\leq\alpha_i\leq C \tag{2.2}$$

denoting $C=1/\lambda n$. Let (w^*,ξ^*) and α^* be respectively the optimal primal and dual solutions. We have $w^*=\sum_{i=1}^{n}\alpha_i^* y_i X_i$. At the optimum value α^*, the objective in (2.2) can be rewritten as $\|\alpha^*\|_1-\frac{1}{2}\|w^*\|_2^2$. Moreover, by strong duality we obtain

$$\frac{1}{2}\|w^*\|_2^2+C\|\xi^*\|_1=\|\alpha^*\|_1-\frac{1}{2}\|w^*\|_2^2$$

The constraint in (2.2) gives $\|\alpha^*\|_\infty\leq C=1/\lambda n$ hence $\|\alpha^*\|_1\leq n\|\alpha^*\|_\infty\leq 1/\lambda$. This yields

$$\frac{1}{2}\|w^*\|_2^2\leq\frac{1}{2}\|w^*\|_2^2+C\|\xi^*\|_1=\|\alpha^*\|_1-\frac{1}{2}\|w^*\|^2\leq\frac{1}{\lambda}-\frac{1}{2}\|w^*\|_2^2$$

Applying lemma 1 gives the desired bound. □

Therefore, we obtain an upper bound on the regret multiplying each member of the inequality by T. In particular, the regret is logarithmic in the number of iterations and depends only on λ.
In the other hand, stochastic gradient descent (Algorithm 1) usually comes with a learning rate $\eta_t=G/\sqrt{t}$ with G as in $\|\nabla_t\|_2\leq G$. In this case, we have $\text{regret}_T=O(G\sqrt{T})$ which is worse than the regret bound for Pegasos.

Chapter 3

Numerical experiments on MNIST dataset

3.1 Implementation details

The MNIST database was made famous by deep learning researcher Yann LeCun when he introduced convolutional neural networks (CNN). It contains images of handwritten digits with a training set of 60 000 examples, and a test set of 10 000 examples. It is a subset of a larger set available from NIST.
Each example is a 28x28 matrix of each pixel value (in greyscale, from 0 to 255) from the image.

For computational reasons, we subsampled the training and test sets. We trained our algorithms on 10 000 examples and tested them with 3000 examples (test set size is 30% of training test size). Moreover, the problem was transformed to a binary classification problem with $\{-1,1\}$ as classes (0 becomes -1 and other digits become 1). In terms of preprocessing, we rescaled our examples by dividing each pixel value by 255 as SVMs have better performances on normalized data.

3.2 Pegasos versus other first order optimizers

Recall that the instantaneous cost (or objective) function for Pegasos is :

$$\frac{\lambda}{2}\|\omega\|_2^2 + \max\,(0,\, 1 - y_{i_t}\langle\omega, X_{i_t}\rangle)$$

with i_t the indice randomly selected at iteration t.

We will compare Pegasos with other stochastic subgradient-based algorithms, such as vanilla SGD, stochastic mirrored descent (SMD), stochastic exponentiated gradient +/- (SEGpm) and AdaGrad. Offline gradient descent with and without projection are also included for comparison. Note that even though the original Pegasos algorithm performs a projection on the Euclidean ball, all algorithms are implemented using projection on the l_1-ball of radius $\mathbf{z = 10}$. Indeed, the MNIST data is sparse thus the choice of this ball is convenient due to its shape.

The quantities of interest are:
- the cost function value at each iteration
- the estimated convergence rate at each iteration
More precisely, these quantities are monitored by log-log plots.

The estimated convergence rate, or accuracy, corresponds to the number of well-classified examples divided by n i.e.: $\frac{1}{n}\sum_{i=1}^{n}\mathbf{1}_{y_i\langle w_t, X_i\rangle > 0}$. This quantity is evaluated on the test set hence it measures how well the SVM is able to classify new examples at each iteration. For computational reasons again, the parameter w can **not** be stored at each iteration when T is great.

In our experiments, w is stored every 100 iterations.
As said in Chapter 1, one iteration of GD is equivalent to n iterations of SGD thus with $n = 10\ 000$ here and 100 iterations of GD we should perform 1 000 000 iterations for fair comparison! However in practice the GD is implemented using matrix operations which greatly reduces the complexity. **We runned GD with 100 iterations and online algorithms with 10 000 iterations.**

One interesting comparison is Pegasos versus vanilla SGD (recall that Pegasos uses a learning rate $\eta_t = 1/\lambda t$ while SGD is implemented using $\eta_t = 1/\sqrt{t}$. One can see the influence of the choice of the learning rate below.
First, costs values of Pegasos are always below SGD's ones. The difference between the two algorithms is particularly visible on the second plot. Indeed, the curve of SGD takes time to decrease and at $T = 10\ 000$ is above Pegasos' curve.

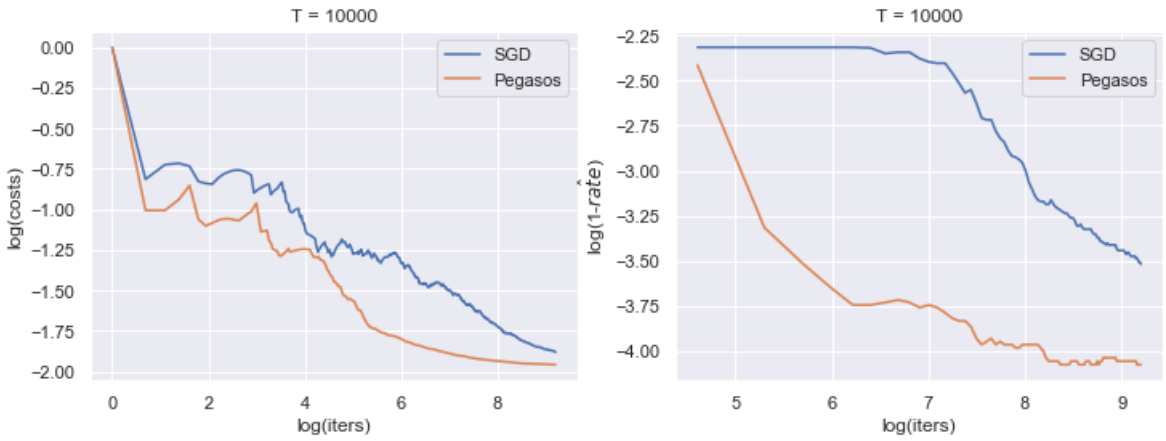

Figure 3.1: Pegasos vs. SGD with $\eta_t = 1/\lambda t$

Below, Pegasos is compared to other first-order optimizers. Pegasos have similar performance as projected GD, SMD, SEGpm and AdaGrad. **Erratum:** the red curve corresponds to SMD.

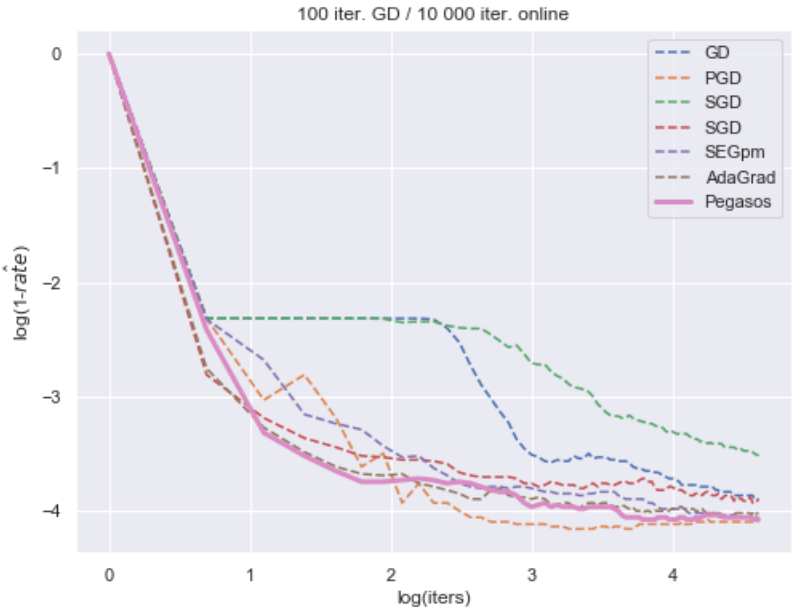

Figure 3.2: Pegasos vs first order optimizers

About training times: one can observe below that the rule of 1 iteration of GD = n iterations of SGD is only validated for SEGpm and Pegasos. This may show how matrix operations for gradient descent helps reduce complexity. The reason why Pegasos outperforms basic SGD in terms of training time is unknown.

	GD	PGD	SGD	SMD	SEGpm	AdaGrad	Pegasos
Training time (s)	38.519044	36.277634	53.345734	56.557502	39.177788	65.460008	43.23147

Figure 3.3: Training times

3.3 About hyperparameters tuning

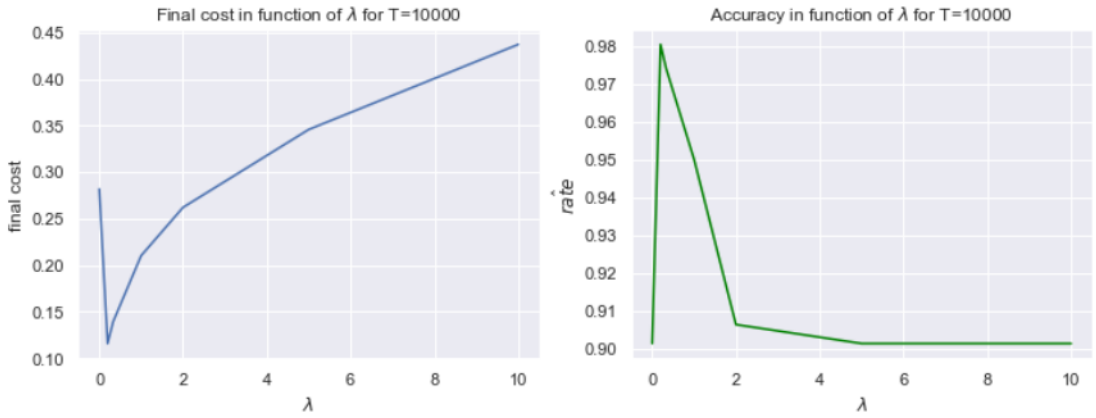

Figure 3.4: Influence of λ

On the figure above, we can observe the influence of the regularization parameter λ on the performance of Pegasos. In particular, we can see what happens in the extreme cases i.e. with under-regularization ($\lambda = 10^{-3}$) and over-regularization ($\lambda = 100$).
Choosing $\lambda = 10^{-3}$ yields poor performances as the model overfits the training data. When $\lambda = 100$, the model is too much penalized and Pegasos can not optimize properly the cost function.

References

[1] S. Shalev-Schwartz, Y. Singer et al., *Pegasos: primal estimated sub-gradient solver*, 2010
[2] E. Hazan, *Introduction to online convex optimization*, 2019
[3] M. Sangnier, Course *Introduction to Machine learning*, from Sorbonne University