SDCA [2] vs. Pegasos [3] for linear SVM fitting

Applications to blood cells classification

Dimitri Bouche, Cyril Verluise

May 9, 2018

Contents

1 Introduction

The aim of this project is to compare two optimization algorithms on the particular problem of fitting a linear SVM :

- Stochastic dual coordinate ascent (SDCA) [2] which is a stochastic version of a Dual coordinate ascent (DCA).
- Primal estimated subgradient solver for SVM (Pegasos) [3] which corresponds to a classic stochastic (sub)gradient descent (SGD) with a given choice of step-size.

As a reminder, the linear (regularized) SVM problem is the following :

$$\min_{w \in \mathbb{R}^d} P(w). \tag{1}$$

Where :

$$P(w) = \left[\frac{1}{n} \sum_{i=1}^{n} \phi_i(w^T x_i) + \frac{\lambda}{2} ||w||^2 \right].$$

d being the number of features, n the number of data points and ϕ_i the hinge loss :

$$\phi_i(w^T x_i) = \max(0, 1 - y_i w^T x_i).$$

With $y_i \in \{-1, 1\}$ being x_i's label.

Regarding theorethical guarantees, the Pegasos algorithm is stated to be faster by [3] yielding an ϵ suboptimal result in $\mathcal{O}(\frac{1}{\lambda\epsilon})$ or less (independant from the size of dataset) whereas the SDCA algorithm is said to yield such result in $\mathcal{O}(n + \frac{1}{\lambda\epsilon})$ or less [2], although it is argued that the latter can reach more precise results.

We will start by a short presentation of the two procedures and will then apply them both to the same problem of image classification (bloodcells classification) in order to see to what extent those theoretical guarantees apply in practise.

2 Brief presentation of the two algorithms

2.1 SDCA

We are here bound to paraphrase [3], so we will only state the updates formula that are of interest to us and perform the computations only when the closed form formula are not given (for instance for the SGD initialization).

2.1.1 SDCA-perm

We focus here on the dual of problem (1) :

$$\max_{\alpha \in \mathbb{R}^n} D(\alpha). \tag{2}$$

Where :

$$D(\alpha) = \left[\frac{1}{n} \sum_{i=1}^{n} -\phi_i^\star(-\alpha_i) - \frac{\lambda}{2} \left\| \frac{1}{\lambda n} \sum_{i=1}^{n} \alpha_i x_i \right\|^2 \right].$$

With $\phi_i^\star$ defined as :

$$\begin{aligned} \phi_i^\star(-a) &= -a y_i \; if \; a y_i \in [0, 1] \\ \phi_i^\star(-a) &= +\infty \; if \; a y_i \notin [0, 1] \end{aligned}$$

Solving problem (2) is equivalent to solving problem (1), since any solution to (2) can be transformed into a solution to (1) using the following function [3] :

$$w(\alpha) = \frac{1}{\lambda n} \sum_{i=1}^{n} \alpha_i x_i$$

We implemented the SDCA-perm version, which runs in epochs instead of employing complete randomization. We also add a stopping criterion based on the duality gap $P(w(\alpha)) - D(\alpha)$ as advised by the authors.

However, we do not apply the "Random option" (returning a randomly chosen value of α among the second half iterations) nor the "Average option" (returning the average value of α over the second half iterations) since it works very well in practice without. The pseudo code for our implementation is the following :

Data: $\alpha^{(0)}$, k_{max} (maximum number of epochs), ϵ (duality gap stopping threshold)
Set $w^{(0)} = w(\alpha^{(0)})$;
Set $g = P(w^{(0)}) - D(\alpha^{(0)})$;
Set $k = 0$;
while $g > \epsilon$ *and* $k < k_{max}$ **do**
  Draw $\{i_1, ..., i_n\}$ random permutation of $\{1, ..., n\}$;
  for $j = 1$ *to* n **do**
    $i = i_j$;
    $t \leftarrow t + 1$;
    $\Delta_i = \Delta_i(\alpha_i^{(t-1)}, w^{(t-1)})$;
    $\alpha^{(t)} \leftarrow \alpha^{(t-1)} + \Delta_i e_i$;
    $w^{(t)} \leftarrow w^{(t-1)} + \frac{1}{\lambda n} \Delta_i x_i$;
  end
  $k \leftarrow k + 1$;
end

Algorithm 1: SDCA Perm

With e_i the vector with 1 in the i-th position and 0s elsewhere, and Δ_i the coordinate update chosen to decrease the dual objective as given in [3]:

$$\Delta_i(\alpha_i^{(t-1)}, w^{(t-1)}) = y_i \max\left(0, \min\left(1, \frac{(\lambda n)(1 - x_i^T w^{(t-1)} y_i)}{||x_i||^2} + \alpha_i^{(t-1)} y_i\right)\right) - \alpha_i^{(t-1)}.$$

2.1.2 SGD initialization

It is advised in [3] to implement a different algorithm based on a modification of stochastic subgradient descent (SGD) just for the first epoch and then to switch back to SDCA. This is useful according to the authors because SDCA tends to perform updates that are too small in the first epoch in comparison to SGD.

The pseudo-code for this first modified epoch is given by :

Set $w^{(0)} = 0$;
for $t = 1$ *to* n **do**
  Find α_t to maximize $-\phi_t^\star(-\alpha_t) - \frac{\lambda t}{2} ||w^{(t-1)} + (\lambda t)^{-1} \alpha_t x_t||^2$;
  $w^{(t)} = \frac{1}{\lambda t} \sum_{i=1}^{t} \alpha_i x_i$;
end

Algorithm 2: Modified SGD

Since the closed form for the step of maximization is not given in [3], we proceed to the computations :

At each step $t \in \{1, ..., n\}$ of the epoch, we wish to find α_t that maximizes :

$$-\phi_t^\star(-\alpha_t) - \frac{\lambda t}{2}||w^{(t-1)} + (\lambda t)^{-1}\alpha_t x_t||^2.$$

Let us remark first of all that we must take α_t to be such that $\alpha_t y_t \in [0, 1]$ since if this is not the case, $-\phi_t^\star(-\alpha_t) = -\infty$.

Now supposing that $\alpha_t y_t \in [0, 1]$, developping the previous expression yields :

$$\alpha_t y_t - \frac{\lambda t}{2}\Big(||w^{(t-1)}||^2 + 2\frac{\alpha_t}{\lambda t}\langle w^{(t-1)}, x_t\rangle + \frac{\alpha_t^2}{\lambda^2 t^2}||x_t||^2\Big).$$

This is a second order polynomial in α_t. With a negative coefficient on the second order term. Thus this is concave. Setting the derivative to 0 with respect to α_t :

$$y_t - \langle w^{(t-1)}, x_t\rangle - \frac{\alpha_t}{\lambda t}||x_t||^2 = 0.$$

This gives us an optimal α_t : $\alpha_t^\star$ defined by :

$$\alpha_t^\star = \frac{\lambda t}{||x_t||^2}(y_t - x_t^T w^{(t-1)}).$$

Let us now ensure that we threshold this value to make sure that it stays within the interval $[0, 1]$: The optimal thresholded value is thus $\tilde{\alpha_t}$:

$$\begin{aligned}
\tilde{\alpha_t} &= \alpha_t^\star \ if \ \alpha_t^\star y_t \in [0, 1] \\
\tilde{\alpha_t} &= \min(0, y_t) \ if \ \alpha_t^\star < \min(0, y_t) \\
\tilde{\alpha_t} &= \max(0, y_t) \ if \ \alpha_t^\star > \max(0, y_t)
\end{aligned}$$

Summing up, the first epoch is performed using Modified-SGD (**Algorithm 2**) and then use the resulting α as initial value for SDCA-Perm (**Algorithm 1**).

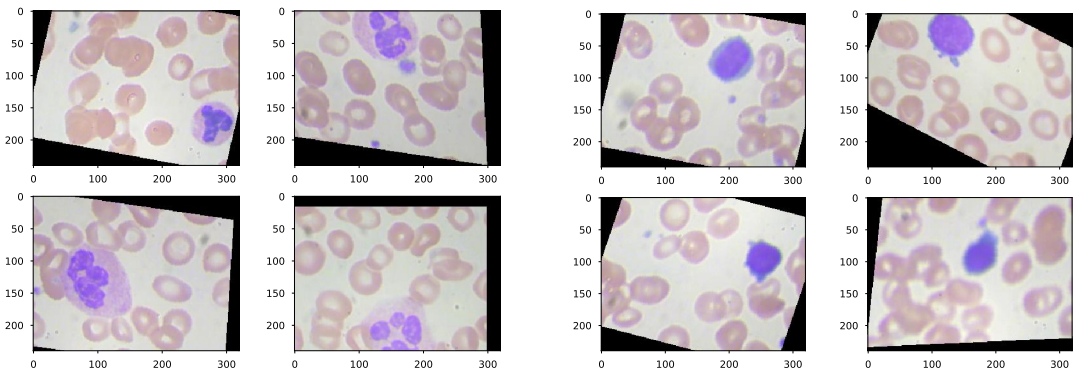
2.2 Pegasos

For this algorithm we have implemented almost exactly the mini-batch version from [3], the only difference being that we run it in epochs which is not the case in the paper. All the closed forms are given by the authors, we thus only gives the pseudo-code that we implemented :

Data: m (mini-batch size), k_{max} (number of epochs)
Set $w^{(0)} = 0$;
Set $t = 0$;
for $k = 0$ *to* $k_{max} - 1$ **do**
  Draw $\{A_1, ..., A_j\}$ random partition of $\{1, ..., n\}$ where $|A_i| = m, \ \ \forall i \in \{1, ..., j\}$;
  for $i = 1$ *to* j **do**
    $A_i^+ = \{l \in A_i : y_i x_i^T w^{(t)} < 1\}$;
    $\eta_t = \frac{1}{\lambda n}$;
    $w^{(t+1)} = (1 - \eta_t \lambda)w^{(t)} + \frac{\eta_t}{k}\sum_{l \in A_i^+} y_l x_l$;
    $t \leftarrow t + 1$;
  end
end

Algorithm 3: Mini-batch Pegasos running in epochs

Remark : we assumed that n is a multiple of m for clarity as trivial modifications can be made if it is not the case.



(a) Neutrophils (b) Lymphocytes

Figure 1: Examples for both classes

3 Application to bloodcells classification

3.1 Bloodcells images dataset

The dataset consists in microscopic pictures of different kind of bloocells, it is available on Kaggle [1]. There are four cells types in the dataset : Eosonophil, Lymphocyte, Monocyte, Neutrophil. We only concerned ourselves with a binary classification between Lymphocyte and Neutrophil. Since the images contain other cells in the background and since we are using a linear method, we made the choice of simplicity. To the human eye, the distinction between Lymphocytes and Neutrophils is the simplest since the former look "uni-nuclear" whereas the latter looks "poly-nuclear" (see **Figure 1**) , so we thought this configuration was the most likely to work.

We have indeed a problem regarding the focus of the images, since the cell of interest is never at the same position in the images so we have to do a little pre-processing. We thus cropped the images around the centre of cell of interest. In order to do so, we used the image processing library **opencv**. We applied the following procedure using this library to find the center (the details can be found in our image processing notebook):

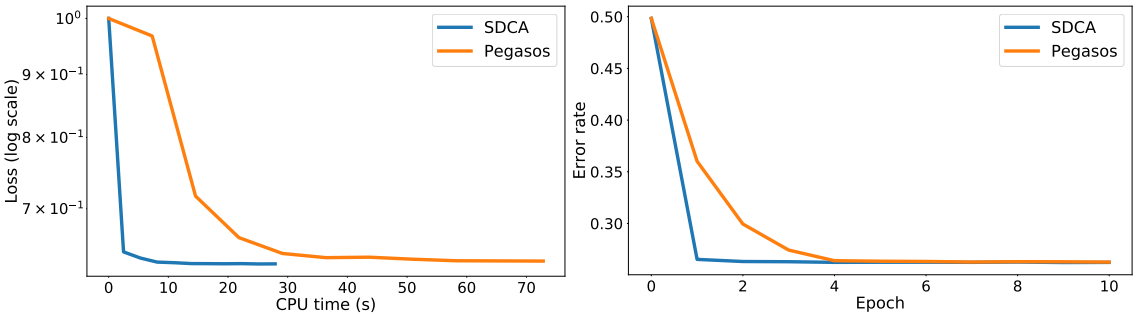
1. Blur the original image to make the main contours clearer.
2. Convert the blurred image to grayscale.
3. Neutralize the black pixels generated by framing issues by given them the value of the average pixel of the rest of the image.
4. Apply thresholding so that everything becomes white except the nucleus of the cell of interest.
5. Apply negative morphology operation to remove the remaining "pepper noise".
6. Find the center of cell by averaging the coordinates of the non white pixels and crop images around that center.

The previous procedure applied to an example can be seen in **Figure 3**. Examples of automatically cropped images can be seen in **Figure 4**. We take the negative of the images so that the pixels of interest (the nucleus) are white and thus have the higher numerical values. From now on we used the transformed dataset containaing the cropped version of all the images, in grayscale and in negative.

3.2 SDCA vs Pegasos directly on the cropped images

We flatten the images in order to work on vectors instead of matrices, our cropped images are 120×120 pixels thus we are working with vectors in $\mathbb{R}^{120\times120} = \mathbb{R}^{14400}$. Since we have only $n = 4982$ distributed approximately evenly between our two classes, that is a lot of features to learn. We plot some of the quantities of interest in **Figure 2**

We set $\lambda = 1.7$ after a few tries (we did not perform cross-validation).



(a) Loss (log scale) as a function of CPU time on train set(b) Error rate on test set as a function of number of epochs

Figure 2: SDCA vs Pegasos - cropped vectorized images - averaged over 100 runs of each

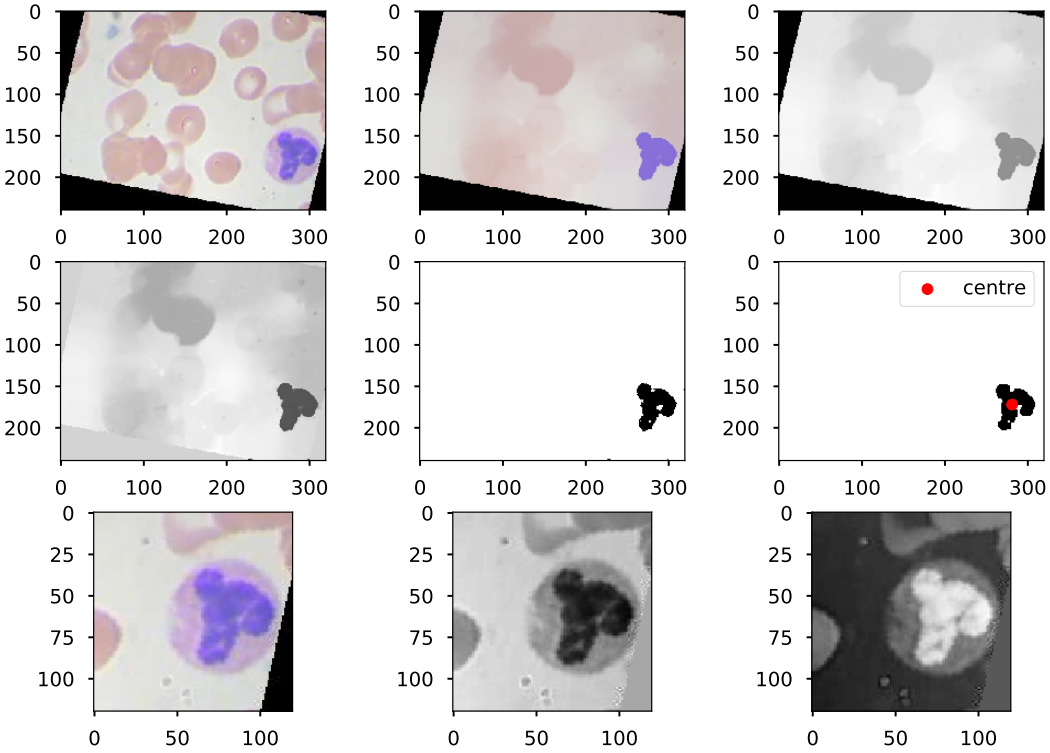


Figure 3: Automatic cropping procedure

SDCA with modified SGD initialization is actually a lot faster than Pegasos, after only the 2nd epoch, SDCA has almost converged whereas Pegasos is still struggling. In term of CPU execution time, we timed only the updates for each epoch (meaning that all the extra computations coming from tracking losses, tracking score on test set etc... do not count in the CPU timer).

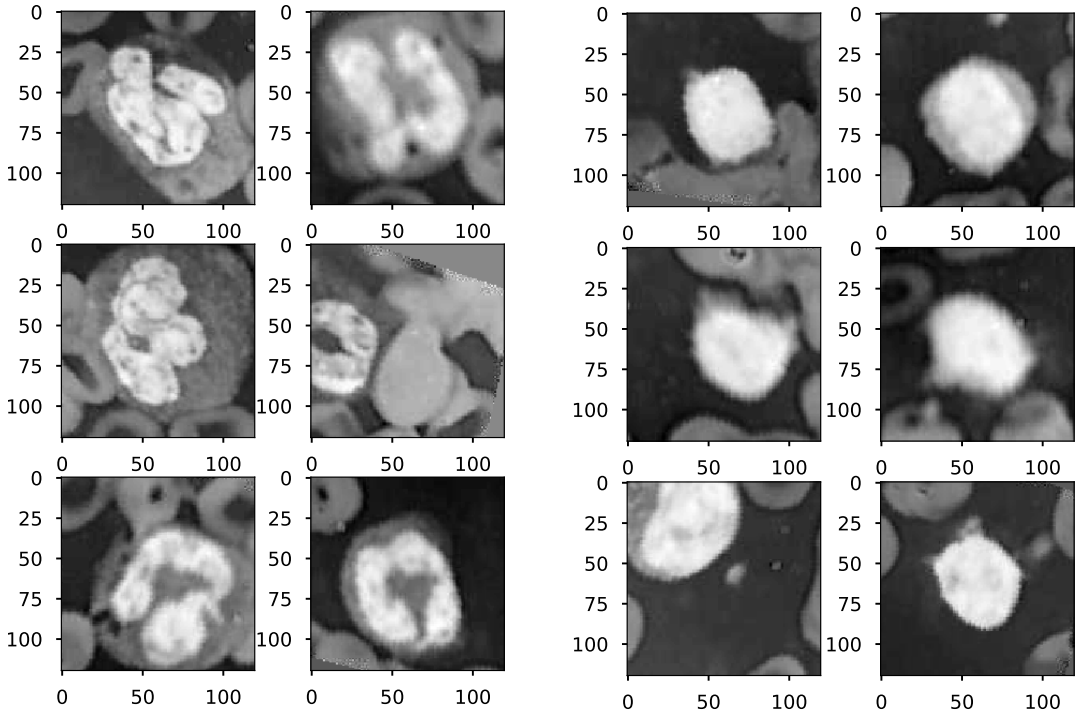
We deliberately set the duality gap stopping criterion too low for SDCA in order to have the same number of epochs for the two algorithms but if we set it to $\epsilon = 0.001$, SDCA stops early around the 4th epoch.

References

[1] *Blood cells images datasest.* https://www.kaggle.com/paultimothymooney/blood-cells.

[2] Shai Shalev-Swhartz and Tong Zhang. "Stochastic Dual Coordinate Ascent Methods for Regularized Loss Minimization". In: *Journal of machine learning research* (2013).

[3] Shai Shalev-Swhartz et al. "Pegasos : Primal Estimated Gradient Solver for SVM." In: *Springer - Journal of mathematical programming* (2011).



(a) Cropped neutrophils

(b) Cropped lymphocytes

Figure 4: Automatically cropped images for both classes