Distributed and Stochastic Optimisation for Machine Learning

Sophie OUILLADE : sophie.ouillade@ensae-paristech.fr
Satyanarayanan VENGATHESA SARMA : satyanarayanan.vengathesa.sarma@ensae-paristech.fr

Friday May, 5th 2017

Project n°1

Implement the SDCA algorithm to estimate Support Vector Machines. Test the algorithm on databases of your choice and compare it with a sub-gradient descent approach such as PEGASOS.

1 Introduction

The goal of this project is to estimate Support Vector Machines (SVM) from scratch. This algorithm was first described in [1] (see the References section).

As a quick reminder, the problem of SVM in binary classification consists in finding the hyperplane that separates the data with the largest margin. A vector $x \in \mathbb{R}^d$ is classified by looking at the sign of a linear scoring function $\langle w, x \rangle$. The goal of learning is to estimate the parameter $w \in \mathbb{R}^d$ in such a way that the score is positive if the vector x belongs to the positive class and negative otherwise.

The parameter w is estimated by fitting the scoring function to a training set of n example pairs $(x_i, y_i)_{i=1,\dots,n}$. Here $y_i \in \{-1, 1\}$ are the ground truth labels of the corresponding example vectors x_i. The goodness of fit is measured by a loss function which, in standard SVMs, is the Hinge loss :

$$l_i(\langle w, x \rangle) = \max\{0,\ 1 - y_i \langle w, x \rangle\}$$

Note that the Hinge loss is zero only if the score $\langle w, x \rangle$ is at least 1 or at most -1, depending on the label y_i. Moreover, the Hinge loss is 1-Lipschitz, but not smooth.

Fitting the training data is usually insufficient. In order for the scoring function to generalize to future data as well, it is usually preferable to trade off the fitting accuracy with the regularity of the learned scoring function $\langle w, x \rangle$. Regularity in the standard formulation is measured by the norm of the parameter vector $\|w\|^2$. Averaging the loss on all training samples and adding to it the regularizer weighed by a parameter λ yields the regularized loss objective :

$$P(w) = \frac{\lambda}{2}\|w\|^2 + \frac{1}{n}\sum_{i=1}^{n} \max\{0, 1 - y_i \langle w, .x \rangle\}$$

Note that this objective function is convex, so that there exists a single global optimum. The primal problem is thus : $\min_w P(w)$.

Learning an SVM amounts to finding the minimizer w^* of the cost function $P(w)$. One classic way to do so is to use the SGD (Stochastic Gradient Descent) method. However in this project we will consider two other solutions :

- The SDCA algorithm : Stochastic Dual Coordinate Ascent
- A modified version of SGD, based on sub-gradients : PEGASOS.

2 Stochastic Dual Coordinate Ascent (SDCA)

2.1 Dual SVM objective

As we have seen it in the lectures, SDCA is based on maximizing the dual SVM objective. To obtain the dual objective, one starts by approximating each loss term (l_i) by its dual conjugate (l_i^*). For the Hinge loss, with a slight change of notation compared to the above formula, we have :

$$l_i(z) = \max\{0,\ 1 - y_i z\} \quad l_i^*(u) = \begin{cases} y_i u & \text{if } -1 \leq y_i u \leq 0 \\ +\infty & \text{otherwise} \end{cases}$$

Since each plane $-z\alpha_i - l_i^*(-\alpha_i) \leq l_i(z)$ bounds the loss from below, by substituting in $P(w)$ one can write a lower bound for the SVM objective for each setting of the dual variables α_i :

$$F(w,\alpha) = \frac{\lambda}{2}\|w\|^2 - \frac{1}{n}\sum_{i=1}^{n}(w^\top x_i\alpha_i + l_i^*(-\alpha_i)) \leq P(w)$$

The dual objective function $D(\alpha)$ is obtained by minimizing the lower bound $F(w,\alpha)$ w.r.t. to w :

$$D(\alpha) = \inf_w F(w,\alpha) \leq P(w)$$

The minimizer and the dual objective are the following :

$$\boxed{w(\alpha) = \frac{1}{\lambda n}\sum_{i=1}^{n} x_i\alpha_i = \frac{1}{\lambda n}X\alpha \quad D(\alpha) = -\frac{1}{2\lambda n^2}\alpha^\top X^\top X\alpha + \frac{1}{n}\sum_{i=1}^{n} -l_i^*(-\alpha_i)}$$

where $X = [x_1, ..., x_n]$ is the data matrix.

Since the dual is uniformly smaller than the primal, one has the duality gap bound :

$$D(\alpha) \leq P(w^*) \leq P(w(\alpha))$$

This bound can be used to evaluate how far off $w(\alpha)$ is from the primal minimizer w^*. In fact, due to convexity, this bound can be shown to be zero when α^* is the dual maximizer (strong duality) :

$$D(\alpha^*) = P(w^*) = P(w(\alpha^*)), \quad w^* = w(\alpha^*)$$

The dual problem is thus : $\max_\alpha D(\alpha)$.

At each iteration of DCA, the dual objective is optimized with respect to a single dual variable, while the rest of the dual variables are kept intact. We focus on a stochastic version of DCA (SDCA) in which at each round we choose which dual coordinate to optimize uniformly at random.

2.2 Algorithm

The algorithm SDCA is described in [4]. We reproduced the corresponding pseudo-code below (see Algorithm 1). In the pseudo-code, the parameter T indicates the number of iterations while the parameter T_0 can be chosen to be a number between 1 to T. The original authors suggest that $T_0 = T/2$ is a good choice. One can also evaluate the duality gap and terminate the algorithm when it is sufficiently small.

In order to calculate $\Delta\alpha_q$ in the algorithm, we can expand the updated dual objective as :

$$\begin{aligned}
D(\alpha^{(t)} + \Delta\alpha_q e_q) &= -\frac{1}{2\lambda n^2}(\alpha^{(t)} + \Delta\alpha_q e_q)^\top X^\top X(\alpha^{(t)} + \Delta\alpha_q e_q) + \frac{1}{n}\sum_{i=1}^{n} -l_i^*(-(\alpha^{(t)} + \Delta\alpha_q e_q)_i) \\
&= -\frac{1}{2\lambda n^2}\left[{\alpha^{(t)}}^\top X^\top X\alpha^{(t)} + {\alpha^{(t)}}^\top X^\top X\Delta\alpha_q e_q + (\Delta\alpha_q e_q)^\top X^\top X\alpha^{(t)} + (\Delta\alpha_q e_q)^\top X^\top X\Delta\alpha_q e_q)\right] \\
&\quad + \frac{1}{n}\left(-l_q^*(-(\alpha_q^{(t)} + \Delta\alpha_q)) + \sum_{i=1, i\neq q}^{n} -l_i^*(-\alpha^{(t)})\right) \\
&= \text{Constant} - \frac{1}{2\lambda n^2}x_q^\top x_q(\Delta\alpha_q)^2 - \frac{1}{\lambda n^2}x_q^\top X\alpha^{(t)}\Delta\alpha_q - \frac{1}{n}l_q^*(-\alpha_q^{(t)} - \Delta\alpha_q) \\
&\propto \frac{1}{n}\left[-\frac{A}{2}(\Delta\alpha_q)^2 - B\Delta\alpha_q - l_q^*(-\alpha_q^{(t)} - \Delta\alpha_q)\right]
\end{aligned}$$

Algorithm 1 Procedure SDCA

Input $\alpha^{(0)} = 0$
Let $w^{(0)} = w(\alpha^{(0)})$
for $t = 0, 1, 2, ..., T$ (or until the duality gap $P(w(\alpha^{(t)})) - D(\alpha^{(t)}) \leq \epsilon$) **do**
  Pick a dual variable q uniformly at random in $1, ..., n$
  Maximize the dual with respect to this variable : $\Delta\alpha_q = \arg\max_{\Delta\alpha_q} D(\alpha^{(t)} + \Delta\alpha_q e_q)$
  Update : $\alpha^{(t+1)} = \alpha^{(t)} + e_q \Delta\alpha_q$ et $w^{(t+1)} = w^{(t)} + \frac{1}{\lambda n}\Delta\alpha_q x_q$
end for
Output (Averaging option)
  Let $\bar{\alpha} = \frac{1}{T-T_0}\sum_{i=T_0}^{T} \alpha^{(t)}$
  Let $\bar{w} = w(\bar{\alpha}) = \frac{1}{T-T_0}\sum_{i=T_0}^{T} w^{(t)}$
  Return $\bar{w}$
Output (Random option)
  Let $\bar{\alpha} = \alpha^{(t)}$ and $\bar{w} = w^{(t)}$ for some random $t \in T_0, ..., T$
  Return $\bar{w}$

with :

$$A = \frac{1}{\lambda n} x_q^\top x_q = \frac{1}{\lambda n}\|x_q\|^2 \quad \text{and} \quad B = x_q^\top \frac{X\alpha^{(t)}}{\lambda n} = x_q^\top w^{(t)}, \quad w^{(t)} = w(\alpha^{(t)})$$

In the case where $l()$ is the hinge loss, this quantity can be maximized.
Thus, by setting the derivative of the objective to zero, one gets the update :

$$\frac{\partial D(\alpha^{(t)} + \Delta\alpha_q e_q)}{\partial \Delta\alpha_q} = 0 \Leftrightarrow -A\Delta\alpha_q - B + y_q = 0 \Leftrightarrow \tilde{\Delta\alpha_q} = \frac{y_q - B}{A}$$

Finally, since we necessarily have $-1 \leq -y_q(\alpha_q^{(t)} + \Delta\alpha_q) \leq 0$, i.e. $0 \leq y_q(\alpha_q^{(t)} + \Delta\alpha_q) \leq 1$, the update becomes :

$$\boxed{\Delta\alpha_q = y_q \max\{0, \min\{1, y_q(\alpha_q^{(t)} + \tilde{\Delta\alpha_q})\}\} - \alpha_q^{(t)}}$$

2.3 Runtime of optimization procedures

Let w^* be the optimum of the primal problem. A solution w is said to be ϵ_P-sub-optimal if $P(w) - P(w^*) \leq \epsilon_P$.
Paper [4] analyzes the runtime of optimization procedures as a function of the time required to find an ϵ_P-sub-optimal solution.

For solving SVM, SGD finds an ϵ_P-sub-optimal solution in time $O(\frac{1}{\lambda\epsilon_P})$. However, SGD reaches a moderate accuracy quite fast but its convergence becomes rather slow for more accurate solutions.
In the dual problem, the duality gap defined as : $P(w(\alpha)) - D(\alpha)$ can be regarded as an upper bound of the primal sub-optimality $P(w(\alpha)) - P(w^*)$. The paper [4] shows that in order to achieve a duality gap of ϵ for L-Lipschitz loss functions, SDCA yields a rate of $O(n + \frac{L^2}{\lambda\epsilon})$. For functions which are almost everywhere smooth (such as the Hinge loss) this result can be further improved.

For SVM, in order to obtain a primal ϵ_P-sub-optimal solution, we need a dual ϵ_D-sub-optimal solution with $\epsilon_D = O(\lambda\epsilon_P^2)$. Therefore, a convergence result for dual solution can only translate into a primal convergence result with worse convergence rate.

2.4 Practical aspects

Paper [4] suggests that, to obtain a duality gap of $\mathbb{E}[P(\bar{w}) - D(\bar{\alpha})] \leq \epsilon_P$ with hinge-loss SVM and SDCA, it is enough to have a total number of iterations of :

$$T \geq T_0 + n + \frac{L^2}{\lambda \epsilon_P} \geq \max\{0, \lceil n \log(0.5\lambda n L - 2) \rceil\} + n + \frac{5L^2}{\lambda \epsilon_P}$$

Moreover, when $t \geq T_0$, we have dual sub-optimality bound of $\mathbb{E}[D(\alpha^*) - D(\alpha(t))] \leq \frac{\epsilon_P}{2}$.

In practice, instead of complete randomization, one should pass through the data multiple times using epochs. In our implementation, we made it possible for each epoch to display a random permutation of the data. This procedure is called SDCA-Perm and is described in [4] as well. In the experimentation part (see the `Python` noteboook in appendix) we compare three possible loops : random, cyclic and permutation. The results are presented in section 4.3.

2.5 Variant : Accelerated Mini-Batch Stochastic Dual Coordinate Ascent (ASDCA)

The paper [3] describes another algorithm called ASDCA, which interpolates SDCA and AGD (Accelerated Gradient Descent), a method based on Nesterov's acceleration principle that we studied during the lectures (see article [2]). Plus, ASDCA uses mini-batches, as opposed to vanilla SDCA. At each iteration of the algorithm, a subset of m indices from $\{1, ..., n\}$ is picked randomly and the dual vectors corresponding to this subset are updated. The algorithm is described below in pseudo-code :

Algorithm 2 Procedure Accelerated Mini-Batch SDCA

Input $\lambda, \gamma, \theta \in [0, 1], m$
Initialize : $\alpha_1^{(0)} = ... = \alpha_n^{(0)} = \bar{\alpha}^{(0)} = 0,\ w^{(0)} = 0$
for $t = 0, 1, 2, ..., T$ **do**
  $u^{(t-1)} = (1 - \theta)x^{(t-1)} + \theta \nabla g^*(\bar{\alpha}^{(t-1)})$
  Randomly pick subset $I \subseteq \{1, ..., n\}$ of size m and update the dual variables in I :
    $\alpha_i^{(t)} = (1 - \theta)\alpha_i^{(t-1)} - \theta \nabla \phi_i(u^{(t-1)})$ for $i \in I$
    $\alpha_j^{(t)} = \alpha_j^{(t-1)}$ for $j \notin I$
  $\bar{\alpha}^{(t)} = \bar{\alpha}^{(t-1)} + \frac{1}{n} \sum_{i \in I} (\alpha_i^{(t)} - \alpha_i^{(t-1)})$
  $w^{(t)} = (1 - \theta)w^{(t-1)} + \theta \nabla g^*(\bar{\alpha}^{(t)})$
end for

with $g(x) = \frac{\lambda}{2}\|x\|^2$. Thus we have : $g^*(\alpha) = \frac{1}{2\lambda}\|\alpha\|^2$ and $\nabla g^*(\alpha) = \frac{1}{\lambda}\alpha$.

For this algorithm, we implemented the optimization procedure using the smooth version of the hinge loss :

$$\phi_i(x) = \begin{cases} 0 & \text{if } y_i x^\top w_i > 1 \\ \frac{1}{2} - y_i x^\top w_i & \text{if } y_i x^\top w_i < 0 \\ \frac{1}{2}(1 - y_i x^\top w_i)^2 & \text{otherwise} \end{cases}$$

but in order to make comparisons with the other algorithms, which use the regular hinge loss, we will plot the value obtained with this usual hinge loss in the primal objective.

This algorithm could also be very useful for it allows parallelization (which we did not try to implement).

3 Primal Estimates sub-GrAdient SOlver for SVM (PEGASOS)

PEGASOS is a stochastic sub-gradient descent algorithm, used for solving the primal problem of SVM. The method is described in [3]. In the basic version, 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. The paper [3] shows that with high probability over the choice of the random examples, this algorithm finds an ϵ-accurate solution using only $O(\frac{1}{\lambda\epsilon})$ iterations, while each iteration involves a single inner product between w and x. So the overall runtime required to obtain an ϵ-accurate solution is $O(\frac{n}{\lambda\epsilon})$ where n is the dimensionality of w and x.

3.1 Algorithm

PEGASOS comes down to performing a stochastic sub-gradient descent on the primal objective with a carefully chosen step-size.

Algorithm 3 Procedure PEGASOS

Input S, λ, T
Initialize : Set $w_1 = 0$
for t=1, 2, ... T **do**
  Choose $i_t \in \{1, \ldots, |S|\}$ uniformly at random
  Set $\eta_t = \frac{1}{\lambda t}$
  if $y_{i_t}\langle w_t, x_{i_t}\rangle < 1$ **then**
    Set $w_{t+1} \leftarrow (1 - \eta_t\lambda)w_t + \eta_t y_{i_t} x_{i_t}$
  else
    (Case : $y_{i_t}\langle w_t, x_{i_t}\rangle \geq 1$)
    Set $w_{t+1} \leftarrow (1 - \eta_t\lambda)w_t$
  end if
  [Projection step (optional) : $w_{t+1} \leftarrow \min\left\{1, \frac{1/\sqrt{\lambda}}{\|w_{t+1}\|}\right\} w_{t+1}$]
end for
Return w_{T+1}

At each iteration, the primal objective is replaced with an approximation based on the training example chosen at random, that is :

$$f(w; i_t) = \frac{\lambda}{2}\|w\|^2 + l(w; (x_{i_t}, y_{i_t}))$$

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

$$\nabla_t = \lambda w_t - \mathbb{I}_{(y_{i_t}\langle w_t, x_{i_t}\rangle < 1)} y_{i_t} x_{i_t}$$

We then update w_{t+1} with ∇_t using a step size of $\eta_t = \frac{1}{\lambda t}$: $w_{t+1} \leftarrow w_t - \eta_t \nabla_t$.

A potential variation of PEGASOS is the gradient-projection approach where the set of admissible solutions is limited to the ball of radius $\frac{1}{\sqrt{\lambda}}$. To enforce this property, w_t is projected after each iteration onto this sphere, by performing the update described in the "Projection" section of the algorithm.

3.2 Variant : Mini-Batch PEGASOS

A more general algorithm that uses k examples at each iteration (*Mini-Batch iterations*) with $1 \leq k \leq m$ is described below (Algorithm 4). When $k = m$, $A_t = S$ on each round t thus we obtain the deterministic

sub-gradient descent method. In the other extreme case, when $k = 1$, we recover the stochastic sub-gradient algorithm (Algorithm 3).

Algorithms belonging to the mini-batch SGD family are widely used in many learning problems (especially deep learning), for they often display fast convergence and good results. We will later see whether or not the three previous algorithms achieve a faster convergence that PEGASOS for SVM.

Algorithm 4 Procedure Mini-Batch PEGASOS

Input S, λ, T, k
Initialize : Set $w_1 = 0$
for t=1, 2, ... T **do**
  Choose $A_t \subseteq \{1, ..., m\}$, where $|A_t| = k$ uniformly at random
  Set $A_t^+ = \{i \in A_t : y_i \langle w_t, x_i \rangle < 1\}$
  Set $\eta_t = \frac{1}{\lambda t}$
  Set $w_{t+1} \leftarrow (1 - \eta_t \lambda) w_t + \frac{\eta_t}{k} \sum_{i \in A_t^+} y_i x_i$
  [Projection step (optional) : $w_{t+1} \leftarrow \min \left\{ 1, \frac{1/\sqrt{\lambda}}{\|w_{t+1}\|} \right\} w_{t+1}$]
end for
Return w_{T+1}

3.3 Convergence

The paper [3] proves the following result about the convergence of Mini-Batch PEGASOS algorithm. Assume that for all $(x, y) \in S$ the norm of x is at most R, let $w^* = \arg\min_w f(w)$ and let $c = (\sqrt{\lambda} + R)^2$ if the projection step is performed or $c = 4R^2$ otherwise, let $T \geq 3$, assume 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 $\lambda \leq \frac{1}{4}$, then if t is selected at random from $\{1, ..., T\}$ we have with a probability of at least $\frac{1}{2}$ that :

$$f(w_t) \leq f(w^*) + \frac{42c \ln(T/\delta)}{\lambda T}$$

So, 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. on average, after two attempts we are likely to find a good solution.

4 Implementation and experiments

We implemented all of the previous algorithms and some of their variations in the `Python` notebook corresponding to this project. Below is the list of those algorithms all the options corresponding to implemented variants :

- SDCA : `SDCA_SVM` (X, y, n_samples, T_0, lambd, nb_epochs, average, keep_full_primal_history, iteration)
- PEGASOS : `PEGASOS_SVM`(X, y, n_samples, T_0, lambd, nb_epochs, average, keep_full_primal_history, proj)
- Mini-batch ASDCA : `SDCA_SVM_mini_batch`(X, y, n_samples, T_0, lambd, nb_epochs, average, keep_full_primal_history, batch_size, theta)

- Mini-batch PEGASOS : `PEGASOS_SVM_mini_batch`(X, y, n_samples, T_0, lambd, nb_epochs, average, keep_full_primal_history, batch_size, proj)

Some parameters are shared by all methods. X and y respectively reference the features matrix and the labels on which the model is trained. *n_samples* corresponds to the number of training instances. The parameter *T_0* represent the limit used for outputing the average. *lambd* corresponds to the regularization parameter in the primal objective. A common default value is $\frac{1}{n\_samples}$. *nb_epochs* is the number of time the algorithm passes throughout the whole dataset. The parameter *average* indicates whether the output is the last iteration or the average of the last ones. The parameter *keep_full_primal_history* was created in order to keep only a part of the history if the dataset is too big (otherwise, computations take significantly longer).

Other parameters are specific to each algorithm. The parameter *iteration* can take 3 values ('random', 'cyclic', 'permutation') depending on the way the iteration is realized. The parameter *proj* indicates whether a projection has to be done on w or not. Finally, the parameter *batch_size* specifies the size of each mini-batch for ASDCA and mini-batch PEGASOS.

4.1 Datasets

For each dataset, we considered the problem of binary classification using linear SVM. The datasets we used are the following :

Table 1 – Data sets description

Dataset	Size	Features
Simulated	1 000	10
Genomic	184	4 654
Skin	245 057	3

The simulated data was used in order to have a general idea of each algorithm's performance. We then chose to test the methods on a high dimensional database (Genomic) i.e. with a very large number of features and very few observations [1]. The goal was to see how high dimension affected the performances. Finally, we also tested all methods on the Skin segmentation dataset, which is commonly used in Machine Learning [2]. We were interested in this dataset for it displayed a large number of observations.

In the upcoming sections, we analyzed the results of all variations of the previous four algorithms on these three databases. For each of them, we report the convergence curve of the primal objective.

4.2 General comparison

First, we have analyzed the overall convergence of the primal objective between the four algorithms (vanilla versions for each one). A batch-size of 8 was used for the simulated and genomic datasets, while a batch size of 2 was used for the skin dataset, in order to speed up computations. The curves corresponding to the simulated data are displayed on Figure 1, those for the genomic data on Figure 2 and those for the skin data are on Figure 3.

Those experiments seem to show that the vanilla PEGASOS algorithm is the slowest of all four methods to reach convergence, no matter the type of data. Moreover, it appears that for the high dimensional genomic dataset, the mini-batch PEGASOS algorithm is the fastest to reach convergence. This is not the case for the

1. This dataset was originally provided by J.P.Vert for his course at ENSAE. It is freely available at the address `http://members.cbio.mines-paristech.fr/~jvert/svn/kernelcourse/course/2017ensae/index.html`

2. This dataset can be found on UCI's machine learning repository `https://archive.ics.uci.edu/ml/datasets/Skin+Segmentation/`

other two datasets, where SDCA and ASDCA are both faster than mini-batch PEGASOS. On the simulated dataset, ASDCA reaches convergence extremely fast, closely followed by SDCA.

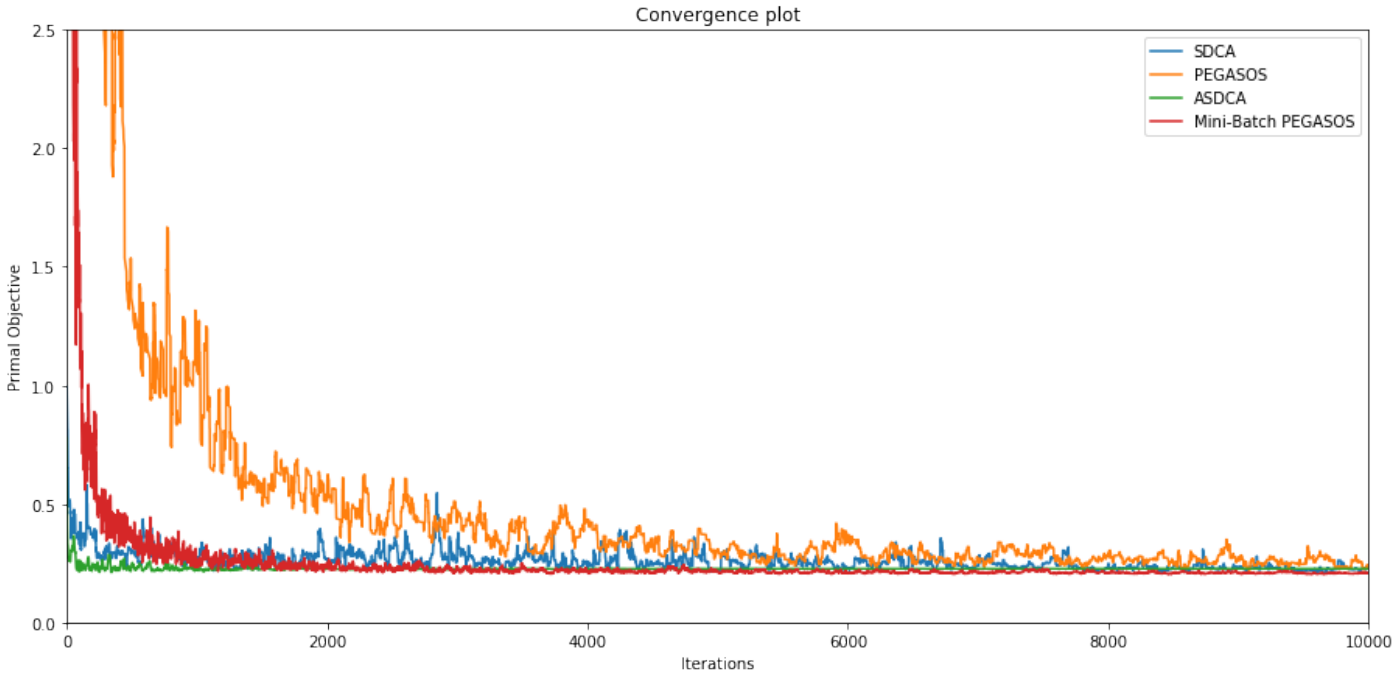


Figure 1 – Convergence comparison for simulated data

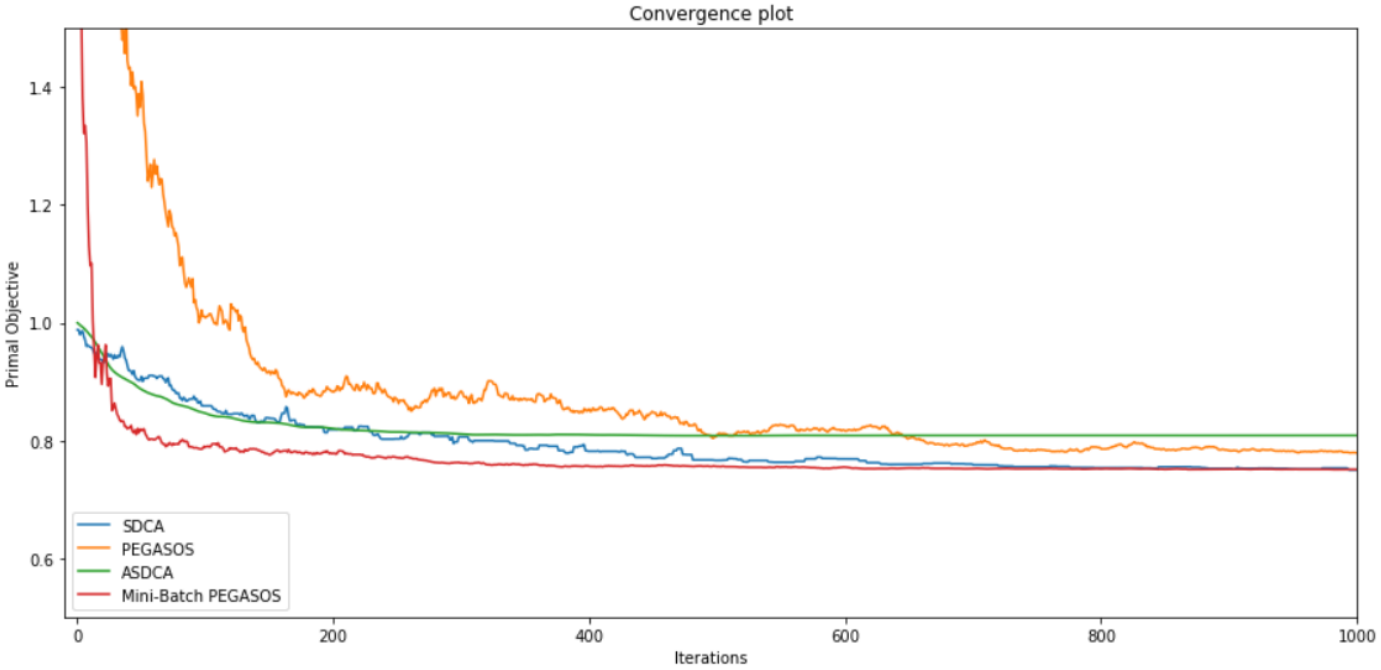


Figure 2 – Convergence comparison for genomic data

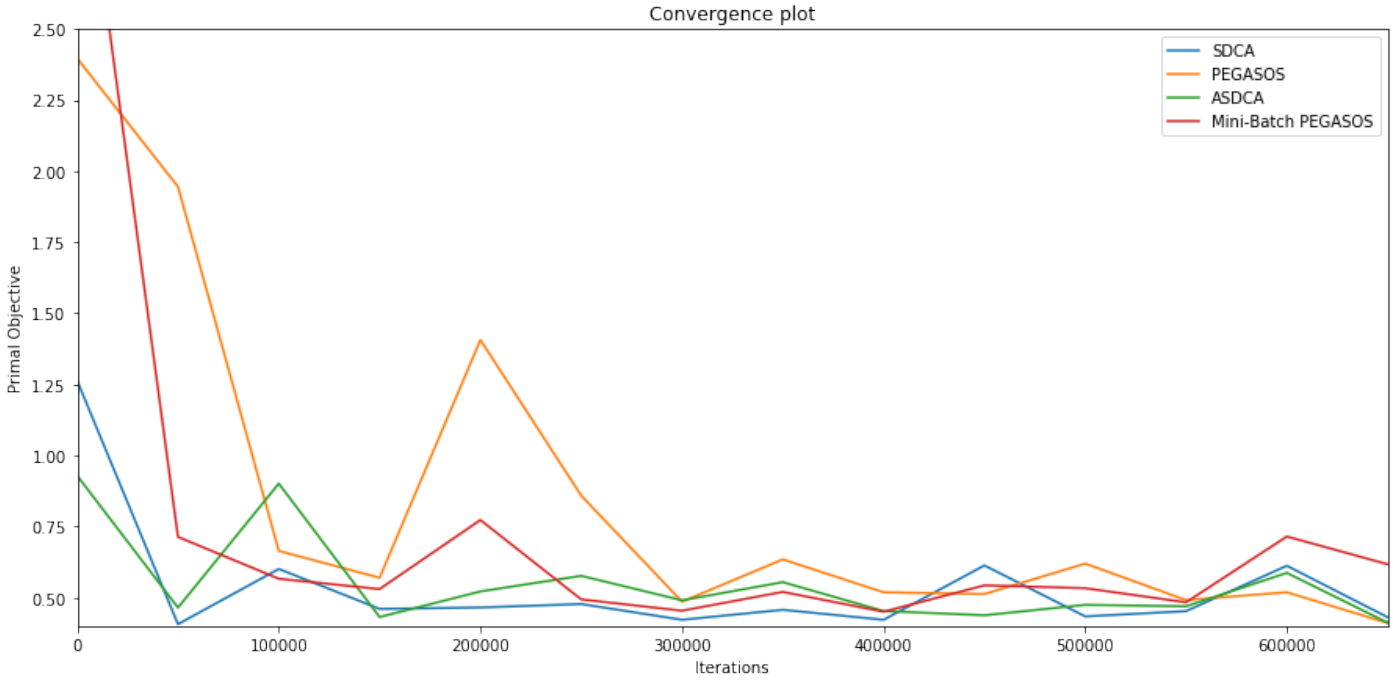


Figure 3 – Convergence comparison for skin data

On the skin datset, the comparison is made difficult since it was too costly (timewise) too compute and save the primal objective at each iteration. This is why we decided to do it only every 50 000 iterations. Thus, on this dataset, SDCA and ASDCA seem to converge towards the final value at the same rate. However, we only used a batches of size 2 for this dataset. By increasing the batch size, one could expect to obtain better performance in terms of convergence, but at the cost of a higher runtime.

These experiments show that it might be preferable to use the mini-batch Pegasos algorithm when training SVM on high-dimensional data. On the other hand, if one is training SVM on a classic dataset with a large number of observations and few features, the ASDCA algorithm should outperform all other methods.

Note that we did not report the runtime of the algorithms, because in all our implementations we compute and save the value of the primal objective (at each step for Simulated and Genomic, once every 50 000 iterations for Skin). This significantly increases the runtime of all procedures, thus it would not have been meaningful to report it.

4.3 Comparaison of SDCA's sampling procedures

In the SDCA algorithm, three different options exist for the sampling step : at random with repetitions, using random permutations at each epoch (this variant is called SDCA-Perm), or with respect to a fixed cyclic order (that was chosen once at random). To analyze the influence of all three possibilities, we have plotted on the same figure the resulting convergence plots, for the Genomic dataset (see Figure 4).

The paper [4] suggests that a cyclic order yields a much slower convergence rate compared to the other sampling methods and that SDCA-Perm is sometimes faster than SDCA. However this is not what we observe on the genomic dataset. Indeed, on Figure 4, we can see that even if they all converge to the same value and at almost the same rate, the random permutation seems to be the slowest. This is probably due to the fact that the Genomic dataset has very few observations. On the skin dataset, the results displayed on Figure 5 seem to confirm what was stated in [4] : the cyclic option results in a slower convergence rate, while the perm option seems to be converging as fast as vanilla SDCA, but with more variability. Hence, the random option seem to

be a good default choice for SDCA.

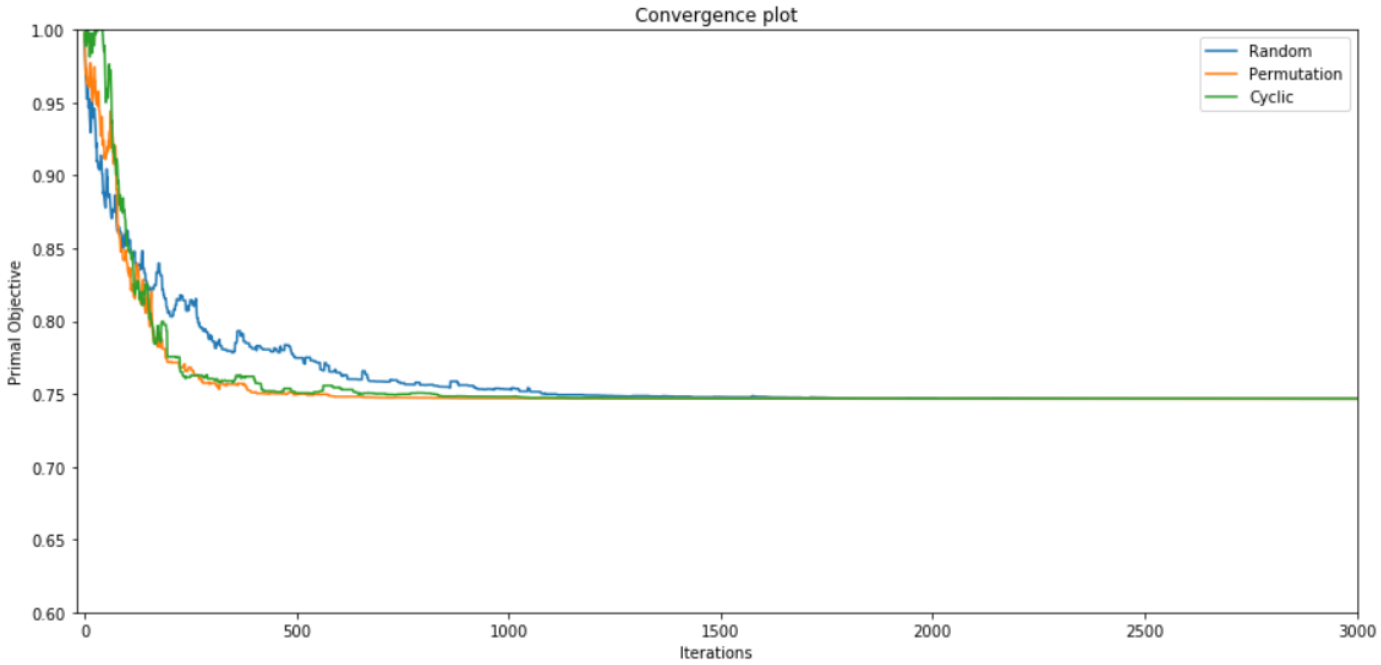


Figure 4 – Influence of the sampling procedure for SDCA (Genomic)

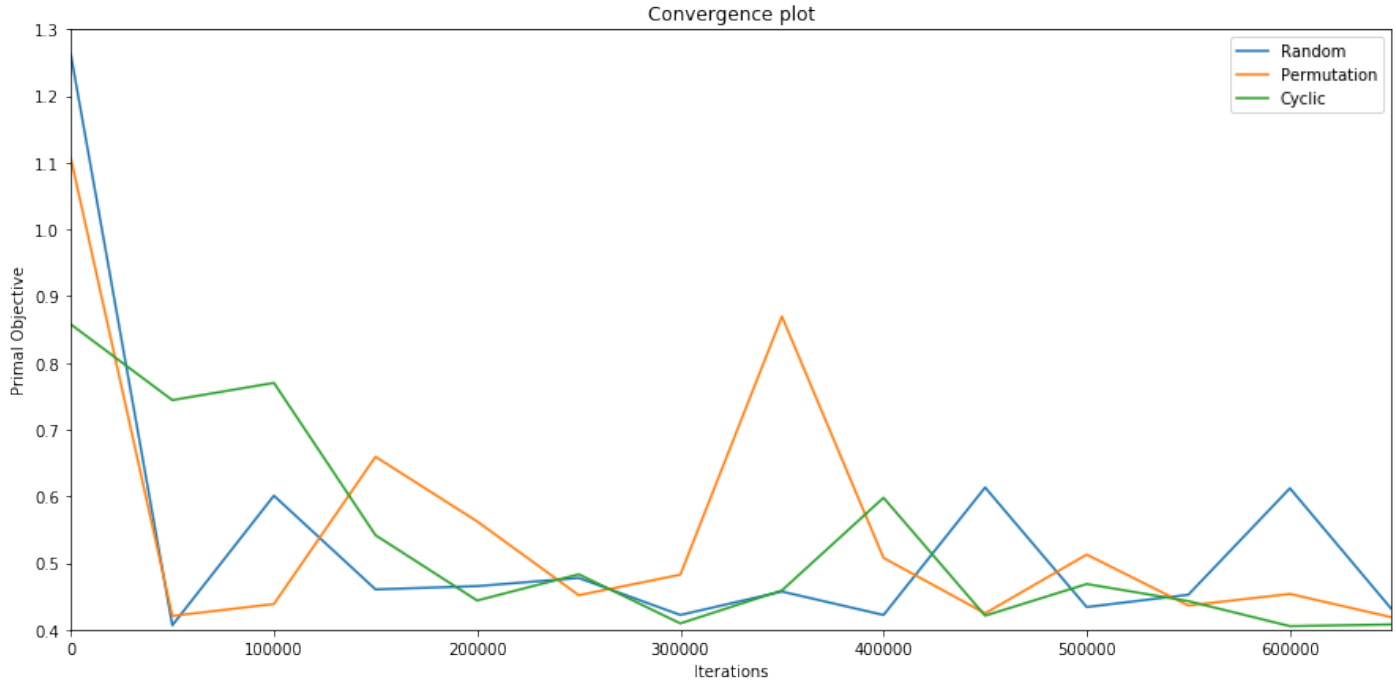


Figure 5 – Influence of the sampling procedure for SDCA (Skin)

4.4 Influence of averaging

Although only SDCA allowed an averaging option for the output, we decided to add this option for all the algorithms we have implemented. Hence, they can return either the average of last parameter vectors or only the final one. Note that this option does not influence the convergence of each algorithm, only their final output. Thus, we have compared those outputs on the simulated data we have created. Indeed, for this dataset, the true value of the parameter vector is known (parameter `beta` in the pseudo-code). After adding some gaussian noise to create corresponding labels, we compare the output of each method with the real `beta`.

We decided to have 10 parameters in the vector `beta`. Comparisons between the true value and our algorithms' estimations (with all possible options) are plotted on Figure 7 in the Appendix section. It appears that using the averaging option does not allow to get much closer to the real parameter vector, compared to not using it.

On top of comparing primal convergence, we also separated the simulated data in train and test sets in order to calculate the accuracy on the test set. All the results are presented in table 2. `beta` has an accuracy of 0.878. We can notice that all the accuracies obtained by the different datasets are around this value, and we can also notice that the accuracy obtained from the averaged output is of the same order than the others. Thus, the average option does not seem to generate an important difference for prediction either.

Table 2 – Accuracy on test set (Simulated)

Algorithm	Accuracy	Averaged accuracy
SDCA random	0.863	0.872
SDCA permutation	0.881	0.875
SDCA cyclic	0.872	0.875
PEGASOS	0.866	0.863
PEGASOS projected	0.860	0.869
ASDCA	0.869	0.875
Mini-batch PEGASOS	0.881	0.866
Mini-batch PEGASOS projected	0.872	0.878

4.5 Influence of the projection step in PEGASOS

As seen in section 3, the PEGASOS algorithm can have an optional projection step. To see the influence of this projection, we have plotted on the same figure the convergence plots of the algorithm with and without projection for the genomic data (see figure 6).

We can notice that both convergence curves are somehow similar : the primal objectives converge towards the same value at the same speed and are close to one another. Furthermore, Figure 7 in Appendix seems to indicated that the output parameter vector is very similar between the two alternatives. However, the projection step seems to make the convergence more smooth (less noisy) and as Table 2 shows, it improves the test accuracy when the averaged output is considered. Hence it should be preferable to use it.

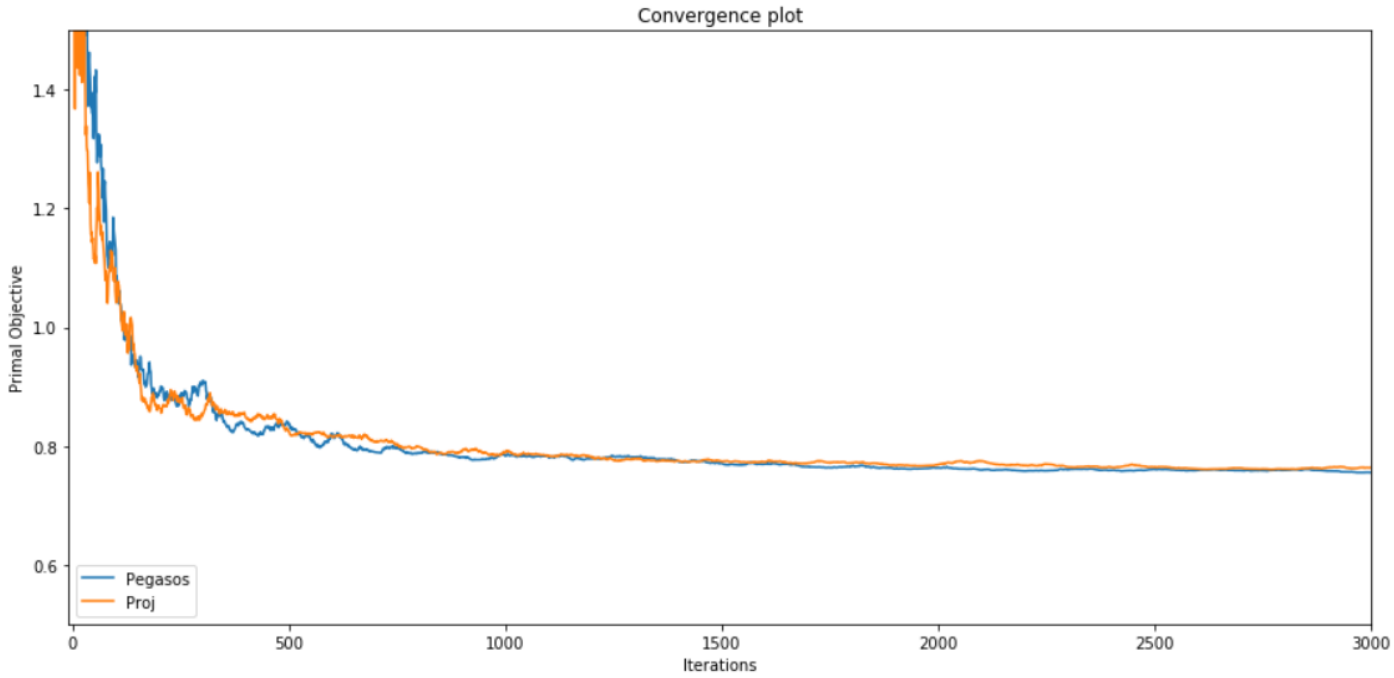


Figure 6 – Influence of the projection step (Genomic)

4.6 Influence of the regularisation parameter λ

Finally, we have tried different values for λ, in order to assess its effect on the convergence. To do so we have considered the genomic dataset, were there is a high chance of overfitting since there are only 184 observations for more than 4 600 features. Hence, we expect the value of λ to have a great impact on convergence for this dataset. Results are presented on Figures 8, 9, 10 and 11 in annex. Thus, according to our experiments, it seems like the higher λ, the faster convergence is reached. But at the same time, the final value of the primal objective increases with λ, which makes sense considering the formulation of $P(w)$ given in our introduction.

Conclusion

[4] stated that "While several experiments have shown that decomposition methods (such as DCA) are inferior to SGD for large scale SVM, [...] SDCA outperforms the SGD approach in some regimes."

Thanks to our experiments, we have shown that in the case of linear SVM, SDCA seems to always converge faster than PEGASOS, which is a variant of SGD. The averaging option for the output does not seem to affect the result in a significant way. However, the mini-batch variants of those algorithms, ASDCA and Mini-Batch PEGASOS seem to yield better results overall. ASDCA appears to be performing very well when the number of samples is bigger than the number of features. However, in the other case, when the number of features is bigger than the number of samples, the Mini-Batch PEGASOS algorithm seems to be more efficient.

Finally, as said previously, for those two variants, we used a batch size of 2 for the skin dataset and a batch size of 8 for the simulated and the genomic datasets. Altough we did not analyze the impact of this parameter, it surely has a significant effect on the convergence, and hence it should be tuned for each dataset.

References

[1] Corinna C. and Vapnik V. "Support-vector networks" *Machine learning 20.3* (1995).

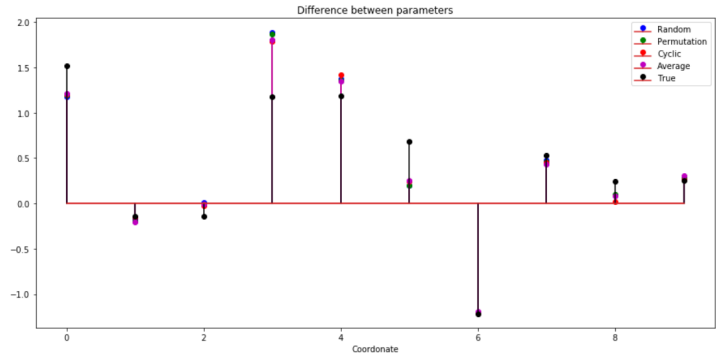
[2] Nesterov Y. "Gradient methods for minimizing composite objective function." (2007).

[3] Shalev-Shwartz S., Singer Y., Srebro N. and Cotter A. "PEGASOS : Primal Estimates sub-GrAdient SOlver for SVM." *Proceedings of the 24th international conference on Machine learning* (2007).

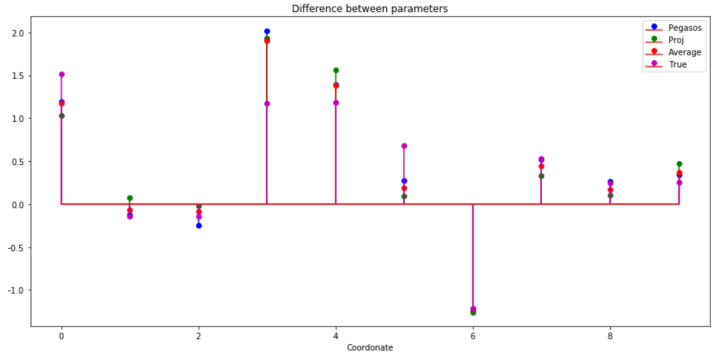
[4] Shalev-Shwartz S. and Zhang T. "Stochastic Dual Coordinate Ascent Methods for Regularized Loss Minimization." *Journal of Machine Learning Research 14* (2013).

[5] Shalev-Shwartz S. and Zhang T. "Accelerated Mini-Batch Stochastic Dual Coordinate Ascent." *Advances in Neural Information Processing Systems* (2013).

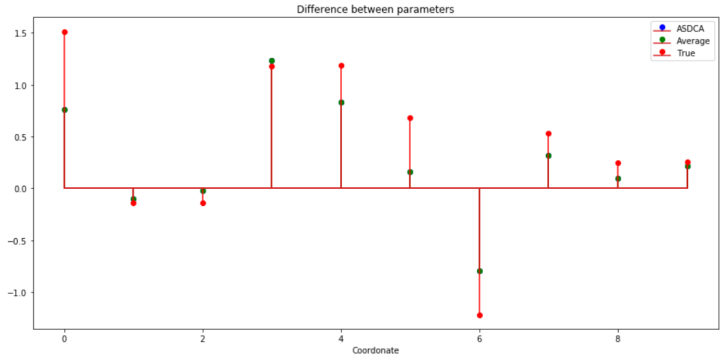
Appendix



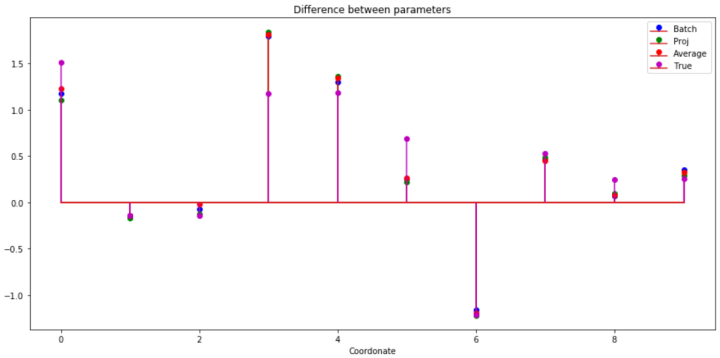
(a) SDCA



(b) PEGASOS



(c) ASDCA



(d) Mini-Batch PEGASOS

Figure 7 – Influence of all options (Simulated)

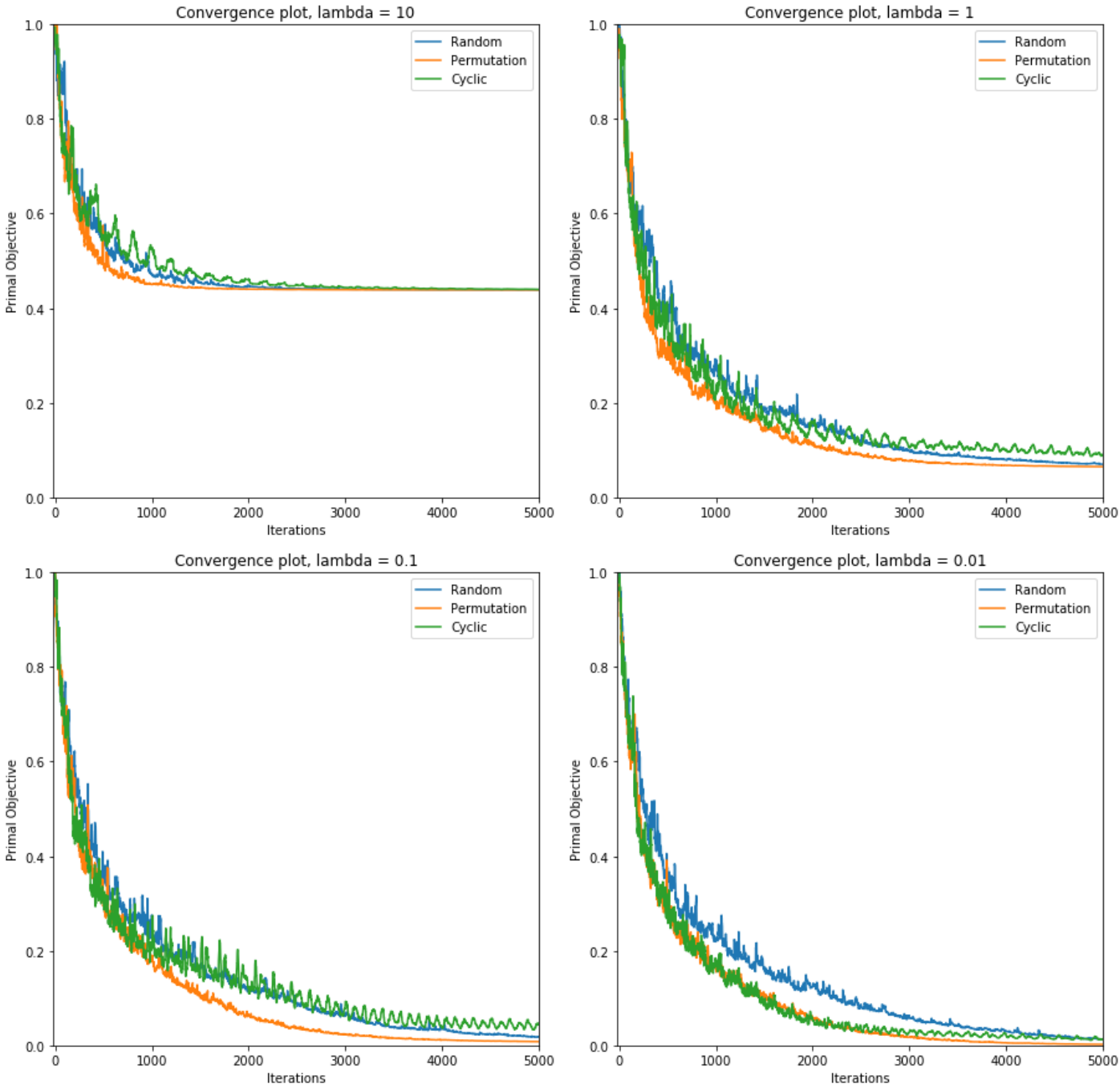


Figure 8 – Influence of λ on SDCA (Genomic)

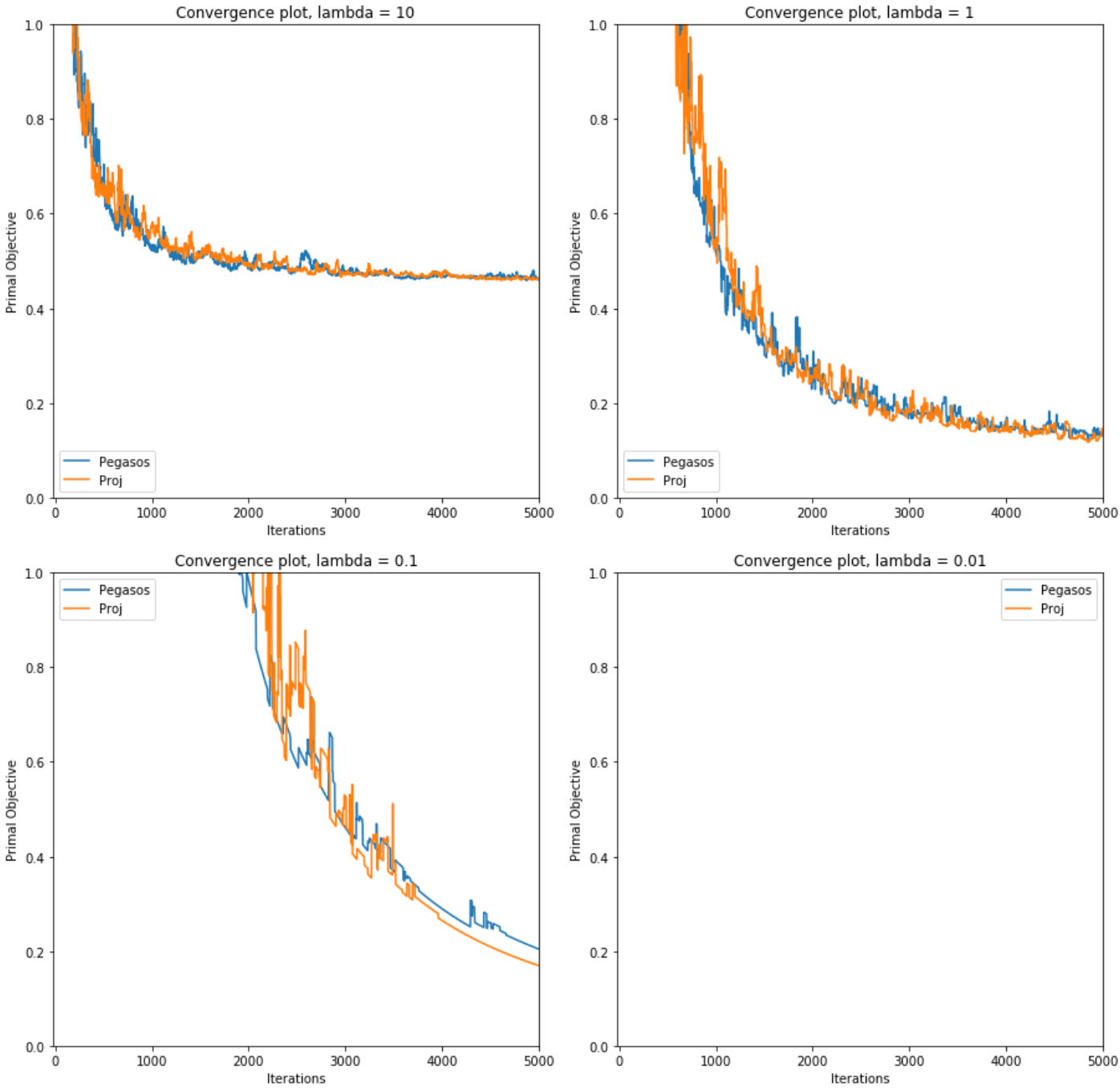


Figure 9 – Influence of λ on PEGASOS (Genomic)

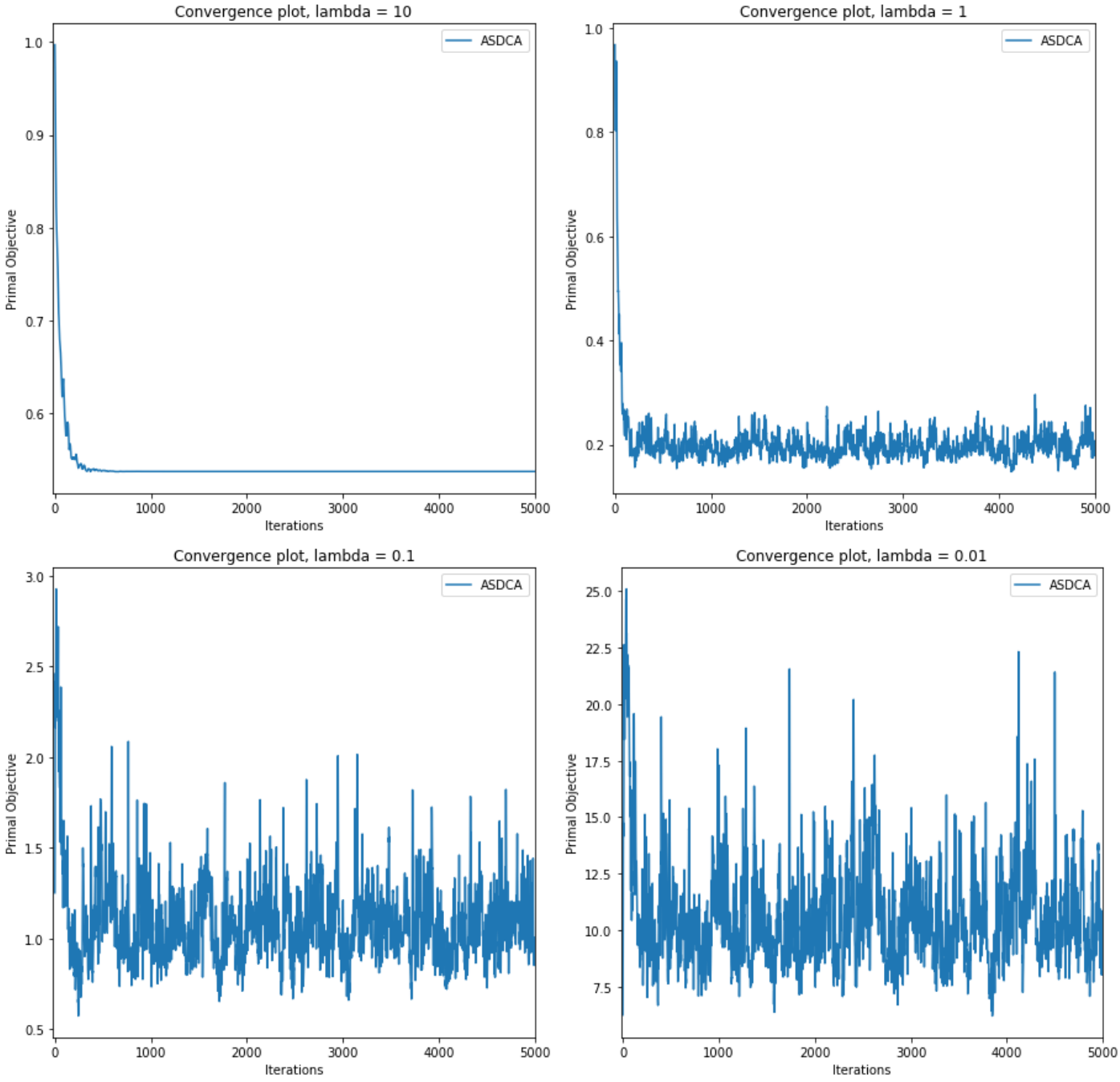


Figure 10 – Influence of λ on ASDCA (Genomic)

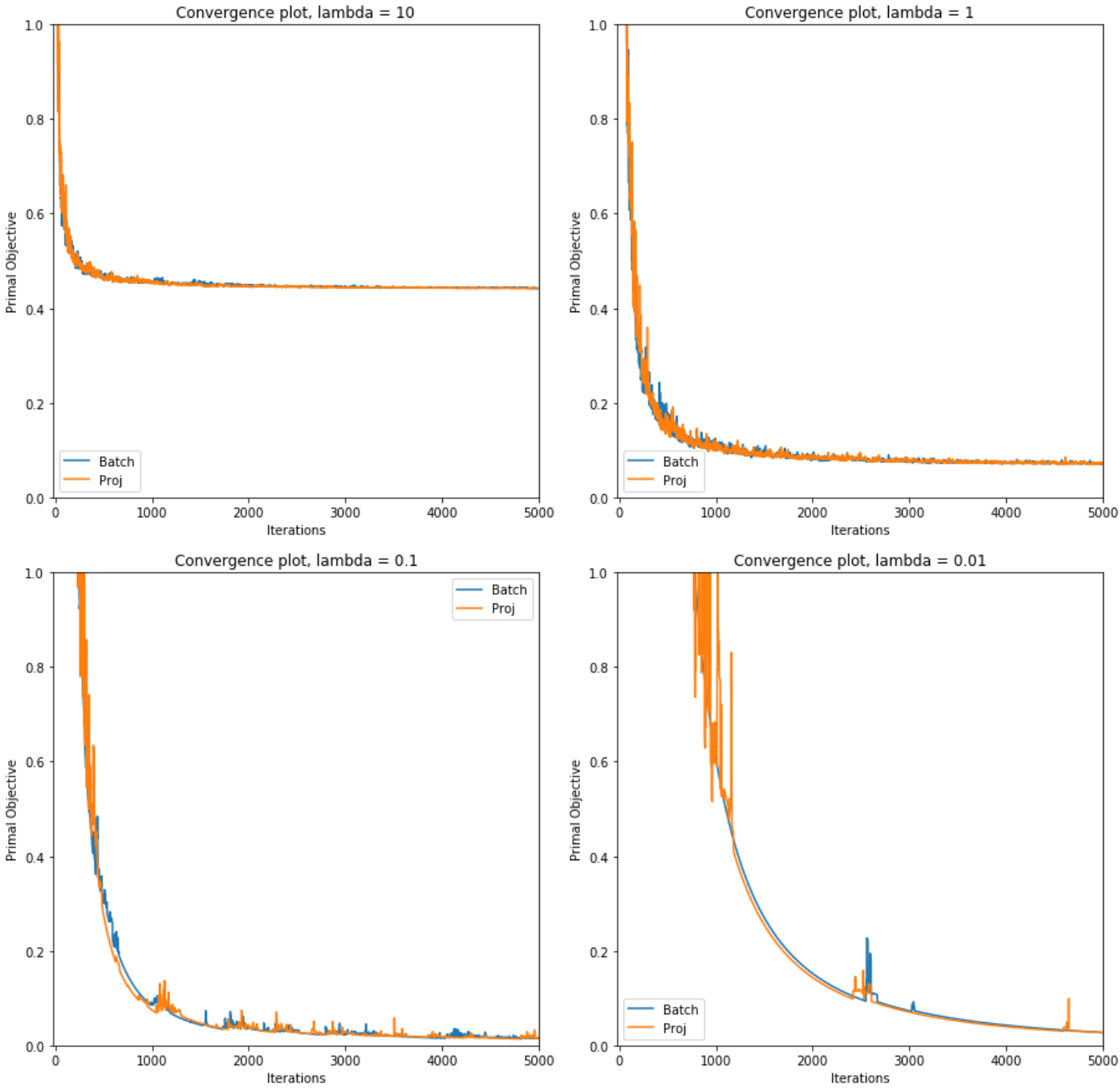


Figure 11 – Influence of λ on Mini-batch PEGASOS (Genomic)