Stochastic Optimization and Automatic Differentiation for machine learning

Performances benchmark of SDCA algorithm against Pegasos

Antoine Grelety

27 avril 2018

Table des matières

1 Introduction

In this report we are going to analyze the performances of two gradient descent algorithms, namely SDCA and Pegasos. These two algorithms are an alternative to the standard stochastic gradient descent and have shown solid good convergence results for optimization problems such as SVM. We will consider an SVM problem with non-smooth hinge loss fonction.
In the first part, we will analyze the results of the SDCA algorithm from different angles such as running time, number of epoch, accuracy or sparsity of coefficients. In the second part, we will apply the same approach with the Pegasos algorithm. Finally We will compare the two algorithms and conclude.

2 Data

In order to evaluate the algorithms we will consider the White Wine Quality dataset. This dataset is based on 12 features and 4898 observations. About two-thirds of the observations will be used for training the models, the remaining third will be used for testing the accuracy of the trained models.

The variable of interest is the grade of the wine, ranging from 1 to 10. In the scope of a SVM hinge loss minimization, the grade variable has been transformed from numeric to binary such that a grade above 5 takes on a value of one, and minus one otherwise.
Another important preprocessing step is the data scaling. We have standardized the input matrix X in order to facilitate the minimization problem.

3 Code

Two files of code have been used in this report :
— An IPython Notebook which loads the data, runs the algorithms and displays the results
— A Python file which contains two classes of fonctions : one for SDCA and another one for Pegasos.

4 SDCA Algorithm

4.1 Problem

The SDCA algorithm aims at reparametrizing the objective function. The primal objective is to find the best linear predictor w solving the regularized loss minimization :

$$min\ P(w) = [\ \frac{1}{n}\sum \phi_i(w^T x_i) + \frac{\lambda}{2}||w||^2\ ]$$

The dual objective aims at finding the linear predictor α maximizing :

$$max\ D(\alpha) = [\ \frac{1}{n}\sum -\phi_i(-\alpha) - \frac{\lambda}{2}||\ \frac{1}{\lambda n}\sum \alpha_i x_i\ ||^2\ ]$$

For each epoch, we maximize the dual function by moving its coordinates to the best direction from its previous position at $t-1$. We want to find $\Delta\alpha$ that maximizes $D(\alpha^{t-1} + \Delta\alpha)$.
The primal and dual parameters take different dimensions. In the primal problem, the vector w of interest is of dimension D, the number of features. With SDCA, we are now optimizing with respect to a vector α of dimension n, the number of observations.
In these experiments we will consider the SCDA-Perm procedure that randomizes the sample at each epoch and updates each α_i sequentially.

4.2 Results

We perform these experiments with a non-smooth hinge loss such that $\gamma = 0$. We ran the SDCA algorithm with different values of the regularization parameter $\lambda \in \{10^{-2}, 10^{-3}, 10^{-4}\}$. Also, for the sake of comparison of running times, we have set a stopping criterion so that the algorithm stops as soon as the duality gap goes below $\epsilon = 10^{-3}$.

4.2.1 Convergence

Figure 1 below shows the convergence rates of the primal cost function, the dual cost function, and the duality gap expressed as the difference between these two functions. For the largest λ, it takes only two epochs to reach the optimum of the cost functions. It takes about 20 epoch for $\lambda = 10^{-3}$ to reach the same point. For $\lambda = 10^{-4}$, the algorithm needs to run over 50 epochs and the primal suboptimal function does not decrease smoothly.

λ seems to have a strong influence on the speed of convergence of SDCA. The dual parameter α converges quite smoothly for all λ. However the convergence of the primal parameter w is more noisy for smaller values of λ. One reason might be that λ has non linear effects on the update of w, which is impacted through the term $(\lambda n)^{-1}$ and also $\Delta\alpha_i$. By contrast the update of α depends only on $\Delta\alpha_i$ and it is quite clear that a small λ translates in a smaller step size for the update of α.

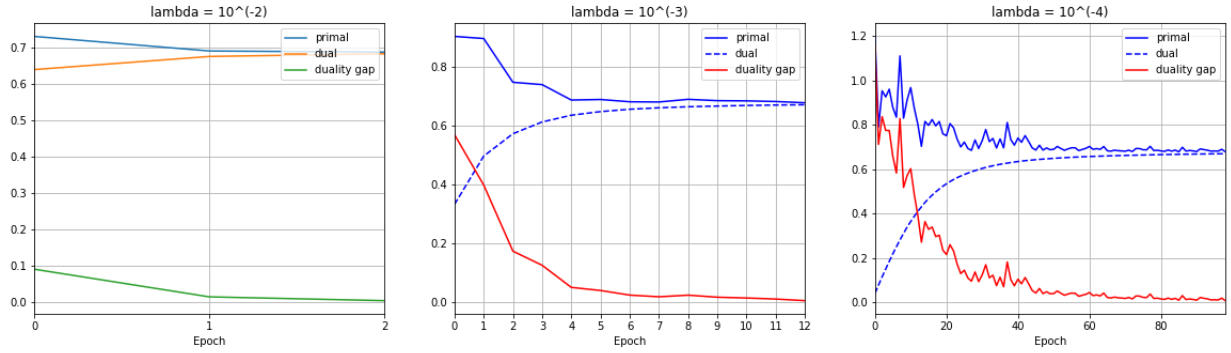

FIGURE 1: SDCA convergence results

In terms of running time, SDCA takes about 1.35 seconds to reach convergence for $\lambda = 10^{-2}$ and 200 ms per epoch. However the running time increases significantly for smaller λ as the number of epoch required is much larger.

4.2.2 Coefficients

As we can observe in Figure 2, all the coefficients w of the primal problem are self-contained in the range [-1,1]. Because data have been standardized, all the coefficients' values are more or less within the same range. Also, we can note that for higher values of the penalization term λ, the coefficients are smaller in absolute value. This makes sense considering that a higher λ is more likely to penalize large coefficients.

In Figure 3, the dual coefficients are much sparser for the smallest value of λ, where many values of α are spread quite evenly over [-1,1]. For the highest λ the results are more clear cut, the vast majority of the coefficients are almost always either -1, 0, or 1.

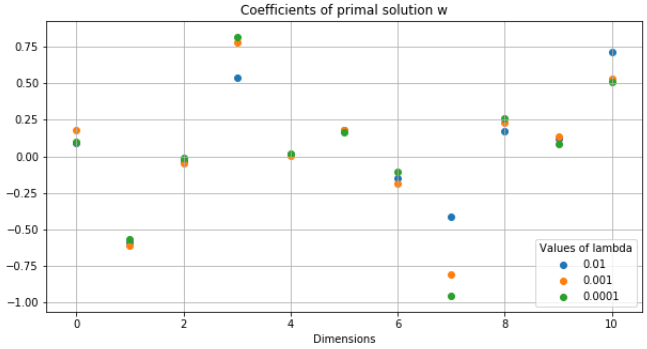

FIGURE 2: SDCA primal coefficients

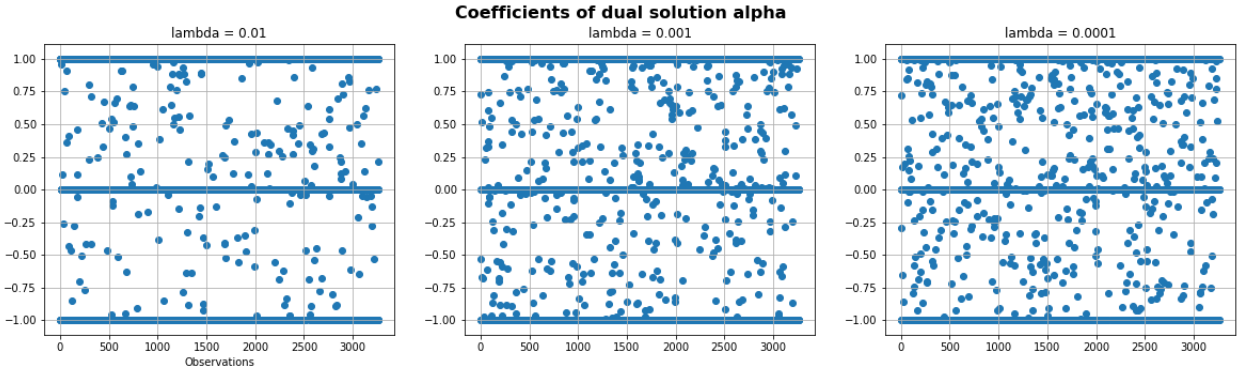

FIGURE 3: SDCA dual coefficients

4.2.3 Accuracy

The trained model's accuracy is about 70% on the test dataset and is robust with various values of λ. Considering that about 64% of the labels are positive, we can argue that this trained SVM model performs slightly better than randomness.

Value of λ	Accuracy %
10^{-2}	70.1%
10^{-3}	70.3%
10^{-4}	70.2%

TABLE 1: Accuracy results SDCA

5 Pegasos Algorithm

5.1 Problem

Pegasos is a simple model for solving sub-convex optimization problems such as SVM. It is a close variation of the stochastic gradient descent algorithm. The central idea consists in computing the sub-gradient of the objective fonction and updating the primal solution w with a step size dependent on the regularization parameter. The main difference with SDCA is that the step size here is predetermined, unlike SDCA that chooses the step size which minimizes the dual objective function.

5.2 Results

We use the same values of λ and stop the algorithm with the same stopping criterion $\epsilon = 10^{-3}$ as with SDCA.

5.2.1 Convergence

The algorithms are converging after about 10 to 20 gradient descent steps. In Figure 4, the convergence results are quite similar for all three values of λ but we can still note that convergence is faster for the smallest λ. Again this is quite intuitive as a smaller regularization term translates in bigger step size.
The computer performs the task relatively fast : about 534 milliseconds and 16ms per loop.

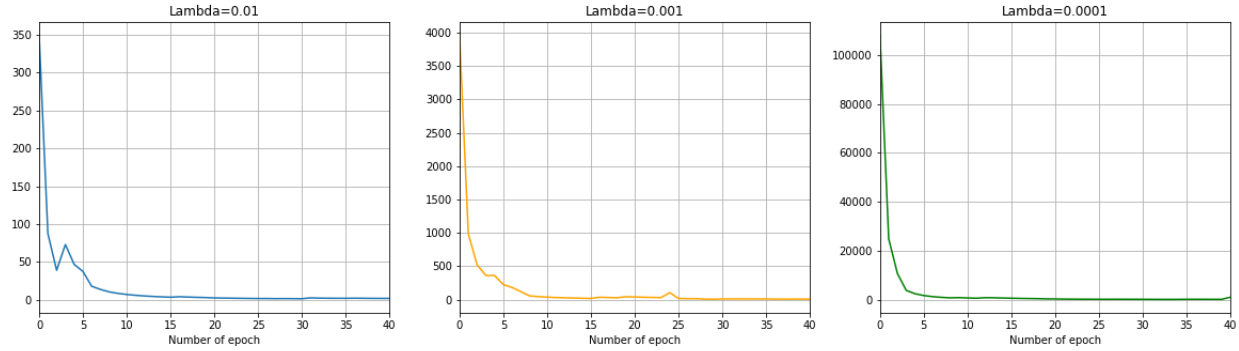

FIGURE 4: Pegasos convergence results

5.2.2 Coefficients

Here again, the estimated coefficients are closer to zero where the penalization term is the largest.

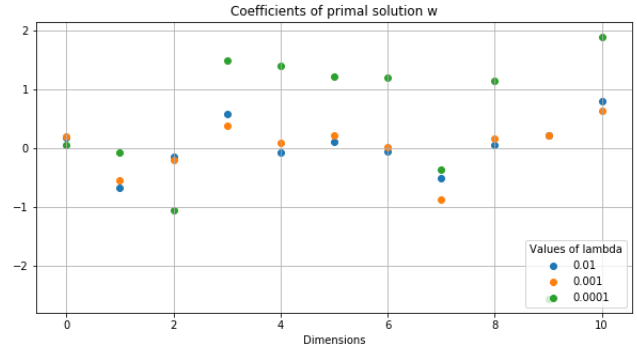

FIGURE 5: Pegasos primal coefficients

5.2.3 Accuracy

In our experiment Pegasos' accuracy is around 69%-71% for $\lambda = 10^{-2}, 10^{-3}$ and it drops at 53% for $\lambda = 10^{-4}$. As the latter regularization parameter is too small, the step of its gradient descent is too large and w has probably fallen in a sub-optimal region.

Value of λ	Accuracy %
10^{-2}	71.2%
10^{-3}	69.3%
10^{-4}	53.5%

TABLE 2: Accuracy results Pegasos

6 Comparison SDCA-Pegasos and Conclusion

In the light of our experiments, Pegasos performs better in term of convergence and running time. The running time is at least twice faster for Pegasos and the gap is deeper for smaller values of λ. In terms of accuracy, the results are similar for well chosen regularization term. However Pegasos' accuracy drops for small λ while SDCA seems to be more robust and is less likely to fall in a suboptimal region.

SDCA has the interesting property to have an adaptive step size which allows to control the gradient descent but it seeems to have a cost in terms of speed of convergence. By contrast, Pegasos' step size needs to be fixed by hand and one has to carefully chose the right value of λ.

We have thus compared the performances of two gradient descent algorithms based on a Support Vector Machine with hinge loss. For each algorithm, we have tested the robustness of the results with different values of the penalization term. These methods have been tested on a a dataset of 12 features. For further experiments, it would be interesting to test the behavior of these models on a dataset with a very large number of features and check if these results still hold.