Optimization Projet - IASD

Yassine ABOU HADID
yassine.abou-hadid@dauphine.eu

January 2023

Contents

INTRODUCTION:

This project was made for the Optimization for Machine Learning course from the M2 IASD cursus at Université Dauphine – PSL. Its goal is to study and apply the different optimization approaches seen during the lectures and the practical sessions to solve an optimization problem. For this purpose, I chose to work on a regression case. In the next chapters, I will present the dataset used in this study, as well as the different approaches implemented and tested.
This report is only a complement of the different Python notebooks where most of the work has been done.

PART 1: Gradient Descent

1. Dataset presentation :

This study looked into assessing the heating load requirement of buildings (that is the amount of heat energy that would need to be added to a space to maintain the temperature in an acceptable range) as a function of building parameters. To obtain the dataset, the energy analysis was performed using 12 different building shapes. The buildings differ with respect to the glazing area, the glazing area distribution, and the orientation, amongst other parameters.

To predict the Heating Load factor, the dataset was comprised of 768 samples and 8 features: Relative Compactness, Surface Area, Wall Area, Roof Area, Overall Height, Orientation, Glazing Area, Glazing Area, Distribution.

The correlation matrix below shows that some features are highly correlated, such as Relative Compactness with Overall Height, or Surface Area with Roof area.

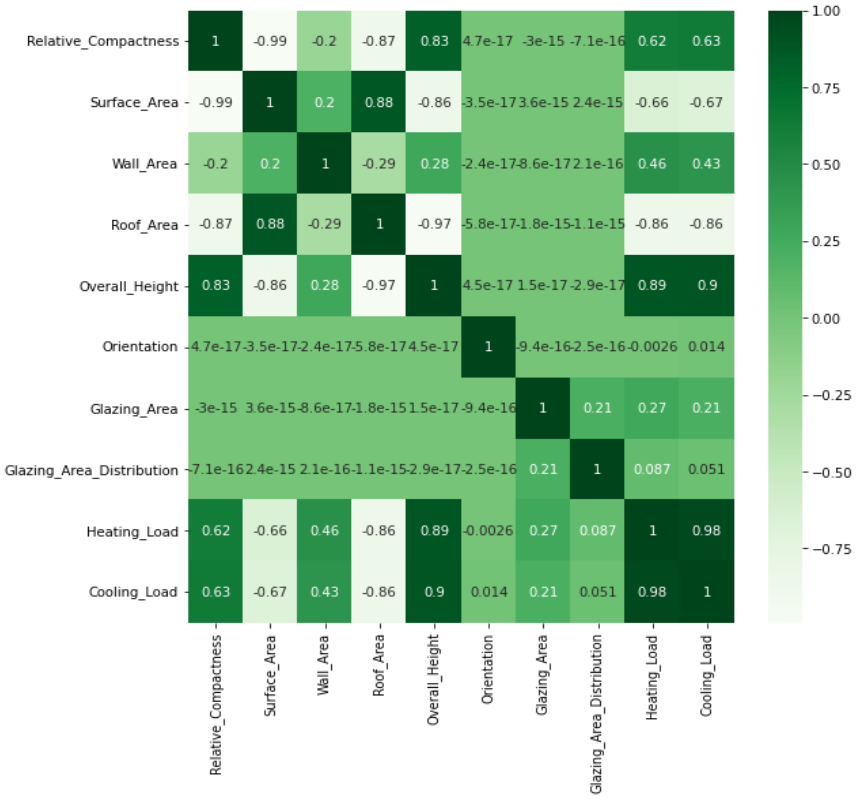

Figure 1: Correlation Matrix

2. Presentation of the problem:

Through this project and this report, we will work on a linear regression problem. The goal is to predict the heating load factor using the 8 features of the dataset. So, our minimization problem is equivalent to:

$$\boldsymbol{minimize}\ \boldsymbol{f}(\boldsymbol{x})_{\boldsymbol{x}\in\boldsymbol{R}^{\boldsymbol{d}}} = \frac{\mathbf{1}}{\mathbf{2}\boldsymbol{n}}\ \|\boldsymbol{Ax}-\boldsymbol{y}\|^{\mathbf{2}} + \frac{\boldsymbol{\lambda}}{\mathbf{2}}\|\boldsymbol{x}\|_{\boldsymbol{p}}$$

Where A is of shape (n,d)=(768,8) and y is of shape (n,1).
$\boldsymbol{\lambda}$ is the regularization term.

3. Gradient Descent with Ridge penalty:

Here, we aim to solve the problem above with p=2.
In a first exploration, we fixed the Ridge penalty term, and we tried to compute the convergence rate using different stepsizes in the GD implementation. The results obtained are showed below:

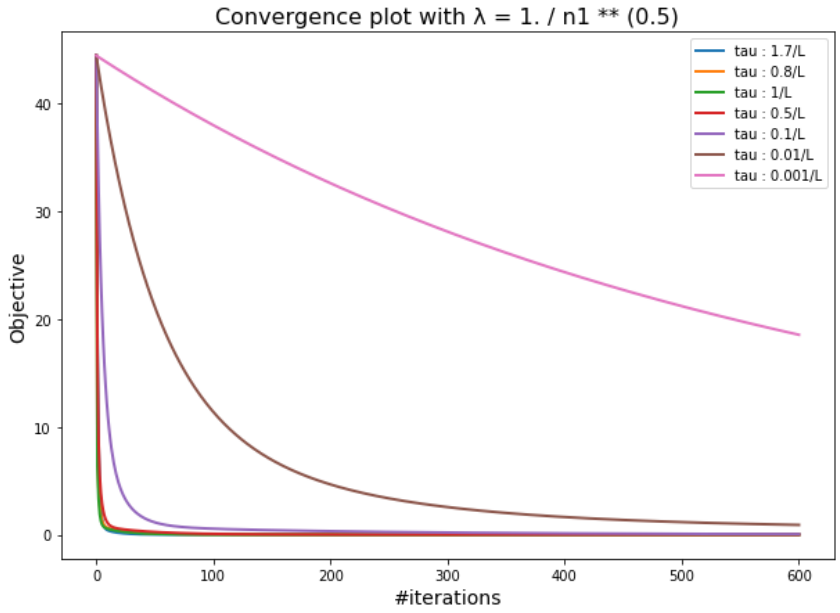

Figure 2: Impact of stepsize in GD

Indeed, this is coherent with the theory. In fact, we can see that for a stepsize equal to $\frac{1}{L}$ (with L the Lipschitz constant), the objective function converges faster. This confirms that $\frac{1}{L}$ is the optimal stepsize choice.

As a second experimentation, we fixed the optimal stepsize $\frac{1}{L}$ and we changed to the Ridge penalty term.

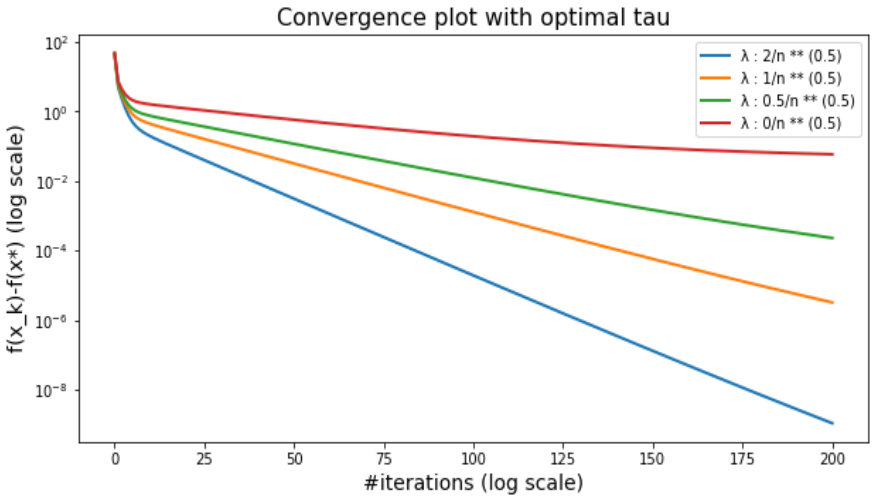

Figure 3: Impact of Ridge penalty term

We can see that a high penalty value prevents the model from overfitting (as this was plotted for the test set).

PART 2: Automatic Differentiation

Automatic differentiation is a powerful tool to automate the calculation of derivatives and is preferable to more traditional methods, especially when differentiating complex algorithms and mathematical functions. In this context, we implemented backpropagation from scratch using NumPy for a feedforward multilayer perceptron (MLP) with arbitrary number of layers.

PART 3: Stochastic Gradient Descent

Here, we explore the SGD algorithm, as it is known to be more efficient than traditional gradient descent; because in one hand, it can deal to non-convex problems. Besides, it is faster because it doesn't require computing all the gradient in every epoch.

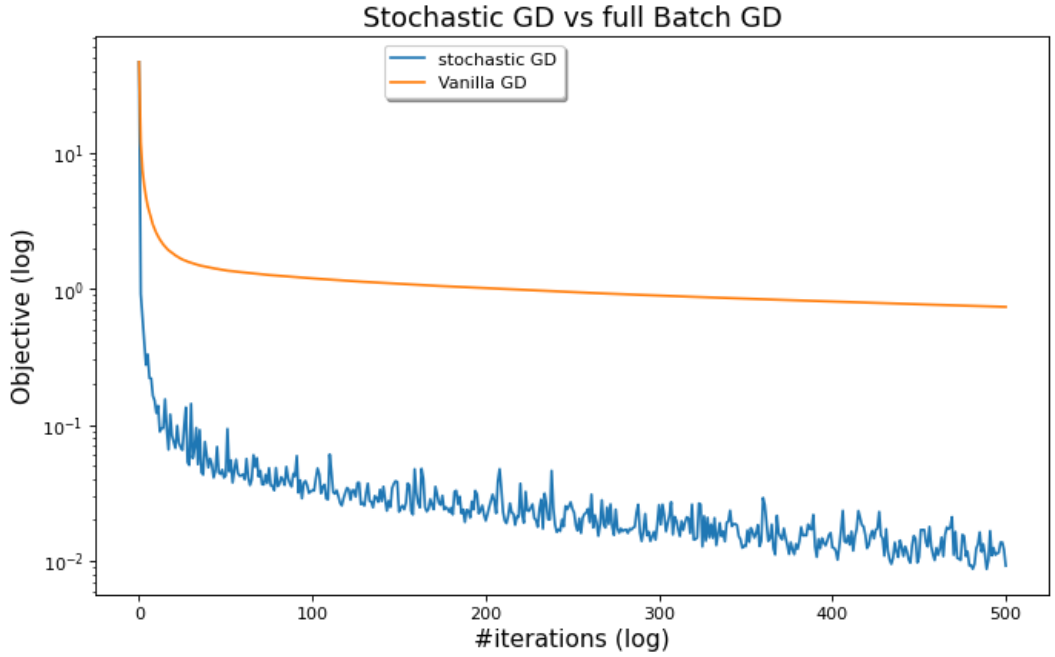

Figure 4: SGD vs vanilla GD

From the plot above, we can confirm that SGD is indeed faster than a natural full batch GD.

Next, we tried to find a compromise between SGD and GD: the mini-batch GD. Here, we define at each epoch our set as a sample of 1<m<n data points, with replacement. To that purpose, we can define m as a divisor of n.

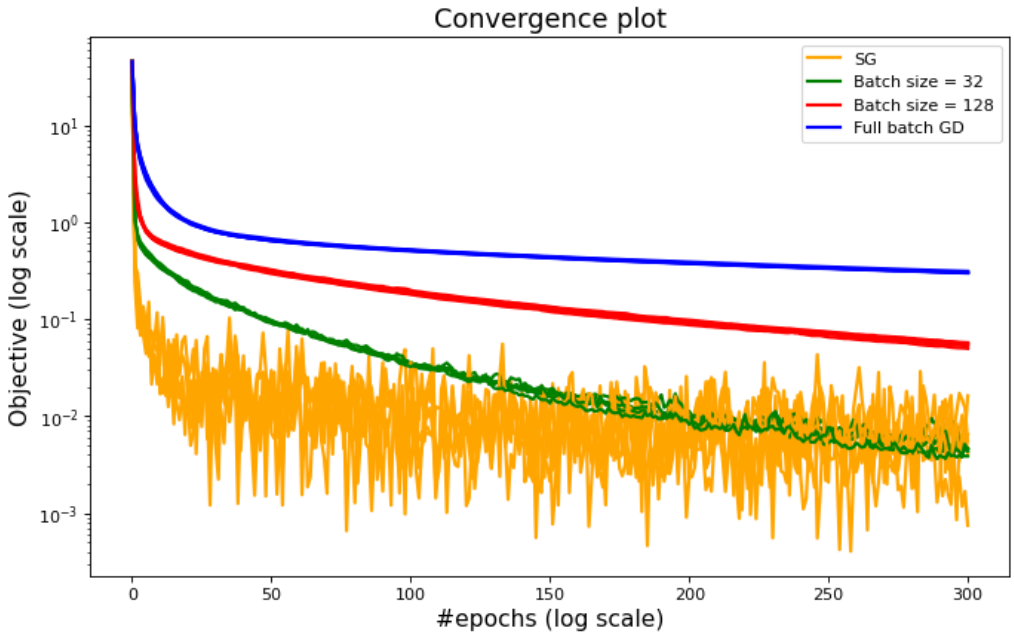

Figure 5: Mini-Batch Gradient Descent

Here, using a batch of 32 (out of 768) seems to be a good compromise. It is faster than GD and more precise than SGD.
There is no general rule for the batch size. It is a task that depends on the problem.

Finally, we implemented a variant of Stochastic gradient: the SVRG (the Stochastic Variance Reduced Gradient). This algorithm consists of two loops, where a reference full gradient is first evaluated in the outer loop and then used to yield a variance reduced estimate of the current gradient in the inner loop.
We compared it with SG with and without the averaging technique.
In one hand, we can see that using SG with averaging can lead to a smoother behavior in terms of function value and convergence. However, this process is not often used because of the computation cost, as it requires to store an additional vector (the average).

In addition, we can see that SVRG is slower than SG in this case, but it is more accurate. It is better at avoiding local minima and converging on global solutions quicker than SGD.

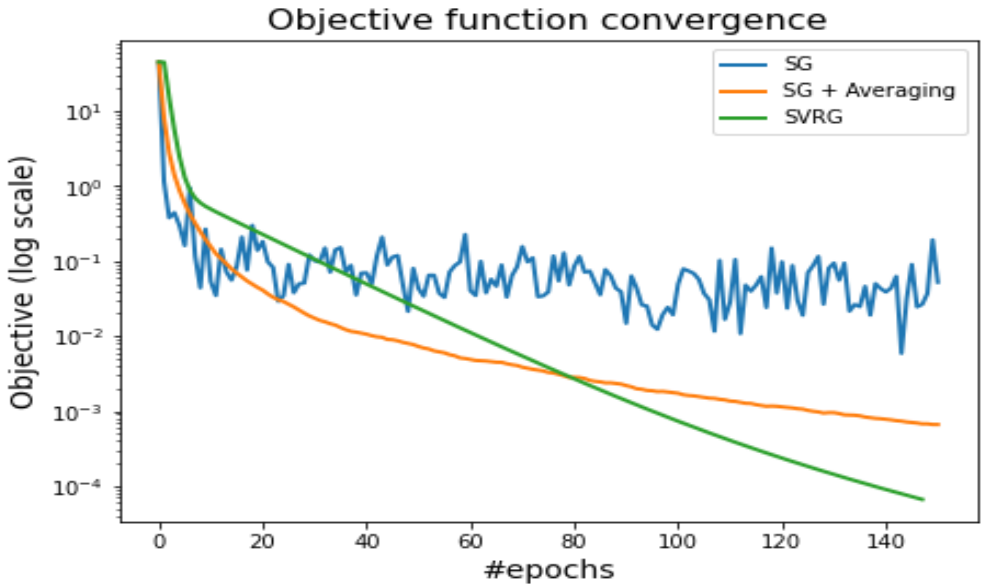

Figure 6: SG vs SVRG

PART 4: Convexity and Constrained Optimization

For this section, we implemented two algorithms which constrained the solutions of the problem to a given set.

1. Conditional Gradient:

The conditional algorithm, or Frank-Wolfe Algorithm is an iterative first-order optimization algorithm for constrained convex optimization. This algorithm considers a linear approximation of the objective function, and moves towards a minimizer of this linear function (taken over the same domain).

Here are the results obtained on Gaussian arrays:

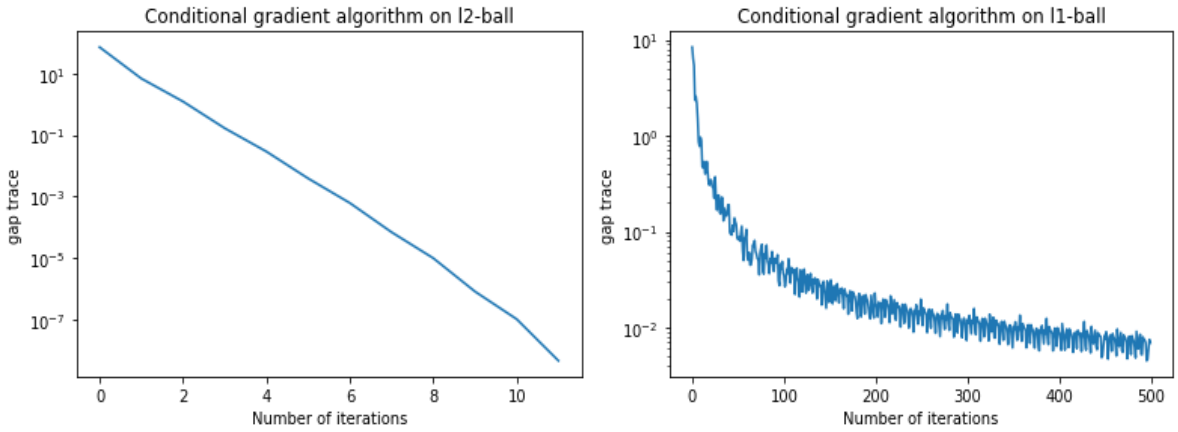

Figure 7: Conditional Gradient on l1 and l2 ball

2. Projected Gradient Descent:

If the Gradient Descent minimizes a function by moving in the negative gradient direction at each step. There is no constraint on the variable, the Projected Gradient Descent in the other hand minimizes a function subject to a constraint. At each step we move in the direction of the negative gradient, and then "project" onto the feasible set.

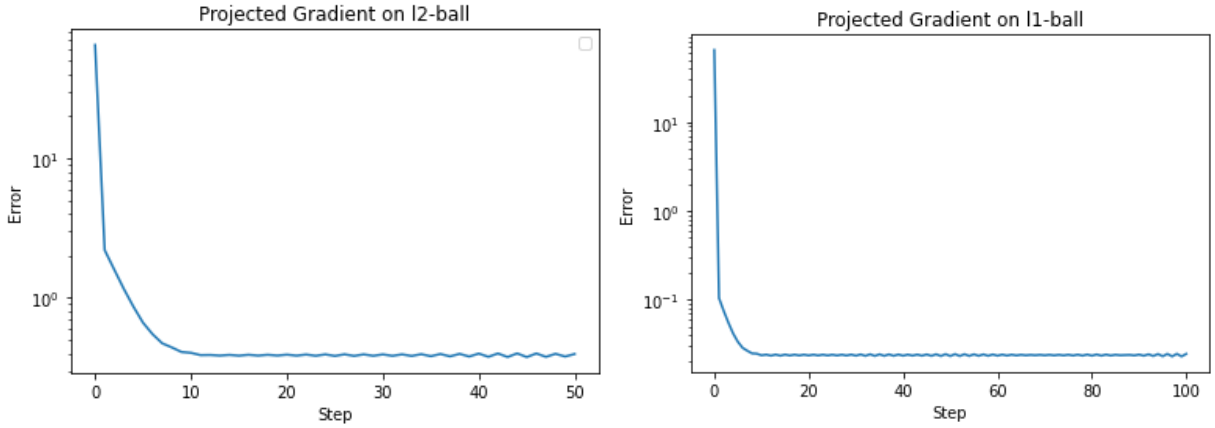

Figure 8: PGD on l1 and l2 ball

Comparing these two algorithms for constrained optimization, we see that PGD offers a faster convergence and overall a better accuracy than Frank-Wolfe algorithm.

PART 5: Proximal Gradient and LASSO

In this part, we implemented two different regularizations: a Ridge regularization (l2) and a LASSO regularization (l1) with penalty term values.
For Ridge regularization:

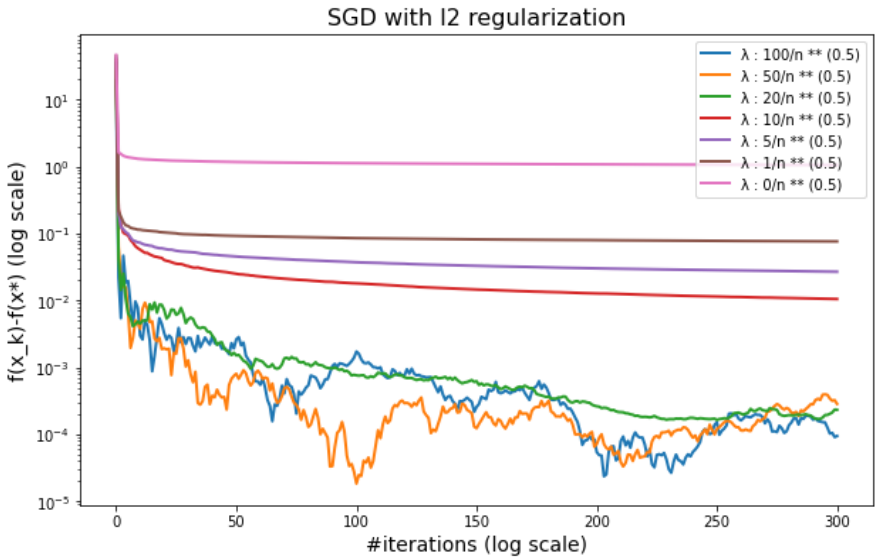

Figure 9: SGD with Ridge Regularization

In order to solve LASSO problem, gradient descent is clearly inefficient as the objective function becomes non-differentiable when adding the regularization penalty. For this end, I used ISTA (Iterative Shrinkage-Thresholding Algorithm) which is more fit to solve this type of problems. Unlike gradient descent, which only updates one variable at a time and does not take into account all the variables in the system, ISTA takes into account all variables when updating each parameter. This allows for better convergence towards a global optimum solution than gradient descent alone.
We implemented ISTA using different values for the penalty term.

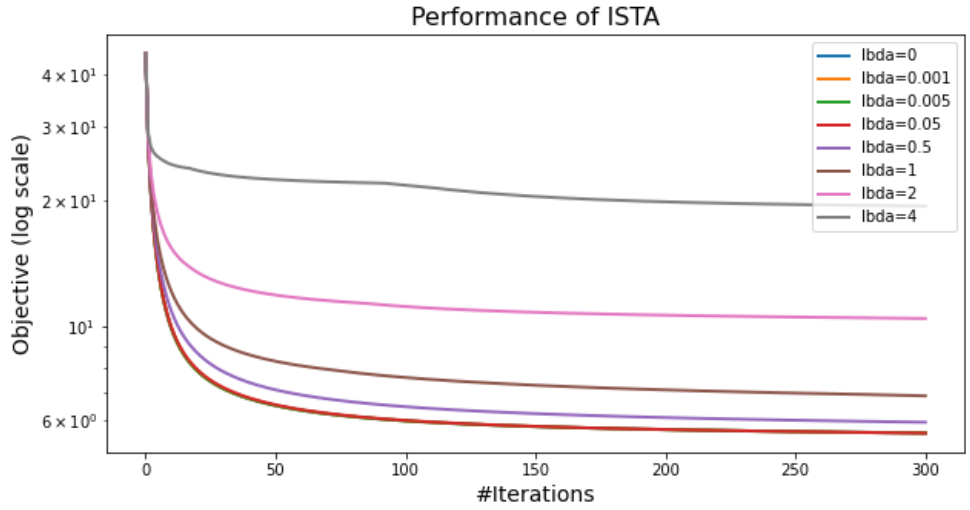

Figure 10: ISTA on LASSO

PART 6: Large-Scale Distributed Optimization

In the first part, we implemented the randomized block coordinate algorithm and tested it using different feature block sizes:

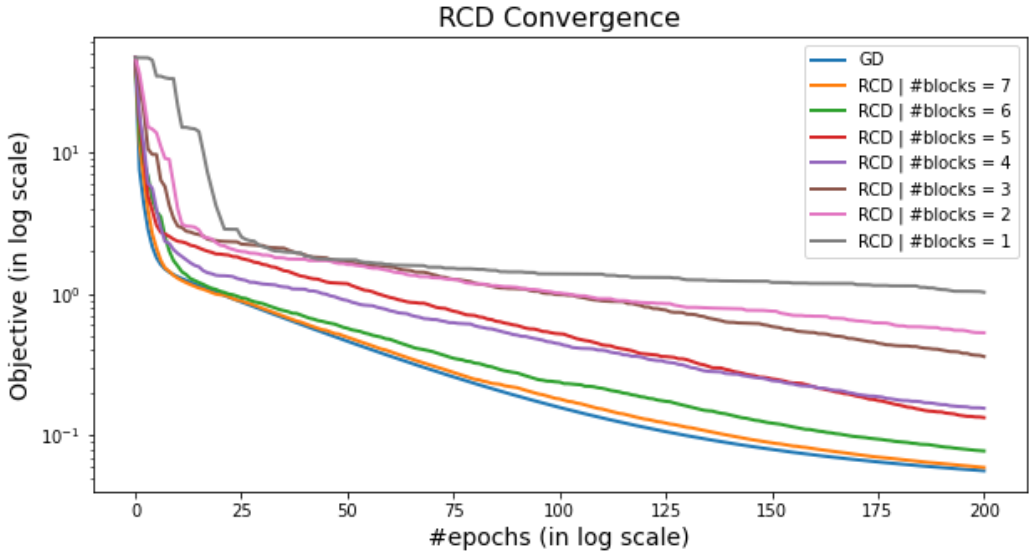

Figure 11: Randomized Coordinate Descent Convergence

In comparison with gradient descent, randomized block coordinate descent is beneficial because it works with small batches instead of full batch sizes. But on the other hand, GD seems to converge faster.
Although RBC is beneficial computation-wise as it only uses small batches instead of full batch, we can see that for our problem, it doesn't outperform GD, which seems to converge faster.

Next, we experimented combining using RCD with Stochastic Gradient.
We observe that RCD+SG is clearly more efficient than RCD alone.

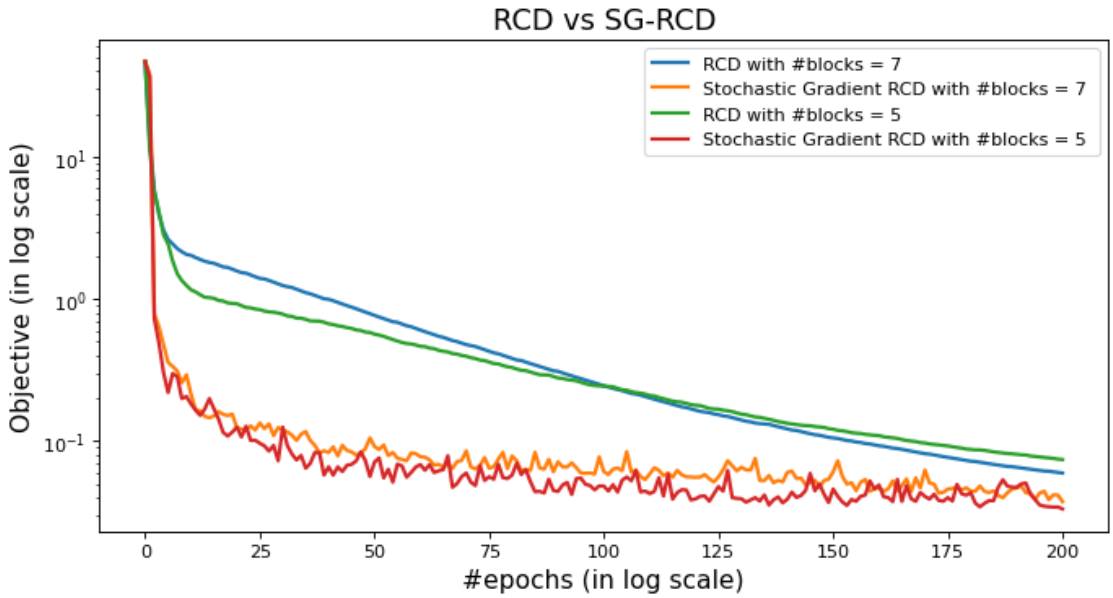

Figure 12: RCD vs SG+RCD

PART 7: Advanced Topics on Gradient Descent

In this part, we used Heavy Ball Algorithm. It is an optimization algorithm combining both GD and momentum. It takes into account the previous step's gradients when calculating the next step size, which enables it to converge faster and to avoid local minimas.

We implemented it using different values for the steps and momentum gamma parameter:

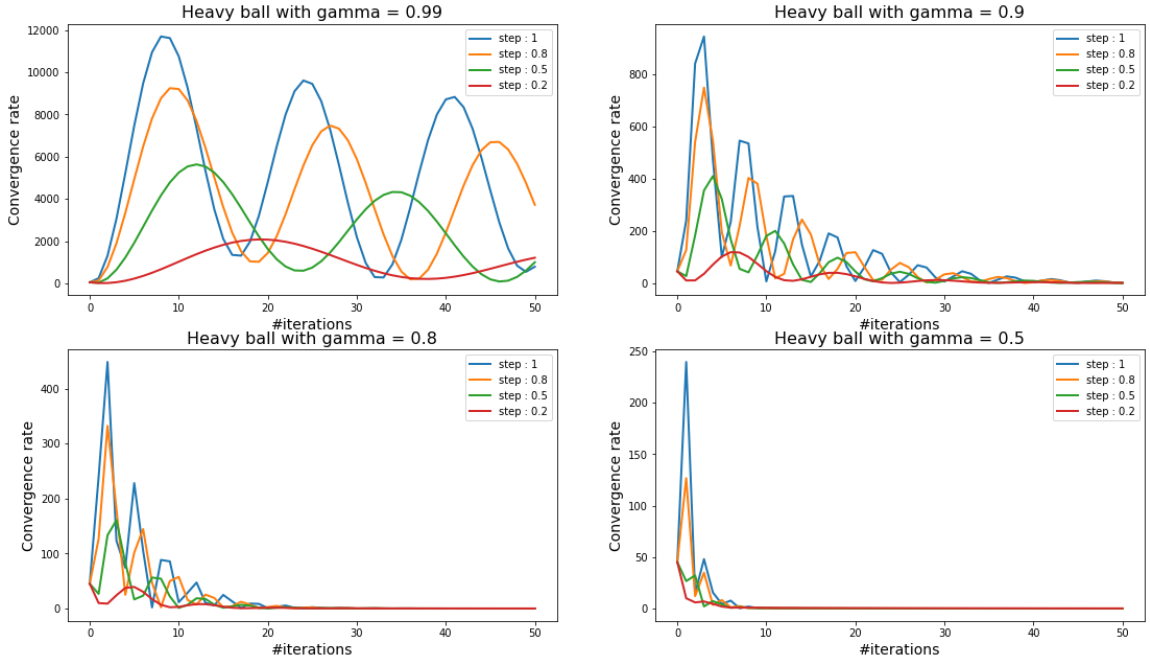

Figure 13: Heavy Ball

Next, we compared Heavy Ball and vanilla Gradient Descent using the best step and gamma obtained previously (0.5 and 0.7 respectively):

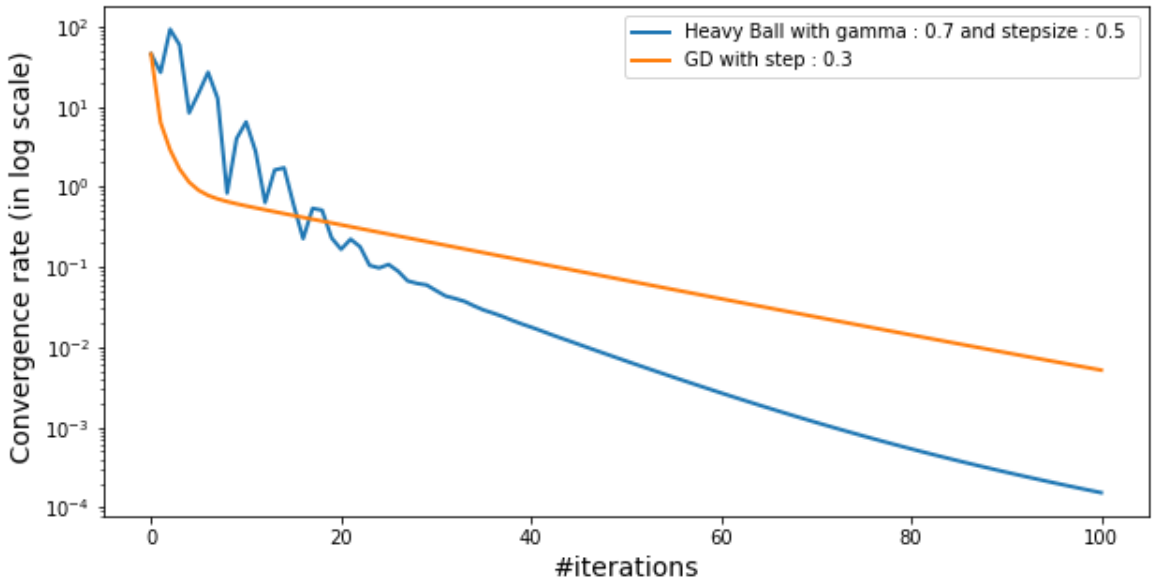

Figure 14: Heavy Ball vs GD

We can see how Heavy Ball outperforms GD after the 20th iteration.
Finally, we tried to minimize a non-convex obtained by adding a p-norm term to the objective function.
With p=0.5, I tried to solve the new minimization problem with both GD and Heavy Ball;

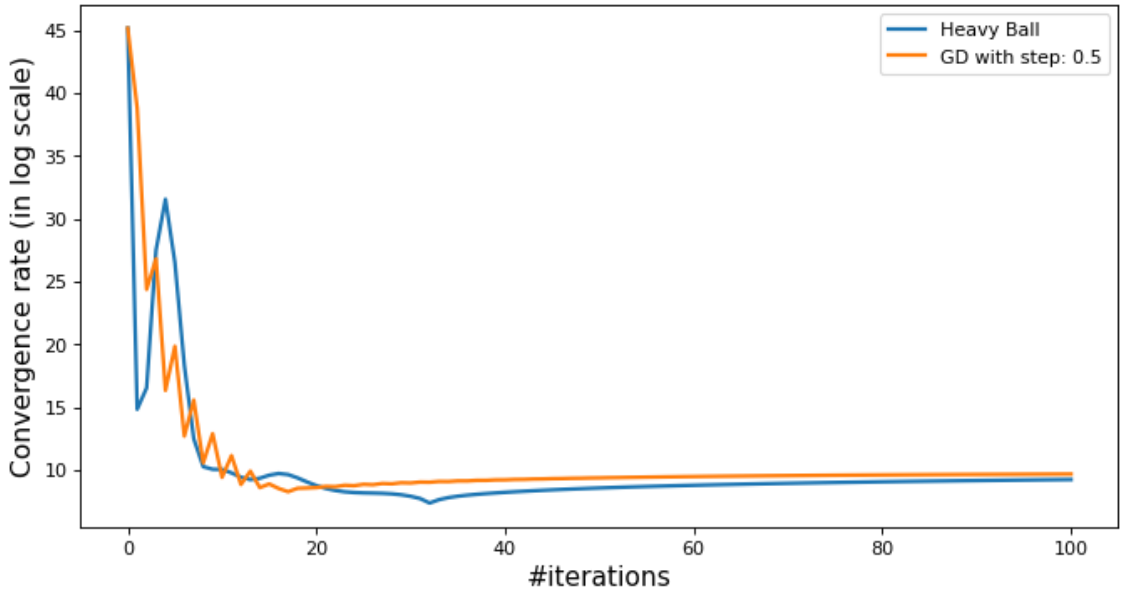

Figure 15: Heavy Ball and GD on non-convex problem

Even though the problem is non-convex, both approaches seems to work as the convergence rate seems to tend a value of 10.

Conclusion:

Throughout this project we had the opportunity to look more in-depth and implement well known optimization algorithms to solve machine learning problems with real-world datasets. The project explores various descent methods and aim to tackle different types of problems (constrained or unconstrained, convex or non-convex, differentiable or non-differentiable). It also exhibits that a combination of different algorithms is often more efficient (for example RCD with Stochastic Gradient).

REFERENCES:

[1] https://www.lamsade.dauphine.fr/~croyer/teach.html
[2] https://fa.bianp.net/blog/2018/notes-on-the-frank-wolfe-algorithm-part-i/
[3] http://people.csail.mit.edu/joanne/WOLA18-slides/Diakonikolas-Jelena