Optimization for machine learning

Marzook Cyril

Monday 10 of January 2022

Contents

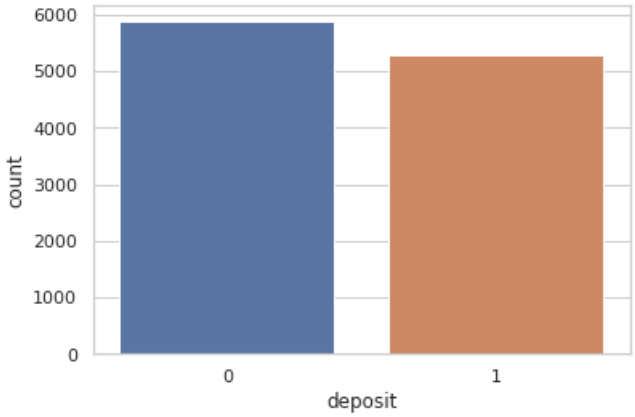
I Introduction

For this project, I have impleted several machine learning algorithms and optimization techniques. I have used the dataset banks.csv which can be found in the zip. I took this dataset from kaggle : `https://www.kaggle.com/janiobachmann/bank-marketing-dataset/version/1`. So, it is composed of more than 11000 samples with 17 features. This is the classic marketing bank dataset uploaded originally in the UCI Machine Learning Repository. The dataset gives information about a marketing campaign of a financial institution. The classification goal is to predict if the client will subscribe to a term deposit. The seventeen columns are : the age, the marital status, the default, the balance, the housing, eventually the loan, the type of contact, the day and month of the campaign, the duration, the title of the campaign, the numbers of days (pdays), the previous campaign, the previous outcome (poutcome) and finally the deposit.

II Data visualization

Fisrt, to gain some insight on the data distribution and to see if the two classes are well balanced :

Figure 1 : Distribution of the deposit column



We can observe here that the data is well balanced and thus we won't have any problems with trainning and testing. Now let's analyze the correlation between all variable and their dependencises with the following graphics :

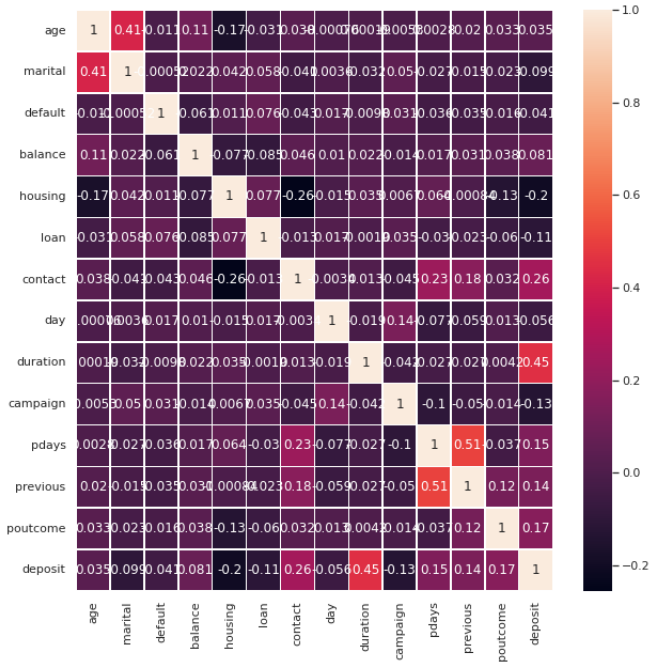
Figure 2 : Pairplot

We can observe that :

- People who subsribe to a deposit are majoritarily people with a high balance.
- The longer the campaign is the more effective it will be. We can see that for duration over 2000 hours (3 months) we have more than 80% people getting a deposit without regarding the other features
- Divorced people are more likely to subscribe to a deposit.
- The account balance does not depend on the age or the marital status. Thus, we can think that those variables are not correlated.

In order to verify if our features are correlated or not we can observe the next figure which corresponds to the feature correlation :

Figure 3 : Feature correlation



We can observe from the feature correlation that the data is not very complex. Furthermore, duration has a correlation of 0.45 with the variable deposit. So this does not mean anything since a correlation of almost 0.5 does not give a lot information we just know that duration may be correlated (positively or negatively) to deposit but we can not be sure here. We can also observe a coefficient of 0.51 between pdays and previous. This is totally normal since those two variables come from a previous campaign. Now that we are sure that no variable are correlated with another we can go through the next part. For the next part, the input space is $\mathbb{R}^d$, with d = 16.

III Gradient Descent

We will study here the optimization problem for logistic regression with gradient descent with a constant stepsize. This correspond to the following problem :

$$w \in \mathbb{R}^d, f(w) := \frac{1}{n}\sum_{i=1}^{n} f_i(w), \quad f_i(w) = \log\left(1 + \exp\left(-y_i x_i^T w\right)\right) + \frac{\lambda}{2}\|w\|^2,$$

where every y_i is a binary label in $\{-1, 1\}$ and $\lambda \geq 0$ is a regularization parameter.
The derivative of $t \mapsto \log(1 + \exp(-t))$ is $t \mapsto \frac{-\exp(-t)}{1+\exp(-t)} = -\frac{1}{1+\exp(t)}$. Combining this with the linear function $\mathbf{w} \mapsto y_i \mathbf{x}_i^T \mathbf{w}$, we get for every i that

$$\nabla f_i(\mathbf{w}) = -\frac{y_i}{1 + \exp(y_i \mathbf{x}_i^T \mathbf{w})}\mathbf{x}_i + \lambda \mathbf{w}.$$

and

$$\nabla f(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} -\frac{y_i}{1 + \exp(y_i \mathbf{x}_i^T \mathbf{w})}\mathbf{x}_i + \lambda \mathbf{w}$$

One way to obtain a Lipschitz constant for the gradient is to bound the second-order derivative, which is given by:

$$\nabla^2 f(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} \frac{\exp(-y_i \mathbf{x}_i^T \mathbf{w})}{(1+\exp(-y_i \mathbf{x}_i^T \mathbf{w}))^2} \mathbf{x}_i \mathbf{x}_i^T + \lambda I,$$

where I is the identity matrix. The result follows by noticing that $t \mapsto \frac{e^t}{(1+\exp(t))^2}$ is always less than or equal to $\frac{1}{4}$ (its value at the origin).
As we have seen in the course, to ensure convergence, the stepsize should be chosen such that

$$\tau < \frac{2}{L}$$

where : $L = \frac{\|A^T A\|}{4n}$ is a Lipschitz constant for ∇f when there is no regularization. In this case, $\tau_{\max} = 2/L$. Here is the implementation of gradient descent with a constant stepsize $\tau = 1/L$

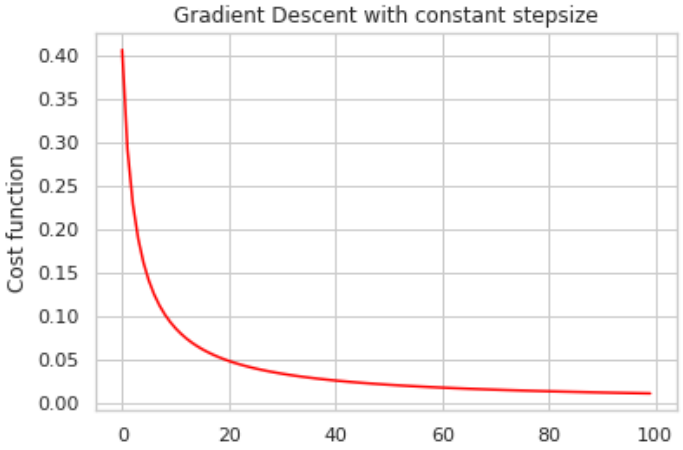


Figure 4 : Gradient Descent with a constant stepsize.

We have also the implementation of gradient descent for several constant stepsize :

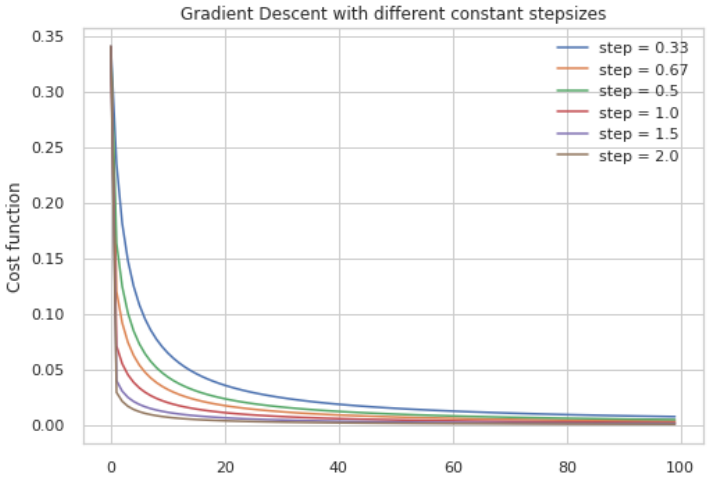


Figure 5 : Gradient Descent with several constant stepsize.

The plots show that gradient descent converges within around 80 iterations. In figure 5 we obtained different curves for different values of stepsize. The curve with a stepsize of 0.33 means that we used a $\tau = \tau_{max}/3$ and 0,67 correspond to $\tau = 2 * \tau_{max}/3$ and so on. We can see that increasing the stepsize make the loss function of gradient descent converge faster. We also know that taking a value a τ higher than τ_{max} will not guarantee the convergence.

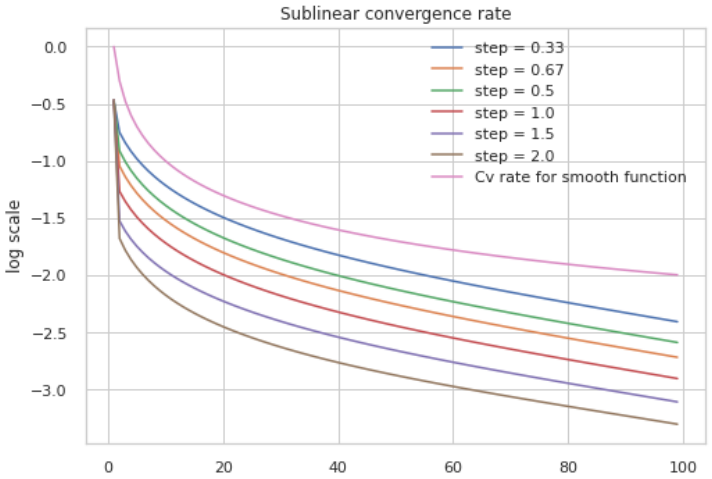


Figure 6 : Convergence rate for Gradient descent

We know that logistic function is a smooth function, convex but not necessarily strongly convex. So in this case the convergence rate is $O(1/\mathrm{k})$. One can also observe that the curves for the different values of the stepsize are quite parallel to the convergence rate line.

III.1 Accelerated Gradient Descent, Nesterov

During the courses we saw that Nesterov's algorithm corresponds to :

$$x_{t+1} = y_t - \frac{1}{L}\nabla f(y_t)$$

$$y_{t+1} = x_{t+1} + \gamma_t(x_{t+1} - x_t)$$

with $x_0 = y_0$ an arbitrary starting point and where $(\gamma_t)_{t\in\mathbb{N}}$ is a carefully chosen sequence of real numbers whose exact definition is :

$$\lambda_{-1} = 0$$

$$\lambda_t = \frac{1 + \sqrt{1 + 4\lambda_{t-1}^2}}{2}$$

$$\gamma_t = \frac{\lambda_t - 1}{\lambda_{t+1}}$$

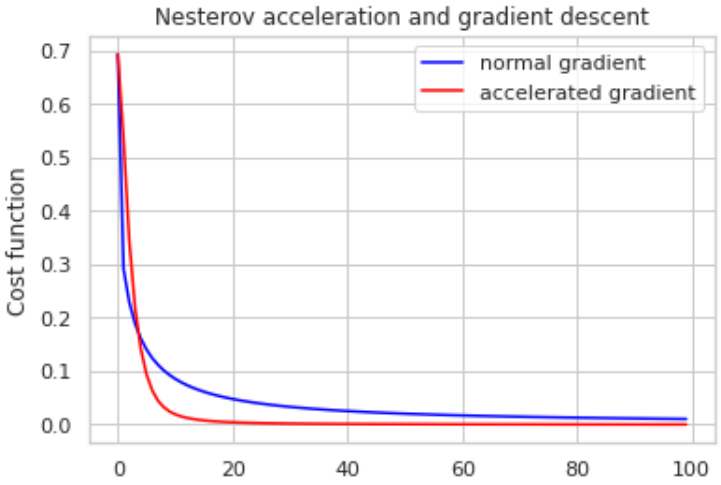


Figure 7 : Comparison between Nesterov and gradient descent.

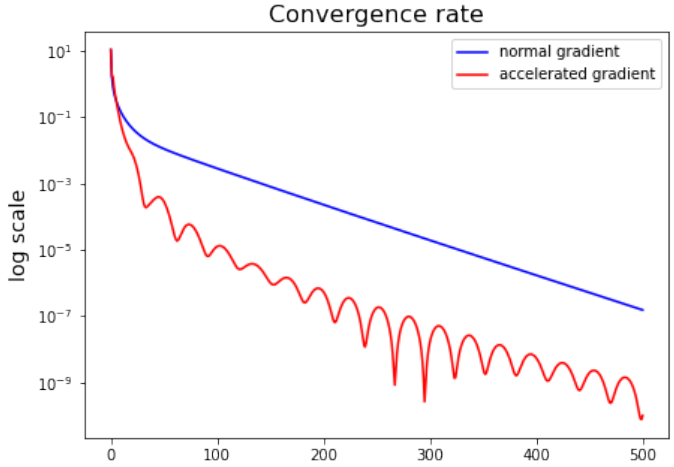


Figure 8 : Convergence rate of gradient descent and accelerated gradient.

As expected, the Nesterov's algorithm converges faster than gradient descent. Also Nesterov's method is not a descent method, so the function value can increase at some iterations (due to the momentum term) as we can see on figure 8.

IV Stochastic Gradient Descent

The iteration of stochastic gradient is given by:

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \alpha_k \nabla f_{i_k}(\mathbf{w}_k),$$

where i_k is drawn at random in $\{1, \ldots, n\}$. There is also a more general version of stochastic gradient, called batch stochastic gradient, which is given by the iteration

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \frac{\alpha_k}{|S_k|} \sum_{i \in S_k} \nabla f_i(\mathbf{w}_k)$$

where S_k is a set of indices drawn uniformly in $\{1, \ldots, n\}$. We know that $|S_k| = n$ results in a full gradient step, while $|S_k| = 1$ corresponds to a basic stochastic gradient step. Here, we only choose a stepsize $\tau < 1/L$.

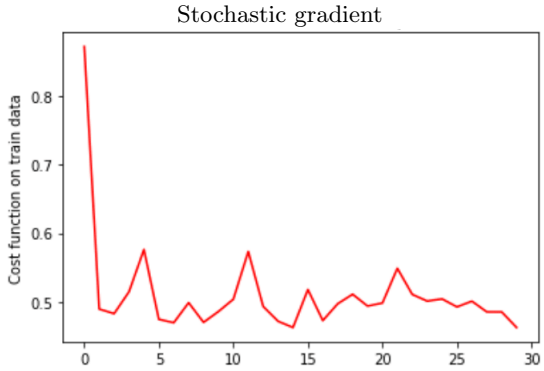
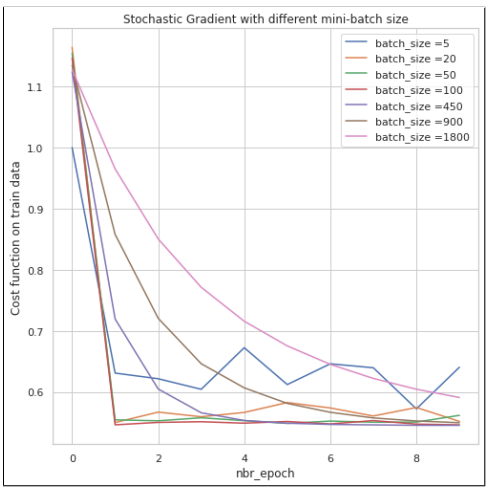


Figure 8 : Stochastic gradient with a constant step size.



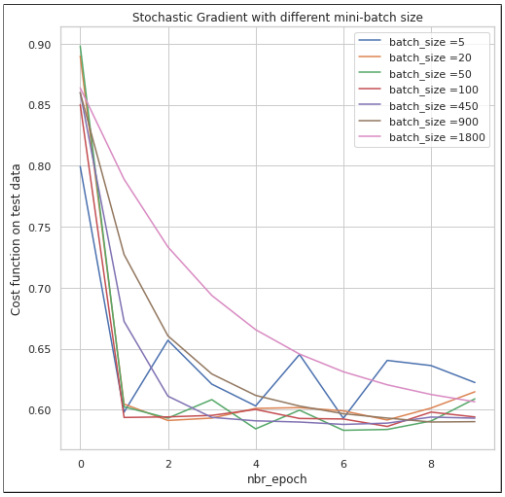


Figure 9 & 10 : Stochastic gradient with different batch size

So stochastic gradient with different values of batch size is implementated here and the evolution of the loss function on the train set as well as on the test set is drawn (figures 9 and 10). One can observe that the best convergence is for batch equal to 50 and batch equal to 100.

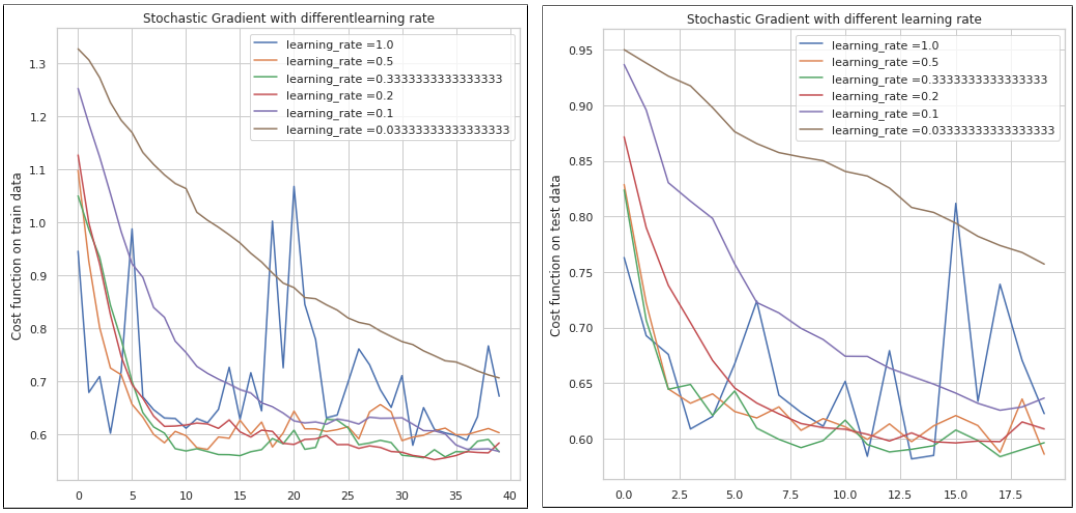


Figure 11 & 12 : Stochastic gradient with different learning rates.

For constant stepsizes, we observe that choosing a "large" stepsize may prevent the method from converging. On the other hand, using a very small stepsize will lead to slow convergence (as illustrated by the 0.033 curve). We expect this slow convergence behavior to be followed by an oscillatory phase, which we can readily see with other stepsize choices. The blue and violet curves (learning rate of 1 and 0.1, respectively) illustrate the trade-off between having a small stepsize, that guarantees convergence to a small neighborhood of the optimum, and having a large stepsize, that leads to faster convergence (but to a large neighborhood).

IV.1 Stochastic gradient with diagonal scaling (optional)

The intuition behind diagonal scaling is to decay the learning rate for parameters in proportion to their update history. I have implemented RMSPROP which correspond to the Root Mean Square Propagation algorithm, maintains an average of the magnitude of every component of the (stochastic) gradient through the following recursion: for each $i = 1, ..d$

$$[R_{-1}]_i = 0$$
$$[R_k]_i = (1-\lambda)\cdot[R_{k-1}]_i + \lambda\cdot[g_k]_i^2$$
$$[w_{k+1}]_i = [w_k]_i - \alpha\cdot\left(1/\sqrt{[R_k]_i + \mu}\right)\cdot[g_k]_i$$

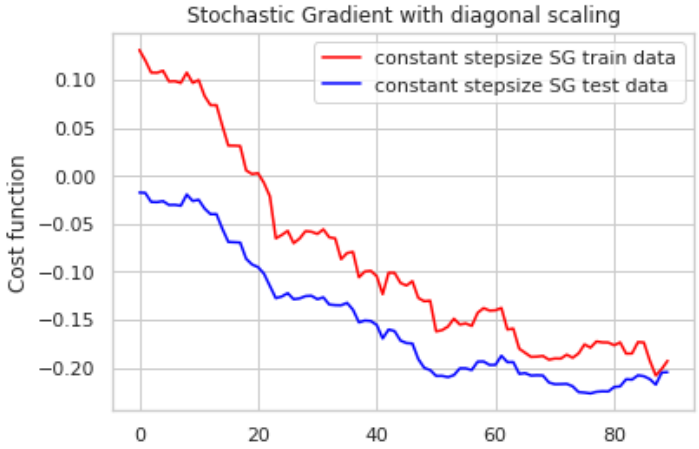


Figure 13 : Graphic of stochastic gradient with diagonal scaling (RMSprop)

IV.2 Stochastic gradient with momentum (optional)

For convex problems, it is known that using momentum leads to accelerated algorithms that converge faster than their accelerated counterparts. Stochastic gradient methods with momentum are typically based on the following recursion:

$$w_{k+1} := w_k - \alpha \nabla f_{ik}(w_k) + \beta(w_k - w_{k-1})$$

With $\alpha > 0$ and $\beta > 0$

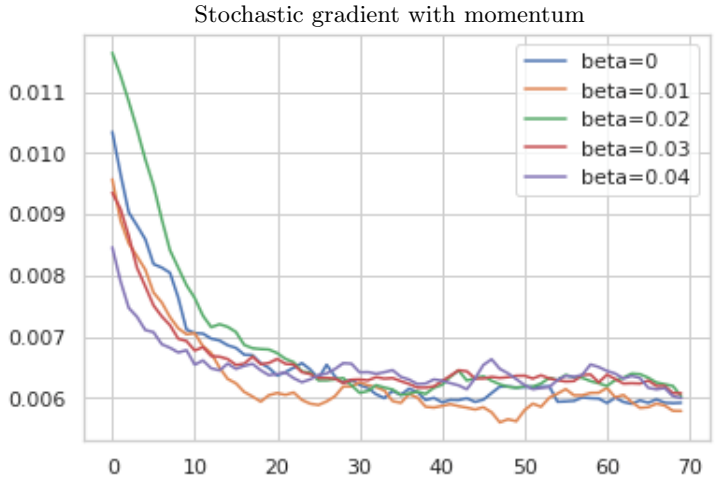


Figure 14 : Graphic of stochastic gradient with momentum

V Ridge and lasso Regularization

V.1 Ridge

Ridge Regularization is by far the most popular regularizer, and corresponds to using $R(x) = \|x_k\|^2_{\mathbb{R}^p}$ and add $\lambda R(x)$ to the function. Here are the regularization paths :

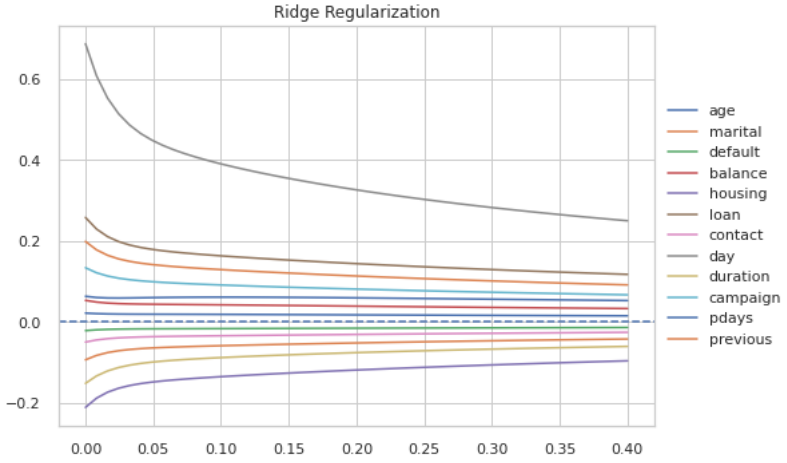


Figure 15 : Graphic of ridge regularization

Ridge Regularization is also used to select the best features in a dataset. On the first graph bellow, you can observe the value of x as a function of the different values of λ. As we have seen in course (with Gabriel Peyré), Ridge does not force the coefficient to go to zero, but it reduces their amplitufes.

V.2 Lasso (ISTA)

The Lasso corresponds to using a l_1 penalty

$$R(x) = \|x\|_1 = \sum_{k=1}^{p} |x_k|$$

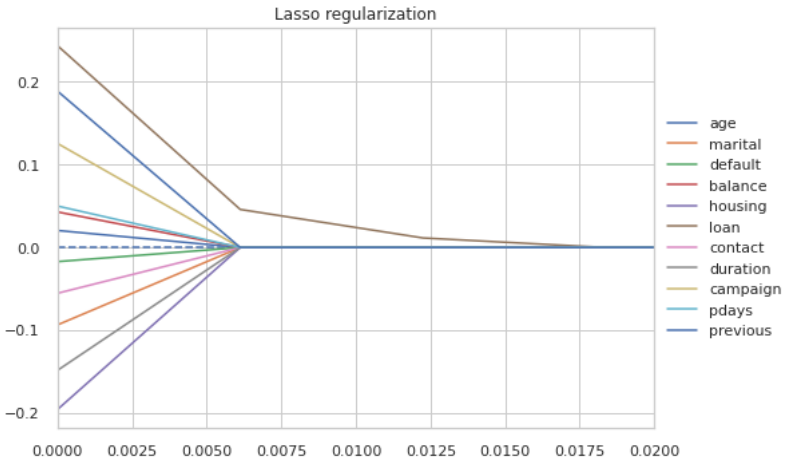


Figure 16 : Graphic of lasso regularization

We can see that this regularization makes the objective function not differentiable. But we have seen the forward backward algorithmto approximate the solution (the corresponding function in my notebook is *forwardbackward(A,y,lbda,tau,d)*)

The figure is coherent with what we saw in the regularization course. In fact, the lasso forces some coefficients to go to zero when we increase the parameter λ . In our case, if we go left to right, the first feature to reach zero is previous, age and marital. Loan goes to zero at last.

VI Newton second order method (optional)

Here is the newton algorithm :

$$x_{k+1} = x_k - \tau(Hf(x_k)^{-1}\nabla f(x_k))$$

with H(f(x)) corresponding to the hessian. The newton second order method converge a lot faster than gradient descent. However, this method takes more computation time at each iteration. In fact, it is generally very expensive costly to compute the hessian and its inverse especially for large dimension p. (I have tried to implement cauchy step and armijo step but I still have a problem in the code in my notebook)

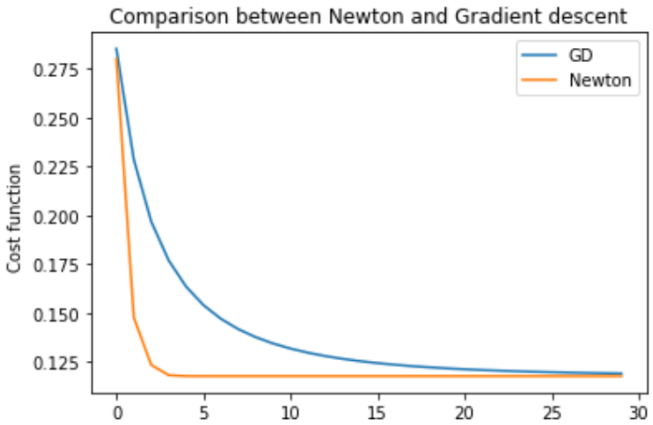


Figure 17 : Graphic of newton method and gradient descent.