# Project : Optimization for Machine Learning
# Bank Marketing Classification Problem

Samy FERRAT
Université Paris Dauphine - PSL
samy.ferrat@dauphine.eu

## Abstract

*This report is doing in response of the middle grade of the course Optimization for Machine Learning. The goal of this project is to apply more techniques we have learnt in the class to the real dataset neither the problem is classification or regression problem. We establish as possible the link between the theoritical results we get in the course and the experiments.*

## 1. Introduction

The study is consist to select one classification or regression problem according to the nature of the dataset and testing more optimization techniques. In this way, we are choosed the *Kaggle dataset*[1] which names **Bank Marketing**. This dataset is related to direct marketing campaigns of a Portuguese banking institution. The marketing campaigns were based on phone calls. Often, more than one contact to the same client was required, in order to access if the product (bank term deposit) would be (or not) subscribed. The classification goal is to predict if the client will subscribe to a term deposit (variable $y$).

## 2. Dataset description

We have in the original dataset two comma-separeted values(csv) files. The first csv file contains the full dataset with 45211 rows or clients and 16 features (except the target variable $y$) and the other, 10% of the examples (4521), randomly selected from the full dataset. The target variable is binary: ”yes”or ”no” where we shall recode in two classes 0 = ”no”,1 = ”yes”. 0 represents the most frequently class of clients in the dataset (see fig 1). The second file allows us to obtain directly the *testing set*. For the training set,

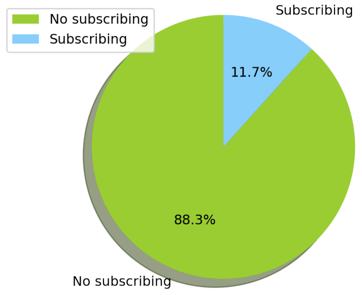

Figure 1: Target distribution

we shall do some manipulations in the terms of reduction of the dataset and keeping the same proportion of classes in the full dataset. Because we know that the large dataset requires more computational ressources and the problem remains not necessarily tractable so we draw an algorithm to select 10000 rows, not random, in the full dataset which shall verify the same or approximate proportion of the distribution of the target variable. This part was doing after the preprocessing which are very important in any data science problem.

## 3. Preprocessing

This part tackle with the preparation of the dataset for the learning. We simply use transformations techniques such Normalization for the numerical variables and one hot encoding for the categorical variables. We have also deleted the variables with number of modalities larger than twelve, and we have used the *train_test_split* of *Scikit-learn* method to select the 10000 rows among the 45211 rows with stratification, regards to the target variable. Finaly, we get 10000 rows and 21 features for the training set and 4521 rows and 21 features for the testing set. For common experi-

[1] https://www.kaggle.com/muhammedsal98/bank-marketing

menting with the course, we replace the class $\{0,1\}$ by $\{-1,1\}$.

## 4. Presentation of the problem

We face the problem with predicting two class $\{-1,1\}$, 1 for predicting the potential subscribing of the client and -1 otherwise. So, the logistic regression appears as good candidate and better suited for classification using the logistic loss. The problem is to minimize the empirical risk minimization (ERM) or objective function $f$ to obtain better approximations of the weights leading to optimal predictions of classes. Mathematically,

$$\text{minimize}_{\mathbf{w}\in\mathbb{R}^d} f(w) := \frac{1}{n}\sum_{i=1}^{n} f_i(\mathbf{w}), \text{where}$$

$$f_i(\mathbf{w}) = \log(1+\exp(-y_i\mathbf{x}_i^T\mathbf{w})) + \tfrac{\lambda}{2}\|\mathbf{w}\|^2, \text{ for Ridge}$$
$$f_i(\mathbf{w}) = \log(1+\exp(-y_i\mathbf{x}_i^T\mathbf{w})) + \tfrac{\lambda}{2}\|\mathbf{w}\|_1, \text{ for Lasso}$$

where every $y_i$ is a binary label in $\{-1,1\}$ and $\lambda \geq 0$ is a regularization parameter. At the beginning we don't consider the regularization term ($\lambda = 0$) involves by the lambda coefficient and we give more details about this in the section talking about **Test of regularization**.

## 5. Test of batch gradient descent

Assuming that the objective function is convex and L-smooth, $f \in \mathcal{C}_L^{1,1}$, the descent property holds and the gradient descent assumes to converges towards an optimal. The Gradient descent (GD) step algorithm follows this step:

$$\begin{aligned} k = 0, &\quad \text{pick } w_0 \in \mathbb{R}^d \\ k \geq 0, &\quad w_{k+1} = w_k - \alpha_k \nabla f(w_k) \end{aligned}$$

The choice of stepsize is crucial but the theory ensures that $\alpha = \frac{1}{L} < \frac{2}{L}$ leading to optimal value. At the point of view of theory, we have this formula for the convergence rate : $f(w_k) - f^* \leq \frac{L}{2}||w_0 - w^*|| \times \frac{1}{K} = O\left(\frac{1}{K}\right)$

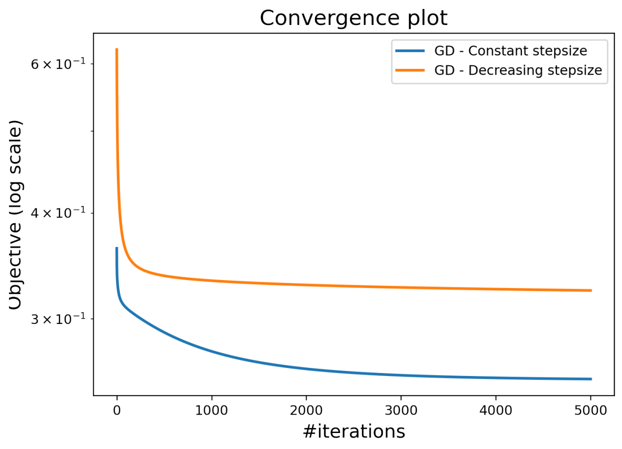

Figure 2: GD with descent stepsize exploration

For this figure above, we observe that the classical gradient descent takes the more time that the decreasing stepsize but we notice a rapid convergence to an optimal in the decreasing stepsize more than the constant stepsize. The convergence is slow in the case of constant stepsize and require more iterations to guarante better results which is not the case of the decreasing stepsize. The comparison has done with the constant stepsize of $0.6/L$ and the decreasing stepsize of $0.6/\sqrt{k+1}$.

## 6. Test of Accelerated gradient descent : Nesterov

The accelerated gradient descent techniques provides a faster algorithm in terms of speed convergence that relies only 1 gradient per iteration. We can distinguish two principals methods such that Heavy ball method(Gradient step before momentum step) and Nesterov (Momentum step before gradient step). We consider only Nesterov for the test of speed convergence and compare it to gradient descent.The description of the Nesterov method follows those steps :

$$\begin{aligned} k = 0, \text{pick } w_0 \in \mathbb{R}^d, w_{-1} = w_0 \\ k \geq 0, w_{k+1} = w_k - \alpha_k \nabla f(w_k + \beta_k(w_k - w_{k-1})) \\ +\beta_k(w_k - w_{k-1}) \end{aligned}$$

As we can see in the course, we have

$$f(w_k) - f^* \leq \frac{L}{2}||w_0 - w^*|| \times \frac{1}{K} = O\left(\frac{1}{K}\right)$$

for GD, and,

$$f(w_k) - f^* \leq \frac{2L||w_0 - w^*||^2}{(K+1)^2} = O(\frac{1}{K^2})$$

for Nesterov's test. So that the convergence speed is very faster in Nesterov.

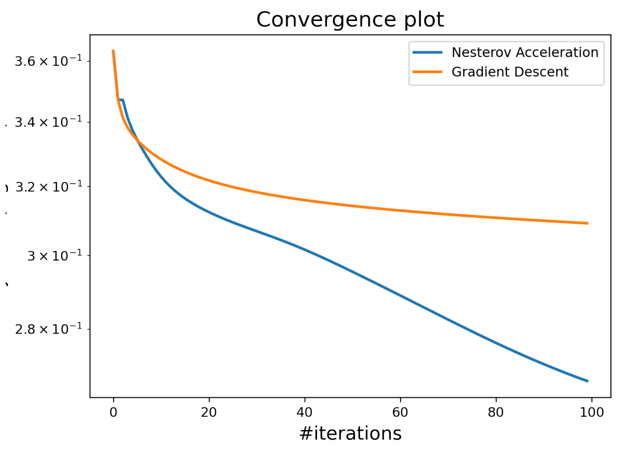

Figure 3: Test of Nesterov

The experiments confirm the results obtained in theory i.e, the Nesterov's test goes faster optimal value than the GD. When we compare the performance, we achieved the accuracy of 90.07% on the testing set with respect to Nesterov, better than the Gradient Descent with accuracy of 89.98%.

| | Training | Testing |
|---|---|---|
| Gradient Descent : | 89.79% | 89.98% |
| Nesterov test | 89.8% | 90.07% |

Table 1: Gradient Descent vs Accelerated Gradient Descent *(constant stepsize)*.

## 7. Test of stochastic gradient descent

The gradient descent provide more deterministically guarantees about the convergence to an optimal value. But in practice, in high dimensional data, the GD can be expensive on very large dataset. An alternative approach is the stochastic gradient. Instead of compute the $\nabla f(w_k)$ i.e all the $\nabla_i f(w_k)$s, we look at one $\nabla_i f(w_k)$ per iteration. The Stochastic gradient algorithm states the following update $w$:

$$\begin{aligned} k = 0, &\quad \text{pick } w_0 \in \mathbb{R}^d \\ k \geq 0, &\quad w_{k+1} = w_k - \alpha_k \nabla f_{i_k}(w_k) \end{aligned}$$

where $i_k$ is drawn uniformly in $\{1, \cdots, n\}$.

Unlike Gradient Descent, this method doesn't guarantee the decreasing of the objective function in monotone fashion. But it works very well in practice much better than the GD. Let's verify this on our dataset.

### 7.1. Gradient descent vs Stochastic gradient

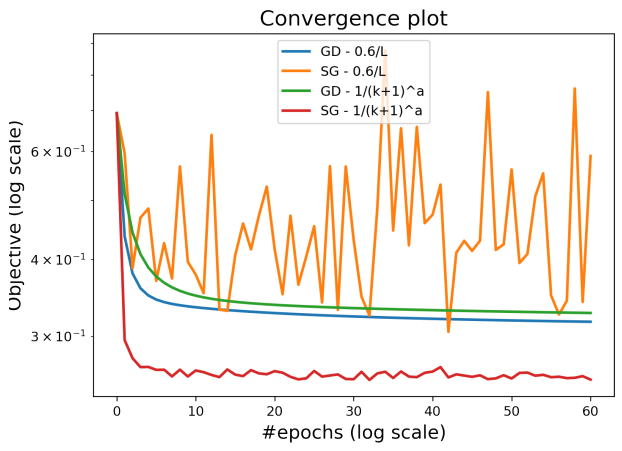

Figure 4: Gradient descent vs Stochastic Gradient

As you can see in the figure above, the first remark is that the stochastic gradient isn't a descent method like the gradient descent. The red curve and the yellow curve (SG) illustrates the common behaviour of SG that converges rapidly in first iterations and oscillating around an value which is not the optimal values. However, the SG provide much better results in the reasonable number of iterations than GD in the case of decreasing stepsize. We display the performance of the algorithms on the training and testing set.

| | Training | Testing |
|---|---|---|
| GD constant stepsize | 88.67% | 88.72% |
| GD decreasing stepsize | 88.30% | 88.47% |
| SG constant stepsize | 89.03% | 88.40% |
| SG decreasing stepsize | 89.85% | 90.02% |

Table 2: Accuracy Comparison

### 7.2. Experimenting with the learning rate

This section get one idea about the choice of the value of learning rate or stepsize. The theory behind the optimization technique doesn't decide the exact value of the learning rate. In practice, we can run several values of the learning and then check the impact of the variation of learning rate before select a good choice according to the data. We run several instances of standard stochastic gradient with constant step size proportional to $1/L$ and with decreasing step size proportional to $1/(k+1)^a$.

### 7.2.1 Constant step size

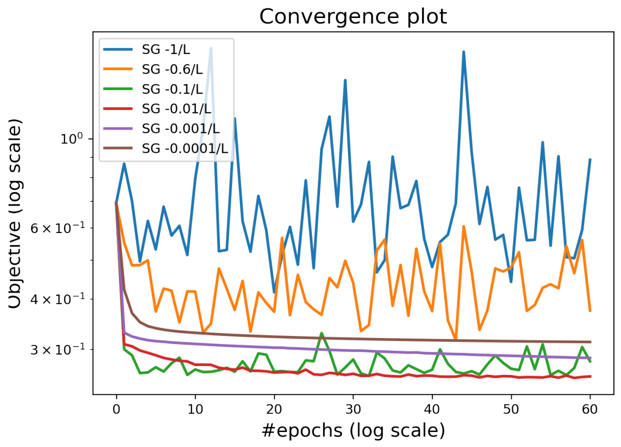

Figure 5: SG - Variation of the constant step sizes

We observe that choosing a "large" stepsize ($\frac{1}{L}$ and $\frac{0.06}{L}$) may prevent the method from converging. On the other hand, using a very small stepsize will lead to slow convergence (as illustrated by the $\frac{0.0001}{L}$ curve). The red and green curves ($\frac{0.1}{L}$ and $\frac{0.01}{L}$) 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).

### 7.2.2 Decreasing step size

We observe that rapidly decreasing stepsize sequence lead to a sublinear convergence rate, without any oscillation around a certain value.

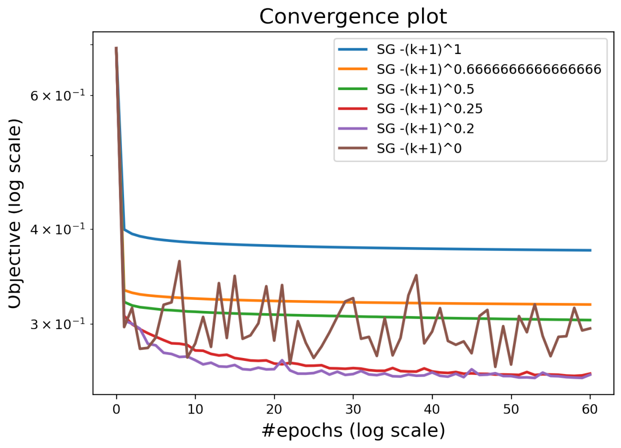

Figure 6: SG - Variation of the decreasing step sizes

This is the case for the choices $\alpha_k = \frac{1}{k+1}$ and $\alpha_k = \frac{1}{(k+1)^{2/3}}$. The lowest values of objective function are attained by using $\alpha_k = \frac{1}{(k+1)^{1/4}}$ or $\alpha_k = \frac{1}{(k+1)^{2/10}}$. In summary, we have the table above which compare the performance of stochastic gradient descent with decreasing step size on training and testing set.

| | Training | Testing |
|---|---|---|
| SG -(k+1)$^{0.25}$ : | 89.70% | 89.91% |
| SG -(k+1)$^{0.2}$ | 89.92% | 89.98% |

Table 3: Stochastic Gradient Decreasing stepsize

We achieve good precisions with this method over only 60 epochs (vs 500 epochs for the Gradient Descent). That proofs the success of Stochastic Gradient in practice.

## 7.3. Experimenting with the batch size

A more general version of stochastic gradient, called batch stochastic gradient (BSG), 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 the batch size. Its use stems from the fact that the accuracy of the stochastic gradient depends on the variability present in the data. According to the course, BSG appears as a variance reduction technique. For experiments, we display in the figure 7 the performance of stochastic gradient with several values for the batch size (and using constant stepsizes).

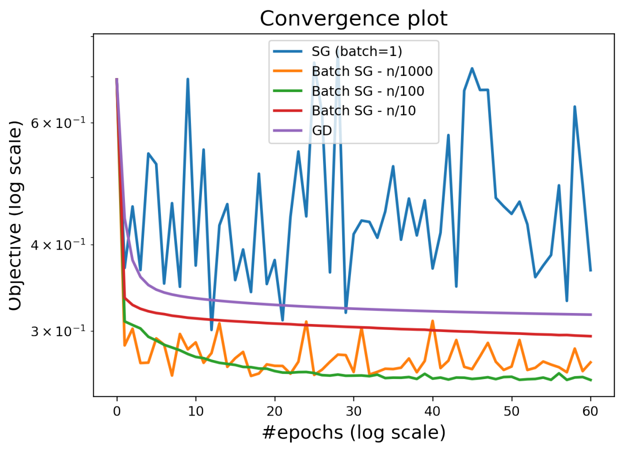

Figure 7: Impact of vary Batch size in SG

It can be seen from the figure that the introduction of the batch size reduces the variability of the stochastic gradient, which means that the oscillation phase is less important in the n/100 curve than in the n/1000 batch.

| | Training | Testing |
|---|---|---|
| Batch SG - n/1000 | 89.68% | 89.82% |
| Batch SG - n/100 | 89.91% | 89.84% |

Table 4: Batch Stochastic Gradient

## 7.4. Variance reduction

### 7.4.1 Stochastic gradient and averaging

To improve somehow the convergence speed, it is possible to average the past iterate, i.e. run a "classical" SGD on auxiliary variables $(\tilde{w_l})_l$. This defines the Stochastic Gradient Descent with Averaging (SGA) algorithm. Running this algorithm implies to store all the iterates which is not used in the computation. To avoid that, it is possible to avoid explicitely storing all the iterates by simply updating a running average as follow

$$w_{k+1} = \frac{1}{k}\tilde{w_k} + \frac{k-1}{k}w_k.$$

The sequence of $\{\tilde{w_k}\}$ has better theoritical properties than $\{w_k\}$ and it can behave more "deterministically" in practice.

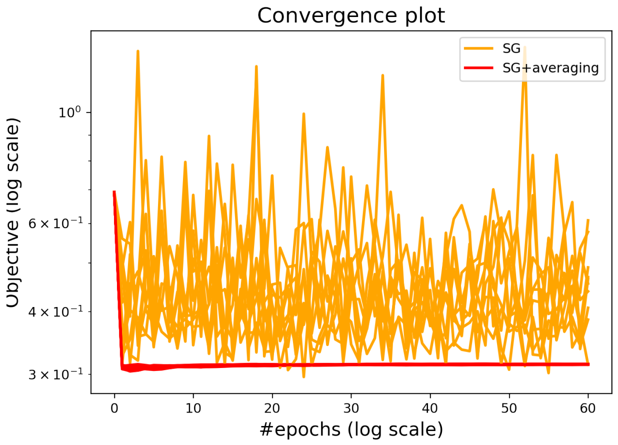

Figure 8: Stochastic gradient and averaging

Averaging the iterates can lead to a smoother behaviour in terms of function value and convergence.

### 7.4.2 Popular practical variants based on diagonal scaling

This section pay attention to the results of an implementation of variants of Batch Stochastic Gradient based on diagonal scaling. We focuses on two popular optimization algorithms which names are RMSProp, Adagrad.

$$[\mathbf{w}_{k+1}]_i = [\mathbf{w}_k]_i - \frac{\alpha}{\sqrt{[\mathbf{v}_k]_i + \mu}}[\nabla f_{i_k}(\mathbf{w}_k)]_i,$$

where $\mu > 0$ is a regularisation parameter, and $\mathbf{v}_k \in \mathbb{R}^d$ is defined recursively by

$$\mathbf{v}_{-1} = 0_{\mathbb{R}^d} \text{ and } \forall k \geq 0,\ \forall i = 1, \dots, d,$$

$$[v_k]_i = \begin{cases} \beta[v_{k-1}]_i + (1-\beta)[\nabla_{i_k}]_i^2, & \text{for RMSProp,} \\ [v_{k-1}]_i + [\nabla f_{i_k}(w_k)]_i^2, & \text{for Adagrad.} \end{cases}$$

(Suggested values: $\mu = \frac{1}{2\sqrt{n}}$, $\beta = 0.8$.). The figure 9 shows that the Deep neural networks optimization method such that Adagrad/RMSProp isn't very suited to our own dataset. RMSProp diverge and the Adagrad appears smoother after few iterations.

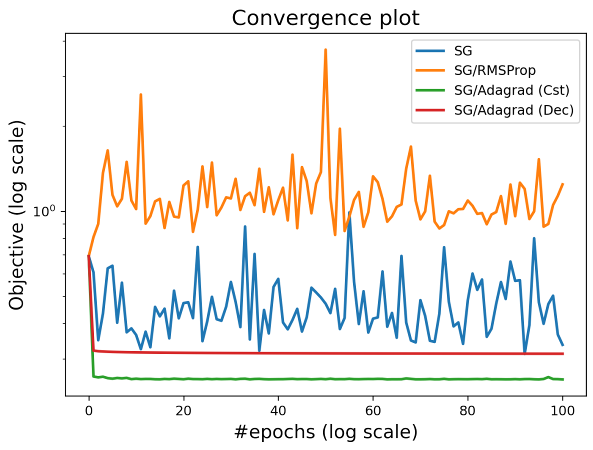

Figure 9: Diagonal Scaling - Stochastic gradient

## 8. Test of Regularization

In this section, we focuse on the regularization term ($\lambda \neq 0$). The regularization techniques are an efficient method to avoid overfitting if the training dataset is not big enough by penalizing the objective functions. We choose the value $1/\sqrt{10000}$ for lambda for the rest of the test.

## 8.1. Ridge regularization

The objective function has the following form $f_i(\mathbf{w}) = \log(1 + \exp(-y_i \mathbf{x}_i^T \mathbf{w})) + \frac{\lambda}{2}\|\mathbf{w}\|^2$. The ridge regression reduces the variation as long as the number of iterations increase. This explains the constant behavior we shall visualize in the curve for large iterations.

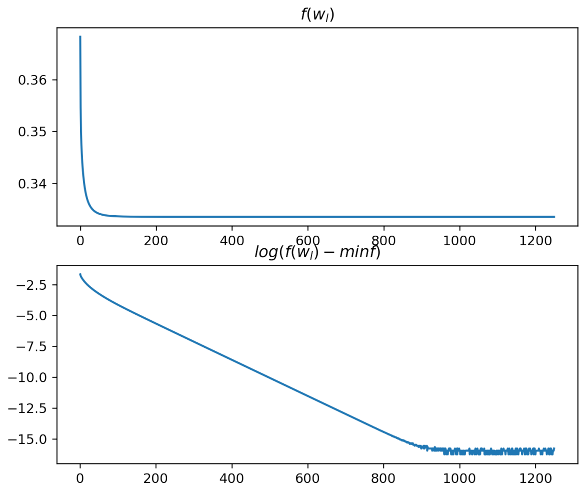

Figure 10: Ridge regularization

We notice that in log domain, the convergence is faster in log domain and for large iterations the objective approaches asymptotically the optimal value.

### 8.2. Lasso regularization

The Lasso regularization can be express mathematically as: $f_i(\mathbf{w}) = \log(1 + \exp(-y_i \mathbf{x}_i^T \mathbf{w})) + \frac{\lambda}{2}\|\mathbf{w}\|_1$.

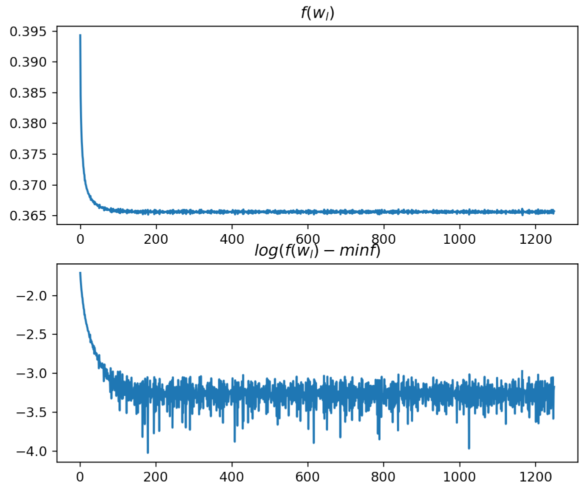

Figure 11: Lasso regularization

The figure above shows that the objective in term of log domain is converge in slow rate follows by an oscillating phase meaning that the distance to minimum still in large confidence interval. We compare the real impact of using the Regularization in our dataset. We use the metric of "accuracy".

| | Training | Testing |
|---|---|---|
| Gradient Descent | 89.29% | 89.36% |
| L2 + GD | 88.70% | 88.74% |
| L1 + GD | 88.48% | 88.63% |

Table 5: Accuracy Comparison

## 9. Scikit Learn

We display the validation curve obtained by using Scikit-Learn and logistic regression model with saga optimization method. we reach with scikit learn an accuracy of 90.02% which is not too far from the performances obtained with the implementation of the different optimization methods in particular the Stochastic gradient.

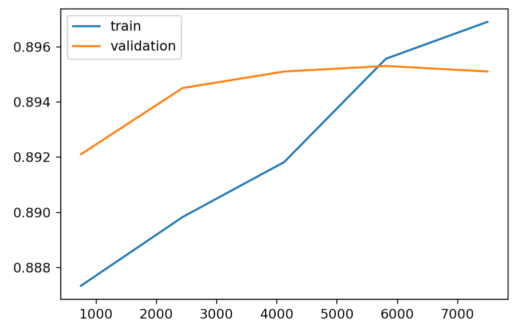

Figure 12: Validation curve with SAGA

## 10. Conclusion

This project was the opportunity to put into practice all the theoretical aspects seen in the course but especially to confront them with the results obtained on a real data set. In a global way, we had found or at least led to the same results seen in the course such as the speed of convergence of the optimization algorithms, the gain of the execution time or the reduction of the variance induced by the Stochastic gradient batch. A small part was devoted to the use of the **Scikit-learn** library for the comparison with the empirical results obtained which are close to an error of $10^{-3}$. Nevertheless, the challenge has not been met with respect to deep learning oriented optimization methods such as Adagrad or RMSProp which diverge or show stationarity after a few iterations. In perspective, it would also be interesting to test other popular deep learning optimisation algorithms such as Adam and evaluate their convergence speed.

## References

[1] G. Peyré, (2021), "Course notes on Optimization for Machine Learning". https://mathematical-tours.github.io/book-sources/optim-ml/OptimML.pdf.

[2] G. Peyré, Numerical tour on logistic classification [online], http://nbviewer.ipython.org/github/gpeyre/numerical-tours/blob/master/python/ml_3_classification.ipynb

[3] C. Royer, (2021), *Advanced aspects of gradient-type methods. The two lectures on gradient descent*, https://www.lamsade.dauphine.fr/~croyer/teachGD.html

[4] C. Royer, (2021), Advanced aspects of gradient-type methods.*The three lectures on stochastic gradient*, https://www.lamsade.dauphine.fr/~croyer/teachSG.html

## Contents