Optimization Project

ELGHOURBAL Salma
M2 IASD

This project consists in implementing optimization algorithms and ML techniques in the context of supervised learning. We will use the dataset `dataset.csv` that you will find in the project zip folder. It is a synthetic dataset created by some website. Each row represents a user and his/her features and whether He/She Clicked on some advertisement or closed It on that website at some time. The link of the dataset is : https://www.kaggle.com/fayomi/advertising. Our goal is to use the features in order to predict if a user would click on an advertisement. Here is a description of the variables of the dataset :

Variable :	Meaning
Daily Time Spent on Site:	consumer time on site in minutes
Age:	customer age in years
Area Income:	Avg. Income of geographical area of consumer
Daily Internet Usage:	Avg. minutes a day consumer is on the internet
Ad Topic Line:	Headline of the advertisement
Male:	Whether or not consumer was male
City:	City of consumer
Country :	Country of consumer
Timestamp:	Time at which consumer clicked on Ad or closed window
Clicked on Ad:	0 or 1 indicated clicking on Ad

Figure 1: Attributes of the dataset

Originally, the data contains $n = 5000$ examples and 9 variables other than the Clicked on Ad variable.

Problem setting : For the sake of simplicity, I only considered the following numerical attributes : Daily time Spent on Site, Age, Area Income, Male and Daily internet usage. So the input space is $\mathbb{R}^d$, with $d = 5$

Given train data A and train labels y, y being labels of examples in A , $y \in \{0, 1\}$ (Clicked on Ad variable) ,

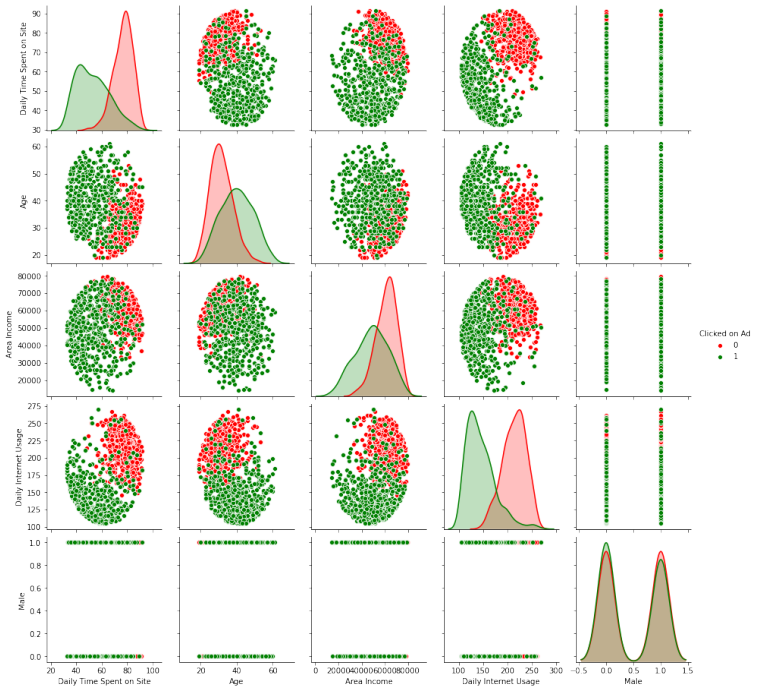
We try to train a binary classification model h to map A to y. We are going to test the generalization capacity of the model h on test data A1 and test labels y1.

1 Data Exploration and Visualization

First of all, we checked that there are no missing values and that the two classes contain the same number of individuals. Thus, we don't have a problem of unbalanced datasets.

Secondly, we drew a pairplot so as to gain some insight on the data distribution :

Figure 2: Pairplot



From the pairplot 2, we can observe :

- Users who do not click on ads are majoritarily young users (< 40) ; The distribution of users who click on ads is centered on age 40.
- The majority of users who do not click on ads belong to an area with high income.
- Users who click on ads mostly use internet less than users who do not click on ads.
- Users who click on ads mostly surf on the site less than users who do not click on ads.

We split the data into train and test sets . We normalized It and standardized It. In addition, we computed attributes correlation and 2D and 3D Projection with PCA. The figure 3 shows a correlation plot and figure 4 shows a 3D PCA.

Figure 4: 3D PCA

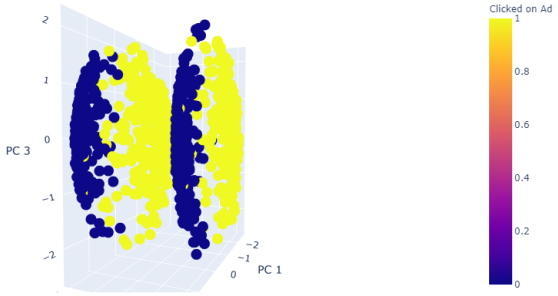


Figure 3: Feature Correlation

As we can see in the 3D projection 4 retrieved by PCA, the data is quite simple. It seems that the two classes are even almost linearly separable.. The explained variance conserved by PCA is of 77.357% which is quite good. That means our previous remark is valid. According to the matrix of correlation 3, attributes are not highly correlated between them; but Daily Internet Usage is a bit strongly and positively correlated with daily time on site, then secondly comes the area income which is positively correlated with Daily Internet Usage as well as Daily time spent on site.

2 Mandatory experiments

2.1 Logistic regression with full Gradient and Constant stepsize

Recall the Optimization problem for Logistic regression and also the formulas of ∇f and ∇^2 f (Formulas taken from M.Clément Royer lab for SGD)

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

where every y_i is a binary label in $\{-1, 1\}$ and $\lambda \geq 0$ is a regularization parameter. The gradients are as follows :

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

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

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

I remembered to adapt the values of y so as $y \in \{-1, 1\}$ instead of {0,1}. As seen in the course, To ensure theoretically the converence, Stepsize should be chosen such as

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

where : $L = \frac{\|A^T A\|}{4n}$ is a Lipschitz constant for ∇f when there is no regularization In our case, $\tau_{max} = 2/L = 3.91$. We implemented Gradient descent algorithm with constant stepsize $\tau = 1/L$

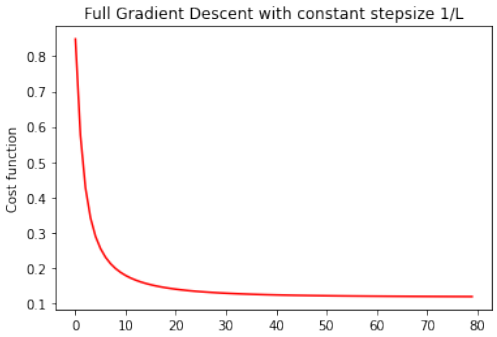


Figure 5: Loss function w.r.t iterations for GD

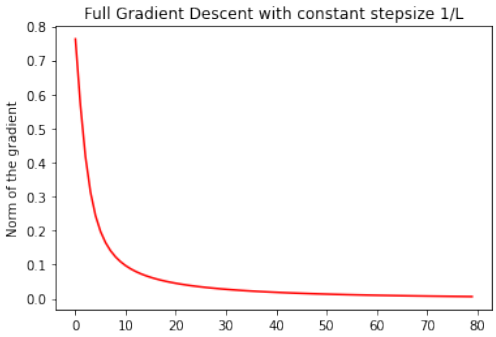
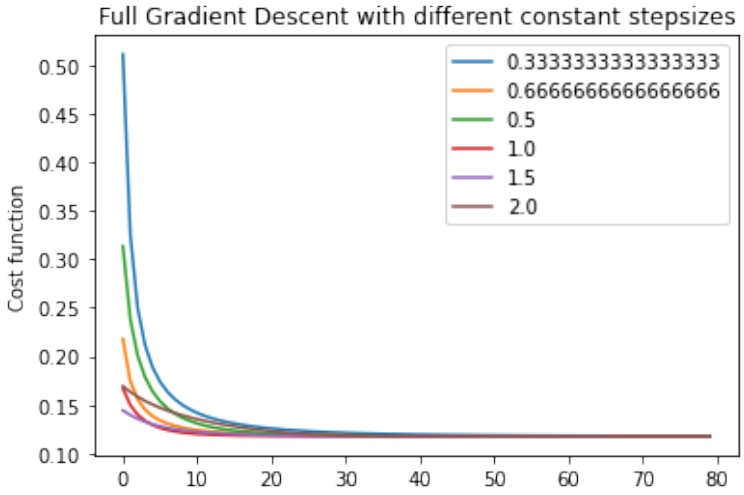


Figure 6: Norm of the gradient w.r.t iterations for GD

Figure 7: Stepsize impact on GD convergence rate



The two plots show that GD converges within 70 iterations. We can see that the norm of the gradient is converging to zero. We studied the impact of descent step size on Gradient descent. We obtained different curves for different values of stepsize in figure 7

The line with value 0.333 means that we used a $\tau = \tau_{max}/3$ and so on. This plot 7 shows that as we increase the stepsize, the loss drops more rapidly. However, if we use a stepsize higher than τ_{max}, we don't have theoretical garantees of convergence. In the case of this simple dataset, the algorithm still converges..

We saw in the course that in the case of smooth functions (Course of Gabriel Peyré or Irène Waldspurger), we have a sublinear convergence rate. In the plot 8, we draw $\log(f(x_k) - \min(f(x_0), f(x_1)...f(x_n))$ for different values of τ

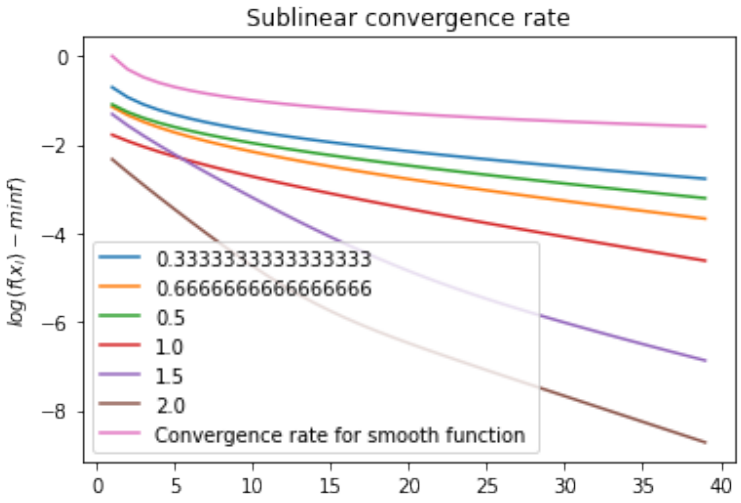
Figure 9: Nesterov Algorithm (caption taken from the course : Advanced Gradient descent)

$$x_{t+1} = y_t - \frac{1}{L}\nabla f(y_t);$$
$$y_{t+1} = x_{t+1} + \gamma_t(x_{t+1} - x_t),$$

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 (and admittedly mysterious, at first sight) definition, is

$$\lambda_{-1} = 0,$$
$$\forall t \in \mathbb{N}, \quad \lambda_t = \frac{1 + \sqrt{1 + 4\lambda_{t-1}^2}}{2},$$
$$\forall t, \quad \gamma_t = \frac{\lambda_t - 1}{\lambda_{t+1}}.$$

Figure 8: Convergence rate for Gradient descent



We saw that logistic function is a smooth function but not necessarily strongly convex. So the convergence rate is shown to be of O(1/k). The curves for $\tau < \tau_{max}$ are quite parallel to the line of the convergence rate..

2.2 Accelerated gradient descent (Nesterov)

We refered to Irène's formulation of Nesterov method for implementation : figure 9

According to the course, In the case of Nesterov Gradient acceleration and for L smooth functions, we have a convergence rate of $1/k^2$ instead of $1/k$ like GD. For this to be garanteed, we have to use a stepsize $= 1/L$ The convergence rate of Nesterov in log scale should be

of $O(2 * log(1/k))$ The plots 10 and 11 show a comparison between Gradient descent and Nesterov's Algorithm.

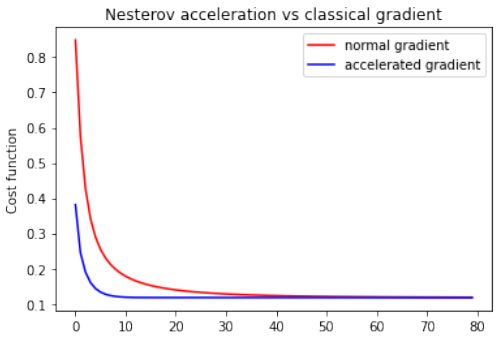


Figure 10: Loss function w.r.t iterations for Nesterov and GD

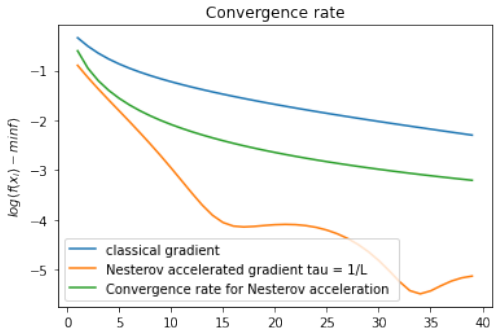


Figure 11: Convergence rate in logscale

As It is expected, the Nesterov algo converges faster than simple Gradient. Furthermore, we obtain a convergence rate better that the theoretical one..

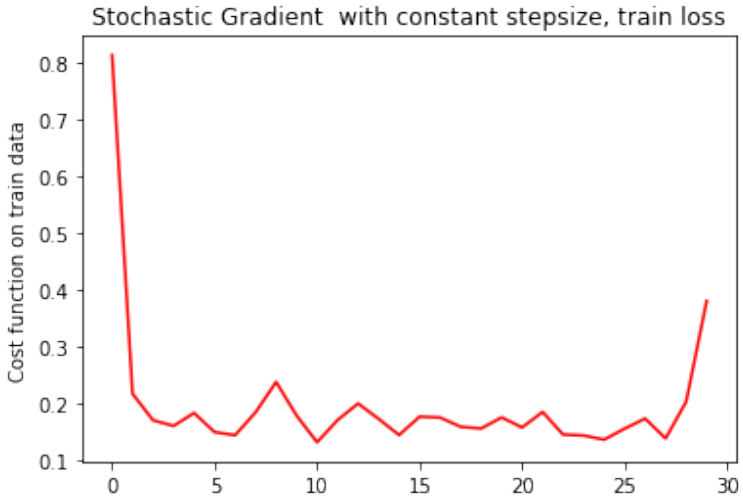
2.3 Mini batch gradient and stochastic Gradient

We implemented Stochastic gradient and also mini-batch stochastic gradient. We saw in Clément's course that in the case of Stochastic gradient, we need some assumptions among which strong convexity for proving theoretical convergence rates. Here, we only choose a stepsize $\tau < 1/L$ but we're not sure to derive the course results. However, some reasonable results were retrieved. The plot 12 shows that stochastic gradient is unstable (noisy) and that It is not a descent method since the loss value can go up and down..

Impact of batch size

We implement Stochastic gradient with different values of batch size and we draw the evolution of the loss function on the train set as well as on the test set (figures 13 and 14). We remembered to give the same budget of iterations to the algorithms . We draw the curves with respect to the number of epochs (1 epoch = 1 pass on the train data)

Figure 12: Stochastic gradient



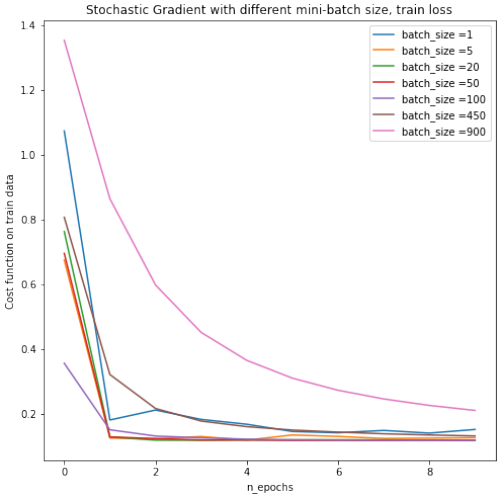


Figure 13: Train Loss function w.r.t epochs for different batch size

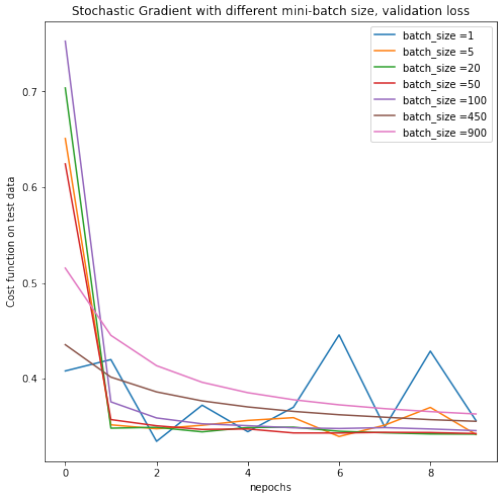


Figure 14: Test Loss function w.r.t epochs for different batch size

In figures 13 and 14, we can see that mini batch algos with intermediate values of mini batch size (5,20,50) have lower train and test error and better convergence rate than mini batch algos with very high batch size (>100) or classical stochastic gradient (batch size = 1). This result is a practical and classical result in Stochastic schemes.

Impact of learning rate

We studied the impact of learning rate on the convergence of mini batch stochastic gradi-

ent with batch size = 20 . The plots 15 and 16 show the evolution of train loss and test loss during training. We can see that for very low learning rates, the convergence on both training set and test set is slow. On the contrary, for larger learning rates that still verify($\tau < 1/L$) the descent on both datasets is faster.

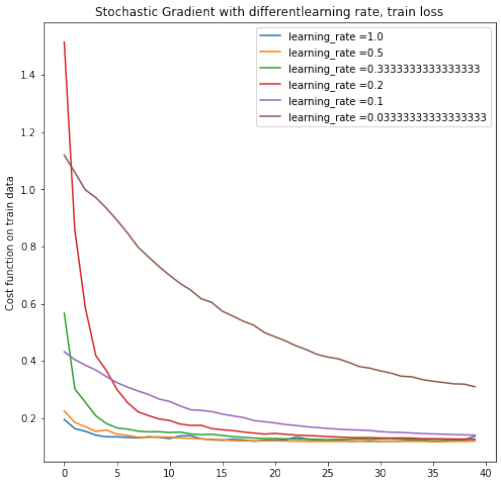


Figure 15: Train Loss function w.r.t iterations for different stepsizes

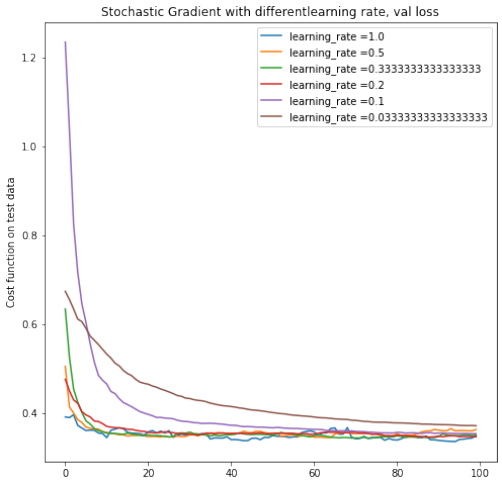


Figure 16: Test Loss function w.r.t iterations for different stepsizes

Test of Diagonal scaling

The intuition behind Diagonal scaling is to Decay the learning rate for parameters in proportion to their update history (more updates means more decay). We implemented two quite similar methods : Adagrad and RMSPROP. The only difference between the two is that RMSPROP decays less aggressively the learning rate compared to Adagrad. Recall the algorithms as were introduced in the course :

RMSPROP for each $i = 1, ..d$

$$[R_{-1}]_i = 0$$

$$[R_k]_i = (1 - \lambda). \cdot [R_{k-1}]_i + \lambda \cdot [g_k]_i^2$$

$$[w_{k+1}]_i = [w_k]_i - \alpha \cdot (1/\sqrt{[R_k]_i + \mu}) \cdot [g_k]_i$$

ADAGRAD for each $i = 1, ..d$

$$[R_{-1}]_i = 0$$

$$[R_k]_i = [R_{k-1}]_i + [g_k]_i^2$$

$$[w_{k+1}]_i = [w_k]_i - \alpha \cdot (1/\sqrt{[R_k]_i + \mu}) \cdot [g_k]_i$$

We chose $\alpha = tau = 1/(2 * L)$, $\lambda = 0.8$ and $\mu = 0.01$. The resulting curves are 17 and 18. We see that diagonal scaling may enhance the performance of stochastic gradient. Here for

instance, RMSPROP converges faster than Stochastic Gradient. Yet, ADAGRAD fails to do so and converges even slower than SG

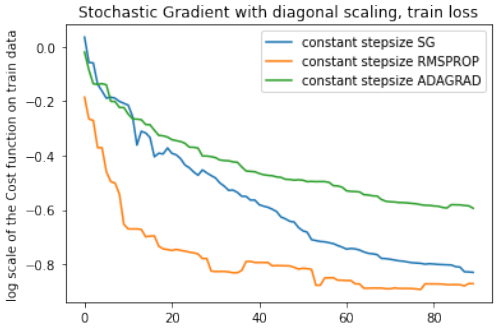


Figure 17: Train Loss function w.r.t iterations for SG, ADAGRAD and RMSPROP

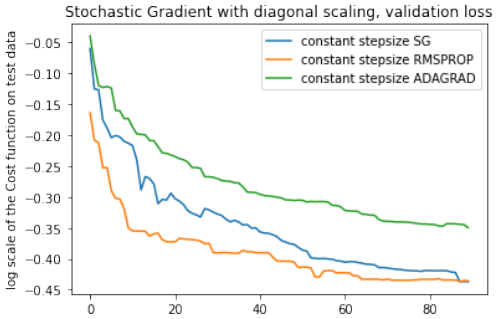


Figure 18: Test Loss function w.r.t iterations for SG, ADAGRAD and RMSPROP

2.4 Ridge and lasso Regularization

We tested Ridge and Lasso on this problem even if We think Regularization is not necessary since We have only 5 features (n»p).

RIDGE

The Ridge or $\ell 2$ regularization is a very popular method for regularization. The implementation is quite simple, we only neeed to specify some value of the regularization parameter and plug IT in the gradient descent function. The graphic of the evolution of the loss function for some given $\lambda < 0.5$ is similar of that of classic Gradient descent. So, It's not interesting to report It in this report. Also, the evolution of the minimum cost function on test set w.r.t λ values was not very interesting . It says that the best cost is given for $\lambda = 0$ which means for no regularization.. Please refer to my notebook in case you want to check the curves.
However, we report here the regularization paths so as we could compare Ridge effect with Lasso effect later. In the figure 19, we plot the resulting weight vectors for each feature w.r.t the values of λ. As we've seen with Gabriel Peyré, the Ridge doesn't force the coefficient to be null , It only reduces their amplitudes altogether, except here for the male variable because maybe It's not so relevant as a feature..

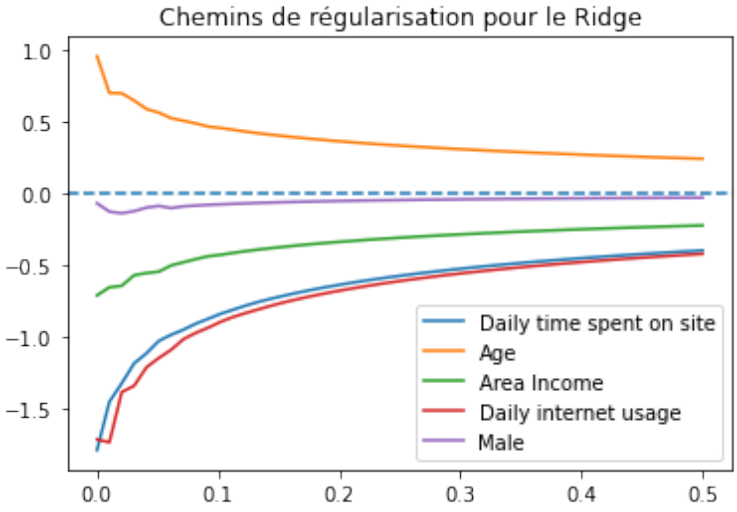
LASSO

The $\ell 1$ regularization also known as lasso

$$\min_{w \in \mathbb{R}^d} f(w) := \frac{1}{n} \sum_{i=1}^{n} f_i(w),$$

$$f_i(w) = \log(1 + \exp(-y_i x_i^T w)) + \lambda \cdot \|w\|_1$$

Figure 19: Regularization paths for Ridge



makes the objective function not differentiable. However, It exists an efficient algorithm to approximate iteratively the solution. We apply Forward Backward algorithm as explained in the course using the Proximal operator :

$$x_{k+1} = Prox_{\tau \cdot g}(x_k - \tau \cdot \nabla(f(x_k)))$$

We have an algorithm similar to ISTA for Lasso linear regression when we used Soft thresholding as the proximal operator, we just had to change the $A.T.(Ax - y)$ by the gradient of the logistic function ∇f :

$$x_{k+1} = Soft_{\tau \cdot \lambda}(x_k - \tau \cdot \nabla(f(x_k)))$$

. we fixed $\tau < 2/L$ so as to ensure theoretical convergence. The figure 20 shows the loss function on train data and the objective function value on test set without the regularization part. The algorithm succeeds in decreasing the train loss. However, the test loss is higher. So as we mentioned before, regularization is not needed for this problem.
In the plot 21, We draw the regularization paths for Lasso in order to study the effect of λ on the sparsity of the final weight vector returned by the Forward Backward algorithm.

Figure 20: Train Loss Test loss function w.r.t iterations for Lasso

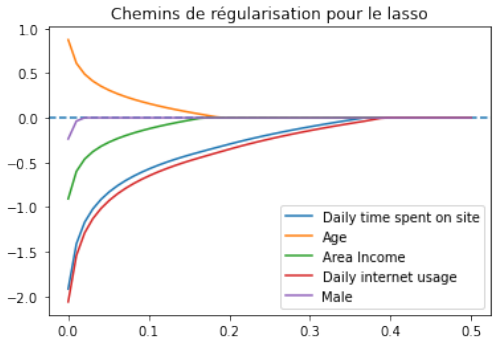


Figure 21: Regularization paths for lasso w.r.t lamda

The figure 21 is coherent with what we saw in the Regularization course. In fact, the Lasso forces some coefficients to zero when we increase the parameter λ . In our case, if we go left to right, the first feature to reach zero is Male, after that, age and area income and finally the two remaining features Daily internet usage and daily time spent on site.

3 Optional experiments

3.1 Newton second order method

Recall the algorithm of Newton :

$$x_{k+1} = x_k - H_k \cdot (f(x_k))$$

In Newton algorithm, $H_k = [\partial^2(f(x_k))]^{-1}$. We implemented this second order algorithm and compared Its convergence speed with Classical Gradient descent. The plots 22 and 23 represent the convergence rate for both algos and also the cost function w.r.t the number of iterations.. The Newton algorithm has the advantage to converge a lot faster that Gradient descent with stepsize $= 1/2L$. However, this method takes more computation time at each iteration. It can be sometimes expensive to compute the hessian especially for large dimension p .

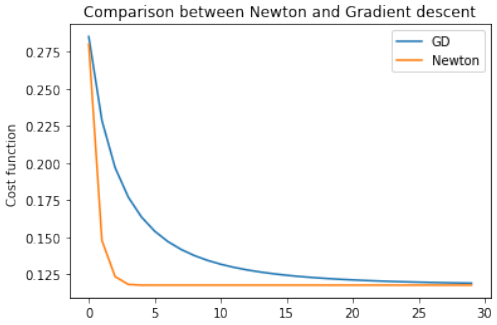


Figure 22: Train Loss function w.r.t iterations for Newton and GD

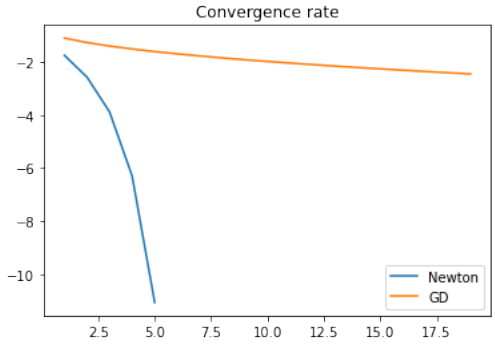


Figure 23: Convergence rate in logscale

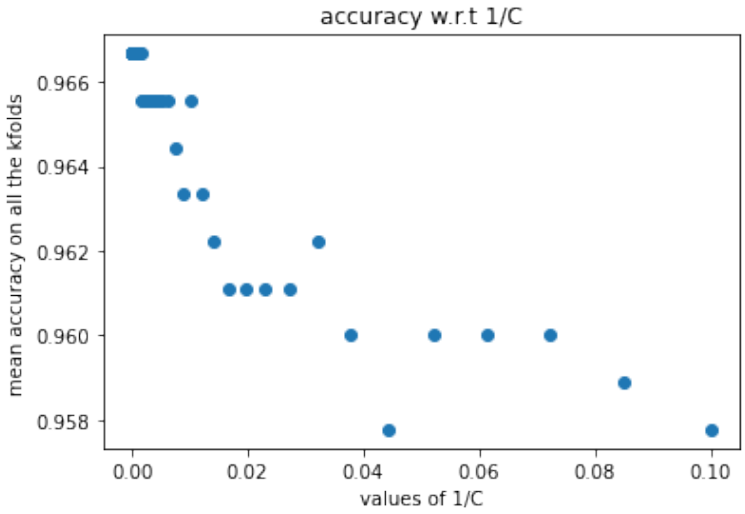
3.2 Support vector machines

In this section, we revisit the Support vector machines large margin classifier. The primal optimization problem can be expressed as follows :

$$\begin{cases} min_{w,b,\xi_i} & \frac{1}{2}||w||^2 + C\sum_{i=1}^{n}\xi_i \\ \text{s.t} & y_i(w^T x_i + b) \geq 1 - \xi_i, \ \forall i = 1...n \\ \text{and s.t} & \xi_i \geq 0, \ \forall i = 1...n \end{cases} \tag{1}$$

C is a hyperparameter that represents the compromise between large margin and error tolerance. We implemented this method. First, We ran this algo on 2D data (we only considered the attributes Daily internet usage and daily time spent on site) for the sake of visualization. Afterwards, we ran SVM on all the features. We applied cross validation for hyperparameter selection so as to find an optimal value of C. We also evaluated the best algo on the test data using classification metrics..
Figures 24 and 25 represent the 2D data and the decision function retrieved by SVM for two different values of C . Large values of C ==> small margin and less erros, while big values of C ==> big margin and a lot of errors..

Figure 26: 2-fold crossvalidation for C selection



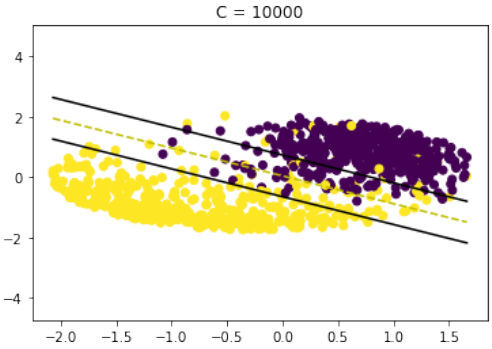


Figure 24: Decision boundary for big C

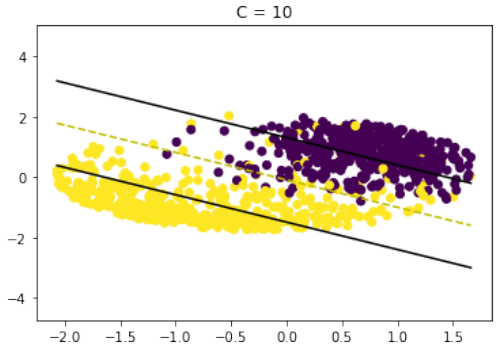


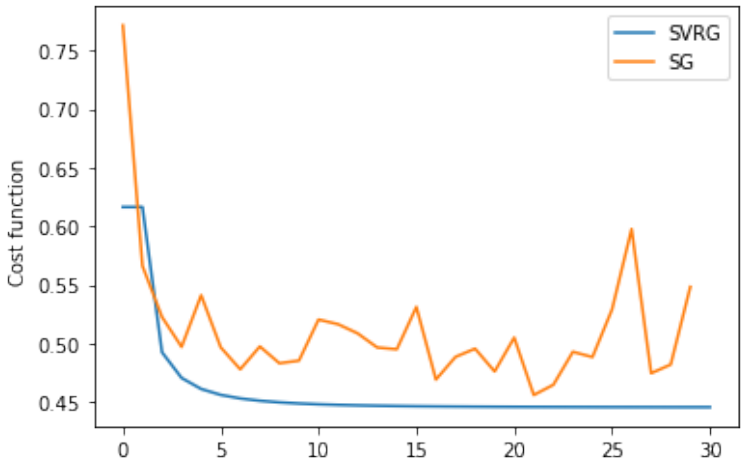
Figure 25: Decision boundary for small C

The cross validation made on all the features of train data yields the curve shown in 26. We implemented this crossval using StratifiedKFold module of sklearn. It shows that for very high values of C, we have a the highest value of mean accuracy. Here is the confusion matrix of the selected model on the unseen test data : $(C = 10^8)$ [[57. 0.],[1. 42.]], this means the algo has only one error on the test set (false positive).

3.3 Variance reduction method SVRG

We implemented the method SVRG-Stochastic variance reduced gradient for variance reduction. We also tried SAGA but the results weren't interesting.. The algo is presented in

Figure 27:



Clément's course (figure 28). The use of SVRG estimate leads to a better bound for the variance of the stochastic gradient. Moreover, one can show that it indeed achieves a linear rate in terms of iterations. In the plot 27, we can see that SVRG doesn't oscillate much unlike SG and that It reaches a lower bound than SG.

3.4 Stochastic gradient with momentum

The Momentum technique uses the following update : $w_{k+1} = w_k - \alpha \cdot \nabla(f_{ik}(w_k)) + \beta \cdot (w_k - w_{k-1})$. We implemented Momentum for different values of β and we fixed α equals $1/2L$. The algorithm was run for 10.000 times and we plotted the average cost function so as to obtain robust curves. In the figure 29, we can see that the lower beta is, the better. However Momentum doesn't improve the stochastic gradient (The difference between the losses is not significant).

Figure 28:

Algorithm Basic SVRG method.

Initialization: $\boldsymbol{w}_0 \in \mathbb{R}^d$, $\alpha > 0$, $m \in \mathbb{N}$.
for $k = 0, 1 \ldots$ **do**
  Compute the full gradient $\nabla f(\boldsymbol{w}_k) = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(\boldsymbol{w}_k)$
  Set $\tilde{\boldsymbol{w}}_0 := \boldsymbol{w}_k$.
  for $j = 0, \ldots, m-1$ **do**
    Draw a random index i_j uniformly in $\{1, \ldots, n\}$.
    Set $\tilde{\boldsymbol{g}}_j := \nabla f_{i_j}(\tilde{\boldsymbol{w}}_j) - \nabla f_{i_j}(\boldsymbol{w}_k) + \nabla f(\boldsymbol{w}_k)$.
    Set $\tilde{\boldsymbol{w}}_{j+1} := \tilde{\boldsymbol{w}}_j - \alpha \tilde{\boldsymbol{g}}_j$.
  end
  Draw j uniformly at random in $\{0, \ldots, m-1\}$ and set $\boldsymbol{w}_{k+1} = \tilde{\boldsymbol{w}}_{j+1}$.
end

Figure 29: Momentum with different values of beta