PTML 4: 15/04/2022

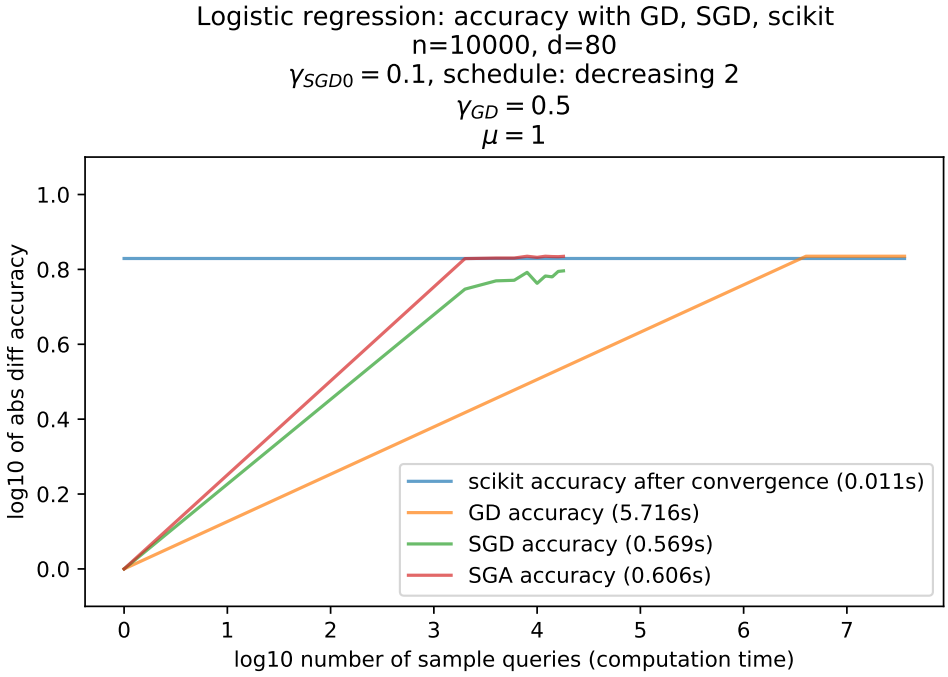


TABLE DES MATIÈRES

1 NUMPY MATRIX OPERATIONS

Run the notebook **np_matrix_operations.ipynb**. The goal is to emphasize the difference between matrix multiplication (produit matriciel) and element-wise multiplication (Hadamard product). Both are often useful.

— matrix multiplication is performed with **numpy.matmul**. https://numpy.org/doc/stable/reference/generated/numpy.matmul.html
"@" can be used as a shorthand.

— Hadamard multiplication is performed with **numpy.multiply**.
https://numpy.org/doc/stable/reference/generated/numpy.multiply.html
"*" can be used as a shofthand.
https://en.wikipedia.org/wiki/Hadamard_product_(matrices)

Conclusion : be careful with the shapes of the input arrays that you give to these methods !

2 GRADIENT DESCENT ON A LEAST-SQUARES PROBLEM

In this section the setting is the same as in the first section of TP3. If you prefer you can do part 3 before.

2.1 The heavy-ball method

When κ is very large, the convergence might become very slow. Some methods exist in order to speed it up, such as **Heavy-ball.** This method consists in adding a **momentum term** to the gradient update term, such as the iteration now writes

$$\theta_{t+1} = \theta_t - \gamma \nabla_{\theta_t} f + \beta(\theta_t - \theta_{t-1}) \tag{1}$$

The update $\theta_{t+1} - \theta_t$ is then a combination of the gradient $\nabla_{\theta_t} f$ and of the previous update $\theta_t - \theta_{t-1}$. The goal of this method it might balance the effet of oscillations in the gradient.

We will use these parameters :

$$\gamma = \frac{4}{(\sqrt{L} + \sqrt{\mu})^2} \tag{2}$$

and

$$\beta = \left(\frac{\sqrt{L} - \sqrt{\mu}}{\sqrt{L} + \sqrt{\mu}}\right)^2 \tag{3}$$

The heavy-ball method is called an *inertial method*. When f is a general convex function (not necessary quadratic), some generalizations exist, such as **Nesterov acceleration.**

2.1.1 *Impact on convergence rate*

Assuming $\mu > 0$, we will show that the characteristic convergence time with the heavy-ball momentum term is $\sqrt{\kappa}$ instead of κ.

Let λ be an eigenvalue of H and u_λ a eigenvector for thie eigenvalue. We are interested in the evolution of $\langle \theta_t - \eta^*, u_\lambda \rangle$.

We note

$$a_t = \langle \theta_t - \eta^*, u_\lambda \rangle \tag{4}$$

Exercice 1 : Show that

$$a_{t+1} = (1 - \gamma\lambda + \beta)a_t - \beta a_{t-1} \tag{5}$$

Exercice 2 : Compute the constant-recursive sequence a_t, and show that there exists a constant C_λ that depends on the initial conditions (trough A and B, and a_0), such that

$$\forall t, a_t \leqslant (\sqrt{\beta})^t C_\lambda \tag{6}$$

https://en.wikipedia.org/wiki/Constant-recursive_sequence

If u_i is a basis of orthogonal normed vectors with eigenvalues λ_i, we have that

$$\begin{aligned} \|\theta_t - \eta^*\|^2 &= \sum_{i=1}^{d} (\langle \theta_t - \eta^*, u_i \rangle)^2 \\ &\leqslant \sum_{i=1}^{d} (\sqrt{\beta})^{2t} C_{\lambda_i} \\ &= (\sqrt{\beta})^{2t} D \end{aligned} \tag{7}$$

with

$$D = \sum_{i=1}^{d} C_{\lambda_i} \tag{8}$$

We can now remark that

$$\begin{aligned} \sqrt{\beta} &= \frac{\sqrt{L} - \sqrt{\mu}}{\sqrt{L} + \sqrt{\mu}} \\ &= \frac{1 - \sqrt{\frac{\mu}{L}}}{1 + \sqrt{\frac{\mu}{L}}} \\ &\leqslant 1 - \sqrt{\frac{\mu}{L}} \\ &= 1 - \frac{1}{\sqrt{\kappa}} \end{aligned} \tag{9}$$

Finally, as $1 - \frac{1}{\sqrt{\kappa}} \leqslant \exp(-\frac{1}{\sqrt{\kappa}})$,

$$\|\theta_t - \eta^*\|^2 = \mathcal{O}(\exp(-\frac{2t}{\sqrt{\kappa}})) \tag{10}$$

Conclusion : with the heavy-ball momentum term, we changed the convergence rate of $\mathcal{O}(\exp(-\frac{2t}{\kappa}))$ to a convergence rate of $\mathcal{O}(\exp(-\frac{2t}{\sqrt{\kappa}}))$. This means that characteristic convergence time went from κ to $\sqrt{\kappa}$. If κ is large, which is the case we are interested in, this can be a great improvement.

Remember that $\kappa = \frac{L}{\mu}$, and that μ may be very small when n or d is large. For instance, in the case of Ridge regression, we have seen in the previous session that for instance, μ can be of order $\mathcal{O}(\frac{1}{\sqrt{n}})$ (see the computation of the optimal regularisation parameter). Hence, κ may be of order $\sqrt{n}$ or higher.

2.1.2 *Simulation*

Exercice 3 : Use the file **TP4_heavy_ball.py** to implement the Heavy-ball method and compare the convergence speed results to that of GD. You should obtain something like figures 1 and 2.

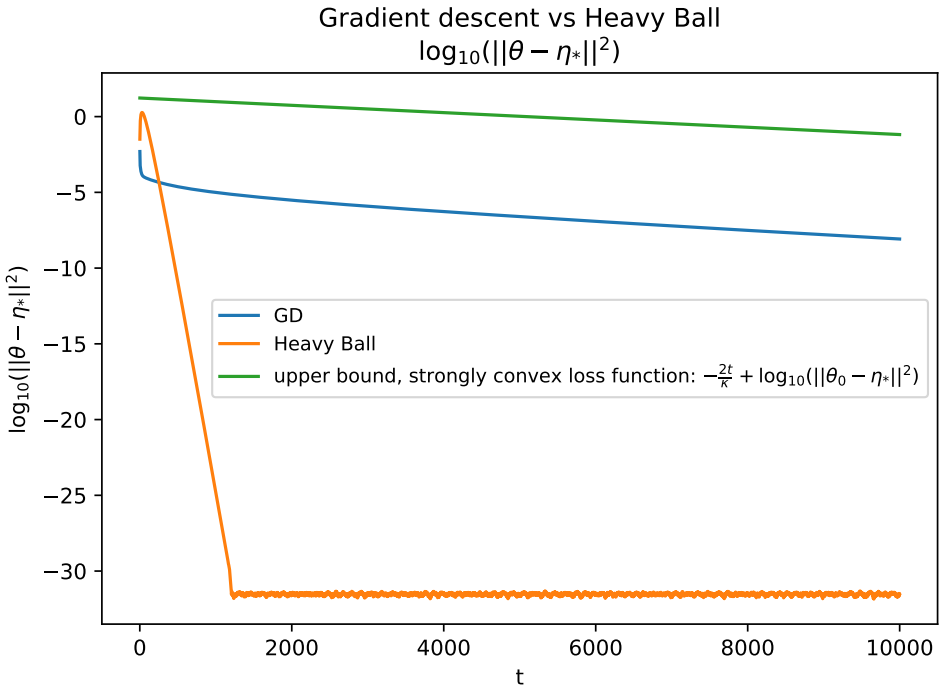


FIGURE 1 – Heavy-ball vs GD

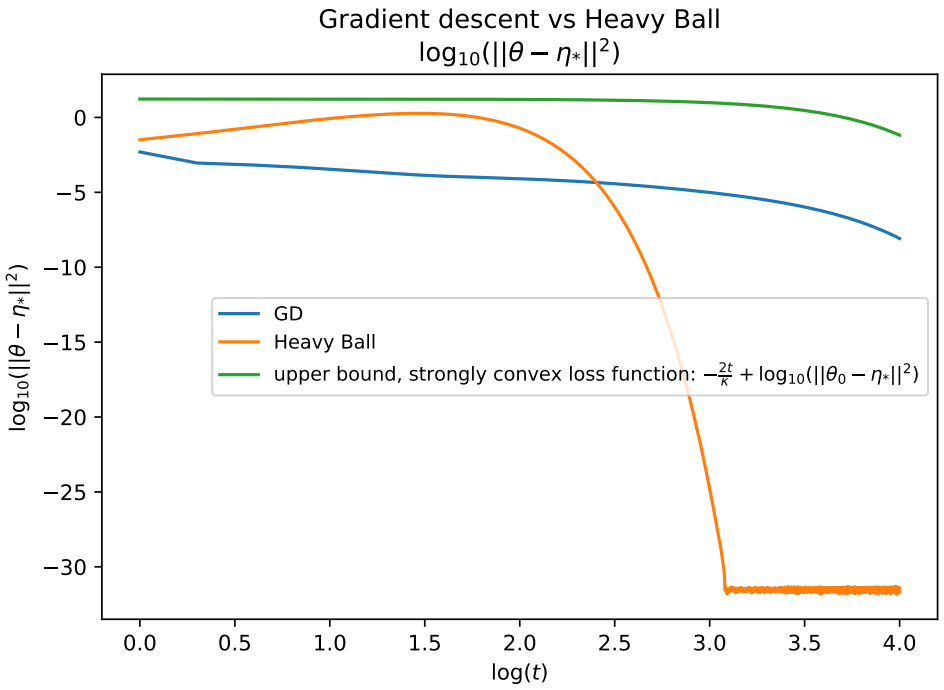
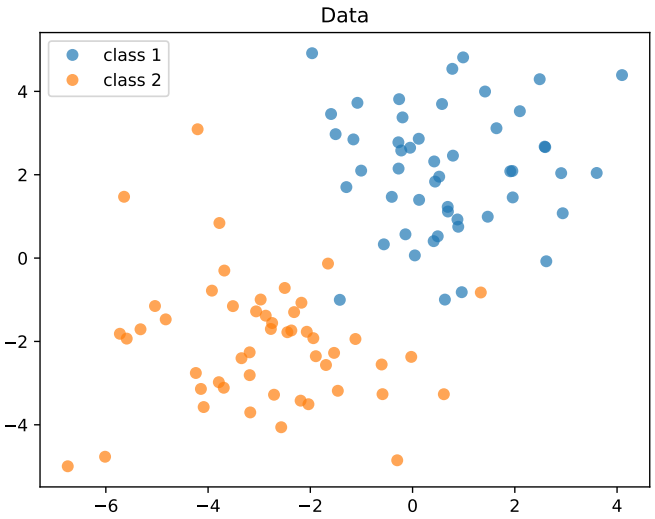


FIGURE 2 – Heavball vs GD, logarithmic scale

3 GD AND SGD FOR LOGISTIC REGRESSION

We will now perform GD, and Stochastic Gradient Descent (SGD) on Logistic regression. We will work on a binary classification problem, for which you can tweak the data using **generate_gaussian_data.py**.



3.1 Setup

We recall the setting of the problem.

— $\mathcal{X} = \mathbb{R}^2$

— $\mathcal{Y} = \{-1, 1\}$ (sometimes it is $\mathcal{Y} = \{0, 1\}$, in which case we change the loss)

— Logistic loss :

$$l(\hat{y}, y) = \log(1 + e^{-\hat{y}y}) \tag{11}$$

$$\begin{aligned} \frac{\partial l}{\partial \hat{y}}(\hat{y}, y) &= \frac{-ye^{-\hat{y}y}}{1 + e^{-\hat{y}y}} \\ &= \frac{-ye^{-\hat{y}y}e^{\hat{y}y}}{(1 + e^{-\hat{y}y})e^{\hat{y}y}} \\ &= \frac{-y}{1 + e^{\hat{y}y}} \\ &= -y\sigma(-\hat{y}y) \end{aligned} \tag{12}$$

With the usual L2 regularization, the empirical risk writes :

$$R_n(\theta) = \frac{1}{n}\sum_{i=1}^{n} l(x_i^T\theta, y_i) + \frac{\mu}{2}\|\theta\|^2 \tag{13}$$

We note $g_i(\theta) = l(x_i^T\theta, y_i) + \frac{\mu}{2}\|\theta\|^2$. Then

$$R_n(\theta) = \frac{1}{n}\sum_{i=1}^{n} g_i(\theta) \tag{14}$$

We have that

$$\nabla_\theta g_i = -y_i\sigma(-x_i^T\theta y_i)x_i \tag{15}$$

Hence

$$\begin{aligned}\nabla_\theta R_n &= \frac{1}{n}\sum_{i=1}^{n} \nabla_\theta g_i + \mu\theta \\ &= \frac{1}{n}\sum_{i=1}^{n} -y_i\sigma(-x_i^T\theta y_i)x_i + \mu\theta\end{aligned} \tag{16}$$

3.2 Files

— **generate_gaussian_data.py** (dimension 2) and **dgenerate_gaussian_data.py** (dimension d) : generate the data

— **TP4_LR.py** : main file, launches the algorithms. You can change several parameters there, like the maximum number of iterations. The file uses the data generated by **generate_gaussian_data.py** and **dgenerate_gaussian_data.py** for larger dimensions.

— and **TP4_algorithms.py** : the file containing the algorithms. You will have to edit it.

— **TP4_utils.py** : other utilities.

3.3 Profiling

Note that some **profiling** is done by the algorithm file and is output to text files, for instance **profiling_gd.txt**. This can be useful to monitor the behavior of the algorithms.

https://docs.python.org/3/library/profile.html

3.4 Gradient Descent (GD)

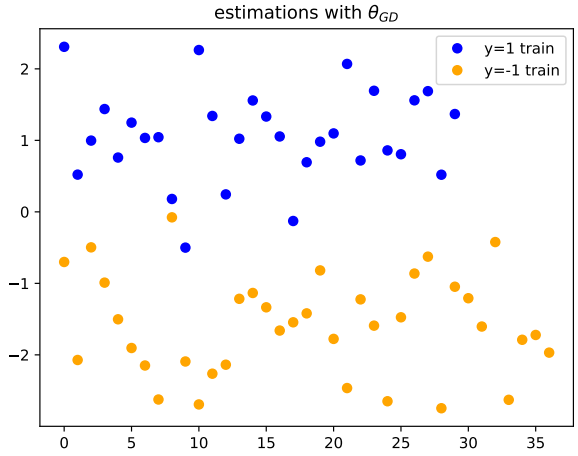
We recall the gradient update

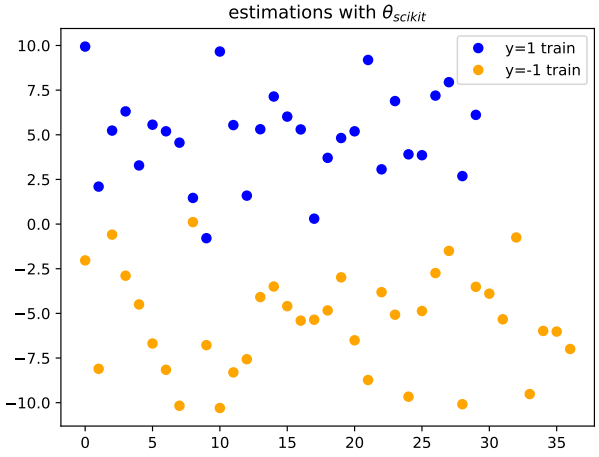
$$\theta_{t+1} = \theta_t - \gamma\nabla_{\theta_t} f \tag{17}$$

3.4.1 *Learning rate*

When performing GD, it is possible to use a constant learning rate γ. A relevant value is computed in **TP4_LR.py**.

Exercice 4 : Run GD (full gradient / batch gradient / deterministic gradient) on the LR problem, by running **TP4 _LR.py**. You can use the dimensions $n = 1000$ and $d = 2$, since this enables a visualization of the obtained predictor. This should work without having to edit the files. You can monitor the algorithm with the following images.


estimations with θ_{GD}
y=1 train
y=-1 train
2
1
0
−1
−2
0
5
10
15
20
25
30
35


estimations with θ_{scikit}
y=1 train
y=-1 train
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
0
5
10
15
20
25
30
35

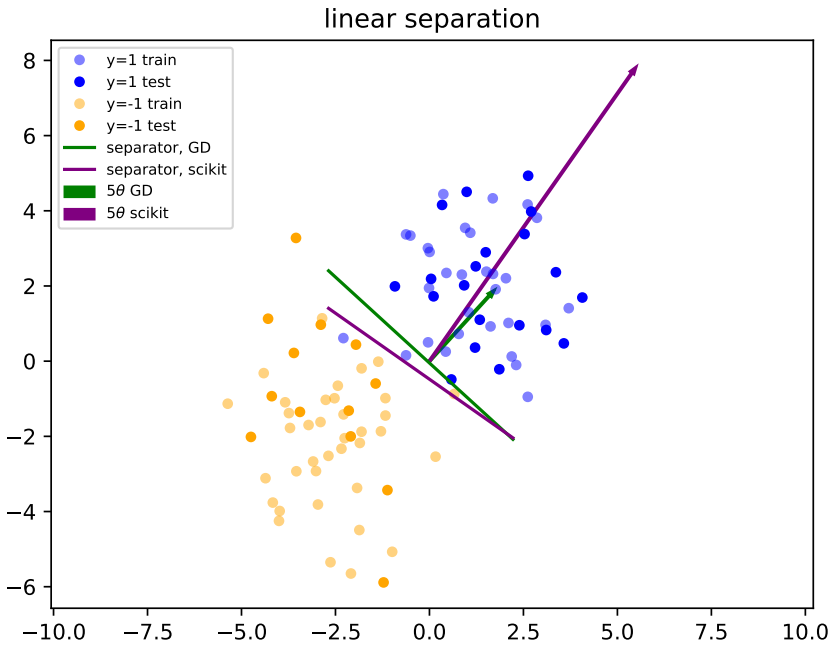


FIGURE 3 – Linear separators returned by the algorithms. Note that if we had used $d > 3$, we could not have produced such a visualization.

3.5 Stochastic Gradient Descent (SGD)

When performing SGD, we replace the computation of the batch gradient $\nabla_\theta R_n$ by the gradient with respect to one of the g_i, with i being uniformly sampled at each iteration. $i \in [1, n]$, where n is the number of samples used for training. If we note $i(t)$ the sample index chosen at iteration t, the SGD update reads

$$\begin{aligned} \theta_{t+1} &= \theta_t - \gamma \nabla_\theta g_i \\ &= \theta_t - \gamma(-y_i \sigma(-x_i^T \theta y_i) x_i) \end{aligned} \tag{18}$$

We will work in larger dimensions, for instance $n = 10000$ and $d = 80$.

3.5.1 *Optimization time and computation time*

The **optimization time** is the number of iterations in the optimization process, while the **computation time** is the optimization time multiplied by the computational complexity of one iteration.

The computationnal complexity of an SGD update is n times smaller than that of GD, where n is the number of training samples. Hence, **if** the optimization time necessary to reach a given precision is of the same order of magnitude for GD and for SGD, the computation time will be way slower for SGD. Actually, it is often the case in practice and it is what we will observe in the following simulations. We will also monitor the elapsed time during the run of the algorithm.

3.5.2 *Learning rate schedule*

In order to benefit from this advantage while still optimizing the empirical risk, it is important to carefully tune the hyperparameters of the algorithm, such the learning rate. Indeed, the choice of $i(t)$ introduces some randomness in the optimization process. It is even possible that after an iteration of SGD, the excess risk is actually **higher** than before this iteration. In order to tackle this randomness, several

strategies exist and the optimal strategy will depend on the problem. A strategy is called a "learning rate schedule". A classical strategy is to use a decreasing learning rate, for instance of the form

$$\gamma_t = \frac{\gamma_0}{1 + \frac{t}{t_0}} \tag{19}$$

where γ_0 is the initial value of γ and t_0 is a parameter controling the speed of the evolution of γ_t as a function of t. We will call this schedule **"decreasing 1"** in the practical session.

Another possibility is the following :

$$\gamma_t = \frac{\gamma_0}{1 + \sqrt{\frac{t}{t_0}}} \tag{20}$$

With this schedule, γ decreases more slowly than with 19, and γ_t is thus larger. It is called **"decreasing 2"** in the practical session.

Another possibility is to use a constant learning rate γ_0, which will be called "constant".

In this session, these strategies are implemented in the function **learning_rate_schedule(gamma_0, iteration, schedule)** in **TP_4_algorithms.py**. Note that other strategies exist, such as piecewise constant γ_t.

Therefore, many parameters are to be tuned :

— choice of the regularization paraterer μ

— choice of γ_0, and t_0.

— choice of the learning rate schedule.

The optimal choice of these hyperparameters will depend on the problem, through the statistical properties of the dataset, and on the convexity or strong convexity of the loss function (which depends on the dataset **and** on μ). Cross-validation is often used to decide the best values to use. In this session we want to observe different possible behaviors of SGD in comparison to GD.

Exercice 5 : Implement SGD and monitor the convergence, as a function of the different hyperparameters. You will need to uncomment the relevant lines in the main file and edit the file containing the algorithms. We will also plot the convergence as a function of the number of sample queries and monitor the execution time of both algorithms. For that purpose, it is important not to compute the test error and empirical risk at each iteration, because this would bias the measured elapsed time. This is controlled by the parameter **nb_empirical _risk_computations** in **TP4_algorithms.py**.

3·5·3 *Convergence and randomness of SGD*

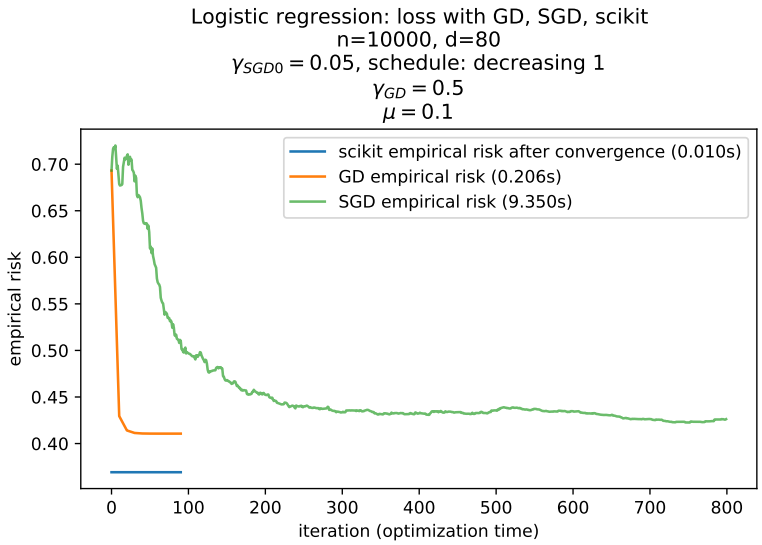


Figure 4 – By computing the empirical risk at each iteration, we observe the randomness of SGD. It can even happen that the empirical risk **increases** at a time step, leading some people to criticize the name Stochastic Gradient "Descent". Here, since we compute the empirical risk at each time step, the measured elapsed time is biased.

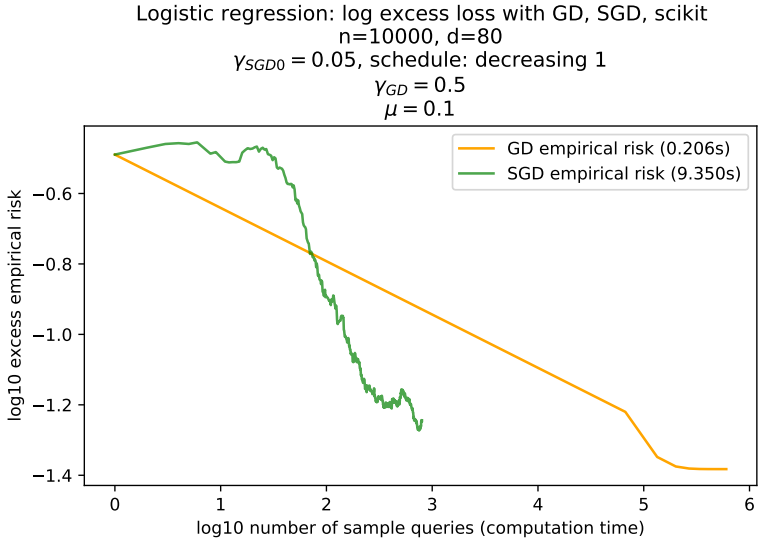


Figure 5 – Same plot, as a function of the log number of sample queries.

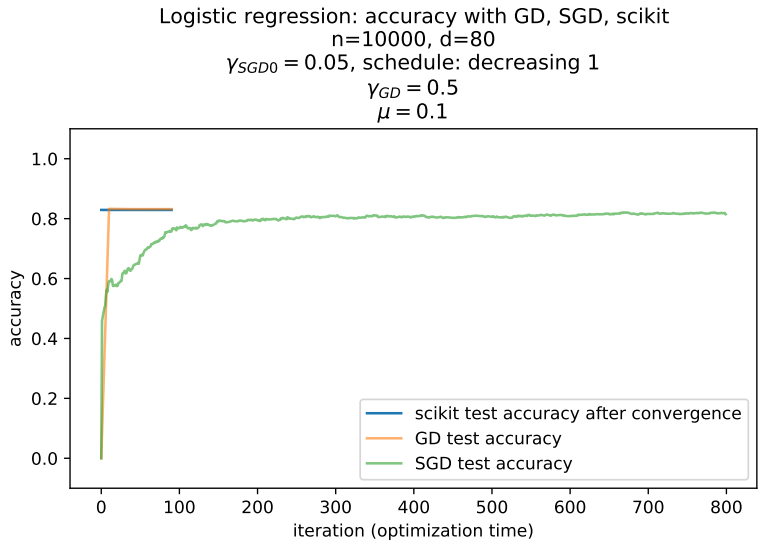


FIGURE 6 – Mean accuracy.

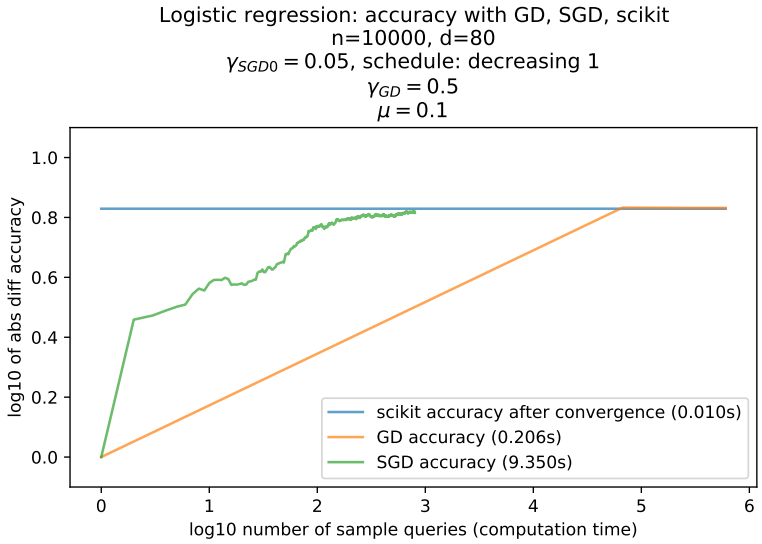


FIGURE 7 – Mean accuracy, as a function of the number of sample queries, in log scale.

3·5·4 *Non convergence of SGD*

If the hyperparameters are not tuned carefully, SGD might not converge or even lead to an increasing loss function. Note that this is also the case for GD.

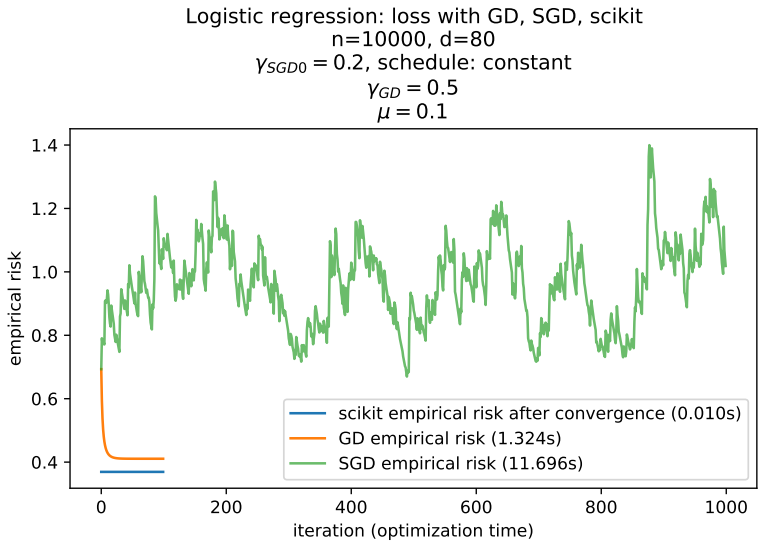


FIGURE 8 – Non convergence of SGD

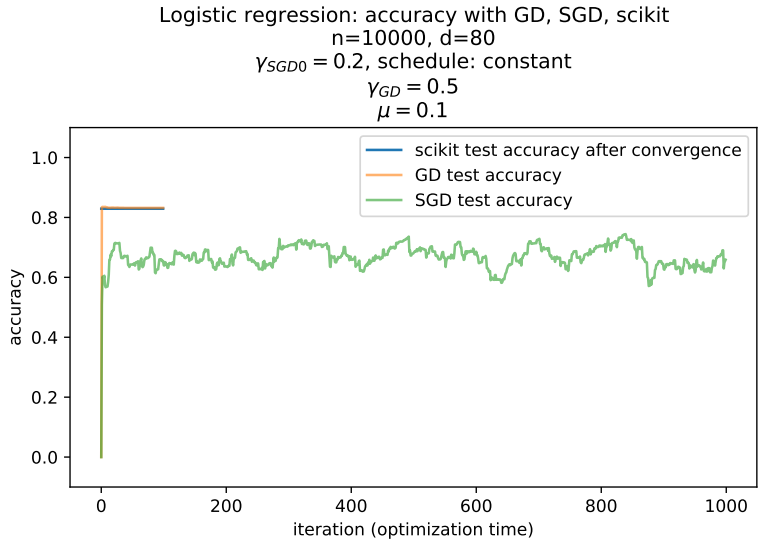


FIGURE 9 – Non convergence of SGD

3·5·5 *Speed of SGD*

Thanks to the reduced number of sample access, with the right tuning SGD might be significantly faster than GD.

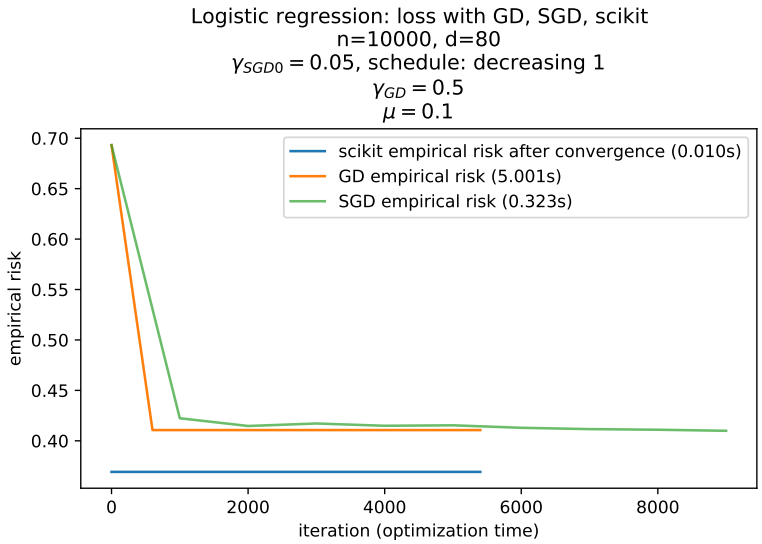


FIGURE 10 – Speed of SGD

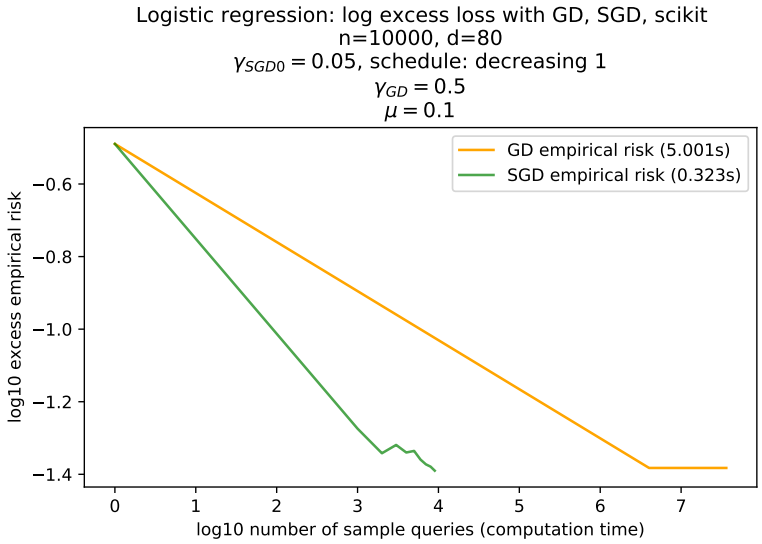


FIGURE 11 – Same plot, as a function of the log number of sample queries.

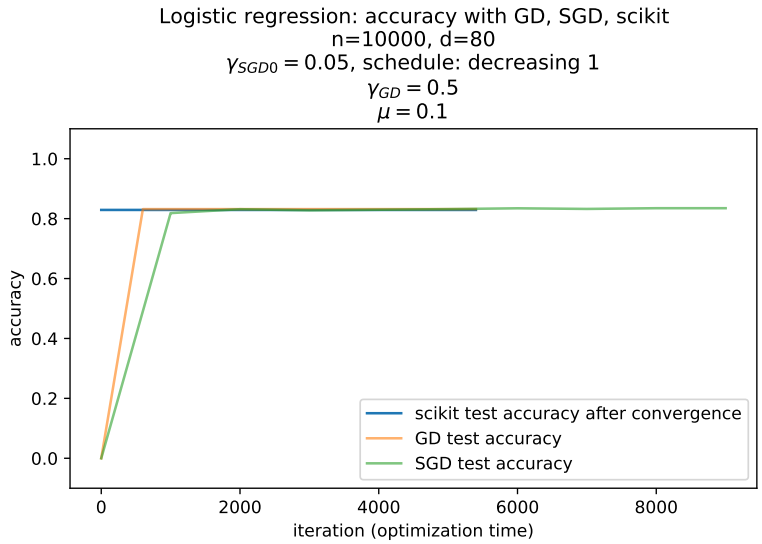


FIGURE 12 – Mean accuracy.

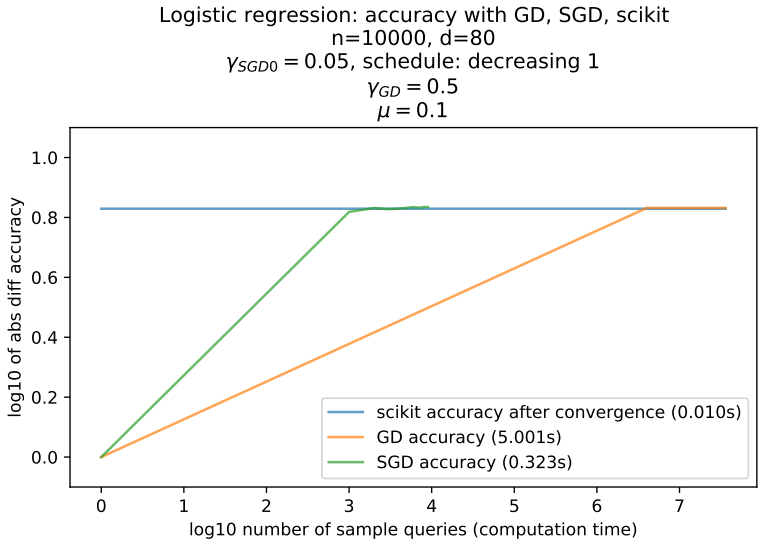


FIGURE 13 – Mean accuracy, as a function of the number of sample queries, in log scale.

3.6 Stochastic Gradient with Averaging (SGA)

In the most general setting, upper bounds on the convergence speeds are obtained for an average of the parameter θ at the end of optimization, such as

$$\hat{\theta} = \frac{1}{t} \sum_{i=1}^{t} \theta_k \tag{21}$$

Note that in some situations it is possible to have garantees **without** averaging, for example in the least-squares setup, in a statistical model called the noiseless model [Varre et al., 2021], and under additional hypotheses such as the existence of a *perfect interpolator*, which means that the optimal loss function value is 0. This setting is also known as *overparametrized, well-specified* or *interpolation regime* and is an important topic in the research community focusing on the theoretical properties

of neural networks. In this case, when no averaging is used, the method is called *last-iterate* SGD.

However, in the case of logistic regression, [Bach, 2014] proves that averaging leads to a faster convergence.

3.6.1 *Averaging strategy*

In practice, more efficient methods exist in order to average the θ_t. For instance, it is possible to only take the last half of the iterates into account, in order to more rapidly forget the initial conditions (**tail averaging**) . More generally, it is possible to use a custom weighting of the iterates θ_t (**weighted averaging**).

Exercice 6 : Experiment with averaging and monitor the impact on convergence speed, in order to observe behaviors like the folliwing figures.

3.6.2 *Comparison of algorithms with learning rate schedule "decreasing 1"*

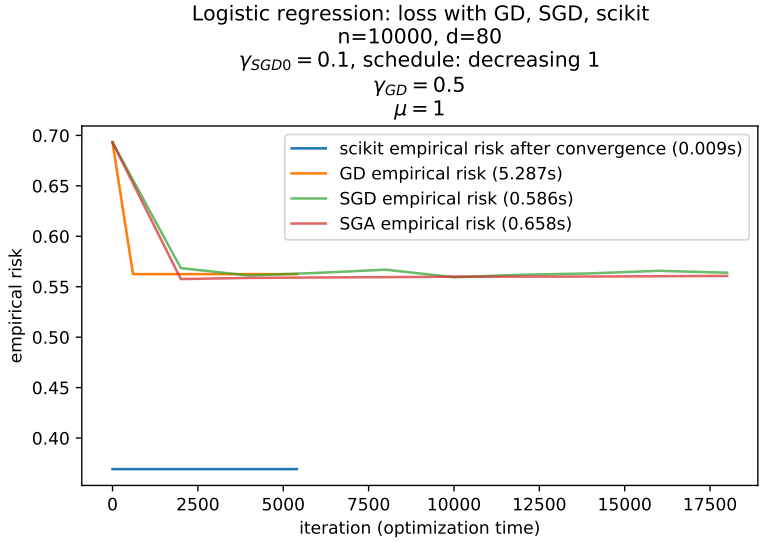


FIGURE 14 – SGA, SGD, GD, with learning rate schedule 1.

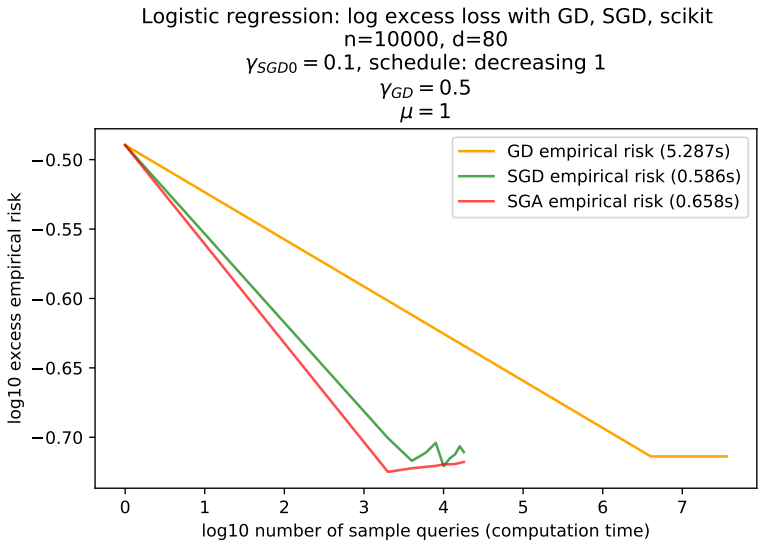


Figure 15 – Same plot, as a function of the log number of sample queries.

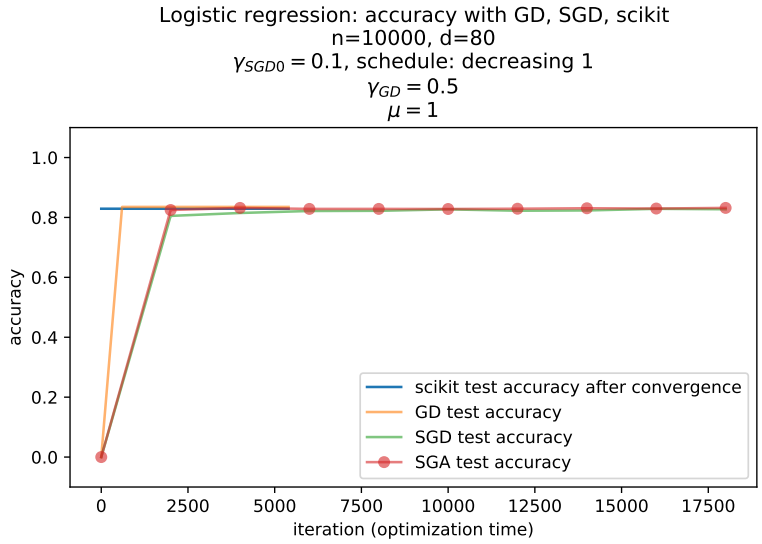


Figure 16 – Mean accuracy.

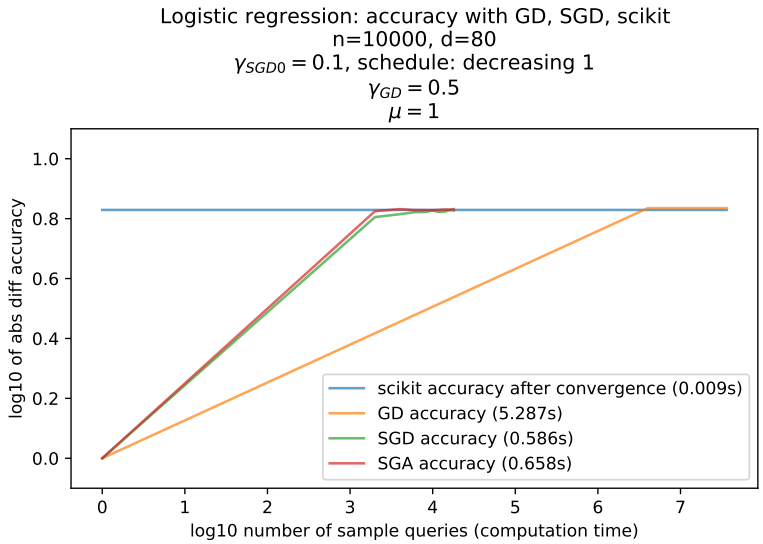


FIGURE 17 – Mean accuracy, as a function of the number of sample queries, in log scale.

3.6.3 *Comparison of algorithms with learning rate schedule "decreasing 2"*

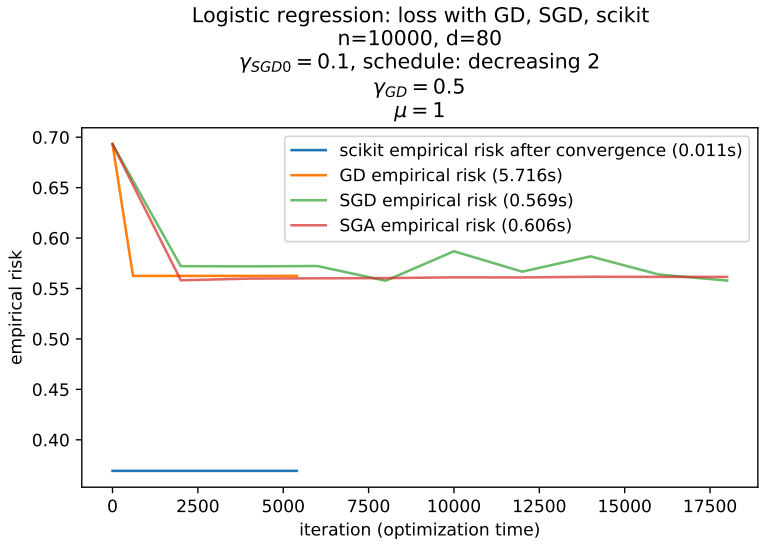


FIGURE 18 – SGA, SGD, GD, with learning rate schedule 2.

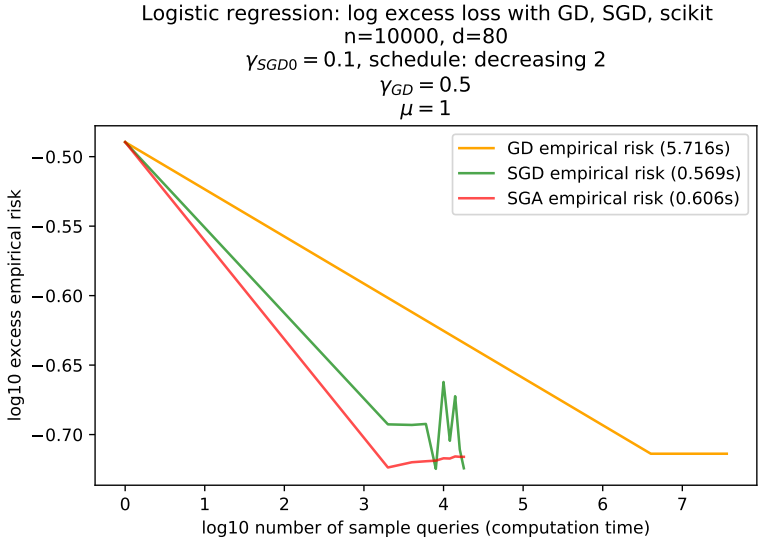


Figure 19 – Same plot, as a function of the log number of sample queries.

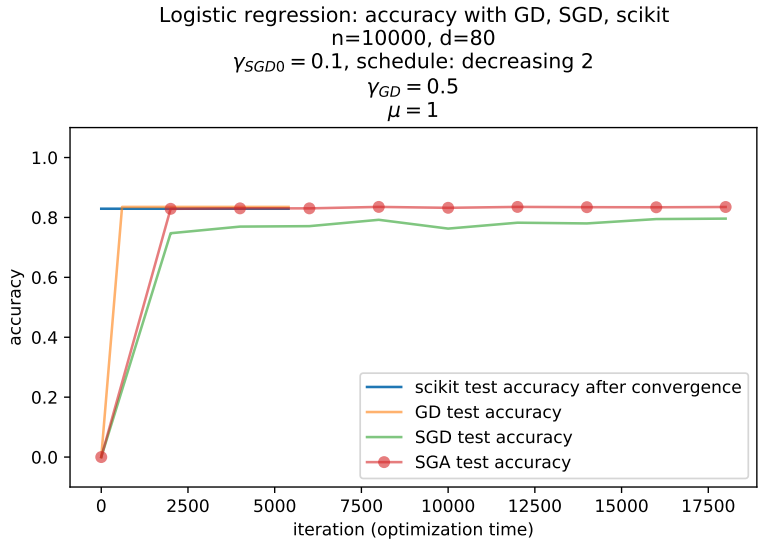


Figure 20 – Mean accuracy.

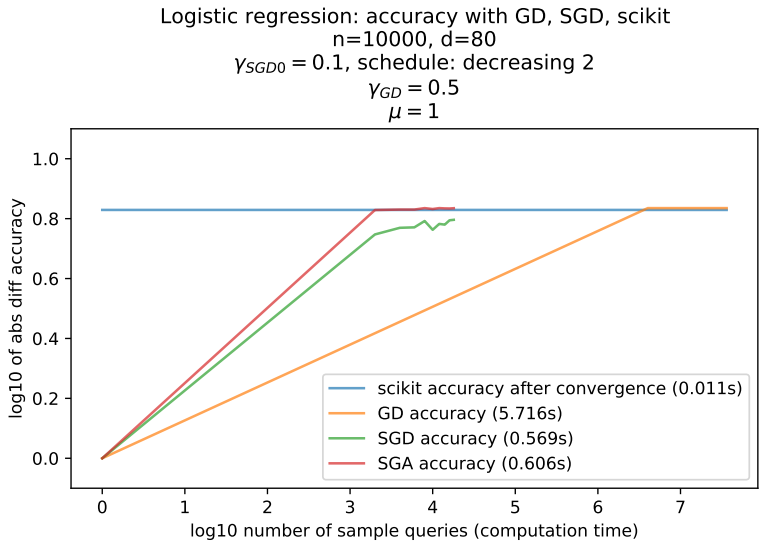


Figure 21 – Mean accuracy, as a function of the number of sample queries, in log scale.

RÉFÉRENCES

[Bach, 2014] Bach, F. (2014). Adaptivity of averaged stochastic gradient descent to local strong convexity for logistic regression. Journal of Machine Learning Research, 15 :595–627.

[Varre et al., 2021] Varre, A., Pillaud-Vivien, L., and Flammarion, N. (2021). Last iterate convergence of SGD for Least-Squares in the Interpolation regime. CoRR, abs/2102.0.