Optimization for Data Science - Homework 1

Luca Dal Zotto

May 18, 2020

Assignment

In this homework, three different techniques are considered to solve the regularized logistic regression problem. To be more specific, the task consists in comparing the performances of the classic gradient descent method (GD) with fixed step-size, the stochastic gradient descent method (SGD) and the stochastic variance reduced gradient method (SVRG). After having implemented them, they have to be tested on a publicly available dataset, analyzing their accuracy vs the CPU time required by the computations.

---

1 Solution

1.1 The dataset

For this homework, I have considered a publicly available dataset regarding a medical study concerning the breast cancer. Basically, for each sample we know the 1000 most informative genes associated with more aggressive and therapeutically refractory tumors: the first 1000 columns are numerical variables; the last column (named code) is a categorical variable with values "case" and "control".

This dataset is quite small, since the number of observations is just 250. In a first attempt, I used this dataset just to check the correctness of the algorithms and to make the first considerations. Afterwards, in order to highlight the differences in terms of computational cost of the three methods, I replicated multiple times the available observations, in order to simulate a larger dataset of 250000 observations, knowing that this leads to sample dependency and, possibly, numerical problems. Anyway, I found out that the results are theoretically reasonable, as we will see later on, and they allows us to compare effectively the three methods without the need to use a larger dataset.

Let's see now the pre-processing steps carried out: the dataset does not contain any missing value, so no operation has to be done in this sense. Since the features have different orders of magnitude, it is advisable to normalize them. Moreover, since we are dealing with a regression problem, it can be useful to add a bias term, simply represented by a column of ones. The feature matrix obtained in this way is denoted by X. As final pre-processing step, the values of the label vector (y) have been converted into -1 and 1, in order to be consistent with the notation seen in class.

1.2 Regularized logistic regression problem

The regularized logistic regression problem consists on the following minimization problem:

$$\min_{w\in\mathbb{R}^n} h(w) = \frac{1}{m}\sum_{i=1}^{m} \log\left(1+\exp\left(-y^i w^\top x^i\right)\right) + \frac{\lambda}{2m}\|w\|^2 = \frac{1}{m}\sum_{i=1}^{m} h_i(w)$$

where w is the vector of the parameters of the model, m is the number of samples and λ the regularization parameter. Note that I added a multiplicative factor of $1/m$ compared to what seen in class. After having defined this loss function and its gradient, it is possible to start with the implementation of the methods.

1.3 Implementation of the methods

The update rule of the classic gradient descent with fixed step-size α is the following:

$$w_{k+1} = w_k - \alpha \nabla h\left(w_k\right) = w_k - \frac{\alpha}{m} \sum_{i=1}^{m} \nabla h_i(w_k).$$

The value of α has been selected after different trials, since it has not been derived from the Lipschitz constant L. At each iteration of the method, the value of the loss function and the cumulative CPU time expired have been stored into two vectors.

Then, I implemented the SGD method. The fundamental part of the implementation is the selection of a random index i and the gradient computation only with respect to the i^{th} sample. Furthermore, in order to ensure convergence, it is necessary to use a diminishing step-size α_k. In this case the updating rule is:

$$w_{k+1} = w_k - \alpha_k \nabla h_i\left(w_k\right).$$

It is crucial to find the learning rate that is decreasing neither too quickly (otherwise the learning phase will terminate very soon), nor too slowly (otherwise the algorithm could fail to converge). For this reason, some alternatives of different forms have been tested:

- $\alpha_{it} = \alpha/(it+1)$
- $\alpha_{it} = \sqrt{\alpha/(it+1)}$
- $\alpha_{it} = \sqrt{\alpha/(it+100)}$
- $\alpha_{it} = \sqrt{2\alpha/(it+1))}$

and so on, where α is a constant.

Since the time required by each iteration is much smaller compared with the GD, the loss function is computed only once every 30 iterations. Personally, I decided to update the vector of the loss function values and the one of the CPU time only when the updated value of the loss function is computed. In this way, the vector of the loss function values does not contain sequences of 30 numbers with the same value, which would be memory inefficient for a high number of iterations. As a consequence, the final plot will not have the step-wise behavior seen during the lectures.

Finally, I implemented the SVRG method using two nested for loop: the outer one is relative to the epochs, while the inner one is relative to the iterations per epoch. In each epoch s, the full gradient is computed only once. Inside the inner loop, for each iteration k the gradient is computed using only one randomly selected sample x_i, and it is evaluated in two different values: the parameter of the previous iteration w_k and the parameter of the current epoch $\tilde{w_s}$. The update rule is the following:

$$w_{k+1} = w_k - \alpha \left[\nabla h_i\left(w_k\right) - \nabla h_i(\tilde{w_s}) + \frac{1}{m} \sum_{i=1}^{m} \nabla h_i(\tilde{w_s}) \right].$$

In this case, the value of the loss function is computed every epoch.

All these functions implement a stopping criterion based on the target value of the loss function. If this value is not reached, a maximum number of iterations specified in input is performed. All the details about the implementations of these functions, the input and the output parameters can be found in the Python notebook named "Luca_Dal_Zotto_OPT_HW1.ipynb".

1.4 Results

In the case of the small dataset, I used the following parameters:

	max number of iterations	α
GD	200	0.1
SGD	1000	0.1
SVRG	20 epochs, 20 iter/epoch	0.05

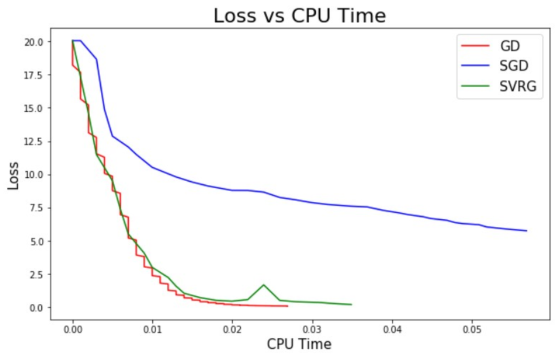
For the SGD method, the diminishing step-size selected was: $\alpha_{it} = \sqrt{2\alpha/(it+1)}$).

The Figure 1(a) shows how the value of the loss function decreases in time for the small dataset. We can already make some relevant observations: SG and SVRG perform better than SGD. The explanation is that the SDG estimate the gradient using only one random sample, but this could not represent the actual gradient effectively. SVRG has a similar behaviour to SG: indeed, remember that it computes the full gradient every epoch (in this case every 20 iterations), so the update rule has also this deterministic component. To sum up, in situations like this where the number of observations is very small, the standard SG method is often preferable, since the full gradient computation is not too costly.

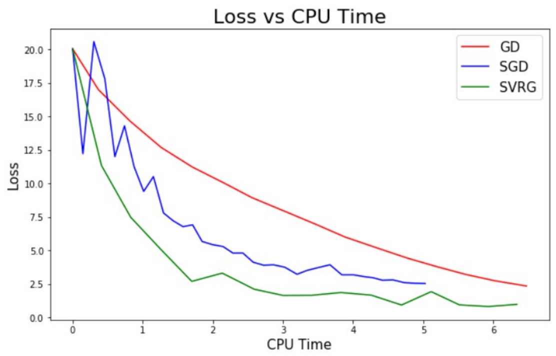
In order to better capture the differences between the 3 methods, we should have a look at the results involving the larger dataset. In this case, the parameters of the models are:

	max number of iterations	α
GD	15	0.5
SGD	1000	0.5
SVRG	15 epochs, 20 iter/epoch	0.1

Even if the dataset is not already huge (250000 samples), in Figure 1(b) it is possible to note the different behaviour of the classic gradient compared with the other methods. The reason is that, at each iteration, it computes the full gradient, a computation whose cost with 250000 samples cannot be neglected. To give some details, with the smaller dataset, 200 iterations were performed in about 0.08 seconds, while now, it takes more than 6.4 seconds to perform only 15 iterations: the oracle cost is bigger.



(a) *Small dataset (250 samples).*



(b) *Large dataset (250000 samples).*

Figure 1: Performances of the three models on the two datasets.

It is also important to remark that the step-size has been determined "empirically" with different trials: smaller ones would have slowdown the method, larger ones would possibly have implied divergence. Knowing the value of the Lipschitz constant could have speed up the computation, using as fixed step-size $\alpha = L^{-1}$.

The performances of the gradient and the SVRG method are quite better. Considering the fact that they are not deterministic algorithms, their results might slightly change at every execution. In order to plot this graphs, I evaluated the loss function once in a while, an operation that has penalized also them. Without this computation, these two methods require almost the same CPU time as before, since SGD is dimension free (it does not depend on the number of samples) and also the SVRG is only partially influenced by the number of samples.

For the sake of completeness, I tried to see how much time was required by the three methods to reach a value of the loss function smaller than the threshold $fstop = 2$. These are the results:

	Loss value	**Iterations required**	**CPU Time**
GD	1.67	18	7.07 s
SGD	1.86	1322	6.50 s
SVRG	1.80	126	2.50 s

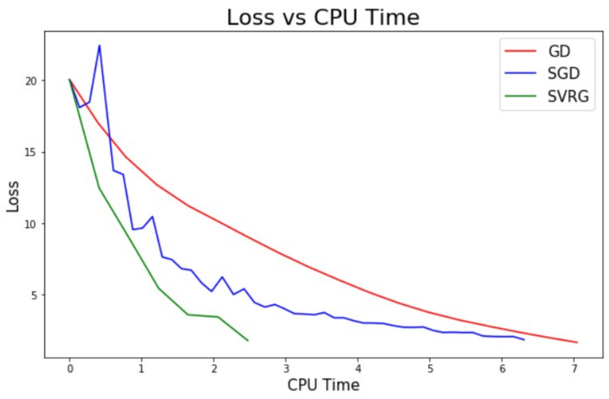
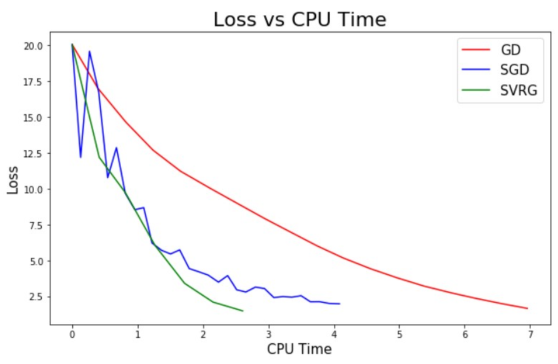


Figure 2: Time required by the three method to reach the same accuracy.

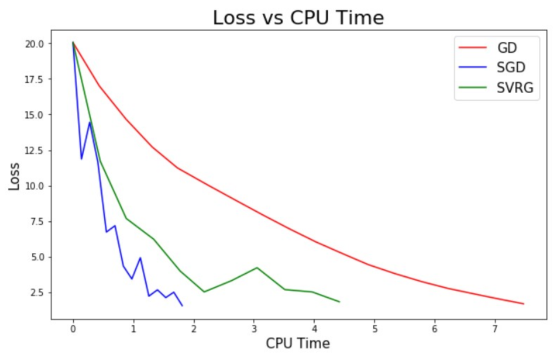
Firstly, it is possible to notice that the SGD method has an irregular behaviour, due to the fact that the update rule is based only on one random sample, and this could move the current solution to a direction which is very different from the optimal one. The gradient descent is the slowest, but his behaviour is very smooth. SVRG seems to be the best compromise in this case, leading to better results.

The last thing I checked was the influence of the regularization parameter λ. According to the theory, the role of this parameter is to control the overfitting of the model, by penalizing large values of the parameters. With a large λ, the complexity of the model should remain low, obtaining a smaller accuracy on the training set.

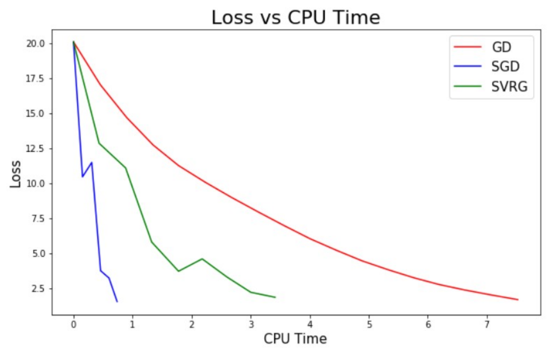
In Figure 3 are reported the performances of the methods for the task of reaching the target objective function $fstop = 2.5$ with different values of λ. It is possible to observe that the GD performances are always comparable, while the other two methods seem to be highly influenced by modifications of λ. In particular, the larger the regularization parameter, the faster becomes SGD. A possible explanation of this result is that λ is nothing but a penalizing value for the parameters, which are kept as small as possible. Therefore, even if a particular iteration of the SGD method provides a wrong (non-optimal) update direction, its consequences are kept under control by limiting large noisy updates.



(a) $\lambda = 1$



(b) $\lambda = 10$



(c) $\lambda = 50$

Figure 3: Performances of the three models using different values of λ.

1.5 Conclusions

To conclude, let us summarize the most important observations obtained with this analysis:

- the standard GD method becomes slower when the number of samples increases, since it computes the full gradient. The results of SDG and SVRG are less affected by a larger number of samples, since most of the times we compute the gradient only with respect to one sample;

- the values of the loss function of the SGD and SVRG methods has an irregular decreasing, caused by their stochastic component;

- in general, SDG seems to guarantee faster reductions of the loss function, but sometimes it is outperformed by SVRG when more accurate results are sought. Probably, this can be motivated by the fact that in this implementation, SGD uses a diminishing step-size, so the learning rate becomes very small after a certain number of iterations. Maybe, a more accurate tuning of this parameter can partially avoid this effect;

- the value of the regularization parameter seems to improve the performances of the SGD method, and, partially, those of SVRG. On the other hand, the SG method seems to be a little bit slower with larger λ. However, in order to check the effectiveness of larger regularization parameters, it would be interesting to see whether the performances of the RLR model improve with larger λ, through an evaluation on a testing set.