Optimization for Data Science

Homework 1

Bernardi Alberto, Cracco Gianmarco,
Manente Alessandro, Squarcina Giuliano

18th May, 2020

1 Dataset

As a dataset for this work, we chose the *Spambase Data Set* from the *UC Irvine Machine Learning Repository*, a data set collecting information about e-mails in order to classify them as spam or non-spam. It consists of 4601 instances, each of which with a set of 57 numerical attributes, including the binary target variable, describing, among the others, the percentage of words in the e-mail matching certain words, the percentage of characters matching certain characters, the average length of uninterrupted sequences of capital letters in the e-mail. For a more specific and detailed description of the dataset, check up the following webpage: `https://archive.ics.uci.edu/ml/datasets/Spambase`.

We proceeded with the scaling of our predictors to speed up and stabilize the calculations of our numerical methods and then exported our scaled dataset in **Matlab** in order to apply our optimization methods: *Full Gradient Method* (*GM*), *Stochastic Gradient Method* (*SGM*) and *Stochastic Variance Reduced Gradient Method* (*SVRGM*).

2 Overview of the Problem

In a generic binary classification problem, we want our model $h(x;w)$ to correctly predict the class associated with an input instance $\mathbf{x} = (x_1, \dots, x_n)$, with n the number of regressors. Given a training set, we want to find the set of parameters $\mathbf{w} = (w_1, \dots, w_n)$ for our model such that the prediction error over our training set is minimum, that is, we want to solve the following:

$$\min_{\mathbf{w}\in\mathbb{R}^n} R_m(\mathbf{w}) = \frac{1}{m}\sum_{i=1}^{m} l\left(y^i \cdot h(x^i;\mathbf{w})\right),$$

where x^i is one of the m instances in our training set, y^i is the true label associated with that instance and $l(\cdot)$ is the so called *loss function.*

In our specific case, $l(\cdot)$ was chosen equal to the *regularized logistic loss function* so that our optimization problem can be reformulated as:

$$\min_{\mathbf{w}\in\mathbb{R}^n} \frac{1}{m}\left[\sum_{i=1}^{m} \log\left(1 + e^{-y^i\mathbf{w}^\top x^i}\right) + \frac{\lambda}{2}\|\mathbf{w}\|^2\right],$$

where $\lambda > 0$ is the regularization coefficient.

It is fundamental to remark the relevance of the regularization term: in fact, in addition to improving the capability of our model of generalizing, it also makes our problem strongly convex, a capital assumption to guarantee the convergence of our optimization methods.

3 Analysis of the Problem

In the following subsections, we are going to provide a description of the used optimization algorithms, analyzing their main features. To obtain the final configuration of the parameters, we proceeded following a *trial-and-error* approach, lead by theoretical tips and our expectation.

Before dealing with our methods, a quick mention about the codes used in our analysis: we implemented our algorithms in **Matlab**, adapting the code for a least squares regression problem to our binary classification task. Notice that in the script of each algorithm the same stopping condition has been implemented: if the objective function becomes smaller than 10^{-9}, the algorithm stops to save time and computational power; otherwise it would continue to run until iterations are completed. Anyway, in our specific problem this condition has never been satisfied.

Full Gradient Method

The first method applied to optimize our problem was the Full Gradient Method, that takes into account all the instances of the training set at each iteration k, according to the following update rule:

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \alpha_k \left[\frac{1}{m} \left(\sum_{i=1}^{m} \frac{y^i x^i e^{-y^i \mathbf{w}^\top x^i}}{1 + e^{-y^i \mathbf{w}^\top x^i}} + \lambda \mathbf{w}_k \right) \right],$$

with α_k a suitably chosen fixed step size.
As it is well known, having to calculate the gradient for each instance of our data set may be very costly, especially when m starts to be pretty large, like in our specific case. This resulted in larger CPU times to obtain a good decrease of the objective function, as we will see more precisely later on. At the same time, the use of the full gradient provides us with a deep understanding of in which direction the objective function decreases, so that, compared to the other methods, we need very fewer iterations to obtain a substantial reduction of our cost function.

To find the best parameters for this method, we tuned the step size until we reached the value of 1: with smaller values, we observed that the decay of the cost function was less marked, while, with much bigger values, convergence was not ensured. For what concerns the number of iterations, we notice that it took less than 100 iterations to really reduce the objective function before it flatten, so we limited them to 300, in order to reduce the time for the computations.

Stepsize	Num. of iterations
1	300

Table 1: Parameters chosen for GM

Stochastic Gradient Method

To speed up things, a good way to proceed is to implement the Stochastic Gradient Method. The update rule in this case is the following:

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \alpha_k \left(\frac{y^{i_k} x^{i_k} e^{-y^{i_k}\mathbf{w}^\top x^{i_k}}}{1 + e^{-y^{i_k}\mathbf{w}^\top x^{i_k}}} + \lambda \mathbf{w}_k \right),$$

with i_k a random index chosen among $\{1, \ldots, m\}$.
By doing so, we update the parameter $\mathbf{w}$ computing the gradient with respect to only one randomly picked sample, so that the huge cost for the computation of the gradient for each instance of the training set, that holds back the Full Gradient Method, is now bypassed. As a drawback, the information captured by the gradient of a single sample is less precise, so the reduction of the cost function will be more subject to fluctuations.

To ensure the possibility of convergence, the step size cannot be kept constant, but has to be such that $\alpha_k \xrightarrow{k\to\infty} 0$. In our specific case we set

$$\alpha_k := \sqrt{\frac{\gamma}{k+\delta}},$$

where γ and δ are properly chosen constants and k is the current iteration. To properly select γ we tried different values, before fixing it equal to 0.1: smaller values lead to a smaller and less steep reduction of the cost objective function, while higher values increased the fluctuations of the gradient, leading even to no convergence.
Regrading δ, instead, we initially observed how the SGM, after a good decreasing for the first iterations, tends to flatten: so we try to increase the value of δ in order to have a bigger step size as the number of iterations increases. But after many trials we opted for $\delta = 1$, that gave good results.

Stepsize	Num. of terations
$\sqrt{\frac{0.1}{k+1}}$	5000

Table 2: Parameters chosen for SGM

Stochastic Variance Reduced Gradient Method

The last method applied was the Stochastic Variance Reduced Gradient Method. The underlying idea of this one is to reduce the high variance associated with the random choice of a component for the SGM. In order to realize this purpose, SVRG introduces the concept of *epoch*, that is a fixed number of iterations after which the current value of $\mathbf{w}_k$ is used to calculate the full gradient for the objective function. More in detail, set the length of an epoch to l; at iteration k of epoch n, we use the value $\tilde{\mathbf{w}}$ obtained from the last iteration of the previous epoch $n-1$ to update the gradient according to the following update rule:

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \alpha_k \left[\nabla f_{i_k}(\mathbf{w}_k) - \nabla f_{i_k}(\tilde{\mathbf{w}}) + \frac{1}{m}\sum_{i=1}^{m} \nabla f_i(\tilde{\mathbf{w}}) \right],$$

where $k = 1, \dots, l$ are the iterations for each epoch, i_k is an index randomly chosen among $\{1, \dots, m\}$ and $f_i(\mathbf{w}) = \log\left(1 + e^{-y^i \mathbf{w}^\top x^i}\right) + \frac{\lambda}{2}\|\mathbf{w}\|^2$ is the loss with respect to a single instance in the training set. Like for the Full Gradient Method, we opted for a fixed step size $\alpha_k = \alpha$ for any k.

To decide the best value l for the length of the epochs, we try different values, aiming, first of all, in not keeping it too small in order to avoid to compute the whole gradient an excessive number of times, thing that would have required longer CPU time. To also escape a high variance similar to the one of the SGM, we decided not to increase l too much, fixing it equal to 100, a value that gave pretty good results.

For the stepsize, we started from values similar to the one used for the SGM and from there we tried slight increasing and decreasing. In the end, as for the previous methods, too large step size lead to no convergence, too small did not provide a sufficient reduction of the objective function. We chose 0.05 since it was the one that gave the best compromise between a good initial decay, a great final value of the cost function and a tail not too flat.

The choice of the number of iterations, finally, was a consequence of the flattening tale: to avoid useless iterations, we stop them at 5000, since going on, there were no significant improvement in the reduction of the function.

Stepsize	Num. of Iterations	Epochs
0.05	5000	100

Table 3: Parameters chosen for SVRGM

4 Results

Before delving into the analysis of the results, a few words about the regularization coefficient: we tried different values, opportunely re-tuning also all the parameters of our methods, but we observed that, the highest reduction of the cost function was obtained with very small values of λ (we fixed it equal to 0.001). We explained this behaviour considering the role of regularization: it is typically used to reduce overfitting and provide a better generalization but, since in this specific case we had just a training set and no test set, increasing regularization would just lead to an increase of the cost function.

The application of the algorithms previously introduced, with the parameters tuned as reported in the above tables, lead to the results displayed in the figures here reported.

Let us start describing Figure 1: here we can see how the different methods reduce the objective function depending on the number of iterations. Before going into details, notice that the cost function for SGM and SVRGM decreases as a step function: this behaviour depends on the fact that, to avoid the computation of the cost function at each iteration (thing that would have been costly), we evaluated it every 100 iterations.
First of all, let's focus on the Full Gradient Method: clearly, since it considers all of the instances in computing the training set (therefore capturing the highest amount of information possible about the loss function behaviour), it is the method that requires the least number of iterations to get a huge reduction

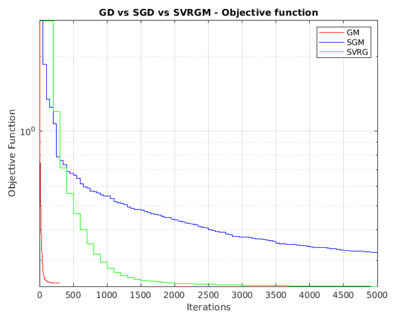

Figure 1: Iterations

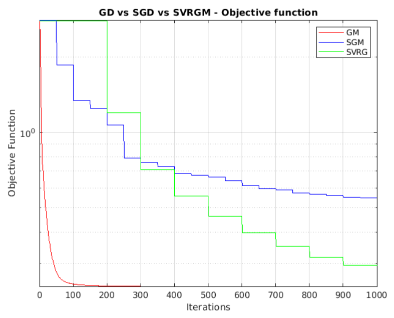

Figure 2: First Iterations Enlargement

of the objective function. As it is more evident in Figure 2, which represent an enlargement of Figure 1, with less than 50 iterations we get to a value well under 0.3 and very close to the final value of 0.243, reached after 500 iterations. Anyway, it is evident that the last 300 iterations give almost no improvement.

Looking at the SGM, instead, we can see that, after 5000 iterations, we obtain a final value of the objective function of 0.270, slightly worse than those achieved by the Full Gradient and the SVRGM. Moreover, the plot displays how there are some cases in which the function increases instead of decreasing: as mentioned above, this depends on the high variance associated with the random choice of the sample on which we compute the gradient for the update rule. Furthermore, observe that, after around 1500 iterations, we have a substantial reduction of the decrease, with the cost function that gets almost flat. Despite all these issues, we will see later on, in the CPU Time analysis, that it is not enough to discredit this method.

The SVRGM, finally, has a very interesting behaviour: in the first 200-300 iterations, it quickly decreases even if a bit more slowly compared to SGM and Gradient Method, but its steep reduction continues even after the 500-th iteration, when the cost function is already smaller than 0.3. At this point, it progressively flatten, but gets anyway to a final value 0.237, much smaller than the one obtained with SGM and even better than the one for GM.

From what we have seen so far, it seems like the best performing method for our task is one between the Gradient Method and the SVRGM. However, the analysis of the reduction of the cost function based on the number of iterations does not take into account the CPU Time needed by the methods to solve the task, and this is a crucial element. In fact, if the iterations analysis seems to discourage the use of the SGM for our problem, Figure 3 and Figure 4 clearly show why this method is so used for big data application: in a very small amount of time, say 0.1 s we have a very steep reduction of the objective function, obviously much better than the Full Gradient Method (slowed down by gradients calculations), but better also than the SVRGM. This really tells us the true importance of the SGM, extremely useful with high dimensional data, because, independently of the number of samples, it guarantees, in such a small amount of time, a great reduction of our cost function. Like what we have seen in the iterations analysis, also in this case the SGM tends to flatten in its final part.

Even if it seemed like the best method by the iterations, we now see how the Full Gradient Method really requires more CPU Time than the other two:

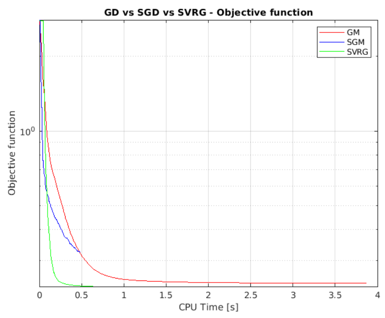

Figure 3: CPU Time

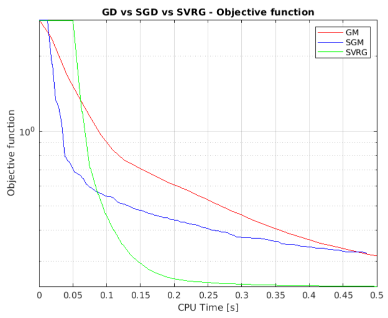

Figure 4: First Second Enlargement

it took 3.897 s to complete 300 iterations, while the others terminated their execution in around half of a second (0.486 s for the SGM and 0.643 s for the SVRGM). At first, we expected that the decrease would have been less steep than the current one; however, as mentioned above, while tuning the step size, we observed that, reducing it to 1 gave the best results, probably because, in this way, we better estimate the optimal step size for this method, that is, as known from theory, $\frac{1}{L}$, where L is the Lipschitz constant for the gradient of our cost function.

Finally, let us focus on the last method, the SVRGM, that really combines the best features of the previous two. Indeed, it joins the steep reduction in a very small amount of time typical of the SGM, with the higher precision of the Full Gradient, achieving the smallest value for the cost function. It took just half of a second (one seventh of the time required by the Full Gradient!) to obtain a value around 0.2, a great result, that definitely let us take it as the best method applied to this data set.

In Table 4 we summarize the parameters and the exact numeric results obtained with our best methods, with the regularization coefficient λ set equal to 0.001.

	GM	SGM	SVRGM
Stepsize	1	$\sqrt{\frac{0.1}{k+1}}$	0.05
Num. of terations	300	5000	5000
Epochs	-	-	100
CPU time [s]	3.879	0.486	0.643
Objective function	0.243	0.270	0.237
Regularization coef.	0.001	0.001	0.001

Table 4: Parameters and results obtained