ENSIMAG, MSIAM/MOSIG

Practical sessions report
Convex & Distributed Optimisation

SIALA Rafik

TRIMECH Abdellatif

LABIADH Mouna

Sunday 18th December, 2016

CONTENTS

ABSTRACT

The purpose of this report is to summarise the work conducted during the two practical sessions "Sparse Regression" and "Matrix Completion" of the "Convex distributed optimisation" course. Moreover, we consolidate the different notions by further detailing certain aspects.

Keywords: Sparse regression, descent gradient, proximal mapping, matrix completion

1 INTRODUCTION

In this practical sessions, we have approached two main problematics which are sparse regression and matrix completion. The sparse regression is a classification model that binary-classify sparse data and is based on a logistic regression loss criterion, subject to a constraint on the weight values. In this part, we t will study the choices of the loss-constraints combinations and approaches to solve them while introducing non-convex constraints, for example, to induce very sparse solutions.

The matrix completion is implemented as a part of a movie recommender system based on the Netflix ratings dataset in order to successfully associate users with matching items and suggest them an ite that they will really like. One way to do that is by the matrix completion strategy by filling in missing entries of a partially observed matrix of ratings(users,items).

2 SPARSE REGRESSION

The main purpose of this session is to successfully classify a small dense dataset D. This dataset is composed by m sample $\{d_i\}$ where each sample i is associated to a binary class label $b_i \in O = \{-1, +1\}$. The vector of d features is denoted by a_i

In order to classify this dataset, we need to estimate a predictor function g with parameter x, such that :

$$g : O \mapsto p_1(a) = \mathbb{P}[d \in \text{ class } +1] = \frac{1}{1 + \exp(-\langle a; x^\star \rangle)}$$

To estimate the optimal value x^*, we consider the maximization of its corresponding log-likelihood :

$$\mathbb{L}(x|a_1, ..., a_m) = \log \left\{ \prod_{i=1}^{m} \frac{1}{1 + \exp(-\langle a_i; x^\star \rangle)} \right\}$$

$$\mathbb{L}(x|a_1, ..., a_m) = \sum_{i=1}^{m} \log \frac{1}{1 + \exp(-\langle a_i; x^\star \rangle)}$$

$$\mathbb{L}(x|a_1, ..., a_m) = -\sum_{i=1}^{m} \log(1 + \exp(-\langle a_i; x^\star \rangle))$$

To find the optimal parameter x, we have, then, to minimize an objective loss function. In our setup, we consider the minimization of the logistic loss function which enable us to define the following optimization problem :

$$\min_{x \in \mathbb{R}^d} f(x) = \frac{1}{m} \sum_{i=1}^{m} \log(1 + e^{-b_i < a_i, x >})$$

At a preliminary step, we have pre-processed our data by normalizing it and adding the intercept value. The normalization, in case of dense data, consists on centering the data around its zero-mean while the addition of the intercept will enable us to tune the dimension of the weight and the sample vectors to match the extra dimension caused by bias weight.

INTERCEPT The simplest classification model for regression involves a linear combination of the input variables :

$$g(x, a) = x_0 + x_1 a_1 + x_2 a_2 + ... + x_m a_m$$

where x_0 is called the bias parameter. This parameter x_0 allows for any fixed offset in the data and is sometimes called. It is often convenient to define an additional dummy bias variable $a_0 = 1$ so that:

$$g(x, a) = \sum_{i=0}^{m} x_i a_i = \mathbf{x}^{\mathbf{T}}\mathbf{a}$$

where $\mathbf{x} = (x_0, x_1, ..., x_m)^T$ and $\mathbf{a} = (a_0, a_1, ..., a_m)^T$

The initial dataset D before and after the pre-processing step is characterized by the following :

Table 1: IONOSPHERE dataset

	Before	After
Number of samples	351	351
Number of features	34	35
Density	0.884	0.859

2.1 Gradient Descent

For the minimization of the logistic loss function, we have implemented a Gradient descent algorithm taking as an input the training samples and the step-size. The descent gradient algorithm is based on the following ingredients :

1. **Initialisation** of x^0 and the parameters t.
2. **Computation of the gradient** $\nabla f(x^k)$.
3. **Update iteration** $x^{k+1} = x^k - t^k \nabla f(x^k)$.
4. **Stopping tests** a maximal number of iterations or a threshold for which the optimality conditions ares almost satisfied.

GRADIENT DESCENT method is a way to find a local minimum of a function. The way it works is we start with an initial guess of the solution and we take the gradient of the function at that point. We step the solution in the negative direction of the gradient and we repeat the process. The algorithm will eventually converge where the gradient is zero.

$$x^{k+1} = x^k - t^k \nabla f(x^k) \quad \texttt{, where } t^k > 0 \texttt{ is the step-size at the } k^{th} \texttt{ step}$$

In order to choose the step-size, two options are presented, whether we choose a fixed step-size fixed from the start or an adaptive step-size that changes at each iteration. In our case, we have worked with a fixed step-size that we denoted γ. Furthermore, a wrong step size may not reach convergence, so choosing the maximum step-size for convergence is an important detail. In our setup, we have assumed that $\nabla f(x^k)$ is Lipschitz. We denote the Lipschitz constant with $L > 0$ such that :

$$\| \nabla f(x) - \nabla f(y) \| \leqslant L \| x - y \| \; \forall x, y$$

We have chosen to work with a fixed step-size $\gamma < \frac{1}{L}$ with respect to theorem 1. The upper bound of the Lipschitz constant in our case would be $L_b = 0.25 max_i \parallel a_i \parallel_2^2$ which was found approximates to 0.99.

Theorem 1. *Gradient descent with a fixed step-size $\gamma < \frac{1}{L}$ satisfies :*

$$f(x^k) - f(x^*) \leqslant \frac{\parallel x^0 - x^* \parallel^2}{2\gamma k}$$

Proof. We begin our from the fact that we are searching to minimize $f(x)$:

$$f(x^{k+2}) \leqslant f(x^{k+1})$$

$$x^{k+1} - \gamma \nabla f(x^{k+1}) \leqslant x^k - \gamma \nabla f(x^k)$$

We have then:

$$\nabla f(x^k) - \nabla f(x^{k+1}) \leqslant \frac{1}{\gamma}(x^k - x^{k+1})$$

We are willing to decrease the value of the gradient until reaching zero:

$$\parallel \nabla f(x^k) - \nabla f(x^{k+1}) \parallel \leqslant \frac{1}{\gamma} \parallel x^k - x^{k+1} \parallel$$

On the other hand and from The Lipschitz gradient, we have the inequality:

$$\parallel \nabla f(x^k) - \nabla f(x^{k+1}) \parallel \leqslant L \parallel x^k - x^{k+1} \parallel$$

The γ step-size is dependant of current iteration's number which means : $\frac{1}{\gamma} \leqslant L$. Then $\gamma \leqslant \frac{1}{L}$, from which the proof. $\square$

At this level, we were all set to execute the descent gradient algorithm on our dataset. The convergence of the functional value towards a minimum is presented by the figure 1.

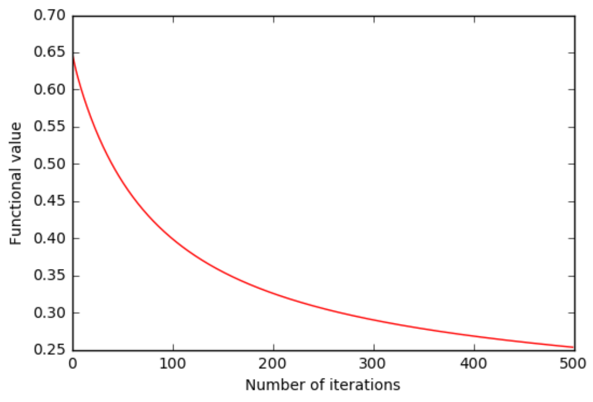

Figure 1: Convergence of the functional value towards a minimum using the descent gradient algorithm for $\gamma = 0.66$

2.2 Logistic Loss Regularization

It is well-known that regularization is required to avoid over-fitting, especially when there is only small number of training examples as in our case, or when there area large number of parameters to be learned. In particular, we have chosen to work with L1 and L2 regularization terms where L2 is the sum of the square of the weights, while L1 is just the sum of the weights. As follows:

$$r(x) = \lambda_1 \parallel x \parallel_1 + \lambda_2 \parallel x \parallel_2^2$$

This regularization term is used for two reasons :

1. **L1 Norm :** which is a sparsity induce norm
2. **L2 Norm :** used to prevent the coefficients to fit so perfectly to overfit.

Un-regularized logistic regression is an unconstrained convex optimization problem with a continuously differentiable objective function. As a consequence, it can be solved fairly efficiently with standard convex optimization methods, such as Gradient Descent implemented previously. However, adding the regularization term to the optimization problem makes the objective not continuously differentiable anymore because of the L1 norm which is not differentiable at 0. The problem is then written as the sum of two functions: a smooth function s_i and a non-smooth n :

$$g(x) = \frac{1}{m} \sum_{i=1}^{m} s_i(x) + n(x)$$

$$= \frac{1}{m} \sum_{i=1}^{m} \log(1 + e^{-b_i < a_i, x >}) + \lambda_2 \parallel x \parallel_2^2 + \lambda_1 \parallel x \parallel_1$$

This precludes the use of, for example, Proximal operator of the non smooth part to solve the optimization problem efficiently.

$$\text{prox}_{\gamma n}(y) = \text{argmin}_x \{ n(x) + \frac{1}{2\gamma} \parallel x - y \parallel_2^2 \}$$

The implementation of this proximal operator takes the form of a simple element-wise operation called soft thresholding and goes as follows :

$$\text{prox}_{\gamma n}(y)_i \begin{cases} x_i - \gamma & \text{if } y_i > \gamma \\ x_i + \gamma & \text{if } y_i < -\gamma \\ 0 & \text{otherwise} \end{cases}$$

At this level, we can apply the proximal algorithm using the gradient of the smooth $(f(x) + s(x))$ and the proximal mapping of the non smooth part $(n(x))$. The admissible step-size γ that have been retained at this part is bounded by $L_b = 0.25 max_i \parallel a_i \parallel_2^2 + 2\gamma_2$ and approximates in

$$x^{k+1} = \text{prox}_{\lambda_1 \gamma n}(x^k - \gamma \nabla (f(x^k) + s(x)))$$

We should note that we have worked on the general form of regularization where λ_1 and λ_2 are different from zeros which is also called the Elastic Net regularization.

In the final step, we are going to play with the values of λ_1 and λ_2. The figure 2 illustrates the results found for each combination (λ_1, λ_2). As we can notice, for each kind of regularization, the output of the proximal gradient algorithm changes remarkably :

- The biggest convergence velocity is seen in the the case of un-regularized loss function $(\lambda_1 = 0, \lambda_2 = 0)$.

- In the *Elastic-Net* case where λ_1 and λ_2 are not zeros, the algorithm converges towards the minimal in a small amount of steps.
- In the case of *Tikonov* and ℓ_1*−regularization*, we have similar results.

In this part and based on our different manipulations of the λ_1 and λ_2 values, we were not able to exactly estimate the optimal values but as we can notice from the graph λ_1 effects remarkably the convergence speed of our proximal gradient algorithm.

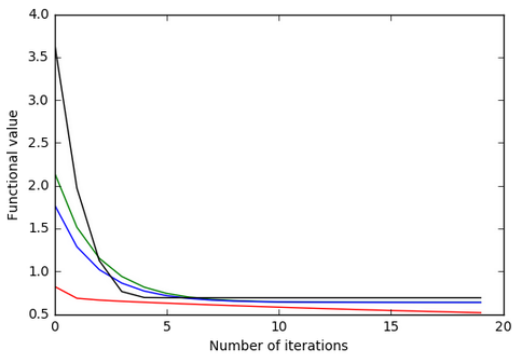

Figure 2: Convergence of the estimation error of the Proximal's algorithm for different values of λ_1 and λ_2. Each graph color represents a different couple of values of (λ_1, λ_2) ; Red : (0, 0), Green : (0, 0.1), Blue : (0.1, 0), and Black : (0.1, 0.1)

3 MATRIX COMPLETION

In this second practical work session, we have approached the Netflix "Matrix Completion" problem which consists on filling in the missing entries of a partially observed matrix.

NETFLIX PROBLEM Users (rows of the data matrix) are given the opportunity to rate movies (columns of the data matrix) but users typically rate only few movies so that there are few scattered observed entries of this data matrix. But, so that Netflix will be able recommend titles that any particular user is likely to be willing to order, this matrix must be filled. In this case, the data matrix of all user-ratings may be approximately low-rank because only a few factors contribute to an individual's tastes or preferences.

The main purpose of this session is then to successfully approximate a large rating matrix $\mathbf{R}$ with the multiplication of two low-dimensional factor matrices $\mathbf{P}$ and $\mathbf{Q}$, i.e. $\mathbf{R} \approx \hat{\mathbf{R}} = \mathbf{P}^{\mathbf{T}}\mathbf{Q}$, that model respectively users and items in some joint factor latent space of dimensionality k.

Accordingly, The rating matrix $\mathbf{R}$ will be of dimension $m x n$, where m is the number of users and n is the number of items. Each item i is associated with a vector $q_i \in \mathbb{R}^k$, the matrix $\mathbf{Q}$ will have then the size $m x k$, where $k <<< m$ is the size of the latent space. Similarly, each user u is associated with a vector $p_i \in \mathbb{R}^k$, the matrix $\mathbf{P}$ will have the size kxn. We will then approximate user u's rating of item i, which is denoted by r_{ui}, such that :

$$r_{ui} = q_i^T.p_u$$

For a pair of user and item (u_i, i_j) for which a rating r_{ij} exists, a common approach approach is based on the minimization of the ℓ_2-regularized quadratic error:

$$\ell_{u_i,i_j}(P,Q) = \left(r_{ij} - p_i^T q_j\right)^2 + \lambda(\|p_i\|^2 + \|q_j\|^2)$$

where p_i is the column vector composed of the i-th line of P and $\lambda \geqslant 0$ is a regularization parameter. The whole matrix factorization problem thus writes :

$$\min_{P,Q} \sum_{i,j:r_{ij}\text{exists}} \ell_{u_i,i_j}(P,Q).$$

Note that the error $\ell_{u_i,i_j}(P,Q)$ depends only on P and Q through p_i and q_j; however, item i_j may also be rated by user $u_{i'}$ so that the optimal factor q_j depends on both p_i and $p_{i'}$.

The initial rating dataset is characterized as follows :

Table 2: Ratings dataset

Number of users	6040
Number of movies	3952 where 3706 are already rated
Number of ratings	1000209
Density	4.19% of non zeros

Before starting the training of our algorithm that estimates the optimal matrices $\mathbf{P}$ and $\mathbf{Q}$, we initialize them by randomly filling them with values $p_{ij}, q_{ij} \in [0,1]$. Our Matrix Completion algorithm goes as follows :

```python
# function that calculate the gradient of the error
# returns the 2 gradient matrix of the function following P and Q
def MSE_gradSum(k,pu,qi,lamb,r,rHat):
  grad_p = np.zeros((nbUsers,k))
  grad_q = np.zeros((k,nbMovies))
  # derevate following each vector and add it to the gradients matrix P and Q
  for u in range(0,nbUsers):
      for i in range(0,nbMovies):
          # Condition of rating existance
          if r[u][i] != -1:
              _grad_p, _grad_q = MSE_grad(pu[u,:],qi[:,i],lamb,r[u,i],rHat[u,i])
              grad_p[u,:] = grad_p[u,:] + _grad_p
              grad_q[:,i] = grad_q[:,i] + _grad_q
  return grad_p, grad_q

# calculate the error function
def MSE_Sum(k,pu,qi,lamb,r,rHat):
  _MSE = 0
  for u in range(0,nbUsers):
      for i in range(0,nbMovies):
            if r[u][i] != -1:
                _MSE += MSE(pu[u,:],qi[:,i],lamb,r[u][i],rHat[u][i])
    return _MSE

# decent gradient algorithm
def grad_algoSum(k,lamb,nb_iter,matrixRating):
   t = 0
   gamma = 0.001

  pu = np.random.rand(nbUsers,k)
  qi = np.random.rand(k,nbMovies)
  # estimation matrix (initialized with random values in [0,1.0])
  rHat = np.dot(pu,qi)
  r = matrixRating # real rating matrix

  # initialize a vector that will contain the errors value at each iteration
  mse_tab = np.zeros(nb_iter)

  while t < nb_iter:
      if t % 1000 == 0:
          print(t, end=',')
      mse_grad = MSE_gradSum(k,pu,qi,lamb,r,rHat)
      mse_tab[t] = MSE_Sum(k,pu,qi,lamb,r,rHat)
      pu = pu - gamma * mse_grad[0]
      qi = qi - gamma * mse_grad[1]
      rHat = np.dot(pu,qi)
      t += 1
  return pu, qi, mse_tab
```

In the test phase, we have proceeded as follows: we have taken a small chunk of the rating matrix for which all the r_ij already exist for all users u_i and items i_j. Then, we have eliminated one single rating and replaced its value by -1. We can, then, study the efficiency of our algorithm and its ability to successfully predict the missing rating. This choice was due to the size of the rating matrix (composed in total of 1000209 ratings).

The testing chunk and the predicted values of this matrix are the presented via the tables 3 and 4. For the estimated chunk of ratings, we have chosen to just present the results found when we run the algorithm for 10000 iterations and for the latent space size corresponding to $k = 5$ which showed the best results among $k = 2, 5, 10, 50$.

Table 3: The testing rating matrix's chunk. The omitted value in this case is $r_{22} = 2$

user/ item	610	1983	3362
651	4	2	4
1128	3	-1	3
1285	3	2	5
1680	4	5	4
1733	3	3	4
1820	3	5	5
1941	3	3	5
2181	1	3	5
2909	3	4	4
3032	3	5	5
3626	4	3	4
3841	3	3	4
4009	1	4	5
4041	2	4	4
4064	2	2	4
4227	4	1	4
4238	3	3	4
4303	4	1	4
5627	4	2	3

Table 4: The estimated rating matrix's chunk for $k = 5$ and number of iteration equals to 10000. The omitted value in this case is $r_{22} = 2$

user/ item	610	1983	3362
651	3.9750711	2.00486042	3.99035857
1128	2.98158043	1.99677079	2.99653664
1285	2.99603254	2.01184164	4.96972839
1680	3.97673371	4.96361329	4.01296484
1733	2.98936068	2.99055717	3.99066794
1820	2.99769514	4.97059451	4.99233466
1941	2.99658674	2.9980926	4.97726381
2181	1.0240575	2.99698419	4.9628117
2909	2.98991489	3.97680813	3.99820336
3032	2.99769514	4.97059451	4.99233466
3626	3.9756253	2.99111138	3.99789399
3841	2.98936068	2.99055717	3.99066794
4009	1.02461171	3.98323515	4.97034712
4041	2.00365027	3.97625393	3.99097731
4064	2.00254186	2.00375201	3.97590646
4227	3.9745169	1.01860946	3.98282315
4238	2.98936068	2.99055717	3.99066794
4303	3.9745169	1.01860946	3.98282315
5627	3.9750711	2.00486042	3.99035857

As we can see from the table 4, the estimated values are so close to the real ratings with an error of order 10^{-2} or less. The missing rating value was so successfully estimated with an error of 10^{-3}.

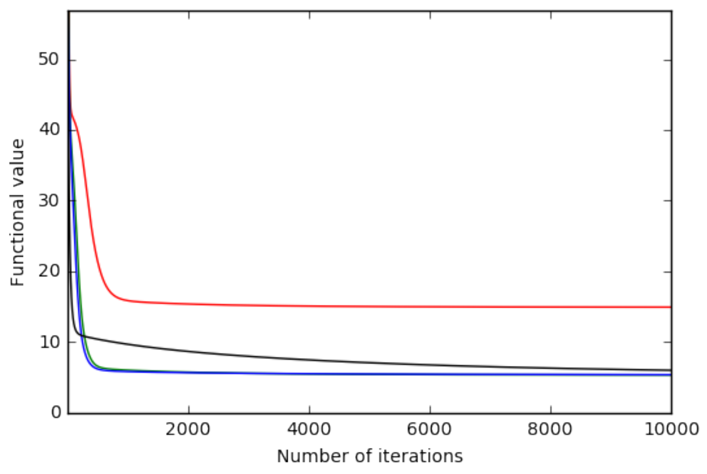

Figure 3: Convergence of the estimation error of the matrix completion's algorithm for different values of k. Each graph color represents a different value of k ; Red : 2, Green : 5, Blue : 10 and Black : 50

As we can notice in the figure 3, The estimation error tends to zero for all the values of k. However, the latent space sizes correponding of $k = 5$ and $k = 10$ present the best estimation results all while presenting an acceptable consensus between velocity of convergence and low estimation error.