Factorization-Based Data Modeling
Practical Work 1 - Report

ZHANG Xin

November 2019

The code is written in Python because there are some problems with the glnxa64 library of the MATLAB installed on Télécom computers.

1 Matrix Factorization with Alternating Least Squares and Gradient Descent

1.1 Implement the ALS update rules.

The update rules are implemented according to the following formulas:

$$W = XH^\top \left(HH^\top\right)^{-1}$$

$$H = \left(W^\top W\right)^{-1} W^\top X$$

```
Wals = X.dot(Hals.transpose()).dot(np.linalg.inv(Hals.dot(Hals.transpose())))
Hals = np.linalg.inv(Wals.transpose().dot(Wals)).dot(Wals.transpose()).dot(X)
```

1.2 Compute the objective function values for each iteration.

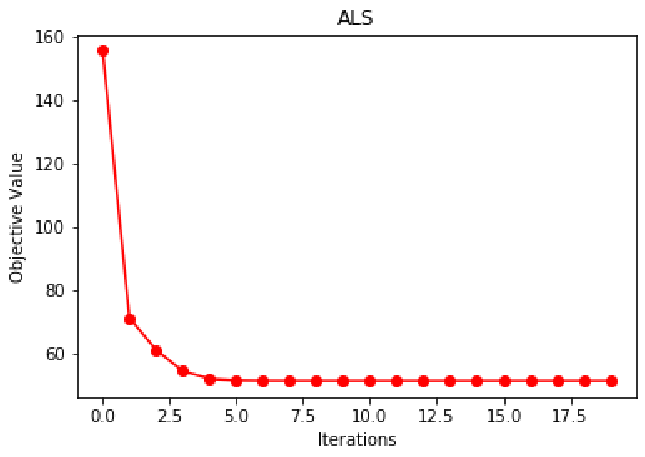
The object function is

$$\frac{1}{2}\|\mathbf{X} - \mathbf{WH}\|_F^2$$

So we have the code;

```
Xhat = Wals.dot(Hals)
obj_als[i] = 0.5 * (np.linalg.norm(X - Xhat))**2
```

And the graph of the value in 20 iterations is as follows:



1.3 Implement the GD update rules

The update rule is

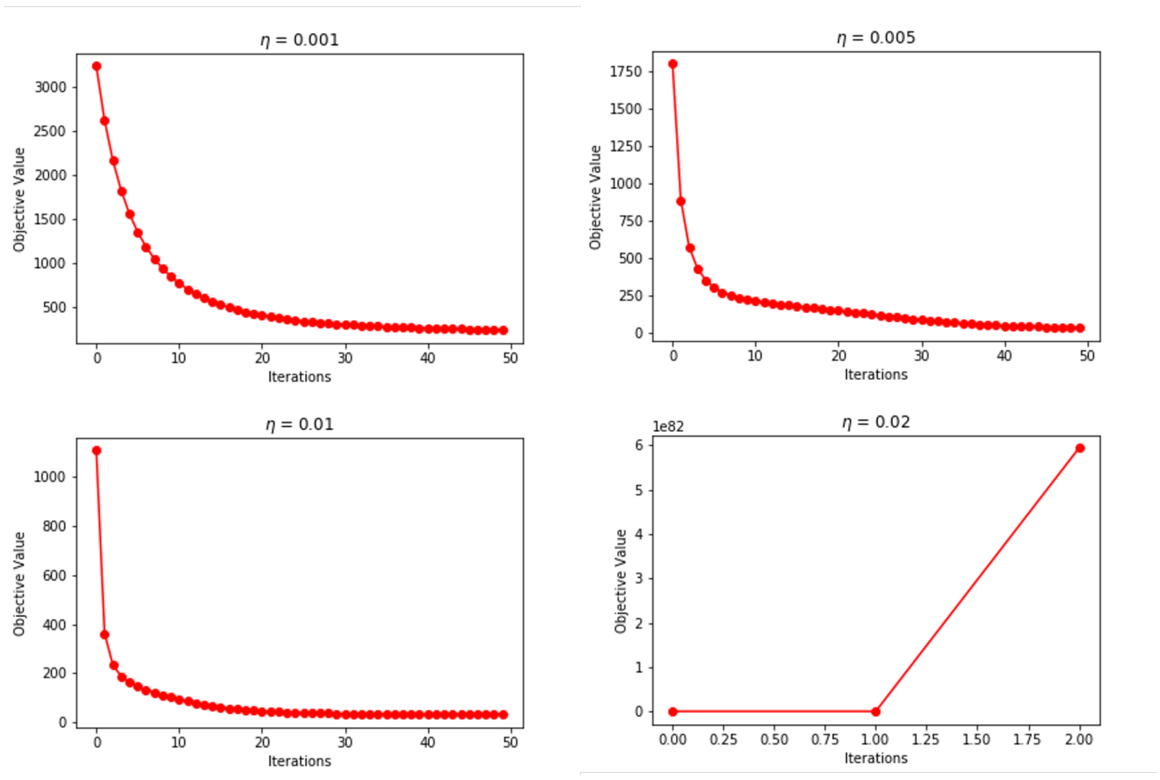
$$\mathbf{W} \leftarrow \mathbf{W} + \eta(\mathbf{X} - \mathbf{WH})\mathbf{H}^\top$$
$$\mathbf{H} \leftarrow \mathbf{H} + \eta\mathbf{W}^\top(\mathbf{X} - \mathbf{WH})$$

So according to this, the implementation is

```
Wgd = Wgd + eta * (X - Wgd.dot(Hgd)).dot(Hgd.transpose())
Hgd = Hgd + eta * Wgd.transpose().dot(X - Wgd.dot(Hgd))
```

1.4 What is the effect of η?

η is the step-size of the gradient descent, it is the learning rate which affects the speed of descent.
We can see the effect from the graphs below:



We can clearly see that when η is small enough, the bigger it is, the quicker the algorithm converges, but when η is too big, the algorithm will not be able to reach a local minimum because the step-size is too big.

1.5 Compute the objective function values for each iteration

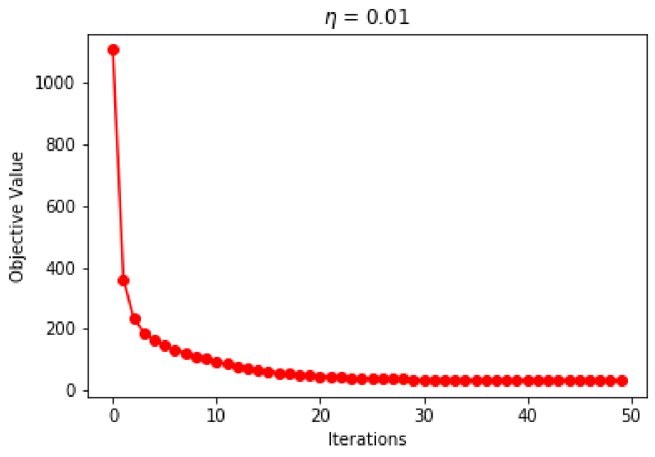
The object function is

$$\frac{1}{2}\|\mathbf{X} - \mathbf{WH}\|_F^2$$

So we have the code;

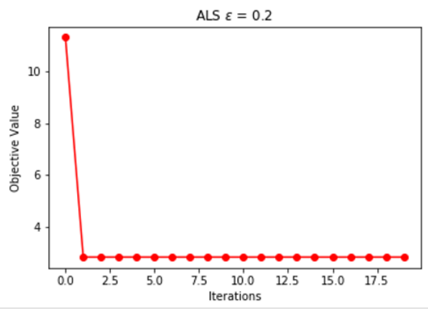
```
Xhat = Wgd.dot(Hgd)
obj_gd[i] = 0.5 * (np.linalg.norm(X - Xhat))**2
```

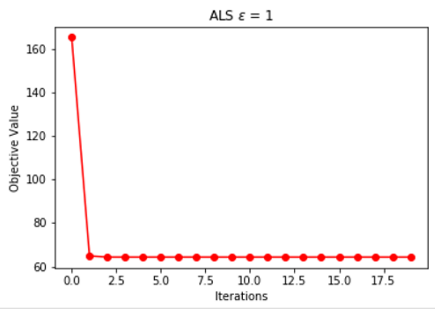
the resulting graph when $\eta = 0.01$ is:

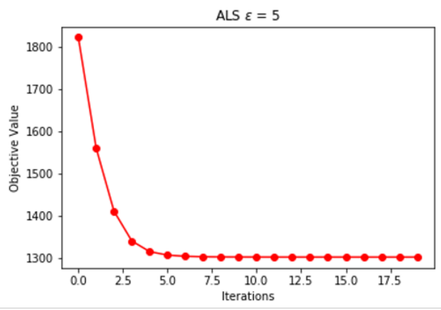


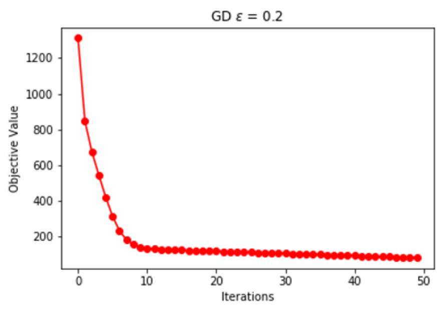
1.6 Play with the variable 'dataNoise'

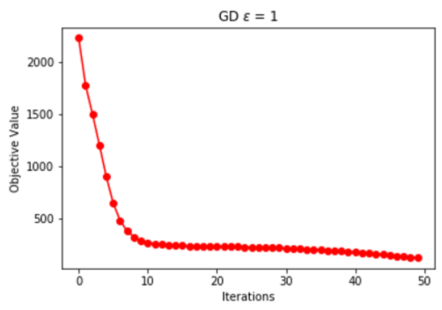
We keep $\eta = 0.01$, and we try with several noises, the result is as follows:

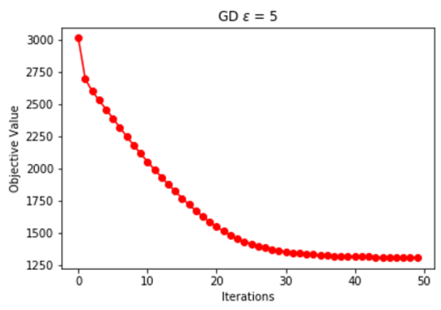








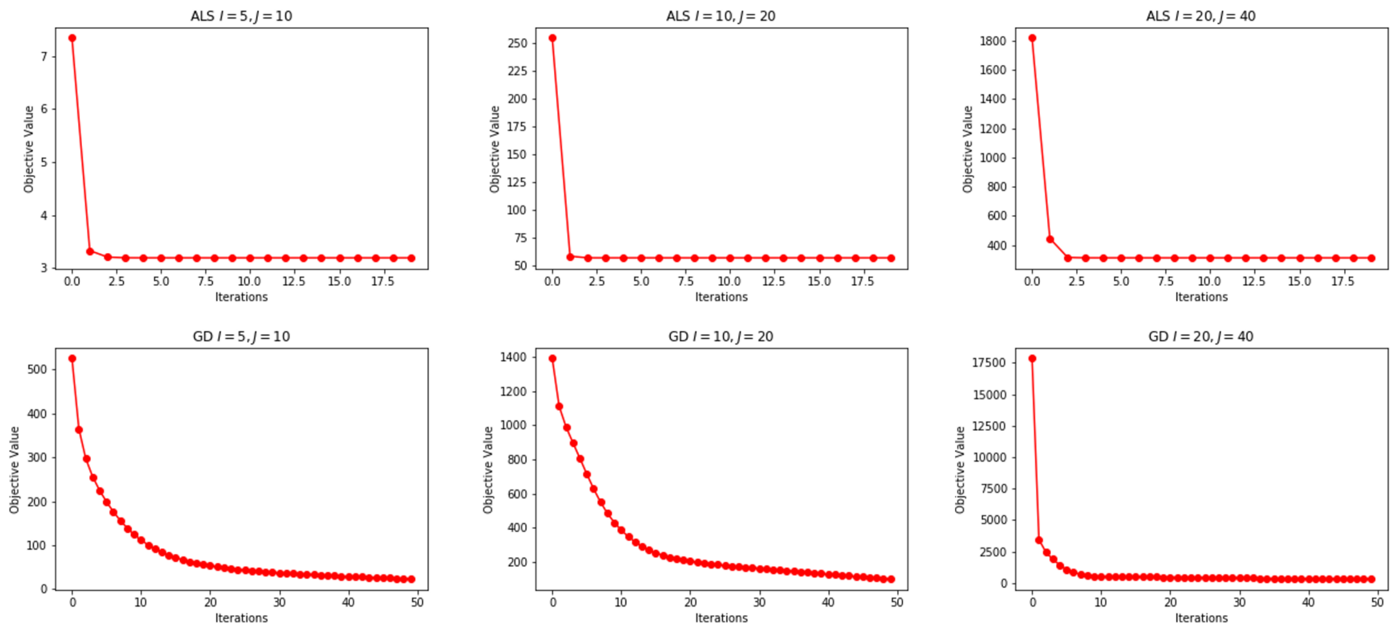




We can see for both algorithms, the bigger the noise, the slower it converges. Moreover, the algorithm will converge at a bigger value as the noise grows.
We can also notice the the ALS algorithm is less effected by the noise than GD, its convergence speed remains more stable at a high level, and the convergence value grows less and samller than that of GD.

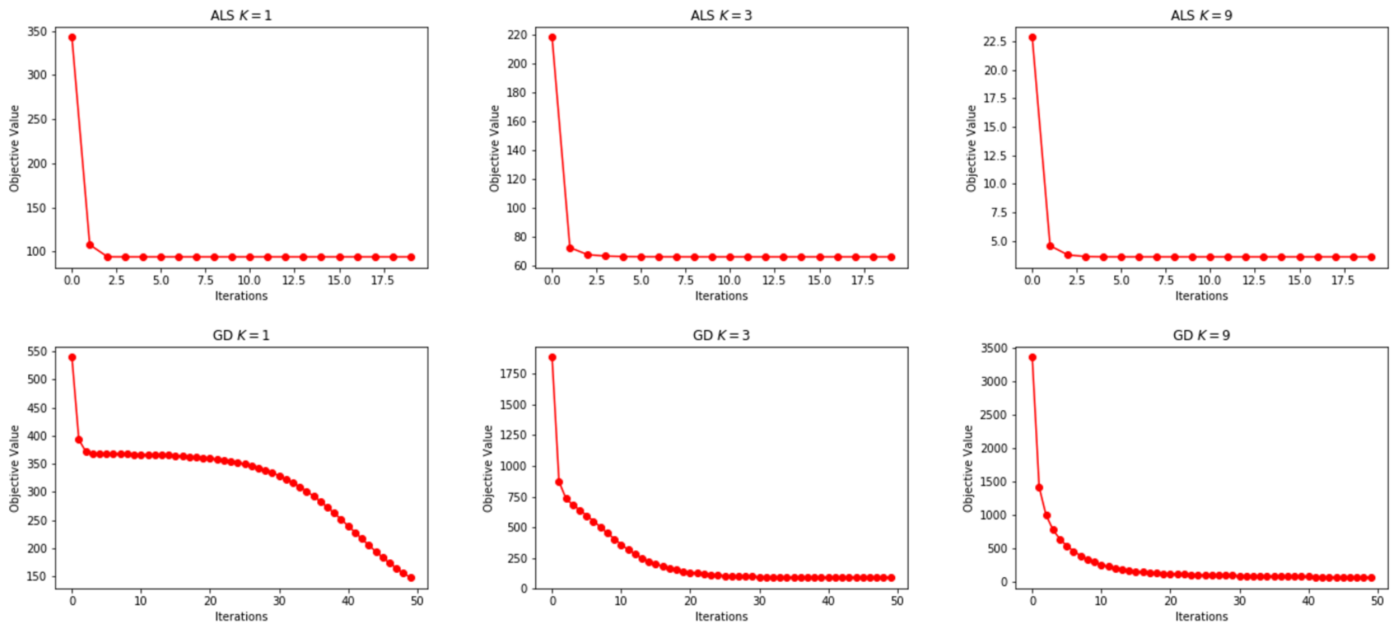
1.7 Play with the size of the data (I, J) and the rank of the factorization K.

Firstly, we keep $K = 3$, and change (I, J), the result is as follows ($\epsilon = 1$):



We can notice that as the size of data grows, the convergence value grows, but generally the two algorithms scale well with the data size. We can see that the gradient descent algorithm converges quicker as larger data size can help it better learn model parameters and improve the optimization process and imparts generalization. So ALS is more robust to small data size.

Now we keep $I = 10, J = 20$ and change the rank K:



We can see that the convergence value is smaller as the desired rank increases, but when the desired rank is smaller, it will be more difficult for GD algorithm to converge, but the ALS algorithm is robust.

1.8 Which algorithm do you think is better? Why?

The **Alternating Least Squares (ALS)** algorithm is better. Firstly, we can see that it is more robust to the changes of noise, data size and the desired rank, and it doesn't require us to choose a proper key parameter (which is η is GD). ALS is easier to parallelize and more practical to deal with implicit datasets, which are usually not sparse.

2 Non-Negative Matrix Factorization with Multiplicative Update Rules

2.1 Implement the MUR update rules.

According to the update rules:

$$W \leftarrow W \circ \frac{(X/\hat{X})H^\top}{OH^\top}$$

$$H \leftarrow H \circ \frac{W^\top(X/\hat{X})}{W^\top O}$$

where $\circ$ denotes element-wise multiplication and / and $\div$ denote element-wise division.
O is of size $I \times J$, such that each element of O will be equal to 1, i.e. $o_{ij} = 1$ for all i and j.
The implementation is as follows:

```
#Update W
Wmur = np.multiply(Wmur,np.divide(np.divide(X,Xhat).dot(Hmur.transpose()),O.dot(Hmur.transpose())))
# Update H
Hmur = np.multiply(Hmur,np.divide(Wmur.transpose().dot(np.divide(X,Xhat)),Wmur.transpose().dot(O)))

Xhat = Wmur.dot(Hmur)
Xhat = Xhat + eps
```

2.2 Compute the objective function values for each iteration.

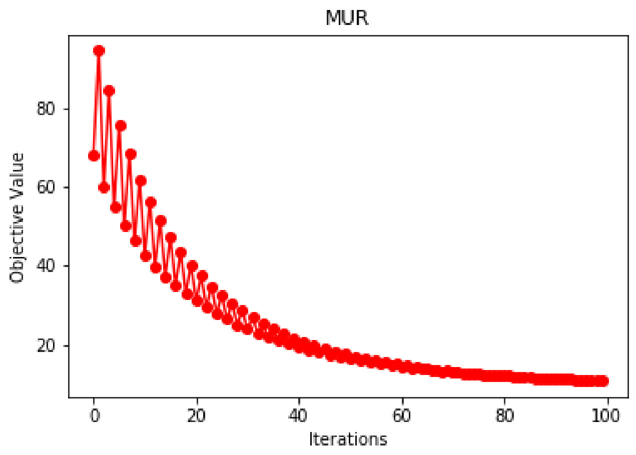
The objective function is

$$\sum_{i=1}^{I}\sum_{j=1}^{J}\left(x_{ij}\log\frac{x_{ij}}{\hat{x}_{ij}} - x_{ij} + \hat{x}_{ij}\right)$$

So we implement:

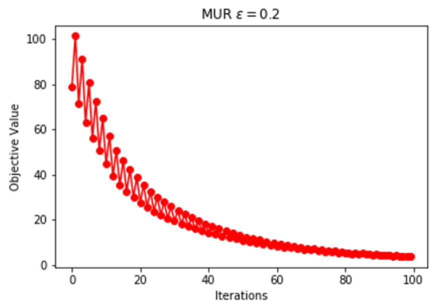
```
 #objective valuve for MUR
obj_mur[i] = np.nansum(np.multiply(X,(np.log(np.divide(X,Xhat)))) - X + Xhat)
```

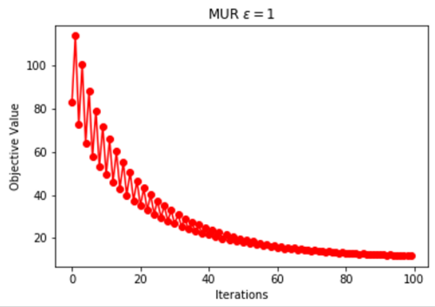
The corresponding graph is as follows:

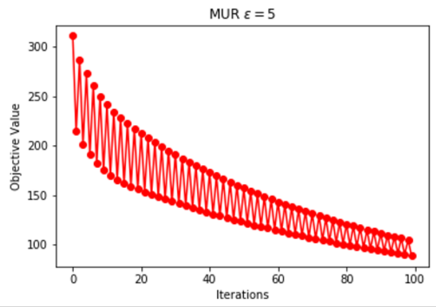


2.3 Play with the variable 'dataNoise'

We keep other parameters unchanged and alter the noise ϵ



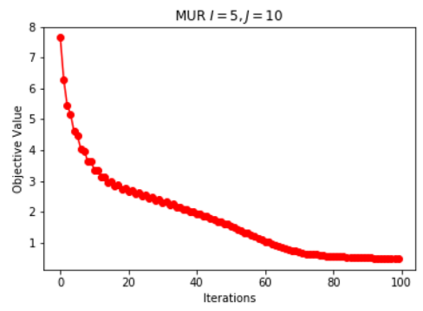


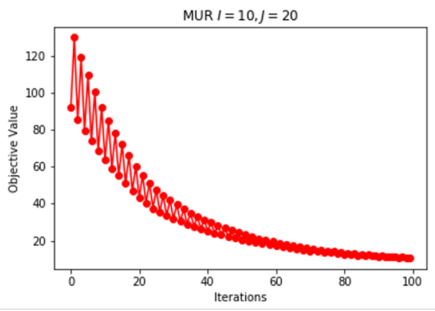


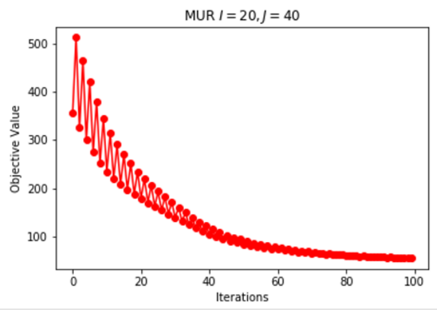
From the results above we can see that as the noise grows, it takes longer for MUR to converge, and and loss scales with the noise.

2.4 Play with the size of the data (I, J) and the rank of the factorization K

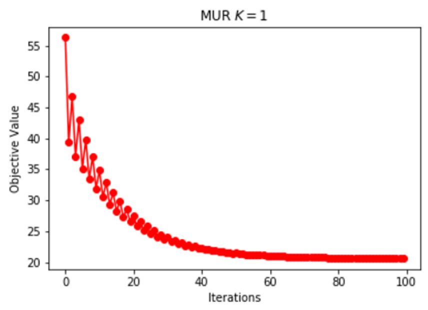
First we keep $K = 3$ and change (I, J)

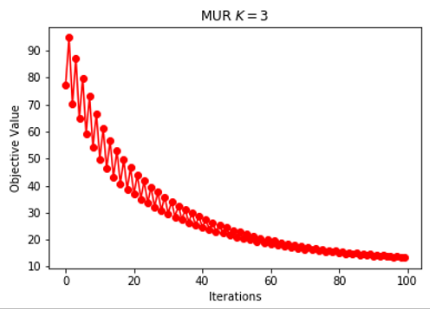


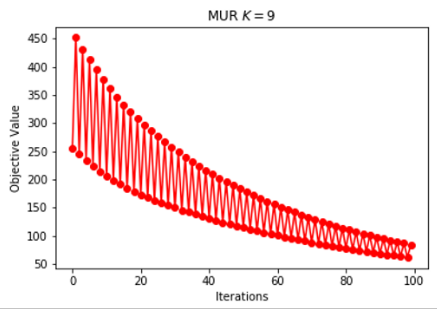




We can see that the data size don't really change the time for the algorithm to converge, but the curve is more stable when the size of data is smaller. And of course the loss will increase with the size of data.
Then we keep $I = 10, J = 20$ and change K







We can observe that it takes longer to converge if we choose a larger rank K, and the curve will be less stable.

2.5 Summary of MUR

We can see that the non-negativity of W and H can be guaranteed in the construction method of MUR, but MUR doesn't guarantee monotonicity, and MUR also has the highest convergence rate, which is also a drawback.