Factorisation de matrice avec PIG

Links: **notebook** (../_downloads/2015_factorisation_matrice.ipynb), **html** (../_downloads/2015_factorisation_matrice.html), **PDF** (../_downloads/2015_factorisation_matrice.pdf), **python** (../_downloads/2015_factorisation_matrice.py), **slides** (../_downloads/2015_factorisation_matrice.slides.html)

auteurs : *Théo Gantzer, Anna Korba*

Ce travail s'appuie sur l'article A Fast Distributed Stochastic Gradient Descent Algorithm for Matrix Factorization (http://www.jmlr.org/proceedings/papers/v36/li14.html), Fanglin Li, BinWu, Liutong Xu, Chuan Shi, and Jing Shi.

```
import pyensae
%nb_menu
```

Plan


```
import pyquickhelper, pyensae
params={"blob_storage":"", "password1":"", "hadoop_server":"", "password2":"", "u
sername":"alias"}
pyquickhelper.ipythonhelper.open_html_form(params=params,title="server + hadoop +
 credentials", key_save="blobhp")
```

server + hadoop + credentials

blob_storage

hadoop_server

password1

password2

username

alias

Ok

```
import pyensae
blobstorage = blobhp["blob_storage"]
blobpassword = blobhp["password1"]
hadoop_server = blobhp["hadoop_server"]
hadoop_password = blobhp["password2"]
username = blobhp["username"]
print(username)
client, bs =  %hd_open
client, bs
```

```
xaviermf
```

```
(<pyensae.remote.azure_connection.AzureClient at 0x8c2b2b0>,
 <azure.storage.blobservice.BlobService at 0x8c2bf98>)
```

```
%blob_ls /$PSEUDO
```

name	url

Téléchargment des données et transfert sur le cluster

```
import pyensae
url = "http://files.grouplens.org/datasets/movielens/"
file = "ml-1m.zip"
pyensae.download_data(file, website=url)
```

```
downloading of  http://files.grouplens.org/datasets/movielens/ml-1m.zip (http://f
iles.grouplens.org/datasets/movielens/ml-1m.zip)  to  ml-1m.zip
creating folder  .ml-1m
unzipped  ml-1m/movies.dat  to  .ml-1m/movies.dat
unzipped  ml-1m/ratings.dat  to  .ml-1m/ratings.dat
unzipped  ml-1m/README  to  .ml-1m/README
unzipped  ml-1m/users.dat  to  .ml-1m/users.dat
```

```
['.\ml-1m/movies.dat',
 '.\ml-1m/ratings.dat',
 '.\ml-1m/README',
 '.\ml-1m/users.dat']
```

```
import os
import pyensae
[ _ for _ in os.listdir() if "ml" in _ ]
```

```
['ml-1m', 'ml-1m.zip']
```

```
# Ici on charge les données puis on les stocke dans un dictionnaire
from itertools import islice

lines = np.genfromtxt('ml-1m/ratings.dat', delimiter="::", dtype=None)
my_dict = dict()
for i in range(len(lines)):
   my_dict[lines[i][0],lines[i][1]] = lines[i][2]
```

```
# Nous avons ((userID, movieID), rating)
def take(n, iterable):
     return list(islice(iterable,n))
print(take(10, my_dict.items()))
```

```
[((822, 1620), 4), ((2488, 1459), 3), ((3389, 423), 2), ((4305, 508), 4), ((1163,
 3752), 3), ((2895, 3070), 5), ((3780, 2916), 5), ((3308, 185), 3), ((3029, 296),
 5), ((1151, 1265), 4)]
```

```
### ICI on crée la matrice sparse R des ratings
import json
from pandas import Series,DataFrame
import numpy as np
from scipy import sparse
data = []
row = []
col = []
for k, v in my_dict.items():
    k=np.asarray(k)
    r = int(k[0])
    c = int(k[1])
    data.append(v)
    row.append(r)
    col.append(c)
R = sparse.coo_matrix((data,(row,col)))

# Et on la met sous forme de trois colonnes indice_ligne, indice_colonne, valeur
data=np.array([R.row, R.col, R.data])
data=np.transpose(data)
print(data)
```

```
[[ 822 1620    4]
 [2488 1459    3]
 [3389  423    2]
 ...,
 [2628 2240    2]
 [5762 2085    4]
 [2895 3173    3]]
```

```
# Attention il faut executer cette fenêtre deux fois (la premiere fois une erreur
 apparaît)
# Ici on créé les matrices P et Q dont le produit doit approximer R.
import pandas as pd
ratings = pd.read_csv('ml-1m/ratings.dat', header = None,sep="::",engine="python"
)
ratings.columns=['user_id','item_id', 'rating', 'timestamp']
ratings=ratings[['user_id','item_id', 'rating']]
#Series(df.values.ravel()).unique()
m=len(Series(ratings['user_id'].values.ravel()).unique()) #m nb d'users
n=len(Series(ratings['item_id'].values.ravel()).unique()) #n nb d'items
# Nous les codons de la même manière que data, c'est-à-dire sous forme de trois c
olonnes.
# Ces matrices sont initialisées avec des nombres aléatoires entre 0 et 1.
d=10 # dimension latente
# P est de dimension m*d
p=np.random.uniform(0,1,((m*d)))
row_p=[x for x in Series(ratings['user_id'].values.ravel()).unique() for j in ran
ge(0,10)]
col_int=[  i for j in range(0,m) for i in range(1,11) ]
P = sparse.coo_matrix((p,(row_p,col_int)))
matrix_p=np.array([P.row, P.col, P.data])
matrix_p=np.transpose(matrix_p)
# q est de dimension n*d
q=np.random.uniform(0,1,((n*d)))
row_q=[x for x in Series(ratings['item_id'].values.ravel()).unique() for j in ran
ge(0,10)]
col_int=[ i for j in range(0,n) for i in range(1,11)]
Q = sparse.coo_matrix((q,(row_q,col_int)))
matrix_q=np.array([Q.row, Q.col, Q.data])
matrix_q=np.transpose(matrix_q)
np.savetxt("sparse_matrix.csv", data, delimiter=",")
np.savetxt("matrix_p.csv", matrix_p, delimiter=",")
np.savetxt("matrix_q.csv", matrix_q, delimiter=",")
```

```
%blob_up sparse_matrix.csv /$PSEUDO/projet_DM/data_full.csv
```

```
'$PSEUDO/projet_DM/data_full.csv'
```

```
%blob_up matrix_p.csv /$PSEUDO/projet_DM/matrix_p_full.csv
```

```
'$PSEUDO/projet_DM/matrix_p_full.csv'
```

```
%blob_up matrix_q.csv /$PSEUDO/projet_DM/matrix_q_full.csv
```

```
'$PSEUDO/projet_DM/matrix_q_full.csv'
```

```
%blob_ls /$PSEUDO/projet_DM
```

	name	last_modified	content_type	content_length	blob_type
0	xaviermf/projet_DM/data_full.csv	Wed, 15 Jul 2015 22:55:22 GMT	application/octet-stream	75015675	BlockBlob
1	xaviermf/projet_DM/matrix_p_full.csv	Wed, 15 Jul 2015 22:56:19 GMT	application/octet-stream	4530000	BlockBlob
2	xaviermf/projet_DM/matrix_q_full.csv	Wed, 15 Jul 2015 22:56:52 GMT	application/octet-stream	2779500	BlockBlob

Implémentation python

```
def matrix_factorization (R, P, Q, K, steps =100 , gamma =0.02 , lambd =0.02) :
    Q = Q.T
    # update des matrices P et Q
    for step in range ( steps ):
        for i in range (len (R)):
            for j in range (len(R[i])):
                if R[i][j] > 0:
                    eij = R[i][j] - np.dot (P[i ,:] ,Q[:,j])
                    for k in range (K):
                        P[i][k] = P[i][k] + gamma * (2 * eij * Q[k][j] - lambd *
P[i][k])
                        Q[k][j] = Q[k][j] + gamma * (2 * eij * P[i][k] - lambd *
Q[k][j])
        eR = np.dot (P,Q)
        e = 0
        # calcul de la fonction de cout
        for i in range (len (R)):
            for j in range (len(R[i])):
                if R[i][j] > 0:
                    e = e + pow (R[i][j] - np.dot(P[i ,:] ,Q[:,j]) , 2)
                    for k in range (K):
                        e = e + ( lambd /2) * ( pow(P[i][k] ,2) + pow (Q[k][j] ,2
) )
        if e < 0.001:
            break
    return P, Q.T
```

Normalement, comme le décrit le code python ci-dessous, pour réaliser la mise à jour de p_{ik} on doit se déplacer sur l'ensemble des colonnes j de Q : en effet, à j fixé, il faut mettre à jour p_{ik} en ajoutant la contribution du coefficient q_{kj} à l'erreur ; la prochaine fois que p_{ik} sera modifié, c'est lorsque l'on sera passé à la colonne suivante $j+1$ et que l'on ajoutera la contribution du coefficient $q_{k,j+1}$ à l'erreur. Cette modification à trois boucles sur i, j, k à été pour nous un vrai-casse tête à implémenter en PIG.

Nous avons finalement choisi de mettre à jour P puis Q. Pour ce faire, nous avons d'abord calculé tous les termes $\gamma e_{ij} q_{kj}$ puis les nouvelles valeurs d'une manière plus simple : $p_{ik} = p_{ik}(1-\lambda\gamma) + \sum_{j=1}^{n} \gamma e_{ij} q_{kj}$. De la même manière, nous avons mis à jour Q avec les coefficients de P mis à jour. Nous sommes conscients que cette façon de modifier les coefficients n'est pas équivalente à la première mais c'est la meilleure solution que nous ayons trouvée. Pour nos expériences, nous avons d'abord testé notre code sur les 100 premières lignes de la base de données, qui contiennent les notes de 2 utilisateurs sur 99 films, et avons regardé la distance euclidienne qui sépare R et PQ sur trois itérations (les calculs étaient déjà relativement lents).

Lors de l'implémentation de cette méthode en Pig, nous nous sommes aperçus d'une divergence du produit PQ par rapport à la matrice R : la somme des coefficients au carré de $R - PQ$ était de 418 après initialisation de P et Q avec une loi uniforme sur $[0, 1]$, puis de 1070 après une itération et de 2893 après deux itérations. Le problème provient vraisemblablement de la mise à jour de P et Q car la méthode est différente de celle présentée en Python. De plus l'implémentation en Pig ne donne pas les résultats escomptés.

Nous avons alors testé une autre méthode, où nous avons calculé indépendemment les termes $p_{ik} = p_{ik}(1-\lambda\gamma) + \gamma e_{ij} q_{kj}$ pour j variant de 1 à n et avons fait la moyenne : $p_{ik} = p_{ik}(1-\lambda\gamma) + \frac{1}{n}\sum_{j=1}^{n} \gamma e_{ij} q_{kj}$. Au bout d'une itération, la distance entre R et PQ est de 349.58, au bout de deux itérations 289.54. La méthode est convergente, que ce soit pour un échantillon réduit d'utilisateurs ou pour l'ensemble de la base de données (100 000 notes).

En effet, nous avons lancé le calcul en Pig sur les données initiales et en près de deux heures, nous avons obtenu après 3 itérations la distance euclidienne des matrices PQ successives à la matrice R. Par souci de clarté dans le code, nous n'avons pas fait figurer davantage d'itérations sachant qu'il s'agit de copier-collers supplémentaires (car Pig ne gère pas les boucles). La distance euclidienne après 0, 1 et 2 vaut respectivement, pour l'ensemble de la base de données 2,9.10^6, 2,6.10^6, 2,2.10^6. La distance semble converger.

Implémentation PIG

```
%%PIG matrix_factorization2.pig
-------------------------------------------------------------------------------
-----------------------------------
--                                                  Iteration 0
-------------------------------------------------------------------------------
-----------------------------------
-----------------------------------------------------------------
-- On charge les matrices R, P et Q;
-----------------------------------------------------------------
R = LOAD '$CONTAINER/$PSEUDO/projet_DM/data_full.csv' USING PigStorage(',') AS (u
serID:int,movieID:int,rate:double);
P = LOAD '$CONTAINER/$PSEUDO/projet_DM/matrix_p_full.csv' USING PigStorage(',') A
S (userID:int, latent_p:int, val_p:double);
Q = LOAD '$CONTAINER/$PSEUDO/projet_DM/matrix_q_full.csv' USING PigStorage(',') A
S (movieID:int, latent_q:int, val_q:double);
--DUMP P ;
--DUMP Q ;
-----------------------------------------------------------------
-- On calcule le produit matriciel entre P et Q
-----------------------------------------------------------------
-- La commande suivant joint P et Q
P_and_Q = JOIN P BY latent_p, Q BY latent_q ;
-- La commande suivante calcule tous les pik*qkj ou i=userID et j=movieID
P_x_Q = FOREACH P_and_Q GENERATE userID, movieID, latent_p, val_p*val_q AS produi
t ;
-- Ici on groupe tous les termes pik*qkj avec le meme couple(i,j) pour pouvoir so
mmer sur les k variables latentes
grouped_P_x_Q = GROUP P_x_Q by (userID, movieID) ;
calcul_PQ = FOREACH grouped_P_x_Q GENERATE group, SUM(P_x_Q.produit) AS ps ;
-- Ici on recupere la matrice PQ :(i,j, ligne pi*colonneqj)
PQ = FOREACH calcul_PQ GENERATE FLATTEN(group) AS (userID_2, movieID_2), ps ;

-----------------------------------------------------------------
-- On calcule lerreur  et lerreur au carre
-----------------------------------------------------------------
-- La commande suivante joint les matrices R et PQ
R_and_PQ = JOIN R BY (userID, movieID), PQ BY (userID_2, movieID_2) ;

-- Dans cette matrice on calcule lerreur au carre faite sur chaque coefficient de
 R
E = FOREACH R_and_PQ GENERATE userID_2, movieID_2, (rate-ps) AS error, (rate-ps)*
(rate-ps) AS error_sq ;
-- syntaxe pour sommer = group all au lieu de group by
resultat_group = GROUP E ALL ;
resultat = FOREACH resultat_group GENERATE SUM(E.error_sq) ;
DUMP resultat ;
--STORE resultat into '$CONTAINER/$PSEUDO/projet_DM/resultat_iteration_0' using P
IGStorage(',','-schema') ;
-------------------------------------------------------------------------------
-----------------------------------
--                                                  Iteration 1
```

```
-------------------------------------------------------------------------------
-----------------------------------
-----------------------------------------------------------------
-- On met a jour P et Q
-----------------------------------------------------------------
-- Tout dabord on joint les matrices E et P
E_and_P = JOIN E BY userID_2, P BY userID ;
E_and_P_bis = FOREACH E_and_P GENERATE userID, movieID_2, latent_p, error,  val_p
 ;
-- On joint ensuite le resultat avec Q
E_and_P_and_Q = JOIN E_and_P_bis BY (movieID_2, latent_p), Q BY (movieID, latent_
q) ;

-------------- Mise a jour de P et Q
-- On calcule la matrice des gamma*e_ij*q_kj (qui vont servir a la mise a jour de
s p_ik) et des p_ik
P_update1 = FOREACH E_and_P_and_Q GENERATE userID, movieID, latent_p, error, (val
_p*(1-$lamb*$gamma)+$gamma*error*val_q) AS val_p, val_q;
Q_update1 = FOREACH P_update1 GENERATE userID, movieID, latent_p AS latent_q, err
or, val_p, (val_q*(1-$lamb*$gamma)+$gamma*error*val_p) AS val_q ;
P_group = GROUP P_update1 by (userID, latent_p) ;
P_new = FOREACH P_group GENERATE group, AVG(P_update1.val_p) AS val_p ;
P = FOREACH P_new GENERATE FLATTEN(group) AS (userID, latent_p), val_p ;
--STORE P into '$CONTAINER/$PSEUDO/projet_DM/matrix_p_1' using PIGStorage(',','-s
chema') ;

-- Idem pour Q
Q_group = GROUP Q_update1 by (movieID, latent_q) ;
Q_new = FOREACH Q_group GENERATE group, AVG(Q_update1.val_q) AS val_q ;
Q = FOREACH Q_new GENERATE FLATTEN(group) AS (movieID, latent_q), val_q ;
--STORE Q into '$CONTAINER/$PSEUDO/projet_DM/matrix_q_1' using PIGStorage(',','-s
chema') ;

-----------------------------------------------------------------
-- On calcule le produit matriciel entre P et Q
-----------------------------------------------------------------
-- La commande suivant joint P et Q
P_and_Q = JOIN P BY latent_p, Q BY latent_q ;
-- La commande suivante calcule tous les pik*qkj ou i=userID et j=movieID
P_x_Q = FOREACH P_and_Q GENERATE userID, movieID, latent_p, val_p*val_q AS produi
t ;
-- Ici on groupe tous les termes pik*qkj avec le meme couple(i,j) pour pouvoir so
mmer sur les k variables latentes
grouped_P_x_Q = GROUP P_x_Q by (userID, movieID) ;
calcul_PQ = FOREACH grouped_P_x_Q GENERATE group, SUM(P_x_Q.produit) AS ps ;
-- Ici on recupere la matrice PQ :(i,j, ligne pi*colonneqj)
PQ = FOREACH calcul_PQ GENERATE FLATTEN(group) AS (userID_2, movieID_2), ps ;

-----------------------------------------------------------------
-- On calcule lerreur  et lerreur au carre
-----------------------------------------------------------------
-- La commande suivante joint les matrices R et PQ
```

```
R_and_PQ = JOIN R BY (userID, movieID), PQ BY (userID_2, movieID_2) ;
-- Dans cette matrice on calcule lerreur au carre faite sur chaque coefficient de
 R
E = FOREACH R_and_PQ GENERATE userID_2, movieID_2, (rate-ps) AS error, (rate-ps)*
(rate-ps) AS error_sq ;

-- syntaxe pour sommer = group all au lieu de group by
resultat_group = GROUP E ALL ;
resultat = FOREACH resultat_group GENERATE SUM(E.error_sq) ;
DUMP resultat ;
--STORE resultat into '$CONTAINER/$PSEUDO/projet_DM/resultat_iteration_1' using P
IGStorage(',','-schema') ;

-------------------------------------------------------------------------------
-----------------------------------
--                                                 Iteration 2
-------------------------------------------------------------------------------
-----------------------------------
-------------------------------------------------------------------
-- On met a jour P et Q
-------------------------------------------------------------------
-- Tout dabord on joint les matrices E et P
E_and_P = JOIN E BY userID_2, P BY userID ;
E_and_P_bis = FOREACH E_and_P GENERATE userID, movieID_2, latent_p, error,  val_p
 ;
-- On joint ensuite le resultat avec Q
E_and_P_and_Q = JOIN E_and_P_bis BY (movieID_2, latent_p), Q BY (movieID, latent_
q) ;

-------------- Mise a jour de P et Q
-- On calcule la matrice des gamma*e_ij*q_kj (qui vont servir a la mise a jour de
s p_ik) et des p_ik
P_update1 = FOREACH E_and_P_and_Q GENERATE userID, movieID, latent_p, error, (val
_p*(1-$lamb*$gamma)+$gamma*error*val_q) AS val_p, val_q;
Q_update1 = FOREACH P_update1 GENERATE userID, movieID, latent_p AS latent_q, err
or, val_p, (val_q*(1-$lamb*$gamma)+$gamma*error*val_p) AS val_q ;
P_group = GROUP P_update1 by (userID, latent_p) ;
P_new = FOREACH P_group GENERATE group, AVG(P_update1.val_p) AS val_p ;
P = FOREACH P_new GENERATE FLATTEN(group) AS (userID, latent_p), val_p ;
--STORE P into '$CONTAINER/$PSEUDO/projet_DM/matrix_p_2' using PIGStorage(',','-s
chema') ;
-- Idem pour Q
Q_group = GROUP Q_update1 by (movieID, latent_q) ;
Q_new = FOREACH Q_group GENERATE group, AVG(Q_update1.val_q) AS val_q ;
Q = FOREACH Q_new GENERATE FLATTEN(group) AS (movieID, latent_q), val_q ;
--STORE Q into '$CONTAINER/$PSEUDO/projet_DM/matrix_q_2' using PIGStorage(',','-s
chema') ;
-------------------------------------------------------------------
-- On calcule le produit matriciel entre P et Q
-------------------------------------------------------------------
-- La commande suivant joint P et Q
P_and_Q = JOIN P BY latent_p, Q BY latent_q ;
```

```
-- La commande suivante calcule tous les pik*qkj ou i=userID et j=movieID
P_x_Q = FOREACH P_and_Q GENERATE userID, movieID, latent_p, val_p*val_q AS produi
t ;
-- Ici on groupe tous les termes pik*qkj avec le meme couple(i,j) pour pouvoir so
mmer sur les k variables latentes
grouped_P_x_Q = GROUP P_x_Q by (userID, movieID) ;
calcul_PQ = FOREACH grouped_P_x_Q GENERATE group, SUM(P_x_Q.produit) AS ps ;
-- Ici on recupere la matrice PQ :(i,j, ligne pi*colonneqj)
PQ = FOREACH calcul_PQ GENERATE FLATTEN(group) AS (userID_2, movieID_2), ps ;

----------------------------------------------------------------
-- On calcule lerreur  et lerreur au carre
----------------------------------------------------------------
-- La commande suivante joint les matrices R et PQ
R_and_PQ = JOIN R BY (userID, movieID), PQ BY (userID_2, movieID_2) ;
-- Dans cette matrice on calcule lerreur au carre faite sur chaque coefficient de
 R
E = FOREACH R_and_PQ GENERATE userID_2, movieID_2, (rate-ps) AS error, (rate-ps)*
(rate-ps) AS error_sq ;

-- syntaxe pour sommer = group all au lieu de group by
resultat_group = GROUP E ALL ;
resultat = FOREACH resultat_group GENERATE SUM(E.error_sq) ;
DUMP resultat ;
--STORE resultat into '$CONTAINER/$PSEUDO/projet_DM/resultat_iteration_2' using P
IGStorage(',','-schema') ;
```

```
client.pig_submit(bs,
                  client.account_name,
                  "matrix_factorization2.pig",
                  params = dict(gamma="0.02", lamb="0.02"),
                  stop_on_failure=True )
```

```
{'id': 'job_1435385350894_0101'}
```

```
st = %hd_job_status job_1435385350894_0101
st["id"],st["percentComplete"],st["status"]["jobComplete"]
```

```
('job_1435385350894_0101', '21% complete', False)
```

```
%tail_stderr job_1435385350894_0101 300
```

```

```

```
%blob_downmerge /$PSEUDO/projet_DM/matrix_p_1 matrix_p_1.csv
```

```
%blob_downmerge /$PSEUDO/projet_DM/matrix_p_2 matrix_p_2.csv
```

```
%blob_downmerge /$PSEUDO/projet_DM/matrix_q_1 matrix_q_1.csv
```

```
%blob_downmerge /$PSEUDO/projet_DM/matrix_q_2 matrix_q_2.csv
```

```
from scipy.sparse import matrix_coo
matrix_p_1 = open("matrix_p_1.csv","r").read()
matrix_p_2 = open("matrix_p_2.csv","r").read()
matrix_q_1 = open("matrix_q_1.csv","r").read()
matrix_q_2 = open("matrix_q_2.csv","r").read()
matrix_p_1 = matrix_coo(matrix_p_1).toarray()
matrix_p_2 = matrix_coo(matrix_p_2).toarray()
matrix_q_1 = matrix_coo(matrix_q_1).toarray()
matrix_q_2 = matrix_coo(matrix_q_2).toarray()
#mettre R, P, Q sous forme "normale" (depuis sparse)

PQ_1 = np.dot(matrix_p_1, matrix_q_1.T)
PQ_2 = np.dot(matrix_p_2, matrix_q_2.T)
#Erreurs au carre
print(np.sum((R-PQ_1)**2))
print(np.sum((R-PQ_2)**2))
```