Recommender system

I. **Introduction**

Matching customers to products is an important practical problem that many companies would be interested in being able to do well. Products information provided by users can be used to develop recommendation systems to help recommend to customers new products that they may like.
Ratings prediction is a common application of such models and is explored in this project. Starting with a database of users/objects ratings, a matrix factorization model is used to predict customers ratings about products they had not rated yet depending on:

- how they rated other products,
- how other customers had rated any of the products.

A rating dataset(1) from the MovieLens website is used for this project.
It contains 100,000 ratings applied to 9,000 movies by 700 users.
Ratings scores are given between 0.5 and 5.

An histogram of the global distribution of the known ratings independently of the type of movies rated, can be seen on Figure 1.

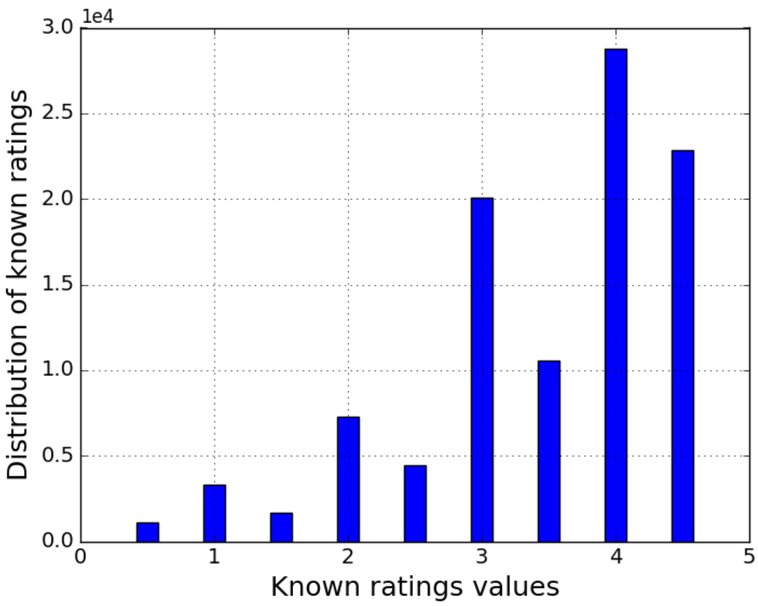

Figure 1: Distribution of known ratings

(1) https://grouplens.org/datasets/movielens/

II. **Methodology**

The ratings matrix characteristic of the recommender system is formed using any possible user/object pair, and the ij^{th} entry in this matrix, m_{ij}, is going to be the rating that the i^{th} user gave to the j^{th} object (Cf. Figure 2).
This matrix will have many missing values, because each user can only rate a fraction of the products. Matrix factorization will try to learn a low-rank factorization of this matrix using only the observed data, while ignoring the data we don't have. From there, it will be possible to fill in all of the missing values that

could be then viewed as predictions for what a user will rate an object and as such, later be used to make recommendations.

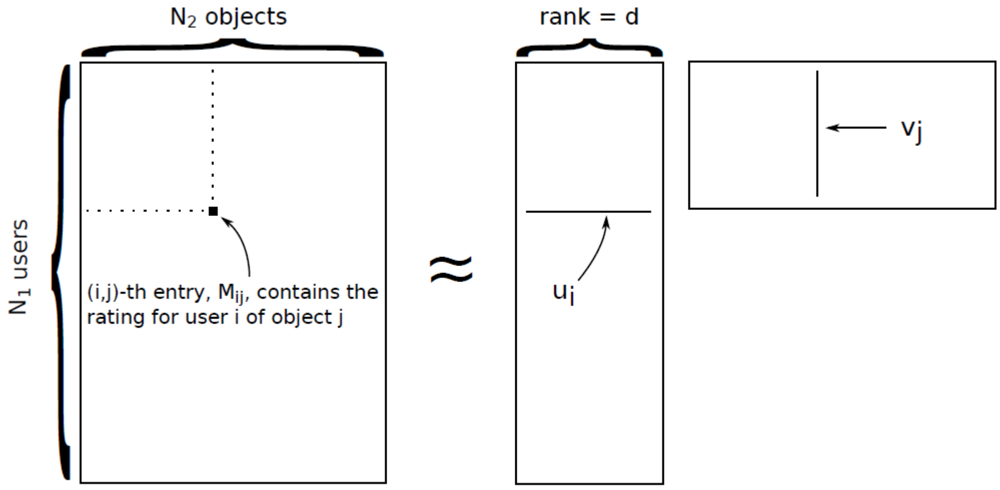

Figure 2: Low-rank matrix factorization

We assume that the ratings matrix M is a N_1 by N_2 matrix of rank d, that can be approximated to the dot product of a N_1 by d U matrix (of objects j rated by users i), times a d by N_2 V matrix (of users i who rated objects j), where $d \ll min(N_1, N_2)$:

$$M \approx UV^T$$

Probabilistic matrix factorization (PMF) is one particular model for learning a low-rank factorization in this missing data problem, and where we make the following assumptions:

- The location $u_i \in \mathbb{R}^d$ of the N_1 users and the location $v_j \in \mathbb{R}^d$ of the N_2 objects are going to be generated from zero mean Gaussians with spherical covariances: $u_i \approx N(0, \lambda^{-1}I)$ and $v_j \approx N(0, \lambda^{-1}I)$.
- The distribution on the data given those two locations is going to be: $M_{ij} \approx N(u_i^\top v_j, \sigma^2)$, where M_{ij} is an observed value if (i, j) is a measured pair of user/object.

Solving this problem would mean to find the matrix $\hat{M} = UV^T$ that minimizes the sum squared distance to the target matrix M. It is equivalent to maximizing the log of the joint likelihood over U and V, which can be written as:

$$\mathcal{L} = -\sum_{(i,j)\in\Omega} \frac{1}{2\sigma^2}||M_{ij} - u_i^\top v_j||^2 - \sum_{i=1}^{N_1} \frac{\lambda}{2}||u_i||^2 - \sum_{j=1}^{N_2} \frac{\lambda}{2}||v_j||^2 + \text{Cst}$$

By taking the derivatives of $\mathcal{L}$ over u_i and v_i independently and setting them to zero, it is then possible to solve for each u_i and v_i individually.

1. Data preprocessing and initialization

   A very important step before doing any analysis is to check and reindex the user/object database in order to not have any missing index between any consecutive users/objects.
   Because it is difficult to make accurate predictions for users who made very few ratings, any users with less than a certain number of ratings were removed from the database before analysis. Similarly, objects with less than a given number of ratings were removed from the database.
   The filtered dataset was split into two sets, a training set used to build the prediction model, and a

testing set containing 10% of the original dataset and that will be used to assess performance. The objects matrix V is initialized as a zero mean Gaussian $v_j \approx N(0, \lambda^{-1} I)$.

2. Iteration process

For each iteration, matrices U then V are respectively calculated and updated in two consecutive steps until convergence of the loss function $\mathcal{L}$:

- Step 1: Update users locations for $i = 1, \cdots, N_1$

$$u_i = \left(\lambda \sigma^2 I + \sum_{j \in \Omega_{u_i}} v_j v_j^\top \right)^{-1} \left(\sum_{j \in \Omega_{u_i}} M_{ij} v_j \right)$$

- Step 2: Update objects locations for $j = 1, \cdots, N_2$

$$v_i = \left(\lambda \sigma^2 I + \sum_{i \in \Omega_{v_j}} u_i u_i^\top \right)^{-1} \left(\sum_{i \in \Omega_{v_j}} M_{ij} u_i \right)$$

III. Results

Many different parameters were tested during analysis, but the results presented in this document used the following parameters:

- Number of iterations: 50
- Shape parameter: $\lambda = 2$
- Variance: $\sigma^2 = 0.1$
- The algorithm was used to learn 5 dimensions: $d = 5$.
- Minimum number of rated objects per user: 10.
- Minimum number of ratings per object: 3.

As seen on Figure 3, the objective function was shown to converge towards a maximum.

The ratings matrix M was calculated as the dot product between matrices U and V. The global distribution of the ratings can be seen on Figure 4.

$R < -2$	$-2 \leq R < 0$	$0 \leq R \leq 5$	$5 < R \leq 7$	$7 < R$
1.66%	8.28%	86.93%	2.58%	0.54%

Table 1: Frequencies of predicted ratings R

The first observation we can make from those results is that 13.06% of the ratings are predicted outside the range of valid ratings values.

Furthermore, the root mean square error was found to be equal to 1.03.

Improvements need to be brought to the model in order to restrain predictions to the valid ratings range, as well as decrease the RMSE by introducing for instance regularizing parameters.

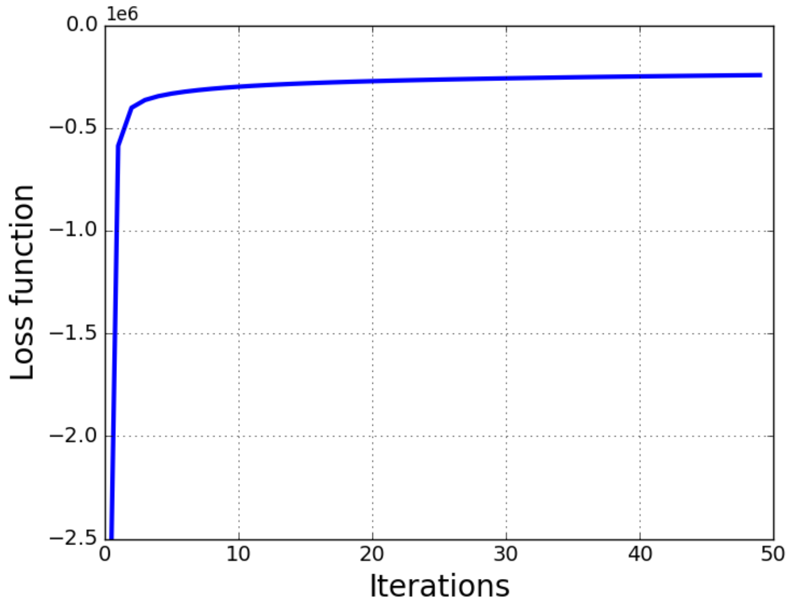

Figure 3: Objective function $\mathcal{L}$ over iteration time

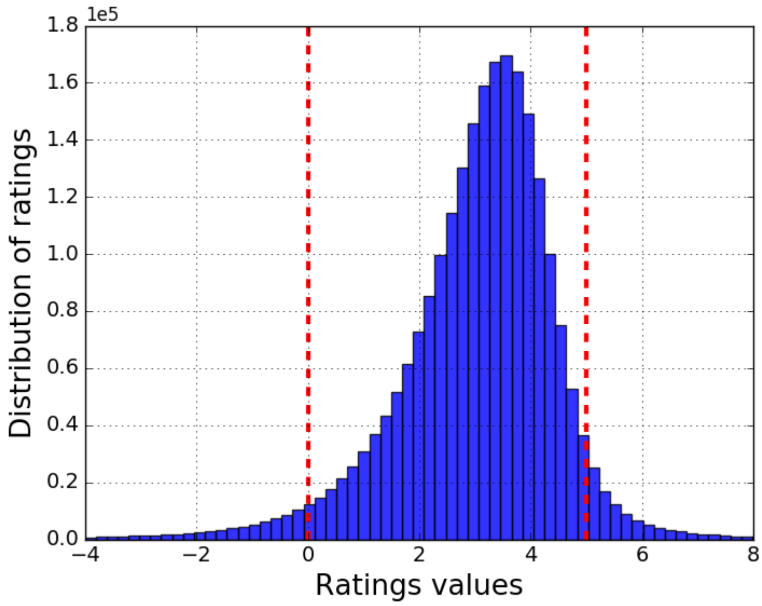

Figure 4: Ratings distribution

IV. Python Code

```
from __future__ import division
import numpy as np
import sys
import pandas as pd
from numpy.random import multivariate_normal as MvN
from matplotlib import pyplot as plt

"""
This file has to be run directly from the terminal using the following command li
    > python PMF.py "ratings.csv"
```

```
where 'hw4_PMF.py' is the name of python file and "ratings.csv" contains the rati
"""

try:
    assert len(sys.argv) > 1, "missing input argument"
    train_data = np.genfromtxt(sys.argv[1], delimiter=",")
except Exception as e:
    print(e)
    exit()

print('data_loaded')

# Parameters of the PMF objective function to maximize
lam = 2          # lambda shape parameter
sigma2 = 0.1     # variance
d = 5            # rank
iterations = 50 # number of iterations


LD = len(train_data)    # nb of data points

# Load data as dataframe
data = pd.DataFrame(train_data)

# Get index set of objects omega_ui rated by user i
omega_ui = {}
set_ui = set()
for user, obj in zip(data[0],data[1]):
    if user not in set_ui:
        set_ui.add(int(user))
        omega_ui[int(user)] = [int(obj)]
    else:
        omega_ui[int(user)].append(int(obj))
N1 = len(omega_ui.keys())

# Get index set of users omega_vj who rated object j
omega_vj = {}
set_vj = set()
for user, obj in zip(data[0],data[1]):
    if obj not in set_vj:
        set_vj.add(int(obj))
        omega_vj[int(obj)] = [int(user)]
    else:
        omega_vj[int(obj)].append(int(user))
N2 = len(omega_vj.keys())

# create dictionary with known data
Mij = {}
set_Mij = set()
for user, obj, mij in zip(data[0], data[1], data[2]):
    Mij[(int(user), int(obj))] = mij

# Initialisation of vj
```

```
def init_V():
    V = MvN(mean=np.zeros((N2)), cov=np.identity(N2) * 1 / lam, size = d)
    return V

# update ui
def update_ui(V,i):
    i += 1
    ui = np.zeros((5,1))
    VT = V.T

    # first sum
    sum1 = 0
    for j in omega_ui[i]:
        vjT = VT[j-1]
        vj = vjT.T
        sum1 += np.outer(vj,vjT)
    sum1 += lam*sigma2*np.identity(5)

    # second sum
    sum2 = 0
    for j in omega_ui[i]:
        vjT = VT[j-1]
        vj = vjT.T
        sum2 += Mij[(i, j)]*vj
    a = np.zeros((d,d))
    for k in range(d):
        a[k][0] = sum2[k]
    sum2 = a

    # product
    prod = np.dot(np.linalg.inv(sum1), sum2)
    for k in range(d):
        ui[k] = prod[k][0]

    return ui.reshape(-1)

# update vj
def update_vj(U,j):
    j += 1
    vj = np.zeros((5,1))
    # first sum
    sum1 = 0
    for i in omega_vj[j]:
        ui = U[i-1]
        uiT = ui.T
        sum1 += np.outer(ui,uiT)
    sum1 += lam*sigma2*np.identity(5)

    # second sum
    sum2 = 0
    for i in omega_vj[j]:
        ui = U[i-1]
        sum2 += Mij[(i, j)]*ui
```

```python
    a = np.zeros((d,d))
    for k in range(d):
        a[k][0] = sum2[k]
    sum2 = a

    # product
    prod = np.dot(np.linalg.inv(sum1), sum2)
    for k in range(d):
        vj[k] = prod[k][0]

    return vj.reshape(-1)

# Calculate loss function
def loss(U,V):
    # first sum
    sum1 = 0
    VT = V.T
    for ij in Mij.keys():
        i = ij[0]
        j = ij[1]
        vjT = VT[j-1]
        sum0 = Mij[ij]- np.dot(U[i-1],vjT)
        sum1 += (sum0)**2
    sum1 *= 1/(2*sigma2)

    # second sum
    sum2 = 0
    for i in range(N1):
        sum2 += np.linalg.norm(U[i], ord=2)**2
    sum2*= lam / 2

    # third sum
    sum3 = 0
    for j in range(N2):
        sum3 += np.linalg.norm(VT[j], ord=2)**2
    sum3 *= lam / 2

    # global sum
    L = - sum1 - sum2 - sum3

    return L

def PMF(V):
    #Initialisation
    VT = np.zeros((N2,d))
    U = np.zeros((N1,d))

    # update ui
    for i in range(N1):
        U[i] = update_ui(V, i)


    # update vj
```

```
    for j in range(N2):
        VT[j] = update_vj(U, j)
    V = VT.T

    # Calculate loss function
    L = loss(U, V)

    return L, U, V


V = init_V()
L_save = []
for iteri in range(iterations):
    # print(iteri)
    L, U, V = PMF(V)
    VT=V.T
    L_save.append(L)
    if iteri == 9:
        np.savetxt("U-10.csv", U, delimiter=",")
        np.savetxt("V-10.csv", VT, delimiter=",")
    if iteri == 24:
        np.savetxt("U-25.csv", U, delimiter=",")
        np.savetxt("V-25.csv", VT, delimiter=",")
    if iteri == 49:
        np.savetxt("U-50.csv", U, delimiter=",")
        np.savetxt("V-50.csv", VT, delimiter=",")

np.savetxt("objective.csv", L_save, delimiter=",")


# Plot objective function
plt.plot(L_save)
plt.grid()
plt.xlabel('Iterations')
plt.ylabel('Loss function')
plt.show()
```