**Technical Report for Recommendation in Social Networks: Probabilistic Graphical Models**

By

Akash Soni (0873602)

Chirag Patel (0883437)

Dhruvi Desai (0884718)

Department of Computer Science

Lakehead University

**Overview :**

Our project is based on the recommendation of movie. When a new user or existing user is introduced to the system it recommends movies related to the existing users rating for different movies and recommends movies with the same category or genre. To implement this we have used Python and a dataset from MovieLens. There are three types of recommender system possible, they are: -

1. The Simple Recommender.
2. Recommendation based on Collaborative Filtering.
3. Recommendation based on Content Based Filtering.

Out of the three, we have chosen to implement Recommendation Based on Collaborative Filtering which is the most popular approach to build Recommendation System and has been successfully employed in many applications. CF is based on the similarity in preference, tastes, and choices of two users. It works by collecting user feedback in the form of ratings for items in a given domain. The main advantage of CF recommender system is that it does not rely on the machine analyzable contents and therefore it is capable of accurate recommendations. In CF, user who had similar choices in the past, will have similar choices in the future as well.

**Language :**

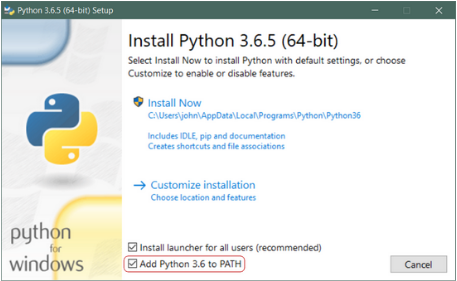
Python is a popular programming language and has many uses like web development, software development, system scripting and so on. Python can be used on server to create web application, software to create workflows, connect to database to read and modify files, can handle big data and perform complex mathematics. Python can used on different software, it is a complete cross-platform application and the language used is simple syntax to English language.

**Download and installation:**

1. Download the Python 3 installer from the below link for windows.

* <https://www.python.org/downloads/windows/>

1. Search for the latest python version and a preferred version upon your choice.
2. Click on download link that says “[Windows x86 executable installer](https://www.python.org/ftp/python/3.7.1/python-3.7.1.exe)” for 32 Bit or “[Windows x86-64 executable installer](https://www.python.org/ftp/python/3.7.1/python-3.7.1-amd64.exe)” for 64 Bit.
3. Run the downloaded file and a window would open as shown below :-



1. Make sure to check the Add Python X.X to path check box in the end as show in the image.
2. Click Install now and the installation process begins.

That should be all to get python up and running.

**Installing the libraries in Python:**

To implement the method, we would require the following library for python along with the command line to install the library: -

1. SciPy

Command: - “pip install SciPy”

1. NumPy

Command: - “pip install NumPy”

1. Matplotlib

Command: - “pip install matplotlib”

1. Panda

Command: - “pip install panda”

Note: - Please run this command in CMD (Command Prompt) and in the directory where python is installed.

**Dataset:**

Our dataset has been taken from MovieLens.com which contains a consolidated data about users and the movie ratings. This is a tab separated list of user ID | item ID | rating | timestamp.

The data is spread into multiple CSV files zipped into a folder. It contains information about the item and the genres of the movies are indicated with 0 or 1 and the date of users such as the demographic of each user.

**Implementation:**

1. **Build simple Recommender System using Collaborative Filtering Technique:**

Frist thing to do is to load the dataset with Pandas to the dataframes data, Item and users.

As shown below.

The code to load the dataset with different items and taking them in matrix is given below. Initially, three basic libraries are imported as shown in first three lines. Comments have been mentioned that explains each part of the code.

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

*#column headers for the dataset*

data\_cols = ['user id','movie id','rating','timestamp']

item\_cols = ['movie id','movie title','release date',

'video release date','IMDb URL','unknown','Action',

'Adventure','Animation','Childrens','Comedy','Crime',

'Documentary','Drama','Fantasy','Film-Noir','Horror',

'Musical','Mystery','Romance ','Sci-Fi','Thriller',

'War' ,'Western']

user\_cols = ['user id','age','gender','occupation',

'zip code']

*#importing the data files onto dataframes*

users = pd.read\_csv('Desktop/ml-100k/u.user', sep='|',

names=user\_cols, encoding='latin-1')

item = pd.read\_csv('Desktop/ml-100k/u.item', sep='|',

names=item\_cols, encoding='latin-1')

data = pd.read\_csv('Desktop/ml-100k/u.data', sep='\t',

names=data\_cols, encoding='latin-1')

You can check if the dataframes are loaded from or not by using the below functions which will print the header of each dataframe.

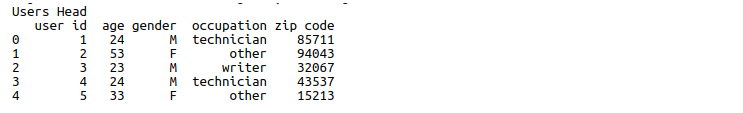
*#printing the head of these dataframes*

print(users.head())

print(item.head())

print(data.head())

The output would be as show for Users Head :-



The code has been run in the python IDLE. In that we need to just press run from the dropdown menu.

**In addition, how to run is being shown in video.**

* **Code For Collaborative Filtering Technique.**

For collaborative Filtering Technique the first step would be to merge all the dataframes together as give below with code.

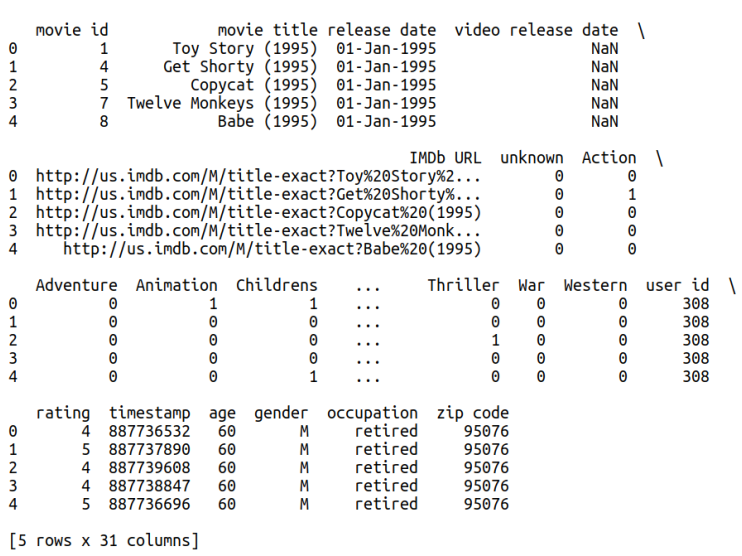
Here we have taken the records for the first 5 movies in the dataset.

*#Create one data frame from the three*

dataset = pd.merge(pd.merge(item, data),users)

print(dataset.head())

Therefore the output of the above code:

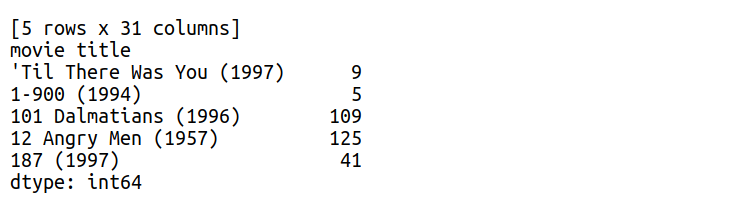


Then we apply group by function to the movies title and using the size function to return the total number of entries under each movie title which would help us find the number of people who rated the movie/ the number of ratings.

ratings\_total = dataset.groupby('movie title').size()

print(ratings\_total.head())

Output :-



Then we apply mean function to take the mean rating of each movie. For that we again apply groupby Movie Title. For the resulting dataframe we select only the movie title and the rating header.

ratings\_mean = (dataset.groupby('movie title'))['movie title','rating'].mean()

print(ratings\_mean.head())

Now if you check the ratings total we would get a series and not a dataframe. So we will convert that into dataframe and the ratings mean we will see that the movie title has been converted from a column to an index. So we can convert that back to a column.

*#modify the dataframes so that we can merge the two*

ratings\_total = pd.DataFrame({'movie title':ratings\_total.index,

'total ratings': ratings\_total.values})

ratings\_mean['movie title'] = ratings\_mean.index

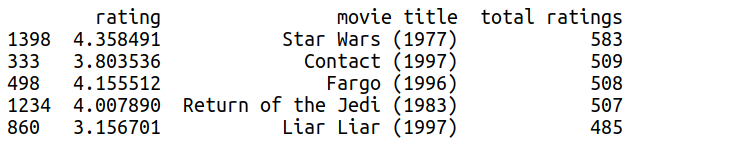
Now we can merge the values. So by sorting the total rating, it would help use sort the data frames by the number of people who views a particular movie.

final = pd.merge(ratings\_mean, ratings\_total).sort\_values(by = 'total ratings',

ascending= False)

print(final.head())

Output :-



**Limitations:**

Collaborative filtering technique algorithms and implementations for applications like recommender system face several challenges :

1. Size of processed dataset.
2. Sparseness of rating matrix, where for each user only a relatively small number of items are rated.
3. Very high complexity.
4. Poor Scalability.

These challenges are very well taken care by Matrix Factorization.

1. **Build Recommender System using Matrix Factorization Technique: (Explanation of the blocks of the code is being mentioned in the comments shown and the equation is the code are being explained below: How they are derived?)**

**The Maths of Matrix Factorization**

It is very important to discuss the mathematics behind the model of matrix factorization. Initially, we have a set U (users), and a set D (items). Let R of size |U|×|D| be the matrix that contains all the user-item ratings that are being assigned. We want to find the K latent features that are required for the model implementation. Our task is to find two matrices: P (of size |U| × |K|) and Q (of size |D| × |K|) such that their product defined as |R|:



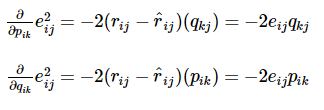
Considering this equation, each row of P would represent the strength of the associations between a user and the features. Similarly, each row of Q would describes the strength of the associations between an item and the features. By calculating the dot product of their vectors, we get the prediction of a rating of an item dj by ui:



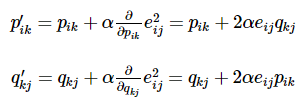
Now, our aim is to obtain P and Q. The approach to this problem is: first initialize the two matrices with some values, find their product and calculate how different their product is to M. Afterwards, try to minimize this difference iteratively. This method is called gradient descent which aims at finding a local minimum of the difference. The difference is called the error between the estimated rating and the real ratings. For calculating, we use the following equation for each user-item pair:



Here we are taking in consideration the squared error because the estimated rating can be either higher or lower than the real rating. For minimizing the error, we need to get an idea in which direction we have to modify the values of pik and qkj. In different words, we should find the gradient at the current values and for doing so we differentiate the above equation with respect to these two variables separately:



After getting the gradient, now we can formulate the update rules for both pik and qkj:



Here, α is a constant whose value which determines the speed of approaching the minimum. Generally, choose a small value for α, say 0.0002. The reason is if we make bigger step towards the minimum we might run into the risk of missing the minimum and ultimately end up oscillating around the minimum.

#Importing the numpy library to use it

import numpy as np

class MF():

def \_\_init\_\_(self, R, K, alpha, beta, iterations):

"""

Perform matrix factorization to predict empty

entries in a matrix.

Arguments

- R (ndarray) : user-item rating matrix

- K (int) : number of latent dimensions

- alpha (float) : learning rate

- beta (float) : regularization parameter

"""

self.R = R

self.num\_users, self.num\_items = R.shape

self.K = K

self.alpha = alpha

self.beta = beta

self.iterations = iterations

def train(self):

# Initialize user and item latent feature matrice

self.P = np.random.normal(scale=1./self.K, size=(self.num\_users, self.K))

self.Q = np.random.normal(scale=1./self.K, size=(self.num\_items, self.K))

# Initialize the biases

self.b\_u = np.zeros(self.num\_users)

self.b\_i = np.zeros(self.num\_items)

self.b = np.mean(self.R[np.where(self.R != 0)])

# Create a list of training samples

self.samples = [

(i, j, self.R[i, j])

for i in range(self.num\_users)

for j in range(self.num\_items)

if self.R[i, j] > 0

]

# Perform stochastic gradient descent for number of iterations

training\_process = []

for i in range(self.iterations):

np.random.shuffle(self.samples)

self.sgd()

mse = self.mse()

training\_process.append((i, mse))

if (i+1) % 10 == 0:

print("Iteration: %d ; error = %.4f" % (i+1, mse))

return training\_process

def mse(self):

"""

A function to compute the total mean square error

"""

xs, ys = self.R.nonzero()

predicted = self.full\_matrix()

error = 0

for x, y in zip(xs, ys):

error += pow(self.R[x, y] - predicted[x, y], 2)

return np.sqrt(error)

def sgd(self):

"""

Perform stochastic graident descent

"""

for i, j, r in self.samples:

# Computer prediction and error

prediction = self.get\_rating(i, j)

e = (r - prediction)

# Update biases

self.b\_u[i] += self.alpha \* (e - self.beta \* self.b\_u[i])

self.b\_i[j] += self.alpha \* (e - self.beta \* self.b\_i[j])

# Update user and item latent feature matrices

self.P[i, :] += self.alpha \* (e \* self.Q[j, :] - self.beta \* self.P[i,:])

self.Q[j, :] += self.alpha \* (e \* self.P[i, :] - self.beta \* self.Q[j,:])

def get\_rating(self, i, j):

"""

Get the predicted rating of user i and item j

"""

prediction = self.b + self.b\_u[i] + self.b\_i[j] + self.P[i, :].dot(self.Q[j, :].T)

return prediction

def full\_matrix(self):

"""

Computer the full matrix using the resultant biases, P and Q

"""

return self.b + self.b\_u[:,np.newaxis] + self.b\_i[np.newaxis:,] + self.P.dot(self.Q.T)

R = np.array([

[5, 3, 0, 1],

[4, 0, 0, 1],

[1, 1, 0, 5],

[1, 0, 0, 4],

[0, 1, 5, 4],

])

mf = MF(R, K=2, alpha=0.1, beta=0.01, iterations=20)

training\_process = mf.train()

print()

print("P x Q:")

print(mf.full\_matrix())

print()

print("Global bias:")

print(mf.b)

print()

print("User bias:")

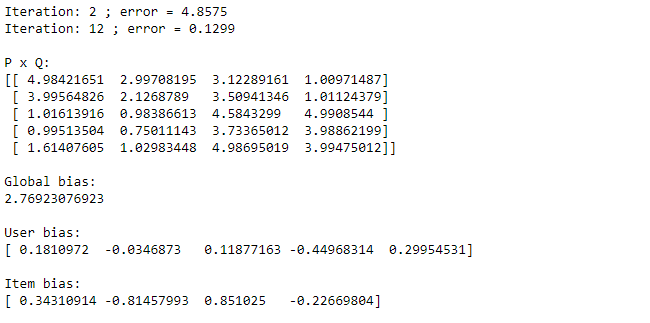
print(mf.b\_u)

print()

print("Item bias:")

print(mf.b\_i)

**Output:**



The mean square error for iteration 2 and 12 has been shown. After that, P x Q matrix (User-Item rating matrix) is been obtained. The global bias, User matrix bias and the item bias has been shown.

**For Plotting a graph of Mean Square Error Vs. Iteration, Use the following code :**

Intially x and y are extracted from the training model. Figure size has been defined. Afterwards, labels have been defined. And at last, the graph of iterations vs. Mean Square Error has been plotted.

x = [x for x, y in training\_process]

y = [y for x, y in training\_process]

plt.figure(figsize=((16,4)))

plt.plot(x, y)

plt.xticks(x, x)

plt.xlabel("Iterations")

plt.ylabel("Mean Square Error")

plt.grid(axis="y")

**Output:**

As seen in output, as the number of iterations increases, the mean square error gets reduced. This means the model is becoming better and gradient descent is working efficiently.

