Final Assignment

Bangladesh University of Engineering and Technology

Department of Electrical and Electronic Engineering

Course Code:	EEE 6608
Course Title:	Machine Learning and Pattern Recognition
Date of Submission:	30/03/2023
Submitted By:	Md. Abrar Zahin
Roll No:	0422062540

Table of Contents

Movie Recommendation System

1.1 Introduction

With the expansion of machine learning and artificial intelligence nowadays it's being used in every sector of our life. Streaming service providers like Amazon prime and Netflix are gaining more and more users day by day. With the growth of new users and demand, there is always a burden for streaming service providers to suggest the right content to the right types of people.

The first stage of creating a recommendation system is to find the correlated users who are similar in terms of movie preferences (which can be denoted by their similar type of rating). The second stage of this problem is to forecast the ratings of movies that have yet to be rated by a user (movies that are not still rated). As a result, we'll need the answers to the following questions:

- How do we know the users are similar?
- How do we predict the movie rating of a user which is not still rated yet?

The answer to the first question is using a technique called collaborative filtering. There are a lot of algorithms available under collaborative filtering. Collaborative filtering can be achieved user-wise or item-wise.

The answer to the second question is quite tricky, but here we tackled it by randomizing a user rating matrix and updating its values by calculating the loss and minimizing the loss from the original user rating matrix and calculating the gradient of each loss and minimizing the loss using the gradient descent algorithm.

1.2 Our Approaches

In this project, we implemented the movie recommendation system by using two approaches.

In the first approach, we used collaborative filtering with matrix factorization. Here we would find out a user feature matrix and another rating feature matrix. Later we would find out the dot product of both of these two and try to figure out the loss between the actual user movie rating matrix and the predicted movie rating matrix. Later by using gradient descent algorithms we would try to minimize the losses for each item. Finally, after minimizing the loss, we would get the preferred movie list for a specific user. Then after, we would plot the loss vs iterations and loss vs feature dimensions graphs to find out the impact of iterations and feature dimensions on the accuracy of our predicted matrix.

In the second approach, we built the recommendation system, using the already existing library for collaborative filtering known as Scikitlearn surprise. We used GridSearchCV and SVD for finding the optimized parameters for the user-movie-rating pivot table. Then we would predict a few movies for a user to recommend.

1.3 Methodology (Collaborative Filtering with Matrix Factorization)

1.3.1 Library Used

Library	Use
Matplotlib	For plotting the graphs
NumPy	For performing different mathematical operations
Pandas	For extracting and searching information from csv files

First, we unzipped the file and extracted all the CSV file information. Later unzipping all the files, we observed closely and then we chose the rating.csv file as our main file because it contains the ratings for different users for different movies. After that, we created our 2D matrix of users and their respective movie ratings by using the panda's pivot table. It's an MxN matrix where m is the number of users and n is the number of movies that exist in the rating.csv file. Then after, we implemented Matrix Factorization.

1.3.2 Matrix Factorization

Let's consider the M number of users and the N of movie ratings for each user. Now that's an MxN=R size matrix. To perform matrix multiplication of such a huge matrix is not only very time-consuming but also very resource intensive. At any time, the number of users m and the number of movies starts to grow exponentially with time, we can't handle such a matrix due to resource constraints. So, to tackle this situation we used a technique called matrix factorization. Here comes an important concept called a latent feature.

Suppose we want to figure out a matrix U such that MxK=U and another matrix V such that Nxk=V. Here U represents the association of the user with the features and their strength. On the other hand, V portrays a relationship between feature K and movie ratings N. In this case, our primary focus is to find the U, the association of user and features matrix and V, an association of movie ratings and feature matrix; where UxV=R (user movie rating matrix).

First, we are going to initialize an MxN with some random values, later we would try to figure out the deviation of the actual user-movie rating matrix. After calculating the deviation, we would use gradient descent algorithms to update the values so that overall error is minimized and we reach the local minima for each item.

All of the above-mentioned steps need to be performed for a few iterations so that we may converge to the local minimum. Later, we would plot the overall average loss for each iteration and see its pattern. Furthermore, the loss vs feature dimensions graphs would be plotted for a fixed iteration, so that we may understand how the feature dimension affects the loss of the user-movie-rating matrix. At the end, we will get a 2d user movie rating predicted matrix where each row represents the different users and each column represents the rating the user might give to that movie.

After implementing our recommendation system, the questions arise, is our recommendation system effective? How to judge it so to do so we also have implemented another recommendation system using Scikit-learn library's GridSearch CV and surprise package. By and large, we choose a few users for both systems and compared the predictions.

1.4 Methodology (Collaborative Filtering with Grid SearchCV)

In this approach, GridSearch CV is used along with SVD (Singular Value Decompositions) to decompose the user rating matrix to a smaller matrix with rank K, so that we may have less computational expense while calculating the dot products of user feature and rating feature matrix.

1.4.1 GridSearchCV

GridSearchCV is a way of tuning the hyperparameters for defining the optimal values of the given model. This GridSearchCV function assists the users to go through different hyperparameters and fits the model based on the training dataset. It uses a cross-validation method for different iterations and calculates loss/accuracy for the model. Finally, it suggests the best combination of hyperparameters to choose for the best performance.

1.4.2 Result & Comparison

After going through the movie.csv file we have found 19 genres for movies. Each movie contains at least one or more genres.

Genre Id	Movie Genres
1	Action
2	Adventure
3	Animation
4	Children
5	Comedy
6	Crime
7	Documentary
8	Drama
9	Fantasy
10	Film Noir
11	Horror
12	Imax
13	Musical
14	Mystery
15	Romance
16	Sci-Fi
17	Thriller
18	War
19	Western

So based on the movie genres we would calculate the F1 score on movie genres for each predicted movie by our recommendation system and the scikit-learn surprise library used recommendation system.

UserId	**Recommended Movie(predicted)**	**Recommended Movie (ground truth)**	**Genre Id(predicted)**	**Genre Id (ground truth)**
1	Star Wars: Episode V	Toy Story (1995)	1,2,16,8,15,5, 6,17	2,3,4,5,9,1, 6,17,14,11
	Eternal Sunshine of the Spotless Mind (2004)	Heat (1995)		
	Austin Powers: International Man of Mystery (1997)	Seven (a.k.a. Se7en) (1995)		
	Planet of the Apes (1968)	Usual Suspects, The (1995)		
	Bourne Ultimatum, The (2007)	From Dusk Till Dawn (1996)		
111	Austin Powers: International Man of Mystery (1997)	Usual Suspects, The (1995)	1,2,5,3,4,9,15, 8,14,16,17	6,14,17,1,2, 16,5,8,18
	Raiders of the Lost Ark (Indiana Jones and the Raiders of the Lost Ark) (1981)	Star Wars: Episode IV - A New Hope (1977)		
	Shrek (2001)	Pulp Fiction (1994)		
	Ice Age (2002)	Schindler's List (1993)		
	Donnie Darko (2001)	Reservoir Dogs (1992)		
211	Star Wars: Episode V - The Empire Strikes Back (1980)	Usual Suspects, The (1995)	1,2,16,5,8	6,14,17,1,8, 18,2,16,5,1 5
	Austin Powers: International Man of Mystery (1997)	Braveheart (1995)		
	Planet of the Apes (1968)	Star Wars: Episode IV - A New Hope (1977)		
	Cool Hand Luke (1967)	Pulp Fiction (1994)		
	Galaxy Quest (1999)	Forrest Gump (1994)		
311	Austin Powers: International Man of Mystery (1997)	Toy Story (1995)	1,2,5,3,4,9,15, 16,17	2,3,4,5,9,1, 6,17,14,15
	Raiders of the Lost Ark (Indiana Jones and the Raiders of the Lost Ark) (1981)	Heat (1995)		
	Shrek (2001)	Seven (a.k.a. Se7en) (1995)		
	Ice Age (2002)	Usual Suspects, The (1995)		

	Independence Day (a.k.a. ID4) (1996)	Bottle Rocket (1996)		
411	Austin Powers: International Man of Mystery (1997)	Fight Club (1999)	1,2,5,8,16,3,4, 6,9,17,15	1,6,8,17,14, 5,18
	Planet of the Apes (1968)	Shawshank Redemption, The (1994)		
	Ice Age (2002)	Rear Window (1954)		
	Independence Day (a.k.a. ID4) (1996)	Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb (1964)		
	Shrek (2001)	Godfather, The (1972)		

Userid	**Precision**	**Recall**	**F1 Score**
1	0.625	0.5	0.55
111	0.636	0.77	0.7
211	1	0.5	0.66
311	0.88	0.8	0.84
411	0.45	0.71	0.55

Now we have chosen five users by their id and calculated their predicted rating's precision, recall and f1 score. Moreover, we have illustrated the movie predictions made by our recommendation system as well as the grid search SVD-based recommendation system. We have considered the genres of the grid search-based recommendation system as the ground truth and genre of our predicted system as predicted output and based on these we have calculated the f1 score considering our predicted movie genres.

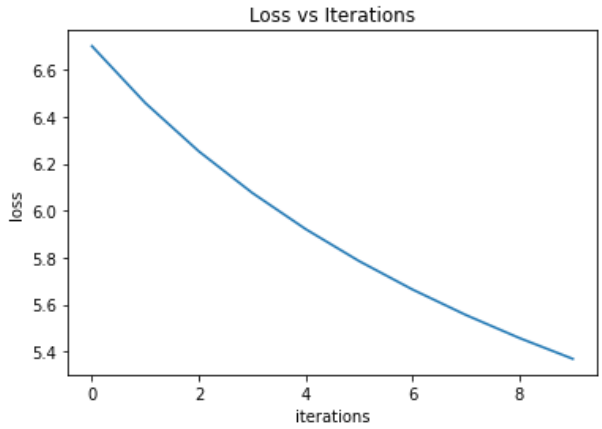

Figure 1 Loss vs Iterations Graph

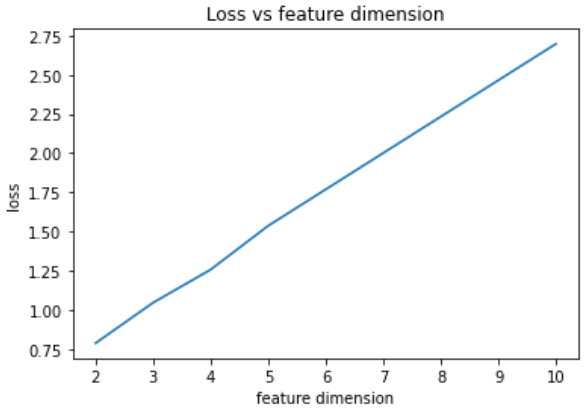

Figure 2 Loss vs Feature Dimension Graph

Observing the loss vs iterations and loss vs feature dimensions graphs from the above figure it's clear that, as we increase the iterations the loss is reduced. On the other hand, as we increase the feature dimensions loss starts to increase. So, we should take a lower feature dimension and a high number of iterations to

get a good result. But if the iteration is very high then again, the loss starts to rise (from our observation for the large iterations).

1.5 Conclusion

By looking at the F1 score of our system and grid-search and SVD based recommendation system we say for 60% of the cases the f1 score was quite high and it was greater than 0.66. Moreover, for the rest of the 40% of cases, the score was 0.55. Likewise, for all of the cases, we got a score above the 50 percentiles. Comparatively, we believe our recommendation system is good enough to suggest a good taste of movies to each user based on their preference.

Annexure

Our implemented recommendation system

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt


!unzip '/content/ml-latest-small.zip'


movie_df = pd.read_csv('/content/ml-latest-small/movies.csv')
df = pd.read_csv('/content/ml-latest-small/ratings.csv')


## ===== get the pivot table of user and movie rating


df[['movieId','userId','rating']] =
df[['movieId','userId','rating']].apply(pd.to_numeric)
movie_user_pivot=df.pivot(index = 'userId', columns ='movieId', values =
'rating').fillna(0)


R = movie_user_pivot.to_numpy()
## function for matrix matrix_factorization
def matrix_factorization(R, P, Q, K, steps=5000, alpha=0.0002, beta=0.02):
    losses=[]
    steps_list=[]
    '''
    R: rating matrix
    P: |U| * K (User features matrix)
    Q: |D| * K (Item features matrix)
    K: latent features
    steps: iterations
    alpha: learning rate
    beta: regularization parameter'''
    Q = Q.T
```

```
    for step in range(steps):
        for i in range(len(R)):
            for j in range(len(R[i])):
                if R[i][j] > 0:
                    # calculate error
                    eij = R[i][j] - np.dot(P[i,:],Q[:,j])

                    for k in range(K):
                        # calculate gradient with a and beta parameter
                        P[i][k] = P[i][k] + alpha * (2 * eij * Q[k][j] - beta *
P[i][k])
                        Q[k][j] = Q[k][j] + alpha * (2 * eij * P[i][k] - beta *
Q[k][j])

        eR = np.dot(P,Q)
        rmse=np.sqrt(np.mean((np.subtract(R, eR))**2))
        losses.append(rmse)
        steps_list.append(step)
        print("step-{}: loss:{}".format(step,rmse))
        e = 0
        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 + (beta/2) * (pow(P[i][k],2) + pow(Q[k][j],2))
        # 0.001: local minimum
        if e < 0.001:
            break
```

```
    return P, Q.T,losses,steps_list

#number of users
N = len(R)
# number of Movies
M = len(R[0])
# Num of Features
K = 2
P = np.random.rand(N,K)
Q = np.random.rand(M,K)

nP, nQ, losses_list,steps_list = matrix_factorization(R, P, Q, K,steps=1)
nR = np.dot(nP, nQ.T)

# function for movie recommendaiton  to search row based on user id
# then we have mapped movie id to movie title and movie genres
# later we choose a few movies based on highest rating

def MovieRecommend(userid,max_movies=10):
  rating_list=nR[userid]
  item_list_dict={}
  for item in rating_list:
    res=np.where(rating_list == item)
    item_list_dict[item]=res[0][0]
  rating_list.sort()
  idx=len(rating_list)-1

  choosen_movies=[]
  while(len(choosen_movies)<max_movies):
    if(item_list_dict[rating_list[idx]] not in choosen_movies):
      choosen_movies.append(item_list_dict[rating_list[idx]])
    idx-=1
  col_movieIds=movie_user_pivot.columns.tolist()
```

```
  for mId in choosen_movies:
    movie_info=movie_df.loc[movie_df['movieId'] == col_movieIds[mId]]

print("{}:{}||{}".format(movie_info.iloc[0]['title'],movie_info.iloc[0]['genres
'],rating_list[idx]))


userlist=[1,111,211,311,411]
for uid in userlist:
    MovieRecommend(uid,5)
    print("====================")


## Plotting loss vs iterations
plt.plot(steps_list,losses_list)
plt.xlabel('iterations')
plt.ylabel('loss')
plt.title('Loss vs Iterations')
plt.show()


## Plotting loss vs feature no
loss_list=[]
k_list=[2,3,4,5,10]
for K in k_list:
  N = len(R)
  M = len(R[0])
  P = np.random.rand(N,K)
  Q = np.random.rand(M,K)
  a,b,loss,ep=matrix_factorization(R, P, Q, K=K,steps=1)
  loss_list.append(loss[len(loss)-1])
  print("one loop completed")


plt.plot(k_list,loss_list)
plt.xlabel('feature dimension')
plt.ylabel('loss')
plt.title('Loss vs feature dimension')
```

```
plt.show()
```

GridSearchCV & SVD-based Movie Recommendation System

```
import numpy as np
import pandas as pd
!unzip '/content/ml-latest-small.zip'

movie_df = pd.read_csv('/content/ml-latest-small/movies.csv')
ratings_df = pd.read_csv('/content/ml-latest-small/ratings.csv')
tags_df = pd.read_csv('/content/ml-latest-small/tags.csv')

print(ratings_df.head())

!pip install surprise

import pandas as pd
from surprise import Dataset
from surprise import Reader
ratings_df=ratings_df[['userId','movieId','rating']]
reader = Reader(rating_scale=(1, 5))
data = Dataset.load_from_df(ratings_df, reader)

from surprise import SVD
from surprise import Dataset
from surprise.model_selection import GridSearchCV

param_grid = {
    "n_epochs": [5, 10],
    "lr_all": [0.002, 0.005],
```

```
    "reg_all": [0.4, 0.6]
}
gs = GridSearchCV(SVD, param_grid, measures=["rmse", "mae"], cv=3,refit=True)
gs.fit(data)



print(gs.best_score)
print(gs.best_params["rmse"])

unique_movies=list(ratings_df['movieId'].unique())

def MovieIdToMovie(mId):
  temp_df=movie_df.where(movie_df['movieId'] == mId)
  temp_df=temp_df.dropna()
  return list(temp_df['title'])[0]

def MovieIdToMovieGenre(mId):
  temp_df=movie_df.where(movie_df['movieId'] == mId)
  temp_df=temp_df.dropna()
  return list(temp_df['genres'])[0]
def
MovieRecommendationSystem(userId,recommend_threshold=4,max_recommendations=10):
  recommended_movies=[]
  recommended_movgenres=[]
  for movie in unique_movies:
    if(gs.predict(userId,movie).est)>=recommend_threshold:
      if len(recommended_movies)==max_recommendations:
        return recommended_movies,recommended_movgenres
      m=MovieIdToMovie(movie)
      g=MovieIdToMovieGenre(movie)
      recommended_movgenres.append(g)
      recommended_movies.append(m)
```

```
  return recommended_movies,recommended_movgenres


def ShowRecommendedMovieForUser(userid,rating,max_recommendations):
  print("Recommended movie for user {}".format(userid))
  print("-------   Top   {}   movie   recommended   for   the   user   -----
".format(max_recommendations))
  rec_movies,recommended_movgenres                                          =
MovieRecommendationSystem(userid,rating,max_recommendations)
  for i in range(len(rec_movies)):
    print("{} : {}".format(rec_movies[i],recommended_movgenres[i]))
  print("=====  End  ======")


user_ids = [1,111,211]
for user_id in user_ids:
  ShowRecommendedMovieForUser(user_id,rating=4,max_recommendations=5)


ShowRecommendedMovieForUser(311,rating=3,max_recommendations=5)
ShowRecommendedMovieForUser(411,rating=3.9,max_recommendations=5)
```