Problem Set 4

Recommendation System Analysis and Implementation

Jiashu Chen

03/08/2023

Content

Part I: DataSet & Research Question

Question 1: DataSet

The dataset I chose is the MovieLens 100k movie rating dataset. It is one of the most popular and famous dataset used for recommendation system. There are many other rating datasets online. I chose this one because my main goal of this project to analyze and compare different recommendation algorithms. Lots of research on recommendation system have been successfully conducted using this dataset, which means that using this dataset will less likely to introduce data bias into my analysis and models.

The dataset consists of 100,000 ratings (1-5 scale) from 943 users on 1682 movies. Each user rated at least 20 movies. The data is collected by the GroupLens Research Project at the University of Minnesota during seven month period from 1997-09 to 1998-04.

The data is individuals'ratings on movies. Features included in the dataset are user_id, item_id, rating, timestamp, movie title, genre, etc. The data set also includes data on users' demographic information. But this part of data is not used in this project.

Question 2: Research Questions

The research question is to analyze and compare performance of different recommendation algorithms in predicting user's rating on certain movies. The rating data is split into training and testing. The training data is used to develop recommendation models and testing data is used to calculate the RMSE of predicted rating and actual rating.

Part II: Data Import

In [17]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import timeit
from scipy.linalg import sqrtm
```

In [3]:
```python
col_names = ['user_id', "item_id", "rating", "timestamp"]
data = pd.read_csv("/Users/jiashu/Desktop/574Final/ml-100k/u.data", sep = '\t', heade
```

In [4]:
```python
d = 'movie id | movie title | release date | video release date | IMDb URL | unknown
col_names2 = d.split(' | ')
items = pd.read_csv("/Users/jiashu/Desktop/574Final/ml-100k/u.item", sep = '|', heade
movies = items[["movie id", "movie title"]]
movies = movies.rename(columns = {"movie id" : "item_id", "movie title" : "movie_titl
df = pd.merge(data, movies, on = 'item_id')
```

In [5]:
```python
df.head()
```

Out[5]:

	user_id	item_id	rating	timestamp	movie_title
0	196	242	3	881250949	Kolya (1996)
1	63	242	3	875747190	Kolya (1996)
2	226	242	5	883888671	Kolya (1996)
3	154	242	3	879138235	Kolya (1996)
4	306	242	5	876503793	Kolya (1996)

In [6]:
```python
n_users = df.user_id.nunique()
n_items = df.item_id.nunique()

print('Num of Users: '+ str(n_users))
print('Num of Movies: '+str(n_items))
```

```
Num of Users: 943
Num of Movies: 1682
```

In [8]:
```python
# Functionality: creating mapping between two columns of the dataframe
# Input:
    # data: rating records
    # col1_name: the name of one column
    # col2_name: the name of the other column

def mapping(data, col1_name, col2_name):
    unique = data[[col1_name, col2_name]].drop_duplicates()
    colmap = dict(zip(unique[col1_name], unique[col2_name]))
    return colmap
```

In [9]:
```python
# map movie_title and movie_id
movie_map = mapping(df, "item_id", "movie_title")
```

PART III: Training Testing Split

To evaluate the Recommendation System model, the entire dataset split into the training data and testing data. Different from evaluation of some other supervised models, there is no split between label and predictors. This is because the RS model is evaluated based on differences between predicted movie rating and actual movie rating. The rating is predictor as well as the label.

The test_size is set at 0.25, which means that 75% of total ratings is in the training set and 25% of total ratings is in the testing set.

In [10]:
```
from sklearn.model_selection import train_test_split
train_data, test_data = train_test_split(df, test_size=0.25)
```

PART IV: Model Development

Question 3: Algorithm Choice

Collabortive filtering is a method used in many recommendation systems. It utilizes similarities among users and items to provide recommendations. Collaborative filtering can be divided into model-based CF and memory-based CF. I think collaborative filtering is a good way to predict user preferences since the model does not rely in hand-engineering features like content-based models. In this part, I developed both the model-based CF model using SVD and the memory-based CF model using KNN.

Question 4, 5, 7 Model Development, Hyperparameter Tuning & Visualization

- 4.1 Model-Based Collaborative Filtering
  - 4.1.1 Sample Model with K = 10
  - 4.1.2 Hyperparameter Tuning on No.Latent Factors
- 4.2 Momery-Based Collaborative Filtering
  - 4.2.1 Sample Model with K = 20
  - 4.2.2 Hyperparameter Tuning on No.Neighbors

4.1 Model-Based Collaborative Filtering

Model-Based Collaborative Filtering is based on matrix factorization. In this project, Singular Value Decomposition(SVD) techniques are used for matrix factorization

with SVD, the original data matrix is decomposed into three matrices.

- U: represents relationships between users and latent factors.
- S: represents strength of each latent factor.
- V: represents relationships between items and latent factors.

4.1.1 Sample Model with K = 10

In the following model, k is set to be 10, which represents the number of latent factors used in the model.

The current data is a collection of rating record. To perform collaborative filtering, a matrix is needed. The columns of the matrix are movies and the rows of the matrix are users. (i, j) value indicates the rating from the ith user to the jth movie. Therefore, the matrix should be in the shape (numOfUsers, numOfMovies).

In [11]:
```
# creating utility matrix
train_data_matrix = np.asarray([[np.nan for j in range(n_items)] for i in range(n_use
for row in train_data.itertuples():
    train_data_matrix[row[1]-1, row[2]-1] = row[3]
```

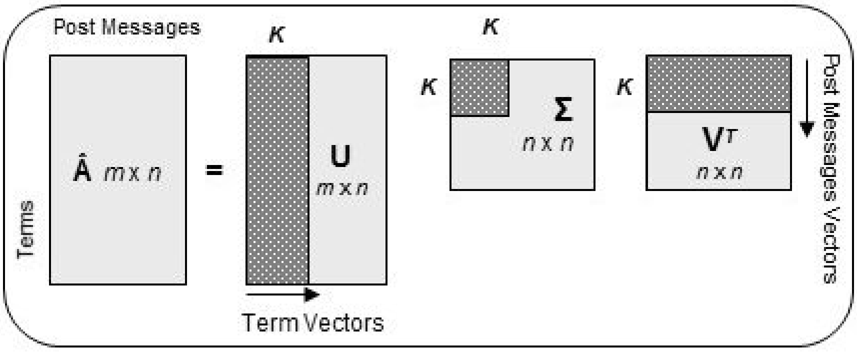
In [12]:
```
# Function: fill in matrix NA values with item mean rating, and normalize each column
# Input: user-item matrix where NA represents no rating record
# Output: normalized matrix, user mean ratings, item mean ratings
def matrixTransform(m):
    mask = np.isnan(m) # mask used to label NA cells
    masked_arr = np.ma.masked_array(m, mask)
    item_means = np.mean(masked_arr, axis=0) # meaning rating for items based on exis
    user_means = np.mean(masked_arr, axis=1) # meaning rating given by users based on
```

```
    filled_matrix = masked_arr.filled(item_means) # fill NA values with item mean rat
    filled_matrix = filled_matrix - item_means.data[np.newaxis,:] # normalize item ra
    util_matrix = filled_matrix/np.sqrt(len(m[0]) -1)
    return util_matrix, user_means, item_means
```

In [13]:
```
train_data_matrix_normed, user_mean, item_mean = matrixTransform(train_data_matrix)
```

In [14]:
```
train_data_matrix_normed
```

Out[14]:
```
array([[ 0.02716518, -0.00371661,  0.        , ...,  0.        ,
         0.        ,  0.        ],
       [ 0.00277494,  0.        ,  0.        , ...,  0.        ,
         0.        ,  0.        ],
       [ 0.        ,  0.        ,  0.        , ...,  0.        ,
         0.        ,  0.        ],
       ...,
       [ 0.        ,  0.        ,  0.        , ...,  0.        ,
         0.        ,  0.        ],
       [ 0.        ,  0.        ,  0.        , ...,  0.        ,
         0.        ,  0.        ],
       [ 0.        ,  0.        ,  0.        , ...,  0.        ,
         0.        ,  0.        ]])
```



In [15]:
```
# Function: perform SVD on matrix, extract k latent factors
# Input: utilMat: normalized user-item matrix,
#        k: the number of latent factors used
#        item_mean: item mean rating
# Output: truncated SVD
def svd(utilMat, k, item_mean):
    U, s, V=np.linalg.svd(utilMat, full_matrices=False)
    s=np.diag(s)
    s=s[0:k,0:k]
    U=U[:,0:k]
    V=V[0:k,:]
    s_root=sqrtm(s)
    Usk=np.dot(U,s_root)
    skV=np.dot(s_root,V)
    UsV = np.dot(Usk, skV)
    # add item mean rating subtracted in previous normalization
    svdout = UsV + item_mean.data[np.newaxis:, ]
    return svdout
```

In [18]:
```
svdout = svd(train_data_matrix_normed, 10, item_mean)
```

In [19]:
```
svdout
```

```
Out[19]: array([[3.88401628, 3.1525543 , 3.18015157, ..., 2.        , 3.        ,
          3.        ],
         [3.88665227, 3.1511246 , 3.17843502, ..., 2.        , 3.        ,
          3.        ],
         [3.88414346, 3.15238381, 3.17783598, ..., 2.        , 3.        ,
          3.        ],
         ...,
         [3.88815029, 3.15275653, 3.17998982, ..., 2.        , 3.        ,
          3.        ],
         [3.8903476 , 3.15344144, 3.17863163, ..., 2.        , 3.        ,
          3.        ],
         [3.88321268, 3.15212644, 3.18075344, ..., 2.        , 3.        ,
          3.        ]])
```

The above SVD output matrix is predicted rating from each user on each item.

To evaluate the model, the performance is calculated by comparing the predicted rating and the test_matrix rating. The metric used for measure performance is the Root Mean Squared Error

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} (Predicted_i - Actual_i)^2}{N}}$$

(RMSE)

In [22]:
```python
import math

# Function: calculate rmse
# Input: true: ground truth, actual movie rating
#        pred: predicted movie rating
# Output: rmse
def rmse(true, pred):
    x = true - pred
    return math.sqrt(sum([xi*xi for xi in x])/len(x))
```

In [23]:
```python
pred = []
for _,row in test_data.iterrows():
    user = row['user_id']
    item = row['item_id']
    # userid and itemid in test_data starts from 1
    # need to subtract 1 from it t get the correct prediction
    pred_rating = svdout[user-1][item-1]
    pred.append(pred_rating)
```

In [24]:
```python
model1_rmse = rmse(test_data['rating'], pred)
model1_rmse
```

```
Out[24]: 1.0267009136257297
```

In [25]:
```python
print("RMSE of the Model-Based Collaborative Filtering Model (k = 10): " + str(model1
```

```
RMSE of the Model-Based Collaborative Filtering Model (k = 10): 1.0267009136257297
```

4.1.2 Hyperparameter Tuning on Latent Factors

The above model uses 10 latent factors. We could adjust the k value for potential improvement.

In [26]:
```python
# Function: perform SVD prediction using k latent factors
# Input: utilMat: normalized user-item matrix,
#        k: the number of latent factors used
#        item_mean: item mean rating
# Output: SVD prediction, rmse
```

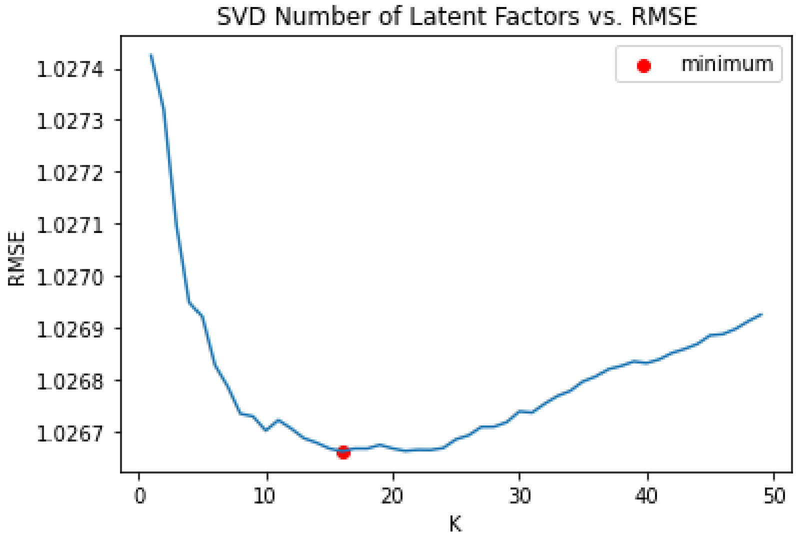
```python
def svd_k(k):
    svdout = svd(train_data_matrix_normed, k, item_mean)
    pred = []
    for _,row in test_data.iterrows():
        user = row['user_id']
        item = row['item_id']
        pred_rating = svdout[user-1][item-1]
        pred.append(pred_rating)
    rmse_k = rmse(test_data['rating'], pred)
    return pred, rmse_k
```

In [27]:
```python
rmse_res = pd.DataFrame({'k':[], 'rmse':[]})
for k in range(1, 50):
    pred, k_rmse = svd_k(k)
    rmse_res = rmse_res.append({'k': k, 'rmse': k_rmse}, ignore_index=True)
```

In [28]:
```python
min_x = np.argmin(rmse_res.rmse) + 1
min_y = np.min(rmse_res.rmse)

plt.plot(rmse_res['k'], rmse_res['rmse'])
plt.scatter(min_x, min_y,c='r', label='minimum')
plt.legend()
plt.xlabel('K')
plt.ylabel("RMSE")
plt.title("SVD Number of Latent Factors vs. RMSE")
```

Out[28]:
```
Text(0.5, 1.0, 'SVD Number of Latent Factors vs. RMSE')
```



In [29]:
```python
min_x, min_y
```

Out[29]:
```
(16, 1.0266608917620779)
```

In [35]:
```python
model2_start = timeit.default_timer()
model2_pred, model2_rmse  = svd_k(min_x)
model2_end = timeit.default_timer()
model2_time = model2_end - model2_start
print("RMSE of the Model-Based Collaborative Filtering Model (k = " + str(min_x) + ")
print("Time Cost of the Model-Based Collaborative Filtering Model (k = " + str(min_x)
```

```
RMSE of the Model-Based Collaborative Filtering Model (k = 16):  1.0266608917620779
Time Cost of the Model-Based Collaborative Filtering Model (k = 16): 1.435460458000307
```

4.2 Memory-Based Collaborative Filtering

Memory-Based Collaborative Filtering utilizes similarity among users and items to provide recommendation. It could be further divided into Item-Based Collaborative Filtering and User-Based Collaborative Filtering.

- Item-Based Collaborative Filtering: movies that are similar to users liked.
- User-Based Collaborative Filtering: movies that liked by similar users

In this part, a weighted user-based model is developed. The KNN algorithm is used to choose neighbors for a specific user. Afterwards, each neighbor is given a weight based on distance. The predicted rating of the user on one movie is calculated by the following formula

$$P_{aj} = \bar{r}_a + \frac{\sum_{i \in NS_a} sim(a,i) * (r_{ij} - \bar{r}_i)}{\sum_{i \in NS_a} |sim(a,i)|}$$

In [37]:
```
from scipy.sparse import csr_matrix
from sklearn.neighbors import NearestNeighbors
```

In [38]:
```
train_data_matrix_zero = pd.DataFrame(train_data_matrix).fillna(0)
```

In [39]:
```
train_data_matrix_sparse = csr_matrix(train_data_matrix_zero)
train_data_matrix_sparse
```

Out[39]:
```
<943x1682 sparse matrix of type '<class 'numpy.float64'>'
	with 75000 stored elements in Compressed Sparse Row format>
```

In [40]:
```
knn_model = NearestNeighbors(metric='cosine', algorithm='brute')
knn_model.fit(train_data_matrix_sparse)
```

Out[40]:
```
NearestNeighbors(algorithm='brute', metric='cosine')
```

In [41]:
```
# Test on a random user
user = 21
n = 10
knn_input = np.asarray([train_data_matrix_zero .values[user-1]])
```

In [42]:
```
# Find n neighbors
distances, indices = knn_model.kneighbors(knn_input, n_neighbors=n+1)
similar_user_list = indices.flatten()[1:]
distance_list = distances.flatten()[1:]
```

In [43]:
```
indices
```

Out[43]:
```
array([[ 20, 813, 603, 366, 365, 371, 254, 421, 801, 387, 117]])
```

In [44]:
```
distances
```

Out[44]:
```
array([[1.11022302e-16, 5.86647226e-01, 6.27207334e-01, 6.44881018e-01,
        6.48348048e-01, 6.49979503e-01, 6.70570365e-01, 6.93740946e-01,
        6.96326705e-01, 6.97565283e-01, 7.12406006e-01]])
```

In [45]:
```
print("Top",n,"users similar to the User",user, ":")
print(" ")
print("    User          Distance")
for i in range(1,len(distances[0])):
    print(i," ", indices[0][i],"     ",distances[0][i])
```

```
Top 10 users similar to the User 21 :

    User         Distance
1   813       0.5866472261032933
2   603       0.6272073343901318
3   366       0.6448810175435709
4   365       0.6483480483664756
5   371       0.6499795026834203
6   254       0.6705703650459579
7   421       0.6937409463364497
8   801       0.6963267051661477
9   387       0.6975652828569034
10   117       0.712406006149418
```

To predict one user's rating on a specific movie, we need to give different weights to different neighbors based on similarity.

In [85]:
```
# Give weights to neighbors based on distances
neighbor_weight_list = distance_list/np.sum(distance_list)
neighbor_weight_list
```

Out[85]:
```
array([0.08851482, 0.09463463, 0.09730128, 0.0978244 , 0.09807055,
       0.10117735, 0.10467339, 0.10506354, 0.10525042, 0.10748962])
```

We need to extract ratings given by neighbors. For movies that neighbors did not see, replace the zero with user mean rating.

In [86]:
```
mask = np.isnan(train_data_matrix)
masked_arr = np.ma.masked_array(train_data_matrix, mask)
train_data_matrix_mean = masked_arr.filled(user_mean.data[:,np.newaxis])
```

In [87]:
```
train_data_matrix_normed = pd.DataFrame(train_data_matrix_mean - user_mean.data[:,np.
```

In [88]:
```
train_data_matrix_normed.head()
```

Out[88]:

	0	1	2	3	4	5	6	7	8	9
0	1.376190	-0.623810	0.0	-0.62381	-0.62381	1.37619	0.37619	-2.62381	1.37619	-0.62381
1	0.295455	0.000000	0.0	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	-1.70454
2	0.000000	0.000000	0.0	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
3	0.000000	0.000000	0.0	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
4	1.098485	0.098485	0.0	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000

5 rows × 1682 columns

In [89]:
```
# Extract ratings given by neighbors
neighbor_rating = train_data_matrix_normed.iloc[similar_user_list]
neighbor_rating
```

Out[89]:

	0	1	2	3	4	5	6	7	8	9	...	1672	1673	16
813	0.000000	0.0	0.0	0.0	0.000000	0.0	1.040000	0.0	0.000000	0.0	...	0.0	0.0	
603	0.000000	0.0	0.0	0.0	-1.291667	0.0	0.000000	0.0	0.000000	0.0	...	0.0	0.0	
366	0.000000	0.0	0.0	0.0	-0.148936	0.0	0.851064	0.0	0.000000	0.0	...	0.0	0.0	
365	0.000000	0.0	0.0	0.0	0.000000	0.0	-2.416667	0.0	0.000000	0.0	...	0.0	0.0	
371	0.000000	0.0	0.0	0.0	-0.339623	0.0	0.000000	0.0	0.000000	0.0	...	0.0	0.0	
254	0.000000	0.0	0.0	0.0	0.000000	0.0	0.000000	0.0	0.000000	0.0	...	0.0	0.0	
421	0.000000	0.0	0.0	0.0	0.000000	0.0	0.000000	0.0	0.000000	0.0	...	0.0	0.0	
801	0.000000	0.0	0.0	0.0	0.000000	0.0	1.379310	0.0	0.000000	0.0	...	0.0	0.0	
387	0.789474	0.0	0.0	0.0	-0.210526	0.0	0.000000	0.0	-1.210526	0.0	...	0.0	0.0	
117	0.000000	0.0	0.0	0.0	0.000000	0.0	0.314815	0.0	0.000000	0.0	...	0.0	0.0	

10 rows × 1682 columns

In [90]:

```
# Broadcast the neighbor weight to a matrix
weight_matrix = neighbor_weight_list[:,np.newaxis] + np.zeros(n_items)
weight_matrix.shape
```

Out[90]: (10, 1682)

In [91]:

```
# Compute neighbor rating with weight
neighor_weight_rating = weight_matrix*neighbor_rating
neighor_weight_rating
```

Out[91]:

	0	1	2	3	4	5	6	7	8	9	...	1672	1673	16
813	0.000000	0.0	0.0	0.0	0.000000	0.0	0.092055	0.0	0.000000	0.0	...	0.0	0.0	
603	0.000000	0.0	0.0	0.0	-0.122236	0.0	0.000000	0.0	0.000000	0.0	...	0.0	0.0	
366	0.000000	0.0	0.0	0.0	-0.014492	0.0	0.082810	0.0	0.000000	0.0	...	0.0	0.0	
365	0.000000	0.0	0.0	0.0	0.000000	0.0	-0.236409	0.0	0.000000	0.0	...	0.0	0.0	
371	0.000000	0.0	0.0	0.0	-0.033307	0.0	0.000000	0.0	0.000000	0.0	...	0.0	0.0	
254	0.000000	0.0	0.0	0.0	0.000000	0.0	0.000000	0.0	0.000000	0.0	...	0.0	0.0	
421	0.000000	0.0	0.0	0.0	0.000000	0.0	0.000000	0.0	0.000000	0.0	...	0.0	0.0	
801	0.000000	0.0	0.0	0.0	0.000000	0.0	0.144915	0.0	0.000000	0.0	...	0.0	0.0	
387	0.083092	0.0	0.0	0.0	-0.022158	0.0	0.000000	0.0	-0.127408	0.0	...	0.0	0.0	
117	0.000000	0.0	0.0	0.0	0.000000	0.0	0.033839	0.0	0.000000	0.0	...	0.0	0.0	

10 rows × 1682 columns

In [92]:

```
# Sum up weighted neighbor rating and add to user mean
user_pred_rating = user_mean[user-1] + neighor_weight_rating.sum(axis =0)
user_pred_rating
```

Out[92]:
```
0       2.766916
1       2.683824
2       2.683824
3       2.683824
4       2.491630
          ...
1677    2.683824
1678    2.683824
1679    2.683824
1680    2.683824
1681    2.683824
Length: 1682, dtype: float64
```

The user_pred_rating gives us predicted ratings on all movies by the user. To test the model, we could compute predicted ratings for all users and then use the test_data to compute rmse

In [96]:
```
# summarize previous procedures into functions

# Function: extract n neighbors close to the user
# Input: user: user_id that we want to predict
#        n: the number of neighbors
# Output: list of n neighbors and their weights
def get_neighbors(user, n = 10):
    knn_input = np.asarray([train_data_matrix_zero .values[user-1]])
    distances, indices = knn_model.kneighbors(knn_input, n_neighbors=n+1)
    similar_user_list = indices.flatten()[1:]
    distance_list = distances.flatten()[1:]
    neighbor_weight_list = distance_list/np.sum(distance_list)
    return similar_user_list, neighbor_weight_list
```

In [97]:
```
# Function: predict one user's rating on all movies based on n neighbors
# Input: user: user_id that we want to predict
#        n: the number of neighbors
# Output: prediction of the user's rating on all movies
def predict_user_rating(user, n = 10):
    similar_user_list, neighbor_weight_list = get_neighbors(user, n)
    neighbor_rating = train_data_matrix_normed.iloc[similar_user_list]
    weight_matrix = neighbor_weight_list[:,np.newaxis] + np.zeros(n_items)
    neighor_weight_rating = weight_matrix*neighbor_rating
    user_pred_rating = user_mean[user-1] + neighor_weight_rating.sum(axis =0)
    return user_pred_rating
```

In [98]:
```
# create a new matrix contain all users' predicted ratings
row_list = []
for user in range(n_users):
    itemPred = predict_user_rating(user)
    row_list.append(itemPred)
knn_pred = pd.DataFrame(row_list)
```

In [99]:
```
knn_pred
```

Out[99]:

	0	1	2	3	4	5	6	7	
0	3.752920	3.260528	3.344408	3.346025	3.395849	3.374046	3.773359	3.123673	3.586
1	4.095067	3.181716	3.336317	3.931057	3.706374	3.623810	3.901201	4.162693	4.040
2	3.401817	3.704545	3.580225	3.737240	3.634486	3.704545	3.683998	3.839994	4.094
3	2.820513	2.820513	2.820513	2.820513	2.820513	2.820513	2.820513	2.820513	2.820
4	4.500000	4.500000	4.500000	4.500000	4.500000	4.500000	4.558325	4.500000	4.454
...	...	...	...	...	...	...	...	...	
938	3.602883	3.229885	3.022630	3.229885	3.229885	3.229885	3.574190	3.229885	2.771
939	4.137863	4.297297	4.297297	4.297297	4.297297	4.297297	4.453451	4.297297	4.232
940	3.357605	3.435305	3.456539	3.748873	3.389484	3.488372	3.666129	4.046399	3.643
941	4.460363	4.125000	4.125000	4.125000	4.125000	4.125000	3.984682	4.125000	4.314
942	4.260485	4.254545	4.254545	4.326320	4.254545	4.301871	4.107370	4.483638	4.230

943 rows × 1682 columns

In [100…
```
knn_pred_list = []
for _,row in test_data.iterrows():
    user = row['user_id'] - 1
    item = row['item_id'] - 1
    pred_rating = knn_pred.iloc[user][item]
    knn_pred_list.append(pred_rating)
```

In [101…
```
model3_rmse = rmse(test_data['rating'], knn_pred_list)
print("RMSE of the Memory-Based Collaborative Filtering Model (n = 10): " + str(model
```

```
RMSE of the Memory-Based Collaborative Filtering Model (n = 10): 1.2023884491108074
```

4.2.2 Hyperparameter Tuning on k Neighbors

The above model with k = 10 gives us a RMSE of 1.035. We could adjust the k value for potential improvement

In [102…
```
# Function: create a matrix that contains all users' top k neighbors
# Input: k: the number of neighbors
# Output: a matrix that contains all users' top k neighbors
def knn_matrix(k):
    row_list = []
    for user in range(n_users):
        itemPred = predict_user_rating(user, k)
        row_list.append(itemPred)
    knn_mat = pd.DataFrame(row_list)
    return knn_mat
```

In [103…
```
# Function: evaluate knn model performance with k neighbors
# Input: k: the number of neighbors
# Output: prediction and rmse using k neighbors

def knn_k(k):
    knn_mat = knn_matrix(k)
    pred = []
    for _,row in test_data.iterrows():
        user = row['user_id']
        item = row['item_id']
        pred_rating = knn_mat.iloc[user-1][item-1]
        pred.append(pred_rating)
```

```
    rmse_k = rmse(test_data['rating'], pred)
    return pred, rmse_k
```

In [104…
```
rmse_res = pd.DataFrame({'k':[], 'rmse':[]})
for k in [10, 30, 50, 100, 200, 300, 500, 800, n_users-1
         ]:
    pred, k_rmse = knn_k(k)
    rmse_res = rmse_res.append({'k': k, 'rmse': k_rmse}, ignore_index=True)
```

In [105…
```
rmse_res
```

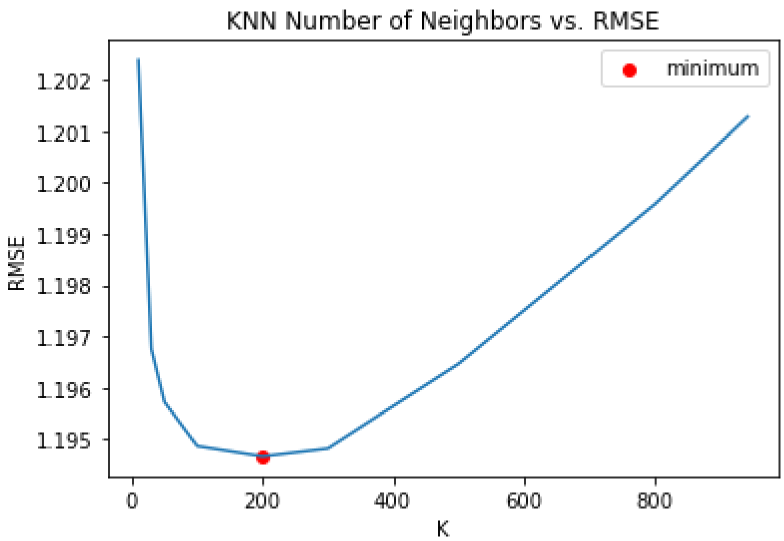
Out[105]:

	k	rmse
0	10.0	1.202388
1	30.0	1.196754
2	50.0	1.195724
3	100.0	1.194867
4	200.0	1.194665
5	300.0	1.194820
6	500.0	1.196462
7	800.0	1.199583
8	942.0	1.201282

In [119…
```
knn_min_x = int(rmse_res.loc[np.argmin(rmse_res.rmse)]['k'])
knn_min_y = np.min(rmse_res.rmse)

plt.plot(rmse_res['k'], rmse_res['rmse'])
plt.scatter(knn_min_x, knn_min_y,c='r', label='minimum')
plt.legend()
plt.xlabel('K')
plt.ylabel("RMSE")
plt.title("KNN Number of Neighbors vs. RMSE")
```

Out[119]: Text(0.5, 1.0, 'KNN Number of Neighbors vs. RMSE')



In [115…
```
knn_min_x, knn_min_y
```

Out[115]: (200, 1.194665264499012)

In [116…
```
# k = 200
model4_start = timeit.default_timer()
model4_pred, model4_rmse = knn_k(knn_min_x)
```

```
model4_end = timeit.default_timer()
model4_time = model4_end - model4_start
print("RMSE of the Memory-Based Collaborative Filtering Model (k = " + str(knn_min_x)
print("Time Cost of the Memory-Based Collaborative Filtering Model (k = " + str(knn_m
```

```
RMSE of the Model-Based Collaborative Filtering Model (k = 200) : 1.194665264499012
Time Cost of the Model-Based Collaborative Filtering Model (k = 200) : 6.1274046660000
75
```

PART V: Model Comparison

Question 6: Model Accuracy

In [117…

```
metric = [['Model 1', 'Model-Based', 'K = 10', model1_rmse, "-"],
          ['Model 2', 'Model-Based', ' K =' + str(min_x), model2_rmse, model2_time],
          ['Model 3', 'Memory-Based', 'K = 20', model3_rmse, '-'],
          ['Model 4', 'Memory-Based', 'K =' + str(knn_min_x), model4_rmse, model4_time
model_compare = pd.DataFrame(metric, columns = ['Model', "Algorithm", "Param", "RMSE"
```

In [118…

```
model_compare
```

Out[118]:

	Model	Algorithm	Param	RMSE	Time
0	Model 1	Model-Based	K = 10	1.026701	-
1	Model 2	Model-Based	K =16	1.026661	1.43546
2	Model 3	Memory-Based	K = 20	1.202388	-
3	Model 4	Memory-Based	K =200	1.194665	6.127405

Comparison Based on RMSE

Following conclusions could be generated based on RMSE metric

- The best model is Model-based CF model with tunned k.
- Even without tuning k, the Model-based models generally perform better than Memory-based models. From previous hyperparameter tunning, we can see that RMSE of Model-based models almost never exceed 1.028 for most k.

Comparison Based on Time Cost

- The time cost difference among different models is quite significant. The time cost of Memory-based model is about 6 times the time cost of Model-based model. However, it might be related to the fact that it is more computational expensive to test the Memory-based model in this case.
- According to some research paper on collabrative filtering (Aditya, P. H., Budi, I.& Munajat, Q., 2016), the computation time varies a lot between the Memory-based model and the Model-based model. The Model-based approach is 10 times faster than the Memory-based approach. (Reference: https://qoribmunajat.github.io/files/comparative-analysis-memory-based-model-based-recommendation-systems.pdf)

Comparison Based on Explanability

- The Memory-based model is easier to explain and interpret since it is based on similaity among items and users.
- The Model-based model is difficult to interpret since SVD performs dimension reduction. Meanings of latent factors are not clear.

Comparison Based on Scalability

- The Memory-based model is less scalable due to the sparsity problem. As the number of users and the number of items grow, there will be lots of zeros in the matrix (Grover, P. 2017).
- The Model-based model handles the sparsity with dimension reduction well. (Reference: https://towardsdatascience.com/various-implementations-of-collaborative-filtering-100385c6dfe0)

Comparison Based on Perceived Recommendation Relevance

- In the same study mentioned above, the researchers also concluded that in terms of users' perceived relevance of recommendation, Model-based is also better than the Memory-based.

PART VI: Problem of Collaborative Filtering

For collaborative filtering algorithms, they all face the same problem: the cold start problem. For new users and new items, there is no past rating records, it is impossible to calculate similarity.

To solve this problem, I implemented the Content-Based filtering in the next section. The recommendation is based on genre, popularity, rating. Similarity is not needed in this approach since it essentially give all other users the same similarity score.

PART VII: Content-Based Filtering

In [121…

```
genre_list = col_names2[-19:]
genre_list
```

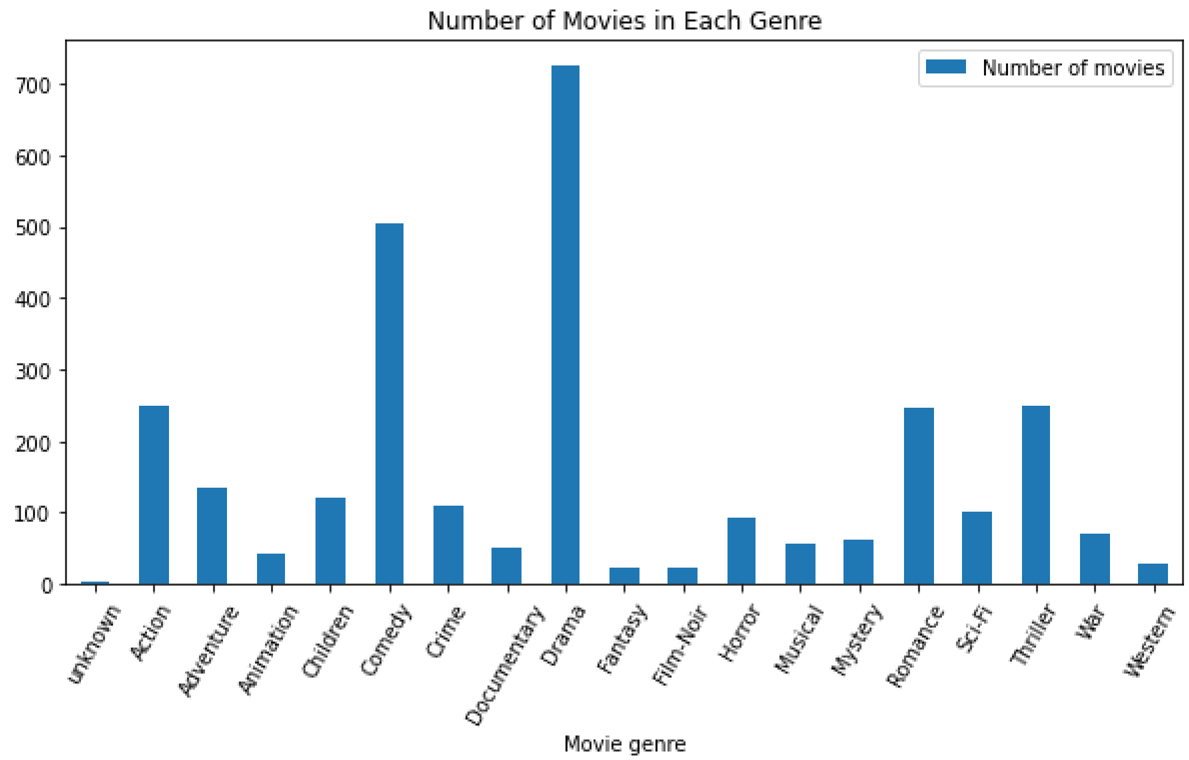
Out[121]:

```
['unknown',
 'Action',
 'Adventure',
 'Animation',
 'Children',
 'Comedy',
 'Crime',
 'Documentary',
 'Drama',
 'Fantasy',
 'Film-Noir',
 'Horror',
 'Musical',
 'Mystery',
 'Romance',
 'Sci-Fi',
 'Thriller',
 'War',
 'Western']
```

In [123…

```
count = []
for i in genre_list:
  genre_based_movies = items[['movie id','movie title',i]]
  genre_based_movies = genre_based_movies[genre_based_movies[i] == 1]
  count.append(len(genre_based_movies))

genre_count = pd.DataFrame({'Movie genre': genre_list, 'Number of movies':count})
ax = genre_count.plot.bar(x='Movie genre', y='Number of movies', rot=60, figsize=(10,
                          title = "Number of Movies in Each Genre")
```



In [124…
```
# merge genre information in the data
merged_df = pd.merge(df, items, how = 'inner', left_on='item_id', right_on='movie id'
merged_df.head()
```

Out[124]:

	user_id	item_id	rating	timestamp	movie_title	movie id	movie title	release date	video release date	
0	196	242	3	881250949	Kolya (1996)	242	Kolya (1996)	24-Jan-1997	NaN	http://us.i exact?
1	63	242	3	875747190	Kolya (1996)	242	Kolya (1996)	24-Jan-1997	NaN	http://us.i exact?
2	226	242	5	883888671	Kolya (1996)	242	Kolya (1996)	24-Jan-1997	NaN	http://us.i exact?
3	154	242	3	879138235	Kolya (1996)	242	Kolya (1996)	24-Jan-1997	NaN	http://us.i exact?
4	306	242	5	876503793	Kolya (1996)	242	Kolya (1996)	24-Jan-1997	NaN	http://us.i exact?

5 rows × 29 columns

In [126…
```
# group movies based on genre
genre_dict = {}
for g in genre_list:
    genre_based_movies = merged_df[['movie id','movie title', 'rating',  g]]
    genre_based_movies = genre_based_movies[genre_based_movies[g] == 1]
    genre_dict[g] = genre_based_movies
```

In [127…
```
# Function: recommending top k movies based on genre and popularity
# Input: k: the number of recommendations
    # gerne: recommendation genre

def recommendations_popularity(genre, k = 10):
        genre_based_movies = genre_dict.get(genre)
```

```
        popular_genre_movies = genre_based_movies.groupby(['movie title']).agg({"rati
        popular_movies_ingenre = popular_genre_movies.to_frame()
        popular_movies_ingenre.reset_index(level=0, inplace=True)
        popular_movies_ingenre.columns = ['movie title', 'Number of Users watched']
        print("Top", k,"popular movies of the genre:", genre)
        print("----- ----- ----- ----- ----- ----- ----- ----- ----- ----- ----- ----
        print(popular_movies_ingenre.sort_values('Number of Users watched', ascending
```

In [128…

```
# Function: recommending top k movies based on genre and rating
# Input: k: the number of recommendations
    # gerne: recommendation genre
def recommendations_rating(genre, k = 10):
        genre_based_movies = genre_dict.get(genre)
        highrate_genre_movies = genre_based_movies.groupby(['movie title']).agg({"rat
        highrate_movies_ingenre = highrate_genre_movies.to_frame()
        highrate_movies_ingenre.reset_index(level=0, inplace=True)
        highrate_movies_ingenre.columns = ['movie title', 'Average Rating']
        print("Top", k,"high rating movies of the genre:", genre)
        print("----- ----- ----- ----- ----- ----- ----- ----- ----- ----- ----- ----
        print(highrate_movies_ingenre.sort_values('Average Rating', ascending=False).
```

In [129…

```
# Function: recommending top k movies based on genre, popularity, and rating
# Input: gerne: recommendation genre

def recommendations_genre(genre):
    print("Chosen Genre: ", genre)
    print("Recommendation by Popularity: ")
    recommendations_popularity(genre)
    print("Recommendation by Rating: ")
    recommendations_rating(genre)
```

In [130…

```
# Test
recommendations_genre("Comedy")
```

```
Chosen Genre:  Comedy
Recommendation by Popularity:
Top 10 popular movies of the genre: Comedy
----- ----- ----- ----- ----- ----- ----- ----- ----- ----- ----- -----
                                    movie title  Number of Users watched
0                              Liar Liar (1997)                      485
1                              Toy Story (1995)                      452
2                     Back to the Future (1985)                      350
3   Willy Wonka and the Chocolate Factory (1971)                      326
4                    Princess Bride, The (1987)                      324
5                           Forrest Gump (1994)                      321
6         Monty Python and the Holy Grail (1974)                      316
7                        Full Monty, The (1997)                      315
8                          Men in Black (1997)                      303
9                          Birdcage, The (1996)                      293
Recommendation by Rating:
Top 10 high rating movies of the genre: Comedy
----- ----- ----- ----- ----- ----- ----- ----- ----- ----- ----- -----
                  movie title  Average Rating
0    Santa with Muscles (1996)        5.000000
1         Close Shave, A (1995)        4.491071
2   Wrong Trousers, The (1993)        4.466102
3    North by Northwest (1959)        4.284916
4       Shall We Dance? (1996)        4.260870
5    As Good As It Gets (1997)        4.196429
6       Cinema Paradiso (1988)        4.173554
7   Princess Bride, The (1987)        4.172840
8   Waiting for Guffman (1996)        4.127660
9         A Chef in Love (1996)        4.125000
```

The Content-Based Filtering comes with several problems

- It is hard to test the recommendation without actual experiment.
- The recommendation is less personalized as it will give the same result for users who choose the same genre.

PART VIII: Implementation

8.1 Existing User Recommdation

Based on previous model comparison and other research studies, I believe the Model-Based Collaborative Filtering would be a better model for predicting preferences of the existing users

In [131…

```
# Functionality: generating user-item matrix with NA filled
# Input:
    # data: rating records
    # n_users: number of users
    # n_items: number of items
    # user_col: column number of user_id, default = 1
    # item_col: column number of item_id, default = 2
    # rating_col: column number of rating_id, default = 3
# Output: utility matrix

def utility_matrix(data, n_users, n_items,
                   user_col = 1, item_col = 2, rating_col = 3):
    data_matrix = np.asarray([[np.nan for j in range(n_items)] for i in range(n_users
    for row in data.itertuples():
        user = row[user_col]-1
        item = row[item_col]-1
        data_matrix[user][item] = row[rating_col]
    return data_matrix
```

In [132…

```
# Functionality: Using svd to generate predition matrix
# Input:
    # util_mat: utility matrix where columns are items, rows are users
    # k: number of latent factors used, default = min_x
# Output: svd prediction

def svd_prediction(util_mat, k = min_x):
    data_matrix_normed, user_mean, item_mean = matrixTransform(util_mat)
    svd_pred = svd(data_matrix_normed, k,item_mean)
    return pd.DataFrame(svd_pred)
```

In [133…

```
# Functionality: generate recommended movies based on svd prediction
# Input:
    # pred: prediction matrix generated from svd
    # user_id: user_id
    # numOfMovies: number of movies to recommend, default = 20
# Output: list of recommended movies

def svd_generate_movies(pred, user_id, numOfMovies = 20):
    user_rating = pred.iloc[user_id - 1, :]
    toplist = user_rating.sort_values(ascending = False).head(numOfMovies)
    toplist = pd.DataFrame(toplist).reset_index().rename(columns = {"index" : "movie_
    recommend_movies = toplist.movie_id.to_list()
    movie_name = []
    for i in recommend_movies:
        movie_name.append(movie_map.get(i+1))
    return movie_name
```

In [134…

```
user_item_matrix = utility_matrix(df, n_users, n_items)
```

In [135…

```
pred = svd_prediction(user_item_matrix)
```

In [136…

```
# Functionality: print out recommendations to users
# Input: user_id: user_id

def recommendation_existingUsers(user_id):
    print(" ")
    numOfRecom = int(input("Please enter the number of movie recommendations needed:
    print("Recommendation for User: ", user_id)
    # if the svd matrix has not been calculated
    if 'svd_matrix' not in globals():
        global user_item_matrix
        global svd_matrix
        user_item_matrix = utility_matrix(df, n_users, n_items)
        svd_matrix = svd_prediction(user_item_matrix)
    rec = svd_generate_movies(svd_matrix, user_id, numOfRecom)
    print("Top", numOfRecom, "Movie Recommendation")
    print("----- ----- ----- ----- -----")
    for i in range(1, len(rec)+1):
        print(i, "   ", rec[i-1])
```

In [138…

```
# Test
recommendation_existingUsers(22)
```

```
Please enter the number of movie recommendations needed: 5
Recommendation for User:  22
Top 5 Movie Recommendation
----- ----- ----- ----- -----
1     Marlene Dietrich: Shadow and Light (1996)
2     Saint of Fort Washington, The (1993)
3     Aiqing wansui (1994)
4     They Made Me a Criminal (1939)
5     Someone Else's America (1995)
```

8.2 New User Recommdation

As explained previously, collaborative filtering has the cold start problem. Therefore, for new users, Content-Based filtering will be used to implement recommendation system for new users.

In [139…

```
# Functionality: generate recommended movies based on chosen genre

def recommendation_newUsers():
    print(" ")
    print("Genre Code         Genre")
    for i in range(1, len(genre_list)):
        print("    ",i, "           ", genre_list[i])
    genreCode = int(input("Please enter the code of movie genre you like: "))
    genre = genre_list[genreCode]
    recommendations_genre(genre)
```

In [140…

```
# Test
recommendation_newUsers()
```

```
Genre Code        Genre
     1            Action
     2            Adventure
     3            Animation
     4            Children
     5            Comedy
     6            Crime
     7            Documentary
     8            Drama
     9            Fantasy
     10            Film-Noir
     11            Horror
     12            Musical
     13            Mystery
     14            Romance
     15            Sci-Fi
     16            Thriller
     17            War
     18            Western
Please enter the code of movie genre you like: 5
Chosen Genre:  Comedy
Recommendation by Popularity:
Top 10 popular movies of the genre: Comedy
----- ----- ----- ----- ----- ----- ----- ----- ----- ----- ----- -----
                                    movie title  Number of Users watched
0                              Liar Liar (1997)                      485
1                              Toy Story (1995)                      452
2                     Back to the Future (1985)                      350
3  Willy Wonka and the Chocolate Factory (1971)                      326
4                    Princess Bride, The (1987)                      324
5                           Forrest Gump (1994)                      321
6        Monty Python and the Holy Grail (1974)                      316
7                        Full Monty, The (1997)                      315
8                          Men in Black (1997)                      303
9                         Birdcage, The (1996)                      293
Recommendation by Rating:
Top 10 high rating movies of the genre: Comedy
----- ----- ----- ----- ----- ----- ----- ----- ----- ----- ----- -----
                  movie title  Average Rating
0   Santa with Muscles (1996)        5.000000
1       Close Shave, A (1995)        4.491071
2  Wrong Trousers, The (1993)        4.466102
3   North by Northwest (1959)        4.284916
4      Shall We Dance? (1996)        4.260870
5   As Good As It Gets (1997)        4.196429
6      Cinema Paradiso (1988)        4.173554
7  Princess Bride, The (1987)        4.172840
8  Waiting for Guffman (1996)        4.127660
9        A Chef in Love (1996)        4.125000
```

8.3 Similar Movie Recommdation

Sometime users would try to find movies similar to one movie. Therefore, the Memory-Item-Based collaborative filtering model would be used to implement recommendation on similar movies

In [164…

```
# Functionality: generating user-item matrix with zero filled
# Input:
        # data: rating data,
        # n_users: number of users,
        # n_items: number of items,
# Output: user-item matrix
def utility_matrix_zero(data, n_users, n_items):
    util_mat = utility_matrix(data, n_users, n_items)
    return pd.DataFrame(util_mat).fillna(0)
```

In [165…

```
# Functionality: develop item-based knn model
# Input:
        # data: rating data,
        # n_users: number of users,
        # n_items: number of items,
# Output: user-item matrix, knn_model
def develop_item_knn_model(data, n_users, n_items):
    util_mat = utility_matrix(data, n_users, n_items)
    util_mat = pd.DataFrame(util_mat).fillna(0).T
    util_mat_sparse = csr_matrix(util_mat)
    item_knn_model = NearestNeighbors(metric='cosine', algorithm='brute')
    item_knn_model.fit(util_mat_sparse)
    return util_mat, item_knn_model
```

In [166…

```
# Functionality: generate recommended movies based on item similarity
# Input:
    # movie_id: movie_id
    # numOfMovies: number of movies to recommend, default = 20
# Output: list of recommended movies
def item_generate_movies(movie_id, numOfMovies = 20):
    if 'item_knn_model' not in globals():
        global movie_user_matrix
        global item_knn_model
        movie_user_matrix, item_knn_model = develop_item_knn_model(df, n_users, n_ite
    knn_input = np.asarray([movie_user_matrix.values[movie_id-1]])
    n = min(n_items-1, numOfMovies)
    distances, indices = item_knn_model.kneighbors(knn_input, n_neighbors=n+1)
    similar_item_list = indices.flatten()[1:]
    distance_list = distances.flatten()[1:]
    movie_name = []
    for i in similar_item_list:
        movie_name.append(movie_map.get(i+1))
    return movie_name
```

In [167…

```
movie_list = df["movie_title"].unique()
case_insensitive_movies_list = [i.lower() for i in movie_list]
```

In [168…

```
movie_lower_map = {}
for key in movie_map:
    val = movie_map.get(key).lower()
    movie_lower_map[val] = key
```

In [169…

```
# Functionality: if user inputs incorrect movie names, search for movies with similar
# Input:
    # movie name
# Output: list of movies with similar names
# Reference Code: https://github.com/rposhala/Recommender-System-on-MovieLens-dataset
def movies_with_similiar_names(inputName):
    name = ''
    searchRange = case_insensitive_movies_list.copy()
    for word in inputName:
        currentRange = []
        name += word
        for movie in searchRange:
            if (name in movie):
                currentRange.append(movie)
        if len(currentRange) == 0:
            return searchRange
        searchRange = currentRange.copy()
    return searchRange
```

In [170…

```
# Functionality: provide recommendation based on user input movie name
# Output: list of movies similar to the input movie by knn model
```

```python
def recommendation_movies():
    print(" ")
    inputName = input("Please enter the name of the movie you like: ")
    inputName = inputName.lower()
    if inputName in case_insensitive_movies_list:
        movie_id = movie_lower_map.get(inputName)
        numOfRecom = int(input("Please enter the number of movie recommendations need
        rec = item_generate_movies(movie_id, numOfRecom)
        print("Top", numOfRecom, "Similar to Movie ", movie_map[movie_id])
        print("----- ----- ----- ----- ----- ----- ----- ----- ----- -----")
        for i in range(1, len(rec)+1):
            print(i, "    ", rec[i-1])
    else:
        suggest_list = movies_with_similiar_names(inputName)
        if len(suggest_list) == len(movie_list):
            print("The movie name you entered is not in the database.")
        else:
            print("The movie name you entered is not found.")
            print("Please check the following suggestions: ")
            for m in suggest_list:
                print(m)
            qs = input("Please press Q to end, press S to restart searching: ").lower
            if qs == 'q':
                return
            else:
                recommendation_movies()
```

In [182…
```python
# Test
recommendation_movies()
```

```
Please enter the name of the movie you like: titanic
The movie name you entered is not found.
Please check the following suggestions:
titanic (1997)
Please press Q to end, press S to restart searching: s

Please enter the name of the movie you like: titanic (1997)
Please enter the number of movie recommendations needed: 5
Top 5 Similar to Movie  Titanic (1997)
----- ----- ----- ----- ----- ----- ----- ----- ----- -----
1      Good Will Hunting (1997)
2      Contact (1997)
3      Apt Pupil (1998)
4      Tomorrow Never Dies (1997)
5      Air Force One (1997)
```

8.4 Recommdation Engine

A recommendation engine that combines previous three recommendation functions

In [174…
```python
rec_list = ["Genre Recommendation", "Similar Movie Recommendation", "General Recommen
```

In [181…
```python
def recommendation_engine():
    print(" ")
    print("Hi, I am here to help you choose movies")
    user_id = int(input("Please enter your user_id. If you are new user, please enter
    print(" ")
    if user_id == -1 :
        print("Hello there and welcome!")
        print("I could recommend some movies to you based on specific movie genre you
        recommendation_newUsers()
    else:
        print("Hello, welcome back!")
        print("Rec Code          Recommendation")
```

```
        for i in range(1, len(rec_list)+1):
            print("    ",i, "            ", rec_list[i-1])
        rec = int(input("Please enter the code of recommendation type you like:"))
        if (rec == 1):
            recommendation_newUsers()
        elif (rec == 2):
            recommendation_movies()
        else:
            recommendation_existingUsers(user_id)
    print(" ")
    print("Here's my recommendations. I hope you find it enjoyable!")
```

In [180…

```
recommendation_engine()
```

```
Hi, I am here to help you choose movies
Please enter your user_id. If you are new user, please enter -112

Hello, welcome back!
Rec Code        Rec
     1             Genre Recommendation
     2             Similar Movie Recommendation
     3             General Recommendation
Please enter the code of recommendation type you like:1

Genre Code         Genre
     1             Action
     2             Adventure
     3             Animation
     4             Children
     5             Comedy
     6             Crime
     7             Documentary
     8             Drama
     9             Fantasy
     10             Film-Noir
     11             Horror
     12             Musical
     13             Mystery
     14             Romance
     15             Sci-Fi
     16             Thriller
     17             War
     18             Western
Please enter the code of movie genre you like: 2
Chosen Genre:  Adventure
Recommendation by Popularity:
Top 10 popular movies of the genre: Adventure
----- ----- ----- ----- ----- ----- ----- ----- ----- ----- ----- -----
                                    movie title  Number of Users watched
0                               Star Wars (1977)                      583
1                      Return of the Jedi (1983)                      507
2                 Raiders of the Lost Ark (1981)                      420
3                               Rock, The (1996)                      378
4                Empire Strikes Back, The (1980)                      367
5                Star Trek: First Contact (1996)                      365
6                     Mission: Impossible (1996)                      344
7     Indiana Jones and the Last Crusade (1989)                      331
8  Willy Wonka and the Chocolate Factory (1971)                      326
9                     Princess Bride, The (1987)                      324
Recommendation by Rating:
Top 10 high rating movies of the genre: Adventure
----- ----- ----- ----- ----- ----- ----- ----- ----- ----- ----- -----
                                movie title  Average Rating
0                            Star Kid (1997)        5.000000
1                           Star Wars (1977)        4.358491
2             Raiders of the Lost Ark (1981)        4.252381
3                  Lawrence of Arabia (1962)        4.231214
4            Empire Strikes Back, The (1980)        4.204360
5                  African Queen, The (1951)        4.184211
6                 Princess Bride, The (1987)        4.172840
7                   Great Escape, The (1963)        4.104839
8  Treasure of the Sierra Madre, The (1948)        4.100000
9                     Wizard of Oz, The (1939)        4.077236

Here's my recommendations. I hope you find it enjoyable!
```

PART IX: Challenges (Question 8)

9.1 Challege on Testing

The most challenging part of the project is testing the model performance. Developing the memory-based, model-based, and content-based models is easy because there are existing algorithms such as SVD and KNN. Testing is especially hard because the only important feature in the data is the rating. Rating is the predictor as well as the label. What makes testing even harder is that the user-item matrix is very sparse. For zeroes in the matrix, we need pay special attention on whether to replace zero with movie mean rating, user mean rating, or let it stay zero.

For the model-based model, I replaced zero with movie mean rating and then normalized columns of the user-item matrix before performing the SVD. This is to make sure that missing rating will not be interpreted as low rating by the algorithm.

For the memory-based model, I keep zero as zero while using KNN to find close neighbors. This is because missing rating will not influence similarity calculation. However, after getting the neighbor list, extra calculations are perfomed to consider neighbor's weight, neighbor's average rating, and the user's average rating.

For the content-based model, it is meaningless to test the model as there isn't any prediction based on one user's preferences.

9.2 Challege on RMSE

The best RMSE score I got from all these models is around 1.026661. However, in some research studeis (Salam & Najafi, 2016) working on the same movie dataset, they achieved a RMSE of 0.905520. Since they did not disclose the code, I could not figure out the exact reason to explain the difference. However, my guess is that the difference is likely to come from different methods of testing or different ways of handeling missing ratings in the matrix. https://kth.diva-portal.org/smash/get/diva2:927356/FULLTEXT01.pdf

PART X: Potential Benefits & Harms (Question 9)

Potential benefits of a good recommendation system:

- For users, it means better using experience as users will spend less time searching for movies.
- For companies, better targeting means more customers and more revenue.

Potential harms of a recommendation system:

- Echo chamber: users may only exposed to content that reinforces their existing beliefs and interests. This could increase bias and lead to more misinformation.
- Privacy concern: recommendation system relies on users' peronsal data. Some companies may use personal data for commerical purposes without users' consent.
- Algorithm manipulation: if the recommendation algorithm is not transparent, some poeple or some group may intentionally manipute the algorithm to gain personal interests.

PART XI: Going Forward (Question 10)

I started working on this topic to complete a homework, but I had lots of lots of fun and excitement along the process. Therefore, I decided to continue exploring on this topic. Future work I have in mind:

- Read more research paper on this topic and discover new ways to improve the model.
- Develop a Neural Network model on Recommendation System.

- Use a cloud server such AWS to run the model on larger datasets and probably go beyond movie data.
- Write a Recommendation System API in Python or Java.

In [ ]:
```

```