Load and Process Trainning Data

In [2]:
```
import numpy as np
import pandas as pd
```

In [3]:
```
case = 1 #1: user-based cosine similarity; 2: user-based Pearson correlatio
```

In [4]:
```
is_IUF = False #set as true when testing IUF Pearson Correlation
```

In [5]:
```
is_std = True #set as true when testing my algorithm(movie controversy)
```

In [6]:
```
case_modification = True# set as true when testing case modification
```

In [7]:
```
trainData = pd.read_csv('train.txt', delim_whitespace=True, header=None)
```

In [8]:
```
trainData.head()
```

Out[8]:

	0	1	2
0	1	1	5
1	1	2	3
2	1	4	3
3	1	5	3
4	1	6	5

In [9]:
```
trainData.columns = ["userId", "movieId", "rating"]
```

In [10]:
```
trainData.head()
```

Out[10]:

	userId	movieId	rating
0	1	1	5
1	1	2	3
2	1	4	3
3	1	5	3
4	1	6	5

In [11]:
```
traindf = trainData.pivot_table(index=["userId"],columns=["movieId"],values
```

In [12]:
```
traindf.head()
```

Out[12]:

movieId	1	2	3	4	5	6	7	8	9	10	...	991	992	993	994	995
userId																
1	5.0	3.0	NaN	3.0	3.0	5.0	NaN	1.0	5.0	3.0	...	NaN	NaN	NaN	NaN	NaN
2	4.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	2.0	...	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
5	4.0	3.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN

5 rows × 994 columns

In [13]:
```
#Run this block when modifying Pearson Correlation with IUF in 1.2
#create a movie map to store IUF for each movie
import math
movieMap = [0] * 1001
for movie in range(1, 1001):
    count = 0
    for user in range(1, 201):
        if movie in traindf.columns and traindf.at[user, movie] > 0:
            count += 1
    if count > 0:
        movieMap[movie] = math.log10(200 / count)
```

In [14]:
```
if (is_IUF): #create a matrix for rating with IUF weight when testing IUF
    traindf_IUF = traindf.copy()
    for user in range(1, 201):
        for movie in range(1, 1001):
            if movie in traindf.columns and traindf_IUF.at[user, movie] > 0
                traindf_IUF.at[user, movie] *= movieMap[movie]
```

Load and Process Test Data

In [15]:
```
test5 = pd.read_csv('test5.txt', delim_whitespace=True, header=None) #when
```

In [16]:
```
test5.head()
```

Out[16]:

	0	1	2
0	201	237	4
1	201	268	5
2	201	306	5
3	201	331	5
4	201	934	5

In [17]:
```
test5.columns = ["userId", "movieId", "rating"]
```

In [18]:
```
test5.head()
```

Out[18]:

	userId	movieId	rating
0	201	237	4
1	201	268	5
2	201	306	5
3	201	331	5
4	201	934	5

In [19]:
```
test5_df = test5.pivot_table(index=["userId"],columns=["movieId"],values="r
```

In [20]:
```
test5_df.head()
```

Out[20]:

movieId	1	2	3	4	5	6	7	8	9	10	...	989	990	991	993	994
userId																
201	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
202	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
203	NaN	NaN	NaN	NaN	NaN	NaN	0.0	NaN	0.0	NaN	...	NaN	NaN	NaN	NaN	NaN
204	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
205	NaN	0.0	NaN	NaN	0.0	NaN	NaN	0.0	NaN	NaN	...	NaN	NaN	NaN	NaN	0.0

5 rows × 936 columns

In [21]:
```
#initialize a 2D array store the movieID with known rating for each test us
test_user_known_movie = []
for i in range(501):
    test_user_known_movie.append([])
```

In [22]:
```
for user in range(201, 301): #for different test data file, could change th
    for movie in range(1, 1001):
        if movie in test5_df.columns and test5_df.at[user, movie] > 0:
            test_user_known_movie[user].append(movie)
```

In [23]:
```
test_user_known_movie[201] #use a sample userId to check the array is creat
```

Out[23]:
```
[237, 268, 306, 331, 934]
```

In [24]:
```
if (is_IUF):#create a matrix for rating with IUF weight when testing IUF
    testdf_IUF = test5_df.copy()
    for user in range(201, 301): #change the user range when testing differ
        for movie in range(1, 1001):
            if movie in testdf_IUF.columns and testdf_IUF.at[user, movie] >
                testdf_IUF.at[user, movie] *= movieMap[movie]
```

In [39]:
```
movie_std = traindf.std(ddof = 0) #used when implementing my own algorithm
movie_std = movie_std.to_dict()
```

In [26]:
```
if (is_std): #create a traindf for rating with std weight (the larger std,
    traindf_std = traindf.copy()
    for user in range(1, 201):
        for movie in range(1, 1001):
            if movie in traindf.columns and traindf_std.at[user, movie] > 0
                traindf_std.at[user, movie] *= math.log10(1 + movie_std[mov
```

In [27]:
```
if (is_std): #create a testdf for rating with std weight (the larger std, t
    testdf_std = test5_df.copy()
    for user in range(201, 301): #change the user range when using differen
        for movie in range(1, 1001):
            if movie in testdf_std.columns and movie in traindf.columns and
                testdf_std.at[user, movie] *= math.log10(1 + movie_std[movi
```

Some helper functions

In [28]:
```
#calculate cosine similarity
def cal_cosine(vector1, vector2):
    if len(vector1) == 1: #edge case: only 1 dimension, add another dimensi
        if abs(vector1[0] - vector2[0]) >= 3:
            vector1.append(1)
            vector2.append(-1)
        elif abs(vector1[0] - vector2[0]) == 0:
            return 1
        else:
            vector1.append(1)
            vector2.append(1)
    numerator = np.dot(vector1, vector2)
    denominator = np.sqrt(np.dot(vector1, vector1)) * np.sqrt(np.dot(vector
    return numerator / denominator
```

In [29]:
```
def getK(similarity): #define k for top k similar user
    if (len(similarity) < 15):
        return len(similarity)
    count = 0
    for i in range(len(similarity)):
        if abs(similarity[i][1]) > 0.9:
            count = count + 1
    return max(count, 15) #return at least 15 most similar user or return a
```

In [30]:
```
def predict_rating(similarity, k, movie): #predict rating based on weighted
    sum_rating = 0
    sum_sim = 0
    if (case_modification): #apply case_modification to similarity/weight
        similarity = [list(simi) for simi in similarity]
        for ele in similarity:
            ele[1] = ele[1] * math.pow(abs(ele[1]), 2.5)

    for i in range(k):
        sum_rating += similarity[i][1] * traindf.at[similarity[i][0], movie
        sum_sim += similarity[i][1]
    return round(sum_rating / sum_sim)
```

In [31]:
```
def get_avg_rating(test_user_known_movie, user): #get test user's average r
    sum = 0;
    for m in test_user_known_movie[user]:
        sum += test5_df.at[user, m]
    return round(sum / len(test_user_known_movie[user]))
```

In [32]:
```
def get_test_avg(test_user, testdf): # get test user's average rating witho
    test_sum = 0
    test_count = 0
    for i in testdf.loc[test_user]:
        if i > 0:
            test_sum += i
            test_count += 1
    return test_sum / test_count
```

In [33]:
```
def normalize_vector(train_vector, test_vector, traindf, train_user, testdf
    if (is_IUF):
        train_avg = get_train_avg(train_user, traindf_IUF)
        test_avg = get_test_avg(test_user, testdf_IUF)
    elif (is_std):
        train_avg = get_train_avg(train_user, traindf_std)
        test_avg = get_test_avg(test_user, testdf_std)
    else:
        train_avg = get_train_avg(train_user, traindf)
        test_avg = get_test_avg(test_user, testdf)

    for rating in train_vector:
        rating -= train_avg
    for rating1 in test_vector:
        rating1 -= test_avg
```

In [34]:
```
def isZero(rating_vector): #check number in vector are all 0s or not
    for rating in rating_vector:
        if rating > 0:
            return False
    return True
```

In [35]:
```
def get_train_avg(train_user, traindf): #get the average rating for user in
    train_sum = 0
    train_count = 0
    for i in traindf.loc[train_user]:
        if i > 0:
            train_sum += i
            train_count += 1
    return train_sum / train_count
```

In [36]:
```
def predict_pearson_rating(similarity, k, movie, test_user,test5_df, traind
    active_avg = get_test_avg(test_user, test5_df)
    sum_rating = 0
    sum_sim = 0
    if (case_modification): #apply case_modification to similarity/weight
        similarity = [list(simi) for simi in similarity]
        for ele in similarity:
            ele[1] = ele[1] * math.pow(abs(ele[1]), 2.5)

    for i in range(k):
        sum_rating += similarity[i][1] * (traindf.at[similarity[i][0], movi
        sum_sim += abs(similarity[i][1])
    result = round(active_avg + sum_rating / sum_sim)

    if result <= 0: #edge case: when Pearson result is <=0 or greater than
        return 1
    if result > 5:
        return 5
    return result
```

Algorithms --User-based Collaborating Filtering

In [37]:
```
#user-based collaborating filtering based on basic cosine similarity and Pe
result5 = []
for user in range(201, 301): #change the range when testing different files
    for movie in range(1, 1001):
        if movie in test5_df.columns and test5_df.at[user, movie] == 0:#fin
            similarity = []
            for train_user in range(1, 201):
                test_user_vector = []
                train_user_vector = []
                if movie in traindf.columns and traindf.at[train_user, movi
                    for m in test_user_known_movie[user]:
                        if m in traindf.columns and traindf.at[train_user,
                            if (is_IUF):#use IUF rating dataframe when requ
                                train_user_vector.append(traindf_IUF.at[tra
                                test_user_vector.append(testdf_IUF.at[user,
                            elif (is_std):#use std rating dataframe when ca
                                train_user_vector.append(traindf_std.at[tra
                                test_user_vector.append(testdf_std.at[user,
                            else:
                                train_user_vector.append(traindf.at[train_u
                                test_user_vector.append(test5_df.at[user, m
                    if len(train_user_vector) > 0: #only select the train u
                        if case == 2:
                            normalize_vector(train_user_vector, test_user_v
                            if isZero(train_user_vector) or isZero(test_use
                                continue
                            else:
                                similarity.append((train_user, cal_cosine(t
                        elif case == 1:
                            similarity.append((train_user, cal_cosine(train
            if len(similarity) > 0:
                similarity.sort(key=lambda x:abs(x[1]), reverse = True)
                k = getK(similarity)
                if case == 1:
                    result5.append((user, movie, predict_rating(similarity,
                if case == 2:
                    result5.append((user, movie, predict_pearson_rating(sim
            else: #if can't find any eligible similar user, use average rat
                result5.append((user, movie, get_avg_rating(test_user_known
```

Algorithms -- Item-based Collaborating Filtering

In [835]:
```
def cal_adjust_cosine(movie, m, similarity):
    #create two vectors with rating for both movie and m on the same user,
    test_movie_vector = []
    train_movie_vector = []
    for train_user in range (1, 201):
        if movie in traindf.columns and m in traindf.columns and traindf.at
            user_avg = get_train_avg(train_user, traindf)
            test_movie_vector.append(traindf.at[train_user, movie] - user_a
            train_movie_vector.append(traindf.at[train_user, m] - user_avg)
    if len(test_movie_vector) > 0:
        if not isZero(train_movie_vector) and not isZero(test_movie_vector)
            similarity.append((m, cal_cosine(train_movie_vector, test_movie
    return similarity
```

In [836]:
```
def predict_item_rating(similarity, movie, user):
    active_avg = get_test_avg(user, test5_df)
    sum_rating = 0
    sum_sim = 0
    for i in range(len(similarity)): #min(len(similarity, 10))
        sum_rating += similarity[i][1] * (test5_df.at[user, similarity[i][0
        sum_sim += abs(similarity[i][1])
    result = round(active_avg + sum_rating / sum_sim)
    if result <= 0: #edge case: when Pearson result is <=0 or greater than
        return 1
    if result > 5:
        return 5
    return result
```

In [866]:
```
result5 = []
for user in range(201, 301): #change the range when testing different files
    for movie in range(1, 1001):
        if movie in test5_df.columns and test5_df.at[user, movie] == 0:
            similarity = [] #movie id with rating for the test user; simila
            for m in test_user_known_movie[user]:
                similarity = cal_adjust_cosine(movie, m, similarity)#only c
            if len(similarity) > 0:
                result5.append((user, movie, predict_item_rating(similarity
            else: #if can't find any eligible similar movie, use user's ave
                result5.append((user, movie, get_avg_rating(test_user_known
```

Algorithms -- The Slope One Algorithm

In [143]:
```
def get_avg_deviation(movie, user): #get avg deviation for current movie to
    movie_dev_map = []
    for i in range(1001):
        movie_dev_map.append([])
    for m in test_user_known_movie[user]:
        count = 0
        sum_dev = 0
        for train_user in range (1, 201):
            if m in traindf.columns and movie in traindf.columns and traind
                count += 1
                sum_dev += traindf.at[train_user, movie] - traindf.at[train
        if count != 0:
            movie_dev_map[m].append(sum_dev / count)
            movie_dev_map[m].append(count)
    return movie_dev_map
```

In [178]:
```
def predict_weighted_slope_one(movie, user): #weighted slope one
    movie_dev_map = get_avg_deviation(movie, user)
    weighted_sum = 0
    num_user = 0

    for m in test_user_known_movie[user]:
        # if can't find same user in trainning data rate both m and movie,
        if len(movie_dev_map[m]) == 0:
            weighted_sum += test5_df.at[user, m]
            num_user += 1
        else:
            weighted_sum += (test5_df.at[user, m] + movie_dev_map[m][0]) *
            num_user += movie_dev_map[m][1]

    result = round (weighted_sum / num_user)
    if result > 5:
        return 5
    if result <= 0:
        return 1
    return result
```

In [203]:
```
result5 = []
for user in range(201, 301): #change the range when testing different files
    for movie in range(1, 1001):
        if movie in test5_df.columns and test5_df.at[user, movie] == 0:
            result5.append((user, movie, predict_weighted_slope_one(movie,
```

Check Result and Write to Output File

In [43]:
```
result5[0:10]
```

Out[43]:
```
[(201, 1, 4),
 (201, 111, 4),
 (201, 283, 4),
 (201, 291, 3),
 (201, 305, 4),
 (201, 361, 5),
 (201, 475, 4),
 (201, 740, 3),
 (202, 259, 3),
 (202, 288, 3)]
```

In [370]:
```
f = open('output.txt', 'w') #write result to output txt files
for ele in result5:
  line = ' '.join(str(x) for x in ele)
  f.write(line + '\n')
f.close()
```

Validate Result Files

In [408]:
```
resultData = pd.read_csv('result5.txt', delim_whitespace=True, header=None)
resultData.columns = ["userId", "movieId", "rating"]
resultData.head()
```

Out[408]:

	userId	movieId	rating
0	201	1	5
1	201	111	5
2	201	283	5
3	201	291	4
4	201	305	5

In [409]:
```
resultData.info() #check if missing some rows
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7997 entries, 0 to 7996
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   userId   7997 non-null   int64
 1   movieId  7997 non-null   int64
 2   rating   7997 non-null   int64
dtypes: int64(3)
memory usage: 187.6 KB
```

In [410]:
```
boollist = pd.isnull(resultData['rating']) #check there's null values or no
resultData[boollist]
```

Out[410]:

	userId	movieId	rating

In [411]:
```
resultData.loc[resultData['rating'] > 5] #validate rating
```

Out[411]:

	userId	movieId	rating

In [412]:
```
resultData.loc[resultData['rating'] <= 0] #vaidate rating
```

Out[412]:

	userId	movieId	rating

Combining multiple algorithms's results to get average rating

In [434]:
```
result5_my_alogrithm = pd.read_csv('result5.txt', delim_whitespace=True, he
result5_my_alogrithm.columns = ["userId", "movieId", "rating"]
result5_my_alogrithm.head()
```

Out[434]:

	userId	movieId	rating
0	201	1	5
1	201	111	5
2	201	283	5
3	201	291	4
4	201	305	5

In [435]:
```
result5_slope = pd.read_csv('result5_slope.txt', delim_whitespace=True, hea
result5_slope.columns = ["userId", "movieId", "rating"]
result5_slope.head()
```

Out[435]:

	userId	movieId	rating
0	201	1	5
1	201	111	4
2	201	283	5
3	201	291	4
4	201	305	5

In [436]:
```
result5_cosine = pd.read_csv('result5_cosine.txt', delim_whitespace=True, h
result5_cosine.columns = ["userId", "movieId", "rating"]
result5_cosine.head()
```

Out[436]:

	userId	movieId	rating
0	201	1	4
1	201	111	4
2	201	283	4
3	201	291	3
4	201	305	4

In [439]:
```
result5 = []
for row in range(len(result5_my_alogrithm)): # each
    combine_rating = round((result5_my_alogrithm.iloc[row]["rating"] + resu
    result5.append((result5_my_alogrithm.iloc[row]["userId"], result5_my_al
```

In [44]:
```
result5[0:10]
```

Out[44]:
```
[(201, 1, 4),
 (201, 111, 4),
 (201, 283, 4),
 (201, 291, 3),
 (201, 305, 4),
 (201, 361, 5),
 (201, 475, 4),
 (201, 740, 3),
 (202, 259, 3),
 (202, 288, 3)]
```

In [441]:
```
f = open('output.txt', 'w') #write result to output txt files
for ele in result5:
  line = ' '.join(str(x) for x in ele)
  f.write(line + '\n')
f.close()
```

In [ ]: