Evan Chou

Professor Sanchez Munoz

Computer Engineering 169

12 March 2021

Project 2

Instructions to run my code if desired:

Simply run predict.py (placed in the Coen169 Project 2 folder) in VScode by hitting the run button (must download the entire folder for it to run correctly). Before running the code, please make sure to comment out the algorithm functions that you don't want to run. For instance, if you want to run ub_cosine_similarity, please comment out ub_pearson_correlation("test5.txt"), ub_pearson_correlation("test10.txt"), ub_pearson_correlation("test20.txt"), pearson_correlation_IUF("test5.txt"), ... ownAlgorithm("test20.txt"). This will ensure that you write in and only get the 3 correct result files for ub_cosine_similarity. Also, please make sure your file_path in line 8 is tailored to wherever you downloaded my Coen169 Project 2 folder (the current path is tailored for my own computer). Please email echou@scu.edu for any problems or questions.

1. Submit your first run by the first deadline via submit.html.

Submitted first run by first deadline on February 21st, 2021:

Hello, Chou, Evan

The following is the summary of your submissions:

MAE of GIVEN 5 : 1.02550956608728
MAE of GIVEN 10 : 0.991333333333333
MAE of GIVEN 20 : 0.98350535352561
OVERALL MAE : 0.999220160893121

You have already submitted 1 times.

GOOD LUCK!

I submitted only to test to see if my pin and files worked.

Supporting Functions For Tasks 2-4:

predict.py

Users › evanchou › Desktop › Coen169 Project 2 › predict.py

```python
#Evan Chou
#COEN169 Project 2
#Due-date: 3/12/2021

import numpy as np #contains functions for linear algebra and matrix that's needed for math formulas
import pandas as pd #for rows and columns

file_path = "/Users/evanchou/Desktop/Coen169 Project 2/"

def train_data():
    with open(file_path+"train.txt", "r") as train_file:
        data = [list(map(int,line.split())) for line in train_file]
    return list(data)

def write_data(data, file_name):
    export_list = []
    i = 0
    for user, movie, rating in data:
        export_string = f"{user} {movie} {rating}\n"
        export_list.append(export_string)

    fp = open(file_path+file_name.replace("test","result"), 'w')
    fp.writelines(export_list)

def test_data(file_name):
    with open(file_path+file_name, "r") as train_file:
        data = [list(map(int,line.split())) for line in train_file]
    return list(data)

def removingZeros(a, b):
    removea = np.array([])
    removeb = np.array([])
    for first1, second2 in zip(a,b):
        if first1 and second2:
            removea = np.append(removea, first1)
            removeb = np.append(removeb, second2)
    return removea, removeb
```

```python
def cosineSimilarity(a, b):
    testa, testb = removingZeros(a, b) #call removing zeroes function to remove 0
    #return 0 if either test a or b is all 0
    if len(testa) == 0 or len(testb) == 0:
        return 0.0
    #cosine similarity formula
    numerator = np.dot(a, b) #dot product
    denominator = np.linalg.norm(testa)*np.linalg.norm(testb) #denominator
    return numerator/denominator

def pearsonCorrelation(a, b):
    testa, testb = removingZeros(a, b) #call removing zeroes function to remove 0
    #return 0 if either test a or b is all 0
    if len(testa) == 0 or len(testb) == 0:
        return 0.0
    #subtract average rating from original rating
    x = np.mean(testa)
    y = np.mean(testb)

    testa = testa - x
    testb = testb - y

    #cosine similarity formula
    numerator = np.dot(a, b) #dot product
    denominator = np.linalg.norm(testa)*np.linalg.norm(testb) #denominator
    return 0.0 if denominator==0 else numerator/denominator #denominator could be 0 if we subtract weight_b_avg

def weightedAverage(sw, r, absValue=False): #sw is similar weights and r is rating
    if np.sum(sw) == 0:
        return 0
    if absValue:
        return np.sum(sw*r)/np.sum(np.absolute(sw)) #Pearson: use absolute value to take into consideration users that are really different
    return np.sum(np.array(sw)*np.array(r))/np.sum(sw) #regular weighted average for basic cosine similarity

```

```python
def user_row(data, size):
    temp = [0] * size
    for d in data:
        if d[1] >0:
            temp[d[0]-1] = d[1]
    return temp

def rounding(val): #how I am rounding the ratings
    #if the rating is 0, put rating of 3
    if val == 0:
        return 3
    #if the rating is less than 1, put rating of 1
    elif val < 1:
        return 1
    #if the rating is greater than 5, put rating of 5
    elif val > 5:
        return 5
    else:
        return round(val)

```

2.1 Implement the basic user-based collaborative filtering algorithms

User-based Cosine Similarity:

Result:

Hello, Chou, Evan

The following is the summary of your submissions:

MAE of GIVEN 5 : 0.824037018509255
MAE of GIVEN 10 : 0.789298216369395
MAE of GIVEN 20 : 0.769438549102836
OVERALL MAE : 0.79224990763926

You have already submitted 6 times.

GOOD LUCK!

Code:

```python
#user-based cosine similarity
def ub_cosine_similarity(filename):
    trainData = train_data() #this is our training data
    testData = test_data(filename) #this is our testing data
    cols = len(trainData[0]) #cols = columns
    userSimilarity = [] #user similarity
    user_ids = list(set(j[0] for j in testData)) #specific user ID
    result=[]
    for u in user_ids:
        a = user_row([ [j[1],j[2]] for i, j in enumerate(testData) if j[0]==u], cols)
        userSimilarity = []
        for b in trainData:
            userSimilarity.append(cosineSimilarity(a,b))

        for m, index in [[j[1], i] for i, j in enumerate(testData) if j[0]==u and j[2]==0]: # movie with rate 0 in index row
            weight_a, weight_b = removingZeros(userSimilarity, [y[m-1] for x, y in enumerate(trainData)])
            result.append([u, m, rounding(weightedAverage(weight_a, weight_b))])
    write_data(result, filename)

ub_cosine_similarity("test5.txt")
ub_cosine_similarity("test10.txt")
ub_cosine_similarity("test20.txt")
```

User-based Pearson Correlation:

Result:

Hello, Chou, Evan

The following is the summary of your submissions:

MAE of GIVEN 5 : 0.880190095047524
MAE of GIVEN 10 : 0.828138023003834
MAE of GIVEN 20 : 0.813332047076982
OVERALL MAE : 0.838922868519355

You have already submitted 7 times.

GOOD LUCK!

Code:

```
#user-based pearson correlation
def ub_pearson_correlation(filename):
    trainData = train_data() #this is our training data
    testData = test_data(filename) #this is our testing data
    cols = len(trainData[0]) #cols = columns
    userSimilarity = [] #user similarity
    user_ids = list(set(j[0] for j in testData)) #specific user ID
    result=[]
    for u in user_ids:
        a = user_row([ [j[1],j[2]] for i, j in enumerate(testData) if j[0]==u], cols)
        averageRating = np.mean([j[2] for i, j in enumerate(testData) if j[0]==u and j[2]>0])
        userSimilarity = []
        for b in trainData:
            userSimilarity.append(pearsonCorrelation(a,b))

        for m, index in [[j[1], i] for i, j in enumerate(testData) if j[0]==u and j[2]==0]: # movie with rate 0 in index row
            weight_a, weight_b = removingZeros(userSimilarity, [y[m-1] for x, y in enumerate(trainData)])
            weight_b_avg = np.mean(weight_b) if len(weight_b)>0 else 0
            weight_b = [x - weight_b_avg for x in weight_b]
            result.append([u, m, rounding(weightedAverage(weight_a, weight_b, True) + averageRating)])
    write_data(result, filename)

ub_pearson_correlation("test5.txt")
ub_pearson_correlation("test10.txt")
ub_pearson_correlation("test20.txt")
```

2.2 Extensions to the basic user-based collaborative filtering algorithms

<u>Pearson Correlation with IUF (Inverse User Frequency):</u>

Result:

Hello, Chou, Evan

The following is the summary of your submissions:

MAE of GIVEN 5 : 0.933091545772886
MAE of GIVEN 10 : 0.882647107851309
MAE of GIVEN 20 : 0.863496044761721
OVERALL MAE : 0.891055375395099

You have already submitted 8 times.

GOOD LUCK!

Code:

```python
#pearson correlation with IUF
def pearson_correlation_IUF(filename):
    trainData = train_data() #this is our training data
    testData = test_data(filename) #this is our testing data
    cols = len(trainData[0]) #cols = columns
    userSimilarity = [] #user similarity
    iuf = [] #IUF empty array in order to implement IUF later
    user_ids = list(set(j[0] for j in testData)) #specific user ID
    result=[]

    #IUF log(m/mj)
    m = len(trainData)
    train_t=pd.DataFrame(trainData).T.values.tolist()
    for x in train_t:
        mj=len([r for r in x if r>0])
        iuf.append(np.log(m/mj) if mj else 0.0)
    trainIUF = trainData * np.array(iuf) #multiply original ratings by IUF

    for u in user_ids:
        a = user_row([ [j[1],j[2]] for i, j in enumerate(testData) if j[0]==u], cols)
        averageRating = np.mean([j[2] for i, j in enumerate(testData) if j[0]==u and j[2]>0])
        userSimilarity = []
        for b in trainData:
            userSimilarity.append(pearsonCorrelation(a,b))

        for m, index in [[j[1], i] for i, j in enumerate(testData) if j[0]==u and j[2]==0]: # movie with rate 0 in index row
            weight_a, weight_b = removingZeros(userSimilarity, [y[m-1] for x, y in enumerate(trainIUF)])
            weight_b_avg = np.mean(weight_b) if len(weight_b)>0 else 0
            weight_b = [x - weight_b_avg for x in weight_b]
            result.append([u, m, rounding(weightedAverage(weight_a, weight_b, True) + averageRating)])
    write_data(result, filename)

pearson_correlation_IUF("test5.txt")
pearson_correlation_IUF("test10.txt")
pearson_correlation_IUF("test20.txt")
```

Pearson Correlation with Case Modification:

Result:

Hello, Chou, Evan

The following is the summary of your submissions:

MAE of GIVEN 5 : 0.910580290145073
MAE of GIVEN 10 : 0.870478413068845
MAE of GIVEN 20 : 0.851919737603704
OVERALL MAE : 0.875744017076475

You have already submitted 10 times.

GOOD LUCK!

Code:

```python
#pearson correlation with case modification
def pearson_correlation_caseModification(filename):
    trainData = train_data() #this is our training data
    testData = test_data(filename) #this is our testing data
    cols = len(trainData[0]) #cols = columns
    userSimilarity = [] #user similarity
    user_ids = list(set(j[0] for j in testData)) #specific user ID
    result=[]
    p = 2.5 #choosing 2.5 for value p

    for u in user_ids:
        a = user_row([ [j[1],j[2]] for i, j in enumerate(testData) if j[0]==u], cols)
        averageRating = np.mean([j[2] for i, j in enumerate(testData) if j[0]==u and j[2]>0])
        userSimilarity = []
        for b in trainData:
            userSimilarity.append(pearsonCorrelation(a,b))

        #pearson case modification formula
        userSimilarity = userSimilarity * pow(np.array(userSimilarity), p-1)

        for m, index in [[j[1], i] for i, j in enumerate(testData) if j[0]==u and j[2]==0]: #movie not rated in index row
            weight_a, weight_b = removingZeros(userSimilarity, [y[m-1] for x, y in enumerate(trainData)])
            weight_b_avg = np.mean(weight_b) if len(weight_b)>0 else 0
            weight_b = [x - weight_b_avg for x in weight_b]
            result.append([u, m, rounding(weightedAverage(weight_a, weight_b, True) + averageRating)])
    write_data(result, filename)

pearson_correlation_caseModification("test5.txt")
pearson_correlation_caseModification("test10.txt")
pearson_correlation_caseModification("test20.txt")
```

3. Item-Based Collaborative Filtering Algorithm

Item-Based Adjusted Cosine Similarity:

Result:

Hello, Chou, Evan

The following is the summary of your submissions:

MAE of GIVEN 5 : 0.872186093046523
MAE of GIVEN 10 : 0.806967827971329
MAE of GIVEN 20 : 0.797800501639977
OVERALL MAE : 0.824473543778991

You have already submitted 11 times.

GOOD LUCK!

Code:

```python
def itemBased_adjustedCosineSimilarity(filename):
    trainData = train_data() #this is our training data
    testData = test_data(filename) #this is our testing data
    cols = len(trainData[0]) #cols = columns
    userSimilarity = [] #user similarity
    user_ids = list(set(j[0] for j in testData)) #specific user ID
    result=[]
    train_avg = []

    #user average to subtract
    for x in trainData:
        x_sum = sum(x)
        x_count = len([r for r in x if r>0])
        train_avg.append(x_sum/x_count if x_count else 0.0)

    for u in user_ids:
        for m0, u0 in [[j[1], i] for i, j in enumerate(testData) if j[0]==u and j[2]==0]: #unknown movie (rate 0)
            item_sim = []
            for m1, u1 in [[l[1], k] for k, l in enumerate(testData) if l[0]==u and l[2]>0]: #known movie (rate>0)
                a = [y[m0-1]-train_avg[x] for x, y in enumerate(trainData)]
                b = [y[m1-1]-train_avg[x] for x, y, in enumerate(trainData)]
                item_sim.append(cosineSimilarity(a,b))
            weight_a, weight_b = removingZeros(item_sim, [y[2] for x, y in enumerate(testData) if y[0]==u and y[2]>0])
            result.append([u, m0, rounding(weightedAverage(weight_a, weight_b))])
    write_data(result, filename)

itemBased_adjustedCosineSimilarity("test5.txt")
itemBased_adjustedCosineSimilarity("test10.txt")
itemBased_adjustedCosineSimilarity("test20.txt")
```

4. Implement your own algorithm

<u>**Own Algorithm:**</u>

Initially, I wanted to improve on my User-Based Pearson Correlation by creating an algorithm to rank the movies and calculate and average using those rankings to subtract instead of average rating. However, this didn't yield good results and many of my code did not function correctly, so I decided to implement a complete new own algorithm.

First, I noticed that we were provided with a few data sets for a user (different for each test.txt). To explain my idea/algorithm better, I think it would be better to use user 201's data in test5.txt as an explanation.

```
201 237 4
201 306 5
201 361 5
201 475 3
201 934 5
201 1 0
201 111 0
201 268 0
201 283 0
201 291 0
201 305 0
201 331 0
201 740 0
```

(user 201 data from test5.txt)

For instance, the above image shows user 201's rating for each movie. If we are calculating user 201's rating for movie 1, my idea was to first look at movie 237's column in the training set data and filter out users that didn't give a rating of 3,4, or 5. In other words, I only looked at the users that gave a rating one-above or one-below the current user's rating (in this case 3,4, or 5 will be looked at because user 201's movie rating for 237 is 4). I would then look at ONLY those users' ratings for movie 1 and store them. Then I'd move on to movie 306 and do the same thing (look at users who gave movie 306 a rating of 4 or 5 and store their movie 1 ratings). In this case, we would be done collecting data after movie 934 (user 201 only rated 5 movies in test5.txt). With this accumulated data (all the ratings stored up), I then took the average of them and put this value as the prediction rating for user 201's movie 1 (this is only an explanation for user 201's movie 1 prediction). This method was used to calculate all the user rating predictions and ultimately gave the best algorithm MAE of 0.787.

Result:

Hello, Chou, Evan

The following is the summary of your submissions:

MAE of GIVEN 5 : 0.826788394197099
MAE of GIVEN 10 : 0.777129521586931
MAE of GIVEN 20 : 0.764132741655412
OVERALL MAE : 0.787898690529945

You have already submitted 12 times.

GOOD LUCK!

Code:

```
def ownAlgorithm(filename):
    trainData = train_data() #this is our training data
    testData = test_data(filename) #this is our testing data
    userSimilarity = []
    data = []
    user = ""
    result=[]
    for tt in testData:
        if user != tt[0]:
            userSimilarity = []
        for tn in trainData:
            if tt[2]>0 and tn[tt[1]-1]>0 and tt[2]>=tn[tt[1]-1]-1 and tt[2]<=tn[tt[1]-1]+1:
                userSimilarity.append(tn)

        if tt[2]==0:
            count=0
            rating=3.0
            sum=0.0
            for d in userSimilarity:
                if d[tt[1]-1]>0:
                    sum+=d[tt[1]-1]
                    count+=1
            result.append([tt[0], tt[1], round(float(sum)/float(count) if count>0 else 3)]) #put rating 3 if there are 0s
        user=tt[0]
    write_data(result, filename)

ownAlgorithm("test5.txt")
ownAlgorithm("test10.txt")
ownAlgorithm("test20.txt")
```

5. Results Discussion

The best result stemmed from using user-based cosine similarity for test5.txt and my own algorithm for both test10.txt and test20.txt, as seen below.

Best Result Combination:

Hello, Chou, Evan

The following is the summary of your submissions:

MAE of GIVEN 5 : 0.824037018509255
MAE of GIVEN 10 : 0.777129521586931
MAE of GIVEN 20 : 0.764132741655412
OVERALL MAE : 0.786995607733673

You have already submitted 13 times.

GOOD LUCK!

Out of the 5 filtering algorithms learned in class, user-based cosine similarity proved to give the best MAE, with a final MAE score of 0.792. For this specific test data, making my code less restricted was more beneficial. I initially was stricter in my cosine similarity, saying that we only care if users rated at least 2 movies, but the MAE was only 0.803. User-based pearson correlation was a bit worse, with an MAE of 0.838. I believe it was slightly worse than user-based cosine similarity because of the given test datas, with volume being the substantive interest. Also, since I rounded ratings, I suspect the average value used in pearson correlation to be less accurate and effective.

Pearson correlation with IUF (inverse user frequency) and pearson correlation with case modification were even worse, with a MAE of 0.891 and 0.875 respectively. This is most likely due to the fact that our data set isn't very large. IUF cares about more rare movies and assumes that popular movies are less impactful, but there were very few users with similar rare movie ratings. Hence, adding more adjustments to our original pearson correlation would reduce the accuracy of our predicted ratings.

Item-based adjusted cosine similarity was the second best filtering algorithm out of the ones taught from class, with a MAE of 0.824. However, compared to the other four algorithms, it had a bigger Big(O) runtime and took way longer to run because we had to traverse through each movie many times. As a result, the algorithm wasn't the best in terms of efficiency and time. Furthermore, it most likely performed worse than user-based cosine similarity because item-based requires a good amount of information of the item's own features rather than the user's interactions and feedback. Once again, we're limited in data, so it is expected for user-based cosine similarity to do better.

Finally, my own algorithm gave the best results compared to the other 5 algorithms, with an MAE of 0.787. I also wrote my code in simpler terms to make it both more efficient and take less time to run and generate the resulting files. With the train and test files provided, the best result was from taking users that gave a rating one-above or one-below the current user's rating, rather than the exact rating. Stricter logic and more restrictions proved to hinder prediction accuracy and give higher MAEs. For this reason, some future ideas would possibly be to compute error to determine how different user ratings are and then apply this to basic user-based cosine similarity.